PHP ET MYSQL MASTERCLASS

Le cours complet
éd. 2020

PIERRE GIRAUD
pierre-giraud.com

Préambule

De quoi traite ce cours ?

Dans ce cours, nous allons découvrir et apprendre à utiliser le langage de programmation PHP ainsi qu'un système de gestion de base de données célèbre basé que le langage SQL qui se nomme MySQL.

PHP et MySQL sont un duo célèbre car ils sont très puissants et relativement simples d'utilisation. On les utilise principalement dans un contexte web, notamment pour manipuler des données envoyées par l'utilisateur et rendre un site dynamique (PHP) et pour stocker des données (MySQL).

Quels sont les objectifs du cours et à qui s'adresse-t-il ?

Ce cours poursuit deux objectifs principaux : présenter de manière exhaustive les notions et fonctionnalités importantes du PHP et du MySQL afin que vous possédiez les outils et compétences nécessaires pour commencer à programmer dès la fin du cours et faire en sorte que vous montiez en compétence et soyez le plus autonome possible d'ici la fin du cours.

Pour servir ces deux objectifs, je vous propose un grand tour d'horizon des langages où chaque grande notion va être présentée, étudiée et illustrée et durant lequel je vais vous pousser à être le plus proactif possible.

En effet, plutôt que simplement effleurer ou pire éviter les notions relativement complexes, nous allons particulièrement nous attarder sur celles-ci et les décortiquer afin que vous compreniez véritablement comment fonctionne les langages.

En plus de cela, je vais vous proposer de nombreux exemples et exercices à réaliser par vous-même durant ce cours afin de vous forcer à vous confronter aux difficultés et afin que vous puissiez vous assurer d'avoir véritablement compris comment fonctionne tel ou tel concept.

Cette façon de procéder est selon moi la meilleure manière de vous rendre rapidement autonome. Si vous faites l'effort de prendre le temps de refaire les exemples et exercices, vous devriez être capable de réaliser la plupart de vos projets dès la fin du cours.

Ce cours s'adresse donc à toute personne curieuse et motivée par l'apprentissage du PHP et du MySQL. Comme le PHP et le SQL ne dépendent d'aucun autre langage et sont des langages relativement simples à apprendre et à comprendre, il n'y a pas de niveau ou de connaissance préalable à avoir pour suivre ce cours ; il est donc ouvert à tous.

Le seul prérequis nécessaire pour suivre ce cours dans de bonnes conditions est de connaitre à minima le rôle et la syntaxe de base du HTML. Avoir des connaissances en CSS est un plus.

Méthodologie et pédagogie

Le domaine de la programmation informatique est un domaine en constante évolution et qui évolue surtout de plus en plus vite. Il est donc essentiel qu'un développeur possède ou acquière des facultés d'adaptation et c'est la raison pour laquelle ce cours a pour but de vous rendre autonome.

Pour servir cet objectif, les différentes notions abordées dans ce cours sont illustrées par de nombreux exemples et exercices. Je vous conseille fortement de passer du temps sur chaque exemple et chaque exercice et de ne pas simplement les survoler car c'est comme cela que vous apprendrez le mieux.

En effet, en informatique comme dans beaucoup d'autres domaine, la simple lecture théorique n'est souvent pas suffisante pour maitriser complètement un langage. La meilleure façon d'apprendre reste de pratiquer et de se confronter aux difficultés pour acquérir des mécanismes de résolution des problèmes.

Ensuite, une fois ce cours terminé, pensez à rester curieux et à vous tenir régulièrement au courant des avancées des langages et surtout continuez à pratiquer régulièrement.

Plan et déroulement du cours

Ce cours contient 17 sections qui s'enchainent dans un ordre logique et cohérent. Je vous recommande donc de les suivre dans l'ordre proposé pour retirer le maximum de ce cours puisque certaines leçons vont réutiliser des notions vues dans les leçons précédentes.

Nous allons commencer par nous intéresser au PHP, en étudiant d'abord des concepts incontournables et communs à de nombreux langages de programmation comme les variables, les fonctions et les structures de contrôle.

Nous irons ensuite progressivement vers des notions plus pointues du PHP en apprenant à manipuler des dates, des fichiers et des données utilisateurs puis verrons finalement ce que sont les expressions régulières et comment utiliser leur puissance en PHP avant d'attaquer une grande partie sur la programmation orientée objet.

Dans la deuxième grande partie de ce cours, nous allons découvrir la syntaxe du SQL puis apprendre à manipuler des bases de données MySQL en utilisant du code PHP.

PARTIE I

Introduction au cours

Introduction au PHP et au MySQL

Dans cette toute première leçon, nous allons déjà définir ce que sont le PHP et MySQL ainsi que leurs rôles respectifs et allons avoir un premier aperçu de ce qu'on va pouvoir réaliser grâce au PHP et au MySQL.

Définition et rôle du PHP

Le terme PHP est l'acronyme de « PHP : Hypertext Preprocessor ». Le premier « P » de PHP est en effet lui-même l'abréviation de « PHP », une curiosité qui ne va pas présenter une grande importance pour nous.

Ce langage a été créé en 1994. Sa version stable la plus récente (au 15 octobre 2019) est la version 7.3.10. C'est la version sur laquelle je vais me baser dans ce cours. Notez qu'il est possible qu'à l'heure à laquelle vous lisiez ces lignes la version 7.4 soit sortie. Cela ne rend pas ce cours obsolète : les changements entre deux versions non majeures sont limités et la rétrocompatibilité est de toutes manières assurée pendant des années.

Le PHP va nous permettre de créer des pages qui vont être générées dynamiquement. En d'autres mots, grâce au PHP, nous allons pouvoir afficher des contenus différents sur une même page en fonction de certaines variables : l'heure de la journée, le fait que l'utilisateur soit connu et connecté ou pas, etc.

Pour illustrer cela, prenons l'exemple d'un espace client sur un site web e-commerce. Un utilisateur arrive sur un site e-commerce sur lequel il a déjà commandé et crée un espace client. Lors de son arrivée sur le site, il dispose d'un formulaire de connexion à son espace client.

Il va alors devoir fournir des informations (généralement un pseudonyme et un mot de passe) pour pouvoir se connecter et accéder à son espace client. Cet espace client est personnalisé : il va certainement contenir l'historique des commandes de l'utilisateur, son profil avec ses informations de facturation et son adresse de livraison, etc.

Ici, lorsque l'utilisateur rentre ses informations de connexion, celles-ci vont être traitées et analysées en PHP. On va vérifier si les informations sont bonnes et si c'est le cas récupérer des informations spécifiques à cet utilisateur et générer dynamiquement les pages de son espace client avec ces informations.

Lorsqu'un utilisateur fournit des informations comme une adresse, un numéro de téléphone ou passe une commande, les données sont généralement enregistrées dans ce qu'on appelle une base de données. Le PHP va également nous permettre d'aller récupérer des données dans une base de données pour s'en resservir.

De plus, notez que le PHP va s'exécuter côté serveur. Il fait ainsi partie des langages qu'on nomme « server side » en opposition aux langages « client side » qui s'exécutent côté client. Nous expliquerons ces notions en détail dans la prochaine leçon.

Sites statiques et sites dynamiques

Les langages de programmation axés web peuvent être catégorisés selon deux grands types de classement :

- Langages statiques VS langages dynamiques ;
- Langages avec exécution côté client VS langages avec exécution côté serveur.

Les sites dits statiques se caractérisent par le fait qu'ils sont… statiques : ils ne possèdent ni interaction, ni la capacité de s'adapter aux visiteurs. Le code des différentes pages ne va pas changer en fonction d'un utilisateur ou d'une autre variable.

Un site de type "CV" par exemple, ou un site servant simplement à présenter ou à donner des informations sur une chose en particulier vont généralement être des sites statiques car il n'y a aucune interaction ni adaptation dynamique avec le visiteur. Un site créé uniquement en HTML et en CSS par exemple sera toujours statique.

Les sites dynamiques, en revanche, vont pouvoir fournir des pages différentes pour chaque visiteur ou selon différentes contraintes et vont nous permettre d'interagir avec l'utilisateur en lui permettant de nous envoyer des données par exemple. De nombreux langages vont nous permettre de créer des sites dynamiques, chacun avec leurs points forts et leurs faiblesses et leur champ d'application.

Dans ce cours, nous nous concentrons sur le binôme certainement le plus connu parmi ces langages : le PHP qui va être utile pour tout ce qui est calcul / traitement des données et MySQL qui va nous servir à gérer nos bases de données.

Nous reparlerons de la distinction client / serveur dans la prochaine leçon. Ici, vous pouvez retenir qu'un site web créé uniquement avec des langages qui s'exécutent côte client sera statique tandis qu'un langage créé avec des langages qui s'exécutent côté client et des langages qui s'exécutent côté serveur sera généralement dynamique.

Définition et rôle du MySQL

MySQL est un système de gestion de bases de données relationnelles. Une base de données est un ensemble structuré de données. Les données vont pouvoir être des informations clients (nom, adresse, mot de passe, etc.), la liste des commentaires de notre blog, le texte de nos articles, etc.

Le problème ici est qu'on ne va pas directement pouvoir interagir avec les bases de données car les données sont stockées d'une manière illisible pour un humain. Pour manipuler les données stockées dans les bases de données, nous allons devoir utiliser un langage de bases de données.

Le langage de bases de données le plus célèbre est le SQL. SQL est l'acronyme de Structured Query Language (Langage de Requêtes Structurées).

Le système de gestion de bases de données MySQL utilise le langage SQL pour la manipulation des données des bases de données.

Les avantages du MySQL sont sa simplicité d'utilisation, sa fiabilité et ses performances en plus du fait qu'on va pouvoir gérer plusieurs types de bases de données différentes si besoin avec MySQL et qu'on va pouvoir l'utiliser conjointement avec PHP.

Je sais que ces notions peuvent être complexes à envisager pour un débutant. Prenons donc un exemple concret. Imaginons que nous voulions créer un site sur lequel les utilisateurs vont pouvoir s'inscrire et s'identifier.

Nous allons créer nos formulaires d'inscription en HTML et allons ensuite récupérer les données des formulaires en PHP. Ici, nous allons vouloir enregistrer ces données dans une base de données. Une base de données n'est pas un objet mystique : ce n'est ni plus ni moins qu'un fichier.

Pour le moment, notre base de données n'existe pas. Nous allons donc devoir la créer. Pour cela, nous avons deux façons de faire : soit on passe par une application spécialisée comme phpMyAdmin (dont nous reparlerons plus tard), soit on envoie nos requêtes SQL depuis un fichier de code.

Pour faire cela, nous allons utiliser une extension PHP (Comme PDO ou MySQLi par exemple) qui va nous permettre de coder en MySQL.

Dans notre code MySQL, nous allons écrire différentes requêtes SQL qui vont nous permettre de créer notre base de données et d'enregistrer les données dedans.

Notre base de données est ici créée en utilisant du MySQL : cette base va donc être une base MySQL. Cela signifie que c'est ce système de gestion qui s'occupe de créer notre fichier « base de données », qui va ordonner les données et qui va le sécuriser.

Notez par ailleurs que MySQL est un système de gestion de bases de données dit « relationnel » car les informations ne vont pas être toutes stockées au même endroit mais plutôt dans plusieurs compartiments appelés « tables » qui vont pouvoir communiquer entre elles.

L'idée principale à retenir ici et ce que je veux que vous compreniez est que nous ne pouvons pas créer ni manipuler de bases de données sans système de gestion de bases de données.

Pourquoi utiliser le PHP et MySQL ?

Contrairement au HTML et au CSS qui sont de véritables standards, le PHP et MySQL ont de nombreux concurrents : Python, Ruby voire JavaScript pour le PHP et PostgreSQL, Microsoft SQL Server ou encore SQLite pour MySQL pour ne citer qu'eux.

Pourquoi préférer le couple PHP / MySQL aux langages concurrents ? Concrètement, il n'y a pas de raison « absolue » au sens où les alternatives citées sont également des langages performants et qui possèdent certains avantages comme certains inconvénients par rapport au PHP et au MySQL.

Cependant, si le couple PHP / MySQL reste de loin le plus célèbre et le choix de référence lorsqu'on veut créer des sites dynamiques et stocker des données, c'est pour de bonnes raisons.

Le premier avantage du PHP concerne la structure de ce langage : c'est un langage à la fois très simple d'accès pour des débutants qui pourront rapidement comprendre sa syntaxe de base et réaliser leurs premiers scripts et qui va également supporter d'un autre côté des structures très complexes.

Ensuite, le PHP est un langage Open Source et donc gratuit. Il est bon de le noter car cela n'est pas forcément automatique même si les utilisateurs du web ont l'habitude du « tout gratuit ». Le PHP est également reconnu et supporté de manière universelle : il va fonctionner quasiment partout et avec l'immense majorité des architectures techniques.

Enfin, le PHP se distingue par ses performances et sa solidité : comme le langage est Open Source, n'importe qui peut contribuer à son évolution, ce qui fait qu'il est sans cesse perfectionné et qu'il ne sera à priori jamais abandonné. En outre, le PHP possède de bonnes performances d'exécution en termes de rapidité et est un langage sûr : les rares failles jamais détectées dans le langage ont toujours été corrigées dans les 24h.

Les systèmes de gestion de base de données sont également nombreux. La plupart d'entre eux sont basés sur du SQL standard. J'ai choisi dans ce cours d'utiliser MySQL car c'est encore une fois le choix le plus populaire parmi les développeurs et cela pour de bonnes raisons.

Tout d'abord, il va être totalement compatible avec PHP et utilise une syntaxe SQL standard ce qui facilitera les opérations si un jour vous devez changer de système de gestion de bases de données. Ensuite et enfin MySQL est à la fois simple d'utilisation, très robuste et offre d'excellente performances que cela soit pour une petite ou pour une grosse structure.

Client et serveur : définitions et interactions

Dans cette nouvelle leçon, nous allons définir ce qu'est un « client » et ce qu'est un « serveur » et allons par la même voir les grandes différences entre les langages dits « client side » et les langages dits « server side ».

Le fonctionnement d'Internet et du Web

L'Internet est un système créé pour transporter de l'information. C'est un réseau de réseaux qui utilisent chacun des protocoles (ensemble de règles établies qui définissent comment formater, transmettre et recevoir des données) différents pour envoyer de l'information.

Le World Wide Web, ou plus simplement « Web », est l'un des réseaux de l'Internet. Le Web n'est donc qu'une partie d'Internet.

Plus précisément, le Web est un réseau de machines interconnectées qui stockent des sites. Lorsqu'une machine est connectée au web et fournit un accès à un site web, on l'appelle un serveur car elle « sert » le site web.

Un serveur est une sorte de super ordinateur, fonctionnant 24h/24 et 7j/7 (en théorie), et étant beaucoup plus puissant que nos ordinateurs. Un serveur dispose de certains logiciels spécifiques et son rôle est de stocker toutes sortes de médias composant les sites (fichiers, images, etc.), et de les rendre accessible pour tout utilisateur à n'importe quel moment, où qu'il soit.

Pour pouvoir se comprendre et échanger des données toutes ces machines doivent parler la même langue, c'est-à-dire utiliser le même protocole.

Le Web repose ainsi sur le protocole HTTP, pour HyperText Transfer Protocol (protocole de transfert hypertexte) et sur son frère qui utilise des clefs de cryptage : le HTTPS (Secure HTTP). Pour utiliser ce protocole, nous allons devoir passer par un navigateur web (Chrome, Safari, etc.) qu'on va alors appeler un « client http ».

Pour accéder directement à une page web, on passe ainsi par un navigateur en utilisant le protocole HTTP. On va passer une adresse au format spécial à notre navigateur : une URL pour Uniform Resource Locator ou « localisateur uniforme de ressource » qui sert à identifier une page web de manière unique.

Ici, notre navigateur (et nous) sommes des « clients » puisque nous sommes ceux qui demandons à accéder à la page web.

Le navigateur va alors chercher où se trouve le serveur hébergeant la page demandée. Pour cela il va utiliser un service de DNS (Domain Name Server) qui sont des serveurs permettant d'associer un nom de domaine (pierre-giraud.com par exemple) à une adresse IP unique.

Notez ici que chaque fournisseur de services Internet fournit une liste d'adresses de DNS à contacter. Si le premier DNS ne reconnait pas le site, alors il envoie la demande à d'autres DNS et ainsi de suite jusqu'à ce qu'un DNS possède le site dans sa liste de noms.

L'adresse IP liée au site va alors être renvoyée au serveur. L'IP (Internet Protocol) est une suite de nombres qui permet d'identifier de manière unique une machine connectée à Internet. Chaque machine va posséder sa propre IP qui va changer en fonction du réseau sur lequel elle est connectée (l'IP est attribuée par le fournisseur de services Internet).

Le navigateur possède donc maintenant l'adresse IP de notre site et donc l'adresse de la machine (le fameux « serveur ») sur lequel il est stocké. Il va ainsi pouvoir directement contacter le serveur en utilisant le protocole HTTP pour lui demander de renvoyer la page en question et va également envoyer notre IP pour que le serveur sache à quelle adresse renvoyer la page demandée.

Lorsque le serveur reçoit la requête, il recherche immédiatement le fichier demandé, effectue éventuellement certaines opérations dessus et le renvoie au navigateur ou renvoie un code d'erreur si le fichier demandé est introuvable ou ne peut pas être envoyé.

Langages client side et server side

Un site Internet n'est qu'un ensemble de fichiers de codes liés entre eux et faisant éventuellement appel à des ressources ou médias comme des images, etc.

Le code écrit dans ces fichiers, encore appelé « script », va pouvoir être exécuté soit côté client (client-side), c'est-à-dire directement dans le navigateur de l'utilisateur qui cherche à afficher la page, soit du côté du serveur (server-side).

Vous devez bien comprendre ici qu'un navigateur (côté client) et un serveur (côté serveur) ne vont pouvoir chacun effectuer que certaines opérations et lire certains langages.

Pour être tout à fait précis, la grande majorité des navigateurs ne sont capables de comprendre et de n'exécuter que du code HTML, CSS et JavaScript. Un navigateur est ainsi incapable de comprendre du code PHP.

Lorsqu'un navigateur demande à un serveur de lui servir une page, le serveur va donc se charger d'exécuter tout code qui ne serait pas compréhensible par le navigateur (en utilisant au besoin un interpréteur).

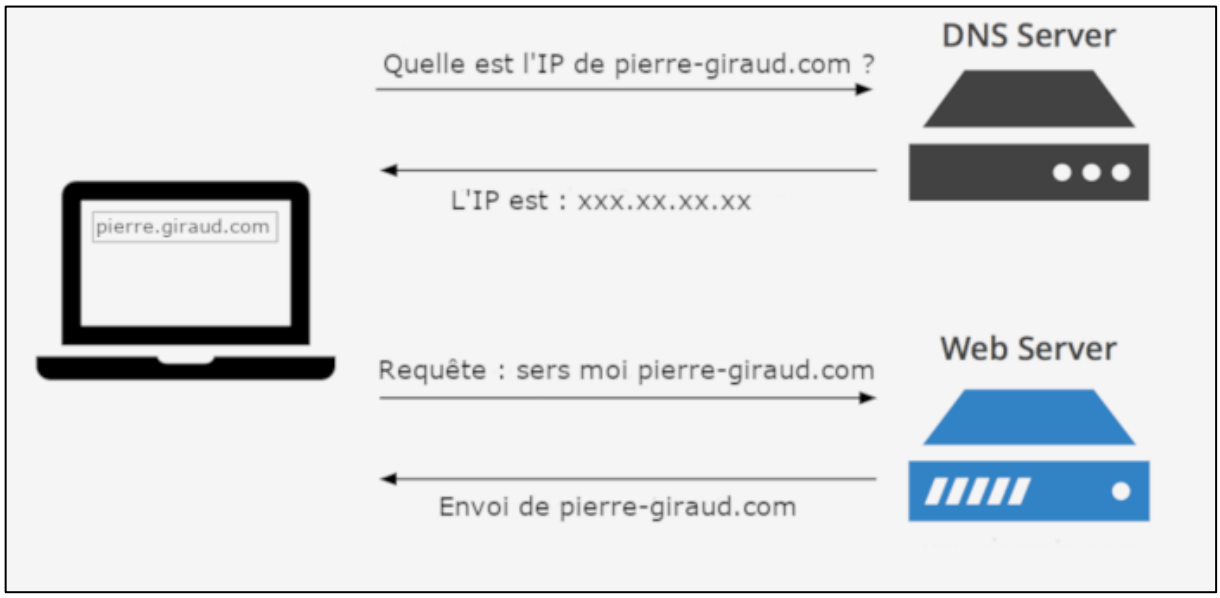
Une fois ces opérations effectuées, le serveur va renvoyer le résultat (la page demandée après interprétation) sous forme de code compréhensible par le navigateur, c'est-à-dire principalement en HTML. Le navigateur va alors afficher la page au visiteur. Cette page va être unique puisqu'elle a été générée par le serveur pour un utilisateur spécifiquement et en tenant compte de données particulières.

Si en revanche la page demandée par le visiteur ne contient aucun code nécessitant l'intervention du serveur, alors le serveur va la renvoyer telle quelle au navigateur qui va l'afficher au visiteur.

Notez bien ici que les opérations réalisées côté serveur vont être transparentes pour le visiteur et que celui-ci ne va jamais avoir accès ni pouvoir voir le code exécuté côté serveur tout simplement car une fois les opérations terminées, le serveur ne va renvoyer que du code compréhensible par le navigateur.

C’est la raison pour laquelle lorsque vous analyser le code d’un page, vous ne trouverez jamais d’instructions PHP mais seulement généralement du code HTML, CSS et JavaScript qui sont des langages qui s’exécutent côté client.

Le schéma ci-dessous résume ce qu’il se passe lorsque vous demandez à accéder à une page web via votre navigateur :



Mise en place de notre environnement de travail

Pour coder en HTML et en CSS et afficher le résultat de son code, un simple éditeur de texte et un navigateur suffisent. Pour le PHP et MySQL, cependant, cela va être un peu plus complexe car le code va s'exécuter côté serveur.

Nous allons détailler dans cette leçon les différentes alternatives qui s'offrent à nous pour coder en PHP et en MySQL et allons voir ensemble quels logiciels installer et pourquoi.

Le travail en local et le travail en production

Lorsque l'on code, on peut travailler soit en local, c'est-à-dire en hébergeant nous-mêmes tous les fichiers sur nos propres machines ou encore « hors ligne », soit en production (ou en préproduction) c'est-à-dire sur des fichiers qui sont hébergés sur un serveur distant et potentiellement accessibles à tous ou en « live ».

On va généralement opposer le travail en local au travail en production. En effet, en travaillant en local, nous sommes les seuls à avoir accès à nos différents fichiers et donc les seuls à voir l'impact des modifications que l'on fait sur ces fichiers tandis que si l'on fait la moindre modification en production, cela impacte directement notre site live et nos visiteurs le voient immédiatement.

Lors de la phase de développement d'un site ou dans des phases de test ou de débogage et sauf cas exceptionnels, un bon développeur préférera toujours travailler en local (ou éventuellement en préproduction) afin de ne pas impacter le fonctionnement normal d'un site web.

On appelle « préproduction » une copie d'un site également hébergée sur serveur. Généralement, on restreint l'accès à la préproduction aux développeurs du site afin qu'ils puissent tester en toute sécurité et en conditions réelles leurs modifications.

Dans le cas où le site est déjà créé et accessible à tous, alors nous ferons une copie locale de toute notre architecture et travaillerons sur cette copie afin de tester et d'implémenter de nouvelles fonctionnalités. Ensuite, une fois seulement qu'on s'est assuré que les différentes modifications ou implémentations fonctionnent et qu'aucun bug n'a été détecté, nous enverrons tous nos changements en production, c'est-à-dire que nous remplacerons les fichiers ou injecteront les modifications sur le serveur.

Dans ce cours, nous ne travaillerons évidemment pas sur un site déjà « live » car c'est une mauvaise pratique et car nous n'en avons pas mais plutôt en local. Cependant, rappelez-vous que le PHP et MySQL vont s'exécuter côté serveur.

Il va donc nous falloir recréer une architecture serveur sur nos propres machines avec les logiciels adaptés afin de pouvoir tester nos codes PHP. Rassurez-vous : cela est très simple et totalement gratuit.

Recréer une architecture serveur sur son ordinateur

Comme je vous l'ai dit précédemment, un serveur dispose de différents programmes lui permettant de lire et de comprendre certains langages informatiques que des ordinateurs « normaux » ne peuvent pas lire.

Nous allons donc devoir installer des programmes similaires afin de pouvoir tester nos codes PHP et MySQL. La bonne nouvelle ici est qu'il existe des logiciels regroupant tous les programmes nécessaires pour faire cela.

Selon le système que vous utilisez, vous devrez installer un logiciel différent. Vous pouvez trouver ci-dessous le logiciel à installer selon votre système :

- Si vous êtes sous Windows, vous allez installer WAMP, disponible à l'adresse http://www.wampserver.com/ ;
- Si vous êtes sous Mac OS, vous allez installer MAMP, disponible à l'adresse http://www.mamp.info/ ;
- Si vous êtes sous Linux, vous allez installer XAMPP, disponible à l'adresse http://www.apachefriends.org/

Pour ma part, étant donné que j'utilise un système Mac OS, je vais travailler avec MAMP pour la suite de ce cours. Si vous devez installer un logiciel différent, pas d'inquiétude : les trois logiciels cités ci-dessus vont fonctionner de la même manière et proposer quasiment les mêmes fonctionnalités.

Commencez donc déjà par télécharger le logiciel correspondant à votre système et par l'installer.

L'éditeur de texte, outil indispensable pour écrire du code

Les logiciels WAMP, MAMP ou XAMPP vont nous être utile pour exécuter notre code PHP. Cependant, avant d'exécuter du code, il va falloir l'écrire et pour cela, nous allons avoir besoin d'un logiciel appelé éditeur de texte ou d'un IDE (Environnement de Développement Intégré).

Un éditeur de texte est un logiciel nous permettant d'écrire des lignes de code dans différents langages et qui possède différentes fonctionnalités pour nous faciliter l'écriture de ce code. Vous devriez déjà savoir ce que c'est si vous possédez des bases en HTML. Un IDE est similaire à un éditeur de texte mais possède généralement des fonctionnalités supplémentaires comme l'intégration possible de plugins et etc.

Pour ma part, j'utiliserai la version gratuite de l'IDE Komodo (Komodo Edit) pour ce cours mais n'hésitez pas à conserver votre propre éditeur si vous en avez installé un différent. Le plus important est que vous soyez à l'aise pour coder. Vous pouvez par exemple utiliser Microsoft Visual Studio , NotePad++, Sublime Text, Brackets, etc.

Créer, enregistrer et exécuter un script PHP

A partir de maintenant, nous allons véritablement rentrer dans le vif du sujet et commencer à pratiquer. Ce cours est divisé en deux parties : nous allons d'abord étudier le PHP puis apprendrons à manipuler nos bases de données avec MySQL.

Dans cette leçon, nous allons déjà découvrir la syntaxe de base du PHP, comment enregistrer et surtout exécuter un fichier PHP.

Où écrire le code PHP ?

Nous allons pouvoir écrire nos scripts PHP soit dans des fichiers dédiés, c'est-à-dire des fichiers qui ne vont contenir que du PHP, soit intégrer le PHP au sein de nos fichiers HTML.

Les fichiers qui contiennent du PHP vont devoir être enregistrés avec l'extension .php. Dans le cas où on intègre du code PHP dans un fichier HTML, il faudra également changer son extension en .php.

La balise PHP

Le serveur, pour être en mesure d'exécuter le code PHP, va devoir le reconnaitre. Pour lui indiquer qu'un script ou que telle partie d'un code est écrit en PHP, nous allons entourer ce code avec une balise PHP qui a la forme suivante : <?php ?>.

Lorsqu'on intègre du PHP dans du code HTML, on va pouvoir placer cette balise et du code PHP à n'importe quel endroit dans notre fichier. On va même pouvoir placer la balise PHP en dehors de notre élément html. De plus, on va pouvoir déclarer plusieurs balises PHP à différents endroits dans un fichier.

Regardez déjà l'exemple ci-dessous :

```php
<!DOCTYPE html>
<?php

?>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php

        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on déclare deux balises PHP au sein d'une page HTML. Pour le moment, nos balises PHP sont vides.

Note : Dans ce cours, vous pourrez copier-coller la plupart des codes directement, comme n'importe quel autre texte. Cela vous permettra de les tester sur votre propre machine ou de les récupérer pour les modifier.

Syntaxe PHP de base

Écrivons une première instruction PHP afin d'analyser et de comprendre la syntaxe générale du PHP.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //Affiche "Hello World" avec un retour à la ligne
            echo 'Hello World <br>'; //Ceci est un commentaire PHP

            /*Affiche
              "Bonjour le Monde
            */
            echo "Bonjour le Monde"; /*Ceci est un commentaire PHP*/
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Il y a plusieurs choses intéressantes dans ce premier script PHP. Décortiquons le ligne par ligne.

La première ligne contient un commentaire PHP monoligne. Les commentaires en PHP vont nous servir à expliquer le fonctionnement d'un script ou de neutraliser une ou plusieurs instructions. En effet, ce qu'il y a entre commentaires va être ignoré durant l'exécution du code.

Donner des informations sur le fonctionnement d'un script peut être utile pour l'auteur du script (dans le cas où notre projet soit long et compliqué, pour nous permettre de nous rappeler de ce que fait chaque partie du code) ou lorsqu'il voudra le partager avec d'autres développeurs.

On va pouvoir écrire des commentaires en PHP en utilisant deux syntaxes différentes : la syntaxe utilisant `//` est réservée aux commentaires monoligne, tandis que la syntaxe utilisant `/* */` peut être utilisée pour créer des commentaires monolignes ou multilignes.

Les parties `echo 'Hello World <br>';` et `echo "Bonjour le Monde";` sont ce qu'on appelle des instructions car on « demande » au code de faire quelque chose.

En l'occurrence, ici, la structure de langage `echo` permet d'afficher une chaine de caractères. Notre script ici sert donc à afficher deux phrases : `Hello World` et `Bonjour le Monde`.

Une instruction en PHP doit toujours se terminer par un point-virgule. Si vous l'oubliez, votre script ne fonctionnera pas correctement.

Ici, vous pouvez remarquer qu'on a ajouté un élément br au sein du premier texte qui doit être affiché. Cet élément br ne sera pas affiché mais va permettre d'indiquer un retour à la ligne au navigateur.

En effet, ici, vous devez bien comprendre l'ordre dans lequel le code s'exécute. Lorsqu'on demande à un serveur de nous renvoyer une page, celui-ci va regarder s'il y a du code à exécuter comme du PHP dans la page.

Si c'est le cas, celui-ci va exécuter le code en question et ne va ensuite renvoyer au navigateur que du code qu'il peut comprendre, c'est-à-dire principalement du HTML. Le navigateur va finalement afficher la page à partir de sa structure HTML.

Dans notre cas, le serveur va exécuter nos deux instructions et renvoyer le texte (et l'élément HTML br qui est inclut dedans) au navigateur qui va lire le HTML pour nous afficher la page.

Si j'affiche ma page, voici le résultat que j'obtiens :

Enregistrer et exécuter un fichier PHP

A partir du moment où on ouvre une balise PHP dans un fichier HTML, nous allons devoir enregistrer le fichier avec l'extension .php. C'est essentiel pour que le serveur comprenne que la page contient du code PHP qu'il devra exécuter.

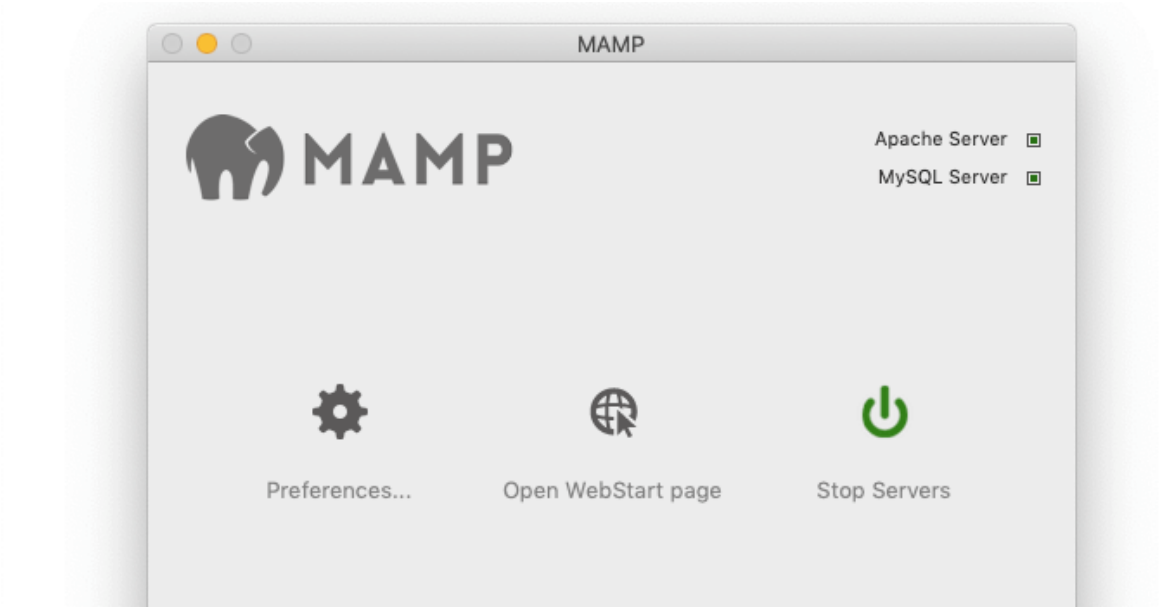
Dans notre cas, nous travaillons en local. Pour exécuter du code PHP, il va falloir utiliser le logiciel qu'on a installé précédemment (WAMP, XAMP ou LAMP). Pour faire cela, il va nous suffire d'enregistrer notre fichier dans le sous dossier dédié du logiciel choisi.

Si vous avez installé WAMP, par exemple, il faudra enregistrer tous vos fichiers PHP dans le sous-dossier « wamp ». Si vous utilisez comme moi MAMP, alors il faudra placer vos fichiers dans le sous-dossier « htdocs ».

Essayons donc d'enregistrer, d'exécuter et d'afficher le résultat d'un premier fichier PHP. Pour cela, commencez déjà par ouvrir votre éditeur de texte, et par recopier le fichier que j'ai créé ci-dessus, puis enregistrez le dans le bon dossier.

Une fois ces opérations réalisées, vous allez devoir démarrer votre logiciel. Pour cela, double-cliquez dessus. Vous arrivez sur l'interface d'accueil de celui-ci.

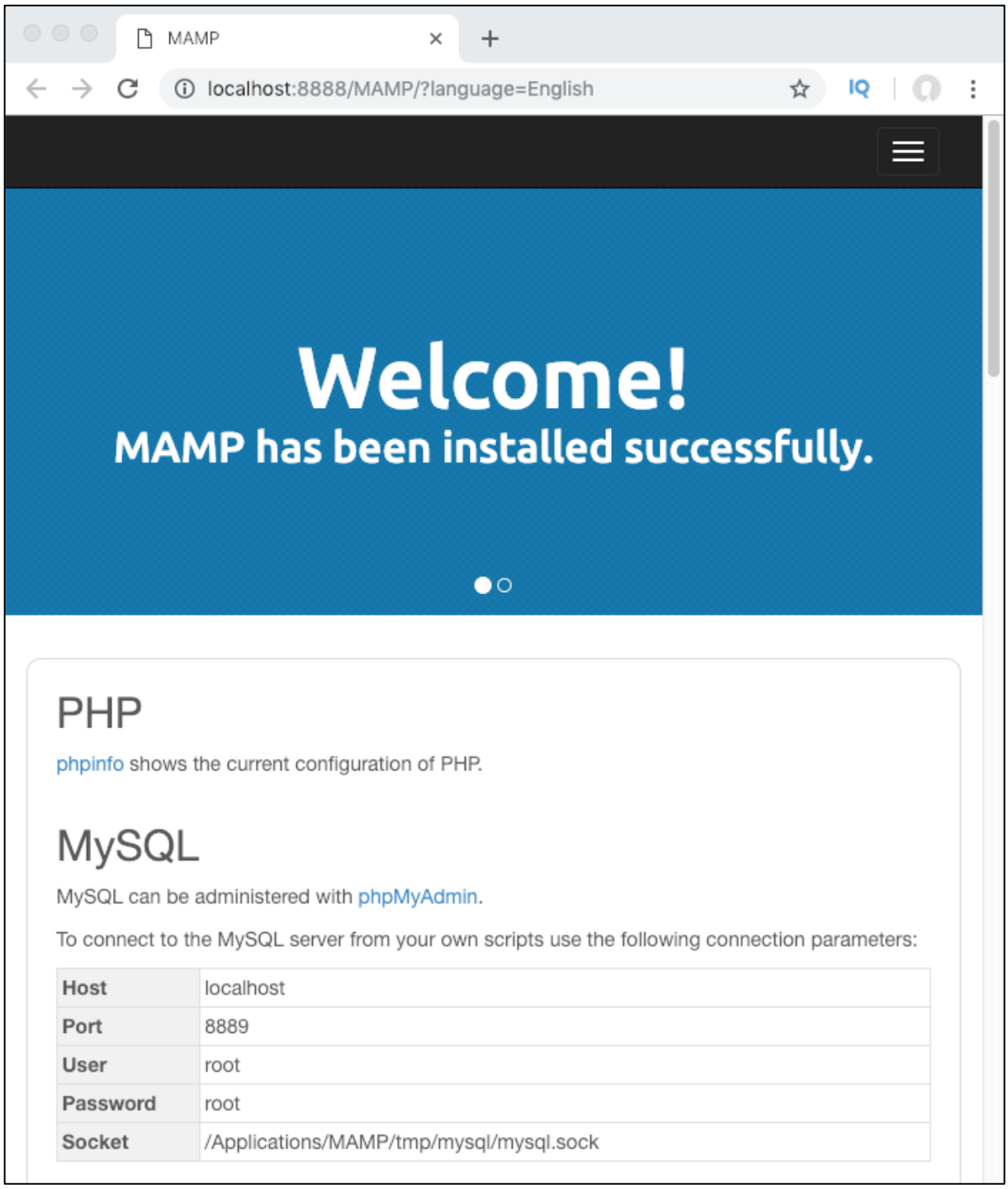
A partir de là, vous devriez avoir un bouton vous proposant de « démarrer les serveurs » ou au contraire de les arrêter, avec des voyants lumineux vous indiquant si les serveurs fonctionnent (voyants verts) ou pas (voyants rouges). Démarrez les serveurs si ceux-ci sont arrêtés.



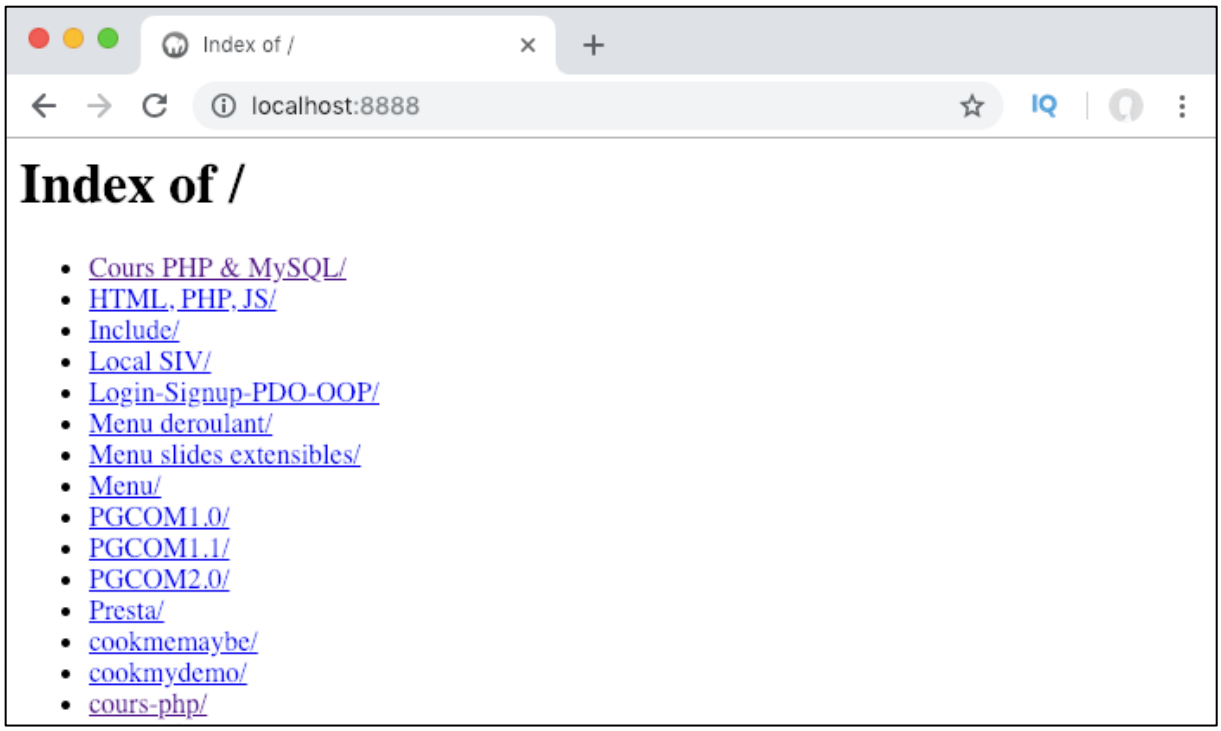
Pour lancer l'exécution du fichier PHP créé et voir le résultat dans notre navigateur, il faut alors accéder au dossier dans lequel le fichier a été stocké via notre navigateur web en passant par nos serveurs.

Pour cela, il suffit d'ouvrir un nouvel onglet dans notre navigateur favori et de renseigner l'adresse localhost ou localhost:8888 ou encore 127.0.0.1 selon votre système.

Pour avoir toutes les informations relatives à votre configuration, vous pouvez vous rendre sur la page d'accueil de votre logiciel (la page d'accueil s'ouvre soit automatiquement lorsque vous démarrez votre logiciel, soit vous pouvez y accéder via un raccourci qui devrait être facile à trouver). La page d'accueil de MAMP par exemple ressemble à cela :



En tapant la bonne adresse dans votre navigateur, vous allez pouvoir accéder au contenu du dossier dans lequel vous avez enregistré le fichier qu'on veut exécuter. Voici à quoi ressemble le contenu de mon propre dossier.



Dans mon cas, mon dossier contient de nombreux autres dossiers et fichiers. De votre côté, vous ne devriez avoir qu'un fichier. Cliquez donc dessus. Le serveur va exécuter le code PHP dans le fichier et renvoyer du code HTML que le navigateur va afficher. Ici, vous devriez avoir le résultat suivant :

Note : si vous tentez d'ouvrir un fichier contenant du code PHP sans passer par un serveur (c'est-à-dire par le localhost dans notre cas), alors le code de toute la page sera affiché tel quel tout simplement car un navigateur n'est pas capable de comprendre ni d'exécuter du code PHP.

Afficher un résultat en PHP : les instructions echo et print

Dans cette nouvelle leçon, nous allons apprendre à afficher un résultat en PHP en utilisant les structures de langage echo et print.

Une première définition : les structures de langage

Commençons déjà par définir ce qu'on appelle « structure de langage » (« language construct » en anglais) afin de partir sur de bonnes bases.

Une structure de langage correspond simplement à une partie de la syntaxe d'un langage. La syntaxe d'un langage informatique correspond à un ensemble de mots clefs au sens bien défini par le langage. A partir de cette syntaxe, et en combinant les mots clefs, nous allons pouvoir construire des expressions.

Les structures de langage sont en d'autres termes les atomes d'un langage informatique : ce sont les unités de base d'un langage qu'on va pouvoir utiliser pour construire des expressions.

Je tenais à commencer ce chapitre avec cette définition car de nombreuses personnes considèrent à tort que les structures de langage echo et print sont des fonctions. Or, les fonctions et les structures de langage sont deux types différents d'objets qui ne vont pas être traités de la même façon.

A votre niveau, il risque néanmoins d'être compliqué de bien comprendre et de bien vous représenter ce qu'est une structure de langage. Ne vous inquiétez pas, c'est tout à fait normal : bien souvent, en code, une notion nécessite de connaitre un ensemble d'autres notions pour être bien comprise tandis que les autres notions ont besoin de cette première notion pour être comprises… C'est toute la difficulté lorsqu'on commence à apprendre un nouveau langage !

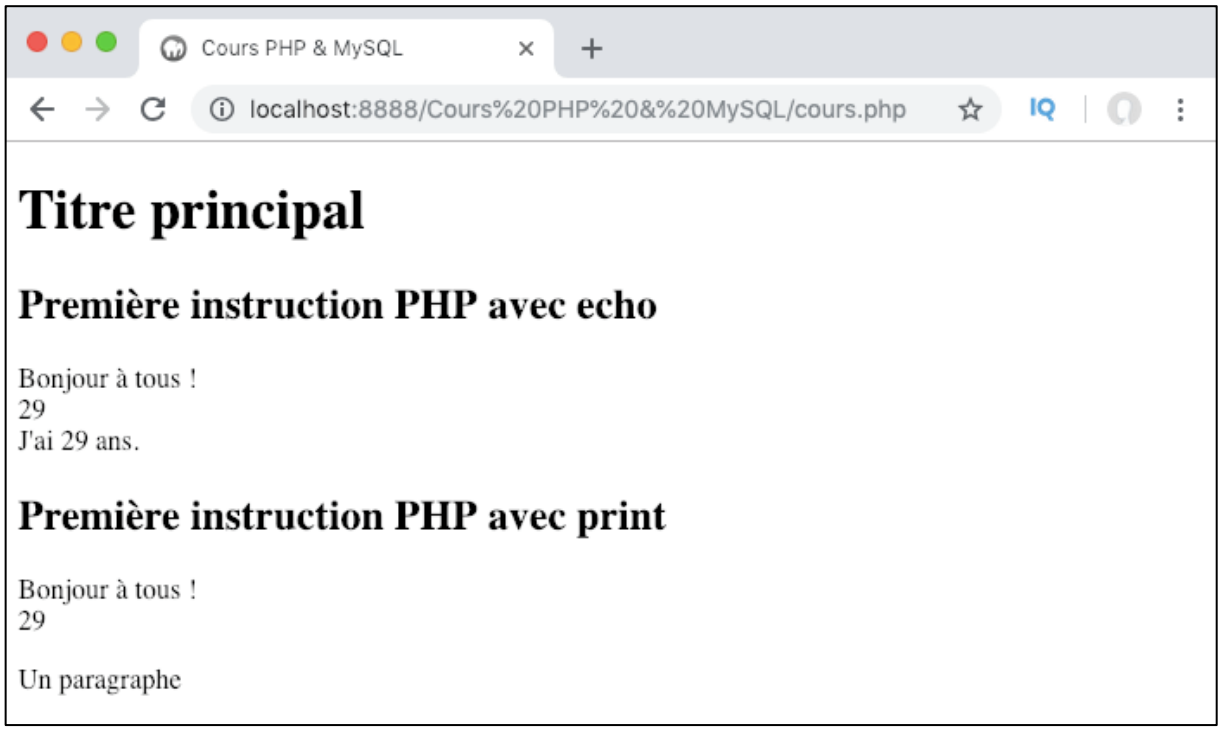
Essayez donc de retenir cette première définition et vous allez voir que celle-ci va devenir de plus en plus claire au fur et à mesure que vous allez avancer dans ce cours. De manière générale, je vous conseille de faire régulièrement des allers retours dans le cours pour retourner sur d'anciennes notions qui n'étaient pas claires pour vous à l'époque et essayer de les comprendre avec vos nouvelles connaissances.

Les structures de langage echo et print

Les structures de langage echo et print vont nous permettre d'afficher un résultat en PHP. Pour cela, nous allons écrire notre echo ou notre print suivi de ce qu'on souhaite afficher et suivi d'un point-virgule pour terminer l'instruction, en plaçant le tout dans une balise PHP.

Regardez plutôt l'exemple suivant qu'on va immédiatement expliquer :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo "<h2>Première instruction PHP avec echo</h2>";
            echo 'Bonjour à tous !<br>';
            echo  29;
            echo "<br>J'ai ", 29, " ans.<br>";

            print "<h2>Première instruction PHP avec print</h2>";
            print 'Bonjour à tous !<br>';
            print  29;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Nous pouvons noter plusieurs choses à partir de l'exemple précédent. Déjà, vous pouvez observer qu'on peut tout à fait passer des balises HTML dans ce qu'on souhaite afficher. Ces balises HTML seront lues comme les autres par le navigateur.

Ensuite, vous pouvez remarquer que j'utilise parfois des apostrophes droites, parfois des guillemets droits et d'autres fois rien du tout pour entourer le contenu qui devra être affiché.

Ici, vous pouvez retenir que les chaines de caractères (c'est-à-dire les contenus de type texte) doivent toujours être entourés d'un couple d'apostrophes ou de guillemets pour pouvoir être affichées. Cela n'est pas nécessaire pour afficher un chiffre. Nous reparlerons des types de données plus tard dans ce cours.

Notez que les structures de langage echo et print ne sont pas strictement équivalentes mais qu'il existe quelques différences mineures entre elles.

La première de ces différences est que echo ne possède pas de valeur de retour à la différence de print qui va toujours retourner 1. Nous reparlerons des valeurs de retour dans une prochaine leçon : il est beaucoup trop tôt pour le moment.

La deuxième différence est qu'on va pouvoir passer plusieurs valeurs à echo tandis qu'on ne va pouvoir en passer qu'une à print. En pratique, cependant, il sera très rare de passer plusieurs paramètres à echo tout simplement car cela n'aura bien souvent aucune utilité.

Si vous regardez bien le dernier exemple utilisant echo, vous pouvez remarquer qu'on passe plusieurs valeurs entre différents couples de guillemets et séparées par des virgules.

Enfin, le dernier point de différence entre echo et print est que echo s'exécute un peu plus rapidement que print.

En pratique, nous utiliserons majoritairement echo pour afficher un résultat en PHP.

Notez finalement qu'une autre syntaxe avec un couple de parenthèses existe pour echo et print :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo ("<h2>Première instruction PHP avec echo</h2>");
            echo ('Bonjour à tous !<br>');
            echo (29);

            print ("<h2>Première instruction PHP avec print</h2>");
            print ('Bonjour à tous !<br>');
            print (29);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Je vais vous demander de ne pas vous préoccuper de cette syntaxe pour le moment et d'utiliser comme moi la syntaxe sans les parenthèses.

Jusqu'ici, vous pouvez vous dire que echo et print ne sont pas très utiles si elles ne servent qu'à afficher du texte qu'on écrit et qu'on pourrait aussi bien directement écrire nos contenus en HTML.

C'est tout à fait vrai : pour le moment, nous faisons un usage très basique de ces structures de langage et pourrions aussi bien écrire directement en HTML. Cependant, echo et print vont nous être très utiles par la suite lorsqu'on devra afficher le contenu de variables.

Nous débutons à peine, il est donc tout à fait normal que nous ne puissions pas encore faire d'utilisation très avancée des éléments du langage que nous étudions mais je vous demande ici de me faire confiance et de faire l'effort de bien comprendre ces premiers concepts qui vont nous servir dans tout le reste du cours.

L'usage des apostrophes et des guillemets pour afficher des chaines de caractères

Pour afficher une chaine de caractère, c'est-à-dire un texte avec echo ou print, vous devez absolument l'entourer d'un couple d'apostrophes droites ou de guillemets droits.

Une nouvelle fois, les écritures avec les apostrophes et avec les guillemets ne sont pas strictement équivalentes mais vont pouvoir être utilisées dans des situations différentes et pour répondre à des problèmes différents.

Nous verrons précisément quand utiliser des guillemets et quand utiliser des apostrophes après avoir étudié les variables en PHP car il est nécessaire de les connaître pour bien cerner et comprendre les différents cas d'utilisation.

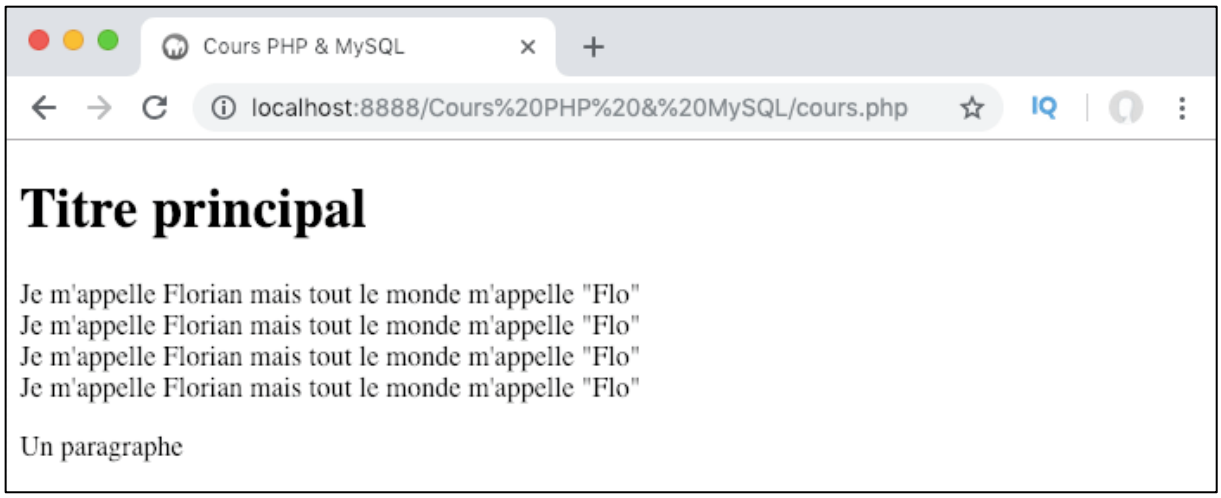
Pour l'instant, nous allons nous intéresser à une autre problématique que nous allons rencontrer lorsqu'un souhaitera afficher une chaine de caractères : que faire si la chaine de caractères qu'on souhaite afficher contient elle-même des apostrophes et / ou des guillemets ? Comment le PHP peut-il faire pour savoir qu'ils appartiennent à notre texte et qu'ils ne servent pas à le délimiter ?

La réponse est simple : il ne peut pas le savoir. Pour cela, nous allons devoir l'aider en « échappant » nos apostrophes ou nos guillemets (selon ce qui a été choisi pour entourer notre chaine de caractères), c'est-à-dire en neutralisant leur signification spéciale en PHP.

Pour faire cela, nous allons simplement devoir utiliser un antislash \ avant chaque caractère qu'on souhaite échapper pour neutraliser sa signification spéciale. L'antislash est en PHP un caractère d'échappement : il sert à indiquer au serveur qu'il ne doit pas tenir compte de la signification spéciale du caractère qui le suit dans le cas où ce caractère est un caractère spécial (c'est-à-dire un caractère qui possède une signification spéciale pour le PHP).

Illustrons immédiatement cela avec un exemple concret :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo 'Je m\'appelle Florian mais tout le monde m\'appelle "Flo"<br>';
            echo "Je m'appelle Florian mais tout le monde m'appelle \"Flo\"<br>";

            print 'Je m\'appelle Florian mais tout le monde m\'appelle "Flo"<br>';
            print "Je m'appelle Florian mais tout le monde m'appelle \"Flo\"<br>";
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Dans cet exemple, notre script contient 4 instructions différentes utilisant `echo` et `print`.

Dans notre première instruction, on utilise des apostrophes pour délimiter notre chaine de caractères à afficher. Toutes les apostrophes à l'intérieur de la chaine de caractères devront donc être échappés à l'aide d'un antislash afin d'éviter toute ambiguïté. Ici, pas besoin d'échapper les guillemets puisque le PHP sait qu'on utilise des apostrophes pour délimiter notre chaine de caractères. Il n'y a donc pas d'ambiguïté sur les guillemets.

Dans notre deuxième instruction, on utilise au contraire des guillemets pour entourer la chaine de caractères qu'on souhaite afficher. Il faudra donc ici échapper tous les guillemets présents au sein de notre chaine de caractères.

On réutilise les mêmes expressions pour nos troisième et quatrième instructions mais en utilisant cette fois-ci `print` plutôt qu'`echo`.

PARTIE II

Découverte des variables en PHP

Introduction aux variables PHP

Dans cette partie, nous allons découvrir ce que sont les variables en PHP et apprendre à les manipuler.

Qu'est-ce qu'une variable ?

Une variable est un conteneur servant à stocker des informations de manière temporaire, comme une chaine de caractères (un texte) ou un nombre par exemple.

Le propre d'une variable est de pouvoir varier, c'est-à-dire de pouvoir stocker différentes valeurs au fil du temps.

En PHP, les variables ne servent à stocker une information que temporairement. Plus précisément, une variable ne va exister que durant le temps de l'exécution du script l'utilisant.

Ainsi, on ne va pas pouvoir stocker d'informations durablement avec les variables (pour cela, nous pourrons par exemple utiliser les fichiers, cookies ou les bases de données dont nous parlerons plus tard dans ce cours).

Note : Dans le début de ce cours, nous allons définir nous-mêmes les valeurs qui vont être stockées dans nos variables, ce qui n'a pas beaucoup d'intérêt en pratique. C'est donc tout à fait normal si vous ne voyez pas immédiatement le but d'utiliser des variables. Ici, vous pouvez retenir que les variables vont être vraiment intéressantes lorsqu'elles vont nous servir à stocker des données envoyées par les utilisateurs (via des formulaires par exemple) puisqu'on va ensuite pouvoir manipuler ces données.

Les règles de déclaration des variables en PHP

Une variable est donc un conteneur ou un espace de stockage temporaire qui va pouvoir stocker une valeur qu'on va lui assigner.

Nous allons pouvoir créer différentes variables dans un script pour stocker différentes valeurs et pouvoir les réutiliser simplement ensuite. Lorsqu'on crée une variable en PHP, on dit également qu'on « déclare » une variable.

On va pouvoir choisir le nom qu'on souhaite donner à chacune de nos variables. Cependant, il y a quelques règles à respecter et à connaitre lors de la déclaration d'une nouvelle variable :

- Toute variable en PHP doit commencer par le signe $ qui sera suivi du nom de la variable ;
- Le nom d'une variable doit obligatoirement commencer par une lettre ou un underscore (_) et ne doit pas commencer par un chiffre ;
- Le nom d'une variable ne doit contenir que des lettres, des chiffres et des underscores mais pas de caractères spéciaux ;
- Le nom d'une variable ne doit pas contenir d'espace.

De plus, notez que le nom des variables est sensible à la casse en PHP. Cela signifie que l'usage de majuscules ou de minuscules va créer des variables différentes. Par exemple, les variables `$texte`, `$TEXTE` et `$tEXTe` vont être des variables différentes.

Enfin, sachez qu'il existe des noms « réservés » en PHP. Vous ne pouvez pas utiliser ces noms comme noms pour vos variables, tout simplement car le langage PHP les utilise déjà pour désigner différents objets intégrés au langage. Nous verrons ces différents noms au fil de ce cours.

Déclarer une variable PHP en pratique

Pratiquons immédiatement et créons nos premières variables ensemble. Ici, nous allons créer deux variables `$prenom` et `$age` qui vont stocker respectivement une chaine de caractères et un nombre.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = "Pierre";
            $age = 28;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Dans le script ci-dessus, nous déclarons donc nos deux variables `$prenom` et `$age`. On assigne ou on affecte la valeur `Pierre` à la variable `$prenom`. Cette variable va donc stocker la valeur `Pierre`. De la même façon, on assigne la valeur `28` à la variable `$age`.

Notez qu'il va falloir utiliser des guillemets ou des apostrophes pour stocker une chaine de caractères dans une variable. En revanche, nous n'en utiliserons pas pour assigner un nombre à une variable.

Il y a une chose que vous devez bien comprendre avec les variables : le signe `=` est dans ce cadre un opérateur d'affectation ou d'assignation et non pas un opérateur d'égalité comme cela est le cas au sens mathématiques du terme.

Ici, on assigne les valeurs `Pierre` et `28` à nos variables `$prenom` et `$age` mais ces variables ne sont pas « égales » à la valeur qu'elles contiennent.

En effet, vous devez bien comprendre que le propre d'une variable est de pouvoir varier, ce qui signifie qu'on va pouvoir assigner différentes valeurs à une variable au cours du script. Notre variable stockera toujours la dernière valeur qui lui a été assignée. Cette valeur écrasera la valeur précédente stockée par la variable.

Afficher et modifier le contenu d'une variable PHP

Nous allons pouvoir réaliser toutes sortes d'opérations avec nos variables. La plus basique d'entre elles consiste à afficher le contenu d'une variable PHP. Pour cela, nous allons généralement utiliser une instruction `echo` comme ceci :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = "Pierre";
            $age = 28;

            echo $prenom;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, vous pouvez noter que nous n'avons pas besoin d'utiliser de guillemets ni d'apostrophes autour de notre variable pour en afficher le contenu avec `echo` : on va réserver cela seulement aux chaines de caractères, c'est-à-dire aux textes.

Comme je l'ai précisé plus haut, le propre d'une variable est de pouvoir stocker différentes valeurs dans le temps. Ainsi, on peut tout à fait affecter une nouvelle valeur à une variable.

Notez cependant déjà qu'une variable PHP « classique » ne va pouvoir stocker qu'une valeur à la fois (nous verrons les différentes exceptions à cette règle au fil de ce cours).

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = "Pierre";
            $age = 28; //$age stocke le nombre 28

            echo "La variable \$age contient : ";
            echo $age;
            echo "<br>";

            $age = 29; //$age stocke le nombre 29
            echo "La variable \$age contient : ";
            echo $age;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Pour bien comprendre l'exemple précédent, vous devez savoir qu'ici notre script PHP est lu ou interprété linéairement, c'est-à-dire ligne par ligne dans l'ordre d'écriture.

Ici, notre variable `$age` commence par stocker le nombre `28`. La première série d'instructions `echo` va afficher un texte, le contenu de la variable et créer un retour à la ligne.

Ensuite, on affecte une nouvelle valeur à notre variable $age. Notre nouvelle valeur, 29, va écraser l'ancienne. La variable $age va donc désormais stocker le nombre 29. La dernière instruction echo affiche la nouvelle valeur de la variable.

Note : Lorsqu'on utilise des guillemets, les variables déclarées au sein d'une chaine de caractères sont interprétées, ce qui signifie que c'est leur valeur qui va être affichée et non pas le nom de la variable dans notre cas.

Pour éviter ce comportement et afficher le nom de nos variables plutôt que la valeur qu'elles contiennent, nous allons devoir échapper le caractère $ avec un antislash comme on l'a fait ici ou utiliser les apostrophes plutôt que les guillemets.

Les opérateurs d'affection et de comparaison

Une nouvelle fois (et j'insiste car c'est très important), vous devez bien comprendre que le signe égal simple utilisé ci-dessus n'est pas un opérateur de comparaison mais bien un opérateur d'affection (ou d'assignation) : il sert à affecter une valeur à une variable.

Cela signifie que l'opérateur = ne représente pas l'égalité d'un point de vue mathématique. L'égalité en termes de valeurs simples est symbolisée en PHP par le double signe égal : ==. L'égalité en termes de valeurs et de types de données, c'est-à-dire l'identité, va être représentée en PHP par le triple signe égal : ===.

En effet, nous allons voir plus tard dans ce cours que les variables peuvent stocker différents types de données : des chaines de caractères, des nombres entiers, des nombres décimaux, etc. En utilisant des guillemets ou des apostrophes, on indique que la valeur stockée par la variable est une chaine de caractères.

Ainsi, si j'écris $age = "28" par exemple, la variable $age stockera la chaine de caractères 28. Cette chaine de caractères va être égale en valeur au nombre 28 mais les types de ces deux valeurs vont être différents (la première valeur est une chaine de caractères, la seconde est un nombre entier). Les deux variables ne vont pas être égales d'un point de vue de l'identité. Nous aurons l'occasion de réexpliquer tout cela dans les prochains chapitres.

A quoi servent de manière concrète les variables PHP ?

Les variables vont être extrêmement utiles en PHP et cela dans de nombreuses situations. Par exemple, elles vont nous permettre de manipuler des données non connues à priori. En effet, imaginons par exemple que vous souhaitiez manipuler (afficher, stocker, etc.) des données récoltées via un formulaire qui sera rempli par vos visiteurs.

Nous ne connaissons pas les valeurs qui vont être envoyées par nos visiteurs à priori. Pour manipuler ces données nous allons utiliser les variables. Ici, nous allons donc créer un script qui va traiter les données envoyées par les utilisateurs. Les valeurs envoyées vont être stockées dans des variables et nous allons manipuler ces variables.

Ainsi, à chaque fois qu'un visiteur enverra le formulaire avec ses données, notre script se déclenchera et les données seront placées dans des variables prédéfinies. Le script en

question sera créé de manière à manipuler ces variables et leur contenu pour les afficher, les stocker, etc.

Les variables vont également être un outil privilégié pour dynamiser notre site grâce à leur faculté de pouvoir stocker différentes valeurs. Imaginons ici que nous souhaitions afficher une horloge sur notre site.

Nous allons alors créer un script qui va recalculer toutes les secondes par exemple l'heure actuelle. Cette heure sera placée dans une variable `$heure` par exemple. Le script sera fait de telle sorte que le contenu de cette variable s'actualisera toutes les secondes (chaque seconde, la variable stockera une valeur actualisée qui sera l'heure actuelle) et affichera le contenu de cette variable.

Nous allons bien évidemment pouvoir utiliser les variables dans de nombreuses autres situations pour créer des scripts plus complexes. Cependant, il serait difficile à votre niveau actuel d'illustrer l'intérêt des variables en prenant appui sur une situation faisant appel à trop de connaissances hors de votre portée pour le moment.

Une nouvelle fois, pour le moment, faites-moi confiance et faîtes votre maximum pour comprendre au mieux chaque notion que je vous présente car elles vont nous être très utiles par la suite.

Les types de données PHP

Les variables PHP vont pouvoir stocker différents types de valeurs, comme du texte ou un nombre par exemple. Par abus de langage, nous parlerons souvent de « types de variables » PHP.

En PHP, contrairement à d'autres langages de programmation, nous n'avons pas besoin de préciser à priori le type de valeur qu'une variable va pouvoir stocker. Le PHP va en effet automatiquement détecter quel est le type de la valeur stockée dans telle ou telle variable, et nous allons ensuite pouvoir performer différentes opérations selon le type de la variable, ce qui va s'avérer très pratique pour nous !

Une conséquence directe de cela est qu'on va pouvoir stocker différents types de valeurs dans une variable au fil du temps sans se préoccuper d'une quelconque compatibilité. Par exemple, une variable va pouvoir stocker une valeur textuelle à un moment dans un script puis un nombre à un autre moment.

Les variables en PHP vont pouvoir stocker 8 grands types de données différents :

- Le type « chaine de caractères » ou `String` en anglais ;
- Le type « nombre entier » ou `Integer` en anglais ;
- Le type « nombre décimal » ou `Float` en anglais ;
- Le type « booléen » ou `Boolean` en anglais ;
- Le type « tableau » ou `Array` en anglais ;
- Le type « objet » ou `Object` en anglais ;
- Le type « NULL » qui se dit également `NULL` en anglais ;
- Le type « ressource » ou `Resource` en anglais ;

Nous allons pour le moment nous concentrer sur les types simples de valeurs. Les autres types feront l'objet de leçons ou de parties dédiées dans ce cours.

Le type chaîne de caractères ou String

Le premier type de données qu'une variable va pouvoir stocker est le type `String` ou chaîne de caractères. Une chaine de caractères est une séquence de caractères, ou ce qu'on appelle communément un texte.

Notez que toute valeur stockée dans une variable en utilisant des guillemets ou des apostrophes sera considérée comme une chaine de caractères, et ceci même dans le cas où nos caractères sont à priori des chiffres comme « 28 » par exemple.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prez = "Je m'appelle Pierre";
            $age = 28; //Stocke le nombre 28
            $age2 = "28"; //Stocke la chaine de caractères "28"
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, notre première variable `$prez` stocke la chaine de caractère « Je m'appelle Pierre ». Notre deuxième variable `$age`, quant à elle, stocke le nombre 28. En revanche, notre troisième variable `$age2` stocke la chaine de caractères « 28 » et non pas un nombre.

En effet, l'utilisation de guillemets ou d'apostrophe fait qu'une valeur est immédiatement considérée comme une chaine de caractères, quelle que soit cette valeur.

Pour s'en convaincre, on peut utiliser la fonction `gettype()` qui nous permet de connaître le type d'une variable (en anglais). Nous verrons plus en détail ce que sont les fonctions plus tard dans ce cours.

Pour le moment, il vous suffit de savoir que la fonction `gettype()` va renvoyer en résultat le type de la valeur stockée dans une variable. Nous allons ensuite utiliser une instruction `echo` pour afficher ce résultat renvoyé.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prez = "Je m'appelle Pierre";
            $age = 28; //Stocke le nombre 28
            $age2 = "28"; //Stocke la chaine de caractères "28"

            echo "La variable \$age contient une valeur de type ";
            echo gettype($age);

            echo "<br>La variable \$age2 contient une valeur de type ";
            echo gettype($age2);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Le code ci-dessus peut vous sembler complexe à votre niveau car nous effectuons deux opérations sur la même ligne : tout d'abord, on demande à `gettype()` va retourner le type de la valeur contenue dans notre variable puis on `echo` le résultat renvoyé par `gettype()`.

Ne cherchez pas forcément à tout comprendre immédiatement. Une nouvelle fois, nous reparlerons des fonctions plus tard dans ce cours. Pour le moment, nous voulions juste démontrer que `$age2` contient bien une valeur de type « chaine de caractères » ou `String` en anglais, ce qui est bien le cas.

Les types de données nombre entier (Integer) et nombre décimal (Float ou Double)

En PHP, on va pouvoir stocker deux types différents de donnée numéraires dans nos variables : le type Integer, qui contient tous les nombres entiers positifs ou négatifs et le type Float ou Double, qui contient les nombres décimaux (nombres à virgule) positifs ou négatifs.

On va donc pouvoir stocker un entier ou un nombre décimal dans une variable. Pour cela, il suffit d'affecter le nombre à stocker à notre variable, sans guillemet ni apostrophe.

Attention cependant : lorsque l'on code, on utilise toujours les notations anglo-saxonnes. Ainsi, il faudra préciser des points à la place de nos virgules pour les nombres relatifs. Voyons immédiatement un exemple ensemble :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prez = "Je m'appelle Pierre";
            $age = 28; //Stocke le nombre 28
            $age2 = "28"; //Stocke la chaine de caractères "28"
            $distance = 2.84;

            echo "La variable \$age contient une valeur de type ";
            echo gettype($age);

            echo "<br>La variable \$distance contient une valeur de type ";
            echo gettype($distance);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Le type de données booléen (Boolean)

Une variable en PHP peut encore stocker une valeur de type booléen (Boolean en anglais). Le type booléen est un type qui ne contient que deux valeurs : les valeurs true (vrai) et false (faux). Ce type n'est pas courant dans la vie de tous les jours mais est très (très) utilisé en informatique.

Nous aurons souvent recours aux booléens dans ce cours car ils vont être à la base de nombreux mécanismes en PHP, et aurons donc largement le temps de comprendre tout l'intérêt de ces valeurs.

Pour stocker une valeur de type booléen dans une variable, pensez bien à ne pas entourer true ou false par des guillemets ou des apostrophes. Si vous utilisez des guillemets ou apostrophes, en effet, les valeurs seront considérées comme étant de type chaine de caractères ou String et nous n'allons pas pouvoir les utiliser pour réaliser certaines opérations.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prez = "Je m'appelle Pierre";
            $age = 28; //Stocke le nombre 28
            $age2 = "28"; //Stocke la chaine de caractères "28"
            $distance = 2.84;
            $vrai = true;
            $faux = false;

            echo "La variable \$vrai contient une valeur de type ";
            echo gettype($vrai);

            echo "<br>La variable \$faux contient une valeur de type ";
            echo gettype($faux);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Le type de données Null

Le type de données `Null` est un type un peu particulier puisqu'il correspond à l'absence de valeur et sert donc à représenter des variables vides en PHP.

Ce type de valeur ne contient qu'une seule valeur : la valeur `NULL` qui correspond elle-même à l'absence de valeur. Il est un peu tôt pour vous faire comprendre l'intérêt de ce type de valeurs ; nous en reparlerons plus tard dans ce cours lorsque nous aurons à l'utiliser.

Notez que si vous déclarez une nouvelle variable sans lui affecter de valeur (ce qui est déconseillé de manière générale), cette variable sera automatiquement de type `Null`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prez = "Je m'appelle Pierre";
            $age = 28; //Stocke le nombre 28
            $age2 = "28"; //Stocke la chaine de caractères "28"
            $distance = 2.84;
            $vrai = true;
            $faux = false;
            $vide = NULL;
            $vide2;

            echo "La variable \$vide contient une valeur de type ";
            echo gettype($vide);

            echo "<br>La variable \$vide2 contient une valeur de type ";
            echo gettype($vide2);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Les types de données PHP tableau (Array) et objet (Object)

Les types de données `Array` et `Object` sont des types de données complexes particuliers qui méritent de faire chacun l'objet de chapitres séparés.

Nous n'étudierons donc pas ces deux types pour le moment car n'avons pas les connaissances suffisantes pour bien les comprendre.

Sachez simplement que l'on va pouvoir stocker plusieurs valeurs d'un coup à l'intérieur d'une variable en lui assignant des valeurs de type `Array` (tableau) ou `Object` (objet).

Le type de données ressource (Resource)

Une ressource est une variable particulière qui contient une référence vers une ressource externe au PHP, comme dans le cas d'une variable qui représente la connexion vers une base de données par exemple.

Là encore, ce type de données est complexe et nécessite d'avoir une bonne vision d'ensemble du langage pour être bien compris. Nous l'étudierons donc plus tard.

Opérateurs et concaténation

Dans cette nouvelle leçon, nous allons définir ce qu'est un opérateur, établir la liste des types d'opérateurs disponibles en PHP et apprendre à en manipuler certains.

Nous allons également préciser les différences entre l'utilisation des apostrophes ou des guillemets lorsqu'on manipule une valeur de type chaine de caractères.

Qu'est-ce qu'un opérateur ?

Un opérateur est un symbole qui va être utilisé pour effectuer certaines actions notamment sur les variables et leurs valeurs.

Par exemple, l'opérateur `+` va nous permettre d'additionner les valeurs de deux variables, tandis que l'opérateur `=` va nous permettre d'affecter une valeur à une variable.

La documentation officielle de PHP classe les différents opérateurs qu'on va pouvoir utiliser selon les groupes suivants :

- Les opérateurs arithmétiques ;
- Les opérateurs d'affectation ;
- Opérateurs sur les bits ;
- Opérateurs de comparaison ;
- Opérateur de contrôle d'erreur ;
- Opérateur d'exécution ;
- Opérateurs d'incrémentation et décrémentation ;
- Les opérateurs logiques ;
- Opérateurs de chaînes ;
- Opérateurs de tableaux ;
- Opérateurs de types ;

Dans cette leçon, nous allons nous concentrer sur les opérateurs arithmétiques, les opérateurs de chaines et les opérateurs d'affectation.

Nous verrons les autres types d'opérateurs au fil de ce cours lorsque cela fera le plus de sens (c'est-à-dire lorsqu'on en aura besoin).

Les opérateurs de chaines et la concaténation en PHP

Concaténer signifie littéralement « mettre bout à bout ». Pour indiquer qu'on souhaite concaténer, c'est-à-dire qu'on souhaite mettre bout à bout deux chaines de caractères en PHP on utilise le point (`.`) qu'on appelle également « opérateur de concaténation ».

Pour bien comprendre comment fonctionne l'opérateur de concaténation et son intérêt, il me semble nécessaire de connaitre les différences entre l'utilisation des guillemets et des apostrophes lorsqu'on manipule une chaine de caractères en PHP.

La différence majeure entre l'utilisation des guillemets et des apostrophes est que tout ce qui est entre guillemets va être interprété tandis que quasiment tout ce qui est entre apostrophes va être considéré comme une chaine de caractères.

Ici, « interprété » signifie « être remplacé par sa valeur ». Ainsi, lorsqu'on inclut une variable au sein d'une chaine de caractères et qu'on cherche à afficher le tout avec un `echo` par exemple en entourant la chaine complète avec des guillemets, la variable va être remplacée par sa valeur lors de l'affichage.

En revanche, lorsqu'on utilise des apostrophes, les variables ne vont pas être interprétées mais leur nom va être considéré comme faisant partie de la chaine de caractères.

Regardez plutôt l'exemple suivant :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = "Pierre";
            $nom = "Giraud";
            $age = 28;

            echo "Je m'appelle $prenom et j'ai $age ans <br>";
            echo "Je m'appelle {$prenom} et j'ai {$age} ans <br>";
            echo 'Je m\'appelle $prenom et j\'ai $age ans <br>';

            $prez = "Je suis $prenom $nom, j'ai $age ans <br>";
            $prez2 = "Je suis {$prenom} {$nom}, j'ai {$age} ans <br>";
            $prez3 = 'Je suis $prenom $nom, j\'ai $age ans';

            echo $prez;
            echo $prez2;
            echo $prez3;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, nous déclarons trois variables `$prenom`, `$nom` et `$age`.

On essaie ensuite d'afficher du texte avec des `echo` en incluant nos noms de variables au sein du texte.

Pour notre premier `echo`, on utilise des guillemets pour entourer le texte. Les variables dans le texte vont être interprétées et c'est leur contenu qui va être affiché.

Notez cependant ici que la syntaxe avec les noms de variables directement au milieu du texte est déconseillée aujourd'hui et qu'on préfèrera utiliser la syntaxe de de notre deuxième `echo` qui utilise des accolades pour entourer les variables.

Dans notre troisième `echo`, on utilise cette fois-ci des apostrophes. Les noms des variables ne vont donc pas être interprétés mais être considérés comme du texte et s'afficher tel quel.

Finalement, on crée de la même façon trois variables `$prez`, `$prez2` et `$prez3` qui stockent à nouveau du texte au sein duquel on inclut les noms de nos variables.

On `echo` alors le contenu de nos trois variables. Sans surprise, les variables `$prez` et `$prez2` stockent le texte donné avec le contenu des variables `$prenom`, `$nom` et `$age` tandis que la variable `$prez3` stocke le nom de ces variables plutôt que leurs valeurs.

L'opérateur de concaténation va nous permettre de mettre bout à bout les différentes données tout en faisant en sorte que chaque donnée soit interprétée par le PHP.

Nous allons l'utiliser pour séparer nos différentes variables des chaines de caractères autour. Regardez l'exemple suivant pour bien comprendre :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = "Pierre";
            $nom = "Giraud";
            $age = 28;
            $prez = "Je suis " .$prenom. " " .$nom. ", j'ai " .$age. " ans";
            $prez2 = 'Je suis ' .$prenom. ' ' .$nom. ', j\'ai '.$age. ' ans';


            echo "Je m'appelle " .$prenom. " et j'ai " .$age. " ans <br>";
            echo 'Je m\'appelle ' .$prenom. ' et j\'ai ' .$age. ' ans <br>';

            echo $prez. '<br>' .$prez2;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Pour concaténer correctement avec l'opérateur de concaténation, la règle est de séparer les différentes variables avec l'opérateur de concaténation (le point) des textes autour. Chaque texte devra être entouré de guillemets ou d'apostrophes selon ce qu'on a choisi.

A ce niveau, il est probable que vous vous demandiez l'intérêt d'utiliser l'opérateur de concaténation qui semble ici compliquer inutilement le code plutôt que simplement des guillemets et des accolades.

Ma réponse va être très simple : ici, vous pouvez utiliser l'une ou l'autre de ces méthodes pour un résultat identique. Cependant, rappelez-vous que l'utilisation d'apostrophes ou de

guillemets n'est pas identique au sens où ce qui est entre guillemets va être interprété tandis que la grande majorité de ce qui est entre apostrophes ne le sera pas.

Ainsi, parfois, on voudra utiliser des apostrophes plutôt que des guillemets et dans ce cas, si on souhaite que certaines de nos variables soient interprétées, il faudra utiliser l'opérateur de concaténation.

De manière générale, il est conseillé de toujours utiliser l'opérateur de concaténation lorsqu'on souhaite mettre bout-à-bout plusieurs chaines de caractères (qui seront généralement séparées par des variables), et ceci qu'on utilise des guillemets ou des apostrophes.

Les opérateurs arithmétiques

Les opérateurs arithmétiques vont nous permettre d'effectuer toutes sortes d'opérations mathématiques entre les valeurs contenues dans différentes variables lorsque ces valeurs sont des nombres.

Le fait de pouvoir réaliser des opérations entre variables va être très utile dans de nombreuses situations. Par exemple, si un utilisateur commande plusieurs produits sur notre site ou plusieurs fois un même produit et utilise un code de réduction, il faudra utiliser des opérations mathématiques pour calculer le prix total de la commande.

En PHP, nous allons pouvoir utiliser les opérateurs arithmétiques suivants :

Opérateur	**Nom de l'opération associée**
+	Addition
–	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division euclidienne)
**	Exponentielle (élévation à la puissance d'un nombre par un autre)

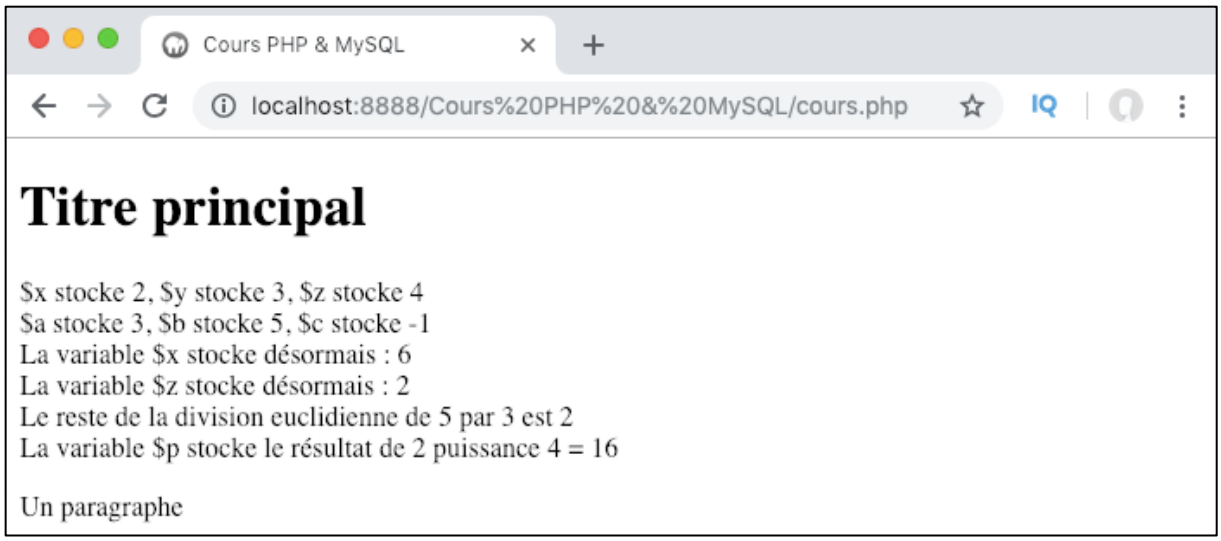
Avant d'utiliser les opérateurs arithmétiques, clarifions ce que sont le modulo et l'exponentielle.

Le modulo correspond au reste entier d'une division euclidienne. Par exemple, lorsqu'on divise 5 par 3, le résultat est 1 et il reste 2 dans le cas d'une division euclidienne. Le reste, 2, correspond justement au modulo.

L'exponentielle correspond à l'élévation à la puissance d'un nombre par un autre nombre. La puissance d'un nombre est le résultat d'une multiplication répétée de ce nombre par lui-même. Par exemple, lorsqu'on souhaite calculer 2 à la puissance de 3 (qu'on appelle

également « 2 exposant 3 »), on cherche en fait le résultat de 2 multiplié 3 fois par lui-même c'est-à-dire 2*2*2 = 8.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 2;
            $y = 3;
            $z = 4;
            echo '$x stocke ' .$x. ', $y stocke ' .$y. ', $z stocke ' .$z. '<br>';

            $a = $x + 1; //$a stocke 2 + 1 = 3
            $b = $x + $y; //$b stocke 2 + 3 = 5
            $c = $x - $y; //$c stocke 2 - 3 = -1
            echo '$a stocke ' .$a. ', $b stocke ' .$b. ', $c stocke ' .$c. '<br>';

            $x = $x * $y; //$x stocke désormais 2 * 3 = 6
            echo 'La variable $x stocke désormais : ' .$x. '<br>';

            $z = $x / $y; //$z stocke désormais 6 / 3 = 2
            echo 'La variable $z stocke désormais : ' .$z. '<br>';

            $m = 5 % 3; //$m stocke le reste de la division euclidienne de 5 par 3
            echo 'Le reste de la division euclidienne de 5 par 3 est ' .$m. '<br>';

            $p = $z ** 4; //$p stocke 2^4 = 2 * 2 * 2 * 2 = 16
            echo 'La variable $p stocke le résultat de 2 puissance 4 = ' .$p;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



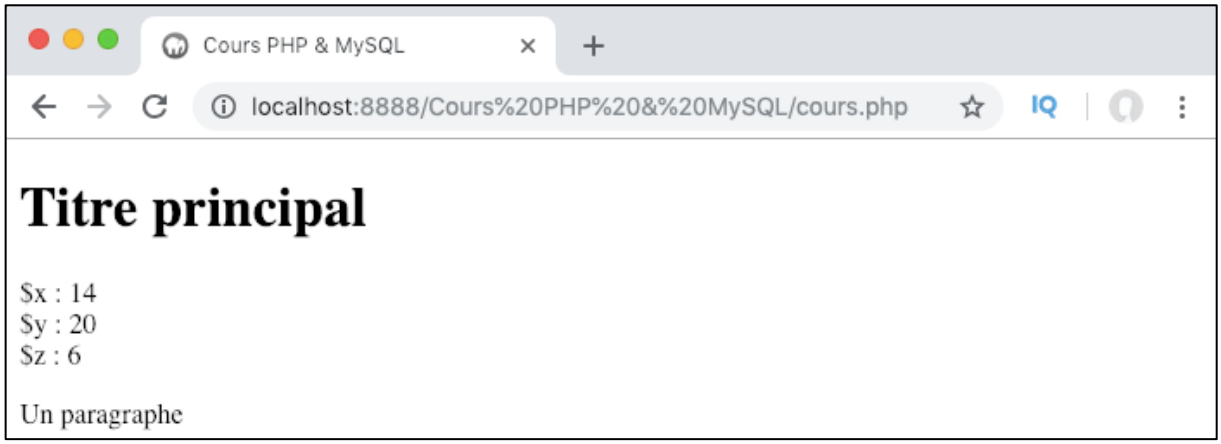
Concernant les règles de calcul, c'est-à-dire l'ordre de priorité des opérations, celui-ci va être le même qu'en mathématiques : l'élévation à la puissance va être prioritaire sur les autres opérations, tandis que la multiplication, la division et le modulo vont avoir le même ordre de priorité et être prioritaires sur l'addition et la soustraction qui ont également le même niveau de priorité.

Si deux opérateurs ont le même ordre de priorité, alors c'est leur sens d'association qui va décider du résultat. Pour les opérateurs arithmétiques, le sens d'association correspond à l'ordre de leur écriture à l'exception de l'élévation à la puissance qui sera calculée en partant de la fin.

Ainsi, si on écrit `$x = 1 – 2 – 3`, la variable `$x` va stocker la valeur -4 (les opérations se font de gauche à droite). En revanche, si on écrit `$x = 2 ** 3 ** 2`, la variable `$x` stockera 512 qui correspond à 2 puissance 9 puisqu'on va commencer par calculer 3 ** 2 = 9 dans ce cas.

Nous allons finalement, comme en mathématiques, pouvoir forcer l'ordre de priorité en utilisant des couples de parenthèses pour indiquer qu'une opération doit se faire avant toutes les autres :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 2 + 3 * 4; //$x stocke 14
            $y = (2 + 3) * 4; //$y stocke 20
            $z = 2 ** 3 - 4 * 4 / 8; //$z stocke 6

            echo '$x : ' .$x. '<br>$y : ' .$y. '<br>$z : ' .$z;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, `$x` stocke la valeur 14. En effet, la multiplication est prioritaire sur l'addition. On va donc commencer par faire 3 * 4 puis ajouter 2 au résultat.

La variable `$y` stocke 20. En effet, on utilise des parenthèses pour forcer la priorité de l'addition par rapport à la multiplication.

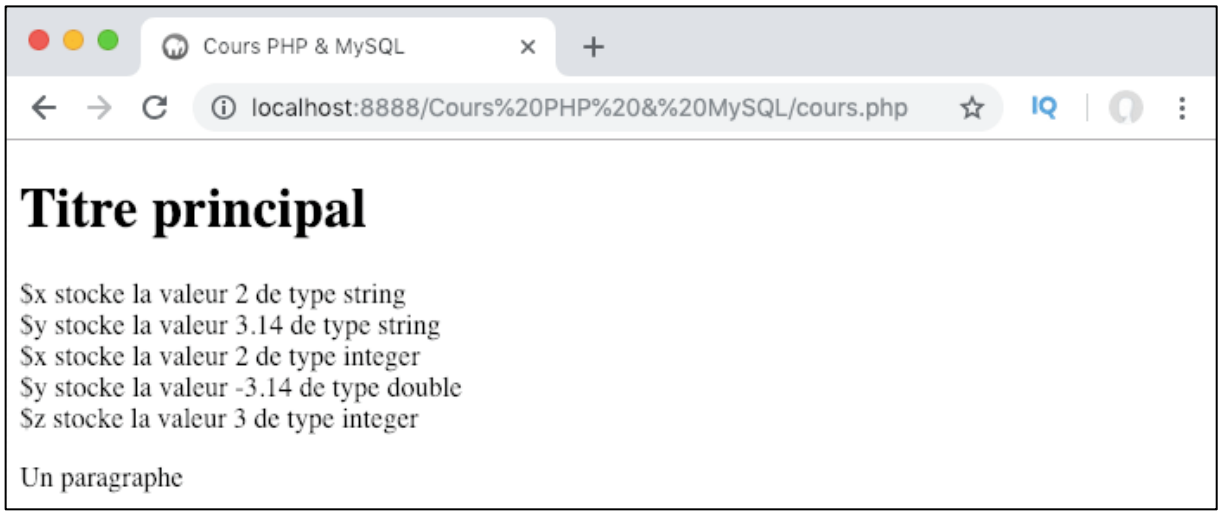
Finalement, `$z` stocke la valeur 6. En effet, on commence ici par calculer 2 puissance 3 (2 * 2 * 2 = 8). Ensuite, on calcule 4 * 4 / 8 = 16 / 8 = 2 car la multiplication et la division sont prioritaires sur la soustraction. Finalement, on calcule 8 – 2 = 6.

Notez également que les opérateurs `+` et `-` peuvent également servir à convertir le type de valeur contenue dans une variable vers `Integer` ou `Float` selon ce qui est le plus approprié.

Cette utilisation des opérateurs va pouvoir nous être utile lorsqu'on aura variables contenant des « nombres » stockés sous le type de chaines de caractères et pour

lesquelles on voudra réaliser des opérations mathématiques. Nous aurons l'occasion de rencontrer ce cas plus tard dans ce cours.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = "2";
            $y = "3.14";

            echo '$x stocke la valeur ' .$x. ' de type ' .gettype($x). '<br>';
            echo '$y stocke la valeur ' .$y. ' de type ' .gettype($y). '<br>';

            $x = +$x;
            $y = -$y;
            $z = +"3";

            echo '$x stocke la valeur ' .$x. ' de type ' .gettype($x). '<br>';
            echo '$y stocke la valeur ' .$y. ' de type ' .gettype($y). '<br>';
            echo '$z stocke la valeur ' .$z. ' de type ' .gettype($z);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Les opérateurs d'affectation et opérateurs combinés

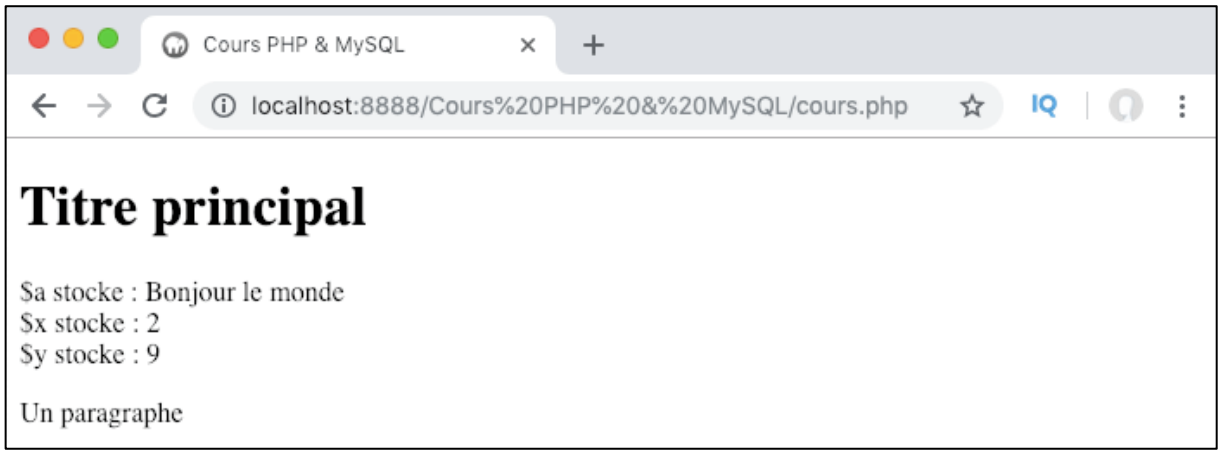
Les opérateurs d'affectation vont nous permettre, comme leur nom l'indique, d'affecter une certaine valeur à une variable.

Nous connaissons déjà bien l'opérateur d'affectation le plus utilisé qui est le signe =. Cependant, vous devez également savoir qu'il existe également des opérateurs combinés notamment pour les opérateurs arithmétiques et l'opérateur de concaténation et qui sont les suivants :

Opérateur	**Définition**
.=	Concatène puis affecte le résultat
+=	Additionne puis affecte le résultat
-=	Soustrait puis affecte le résultat
*=	Multiplie puis affecte le résultat
/=	Divise puis affecte le résultat
%=	Calcule le modulo puis affecte le résultat
**=	Élève à la puissance puis affecte le résultat

Illustrons immédiatement cela et voyons comment se servir de ces opérateurs :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $a = "Bonjour";
            $a .= " le monde"; //$a stocke "Bonjour le monde"
            echo '$a stocke : ' .$a. '<br>';

            $x = 5;
            $x -= 3; //$x stocke désormais 2
            echo '$x stocke : ' .$x. '<br>';

            $y = 3;
            $y **= $x; //$y stocke 3^2 = 3 * 3 = 9
            echo '$y stocke : ' .$y;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ce qu'il faut bien comprendre dans l'exemple précédent est que les opérateurs d'affectation combinés font deux choses à la fois : ils exécutent une opération puis ils affectent une valeur.

Au début, notre variable `$a` stocke `Bonjour`. Ensuite, on utilise l'opérateur d'affectation concaténant `.=` qui va concaténer la valeur à droite avec la valeur contenue dans la variable à gauche avant de lui affecter le résultat.

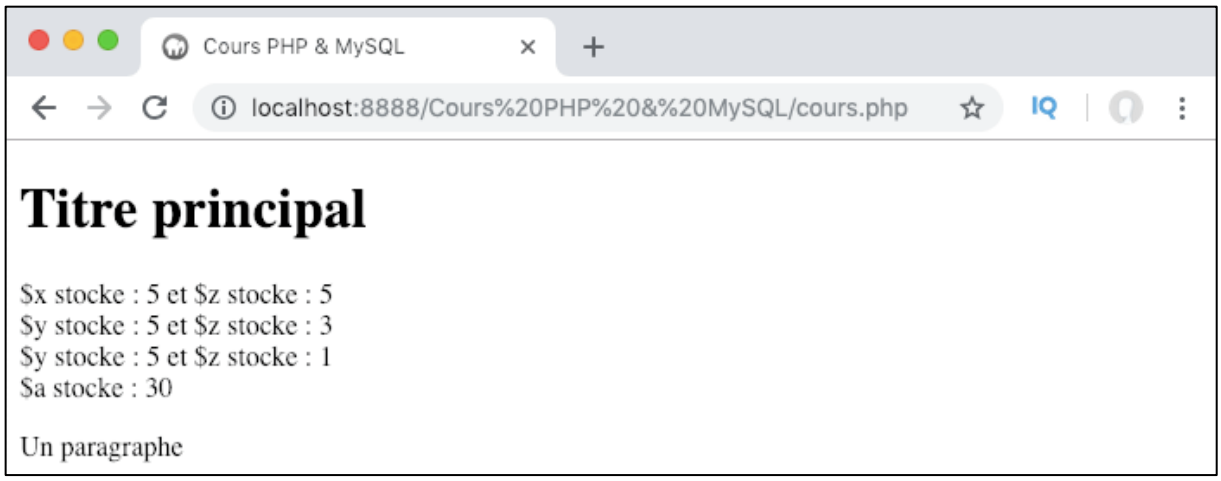
Ici, on concatène donc la chaine de caractères `le monde` avec la valeur `Bonjour` et on affecte le résultat (c'est-à-dire les deux chaines concaténées) dans la variable `$a`. La variable `$a` va donc désormais stocker `Bonjour le monde`.

On fait la même chose avec `$x` en dessous : `$x` stocke au départ 5, puis on lui soustrait 3 avant d'affecter à nouveau le résultat (5 – 3 = 2) à `$x` qui va donc désormais stocker 2.

Par ailleurs, notez que tous les opérateurs d'affectation ont une priorité de calcul égale mais qui est inférieure à celle des opérateurs arithmétiques ou de concaténation.

Lorsque des opérateurs ont des ordres de priorité égaux, c'est le sens d'association de ceux-ci qui va décider du résultat. Pour les opérateurs arithmétiques, on a vu que l'association se faisait par la gauche sauf pour l'élévation à la puissance. Pour les opérateurs d'affectation, l'association se fait par la droite.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
    <h1>Titre principal</h1>
    <?php
        $x = 1;
        $y = 2;
        $z = 3;
        $a = 5;

        $x = $z += 2; //$z stocke 5 et $x stocke 5
        echo '$x stocke : ' .$x. ' et $z stocke : ' .$z. '<br>';

        $y += $z -= 2; //$z stocke 5 - 2 = 3 et $y stocke 2 + 3 = 5
        echo '$y stocke : ' .$y. ' et $z stocke : ' .$z. '<br>';

        $y /= $z -= 2; //$z stocke 1 et $y stocke 5
        echo '$y stocke : ' .$y. ' et $z stocke : ' .$z. '<br>';

        $a *= 4 + 2; //$z stocke 30
        echo '$a stocke : ' .$a;
    ?>
    <p>Un paragraphe</p>
    </body>
</html>
```



Pour notre premier calcul, nous utilisons les deux opérateurs d'affectation `=` et `+=`. L'association va se faire par la droite. On commence donc à ajouter 2 à la valeur de `$z` qui stocke désormais 5 et on stocke la même valeur dans `$x`. Faites bien attention ici : `$x` ne stocke bien évidemment pas la variable `$z` mais seulement la dernière valeur connue de `$z`. Si on modifie ensuite la valeur de `$z`, cela n'a aucun impact sur `$x`.

Les deux exemples suivants utilisent à nouveau deux opérateurs d'affectation. L'association va donc toujours se faire par la droite.

Dans notre dernier exemple, cependant, on utilise à la fois un opérateur d'affectation et un opérateur arithmétique. Les opérateurs arithmétiques sont prioritaires sur les opérateurs d'affectation. On commence donc par réaliser l'opération arithmétique (4 + 2 = 6) et on multiplie ensuite la valeur de `$a` par 6 avant d'affecter la nouvelle valeur dans `$a`.

PARTIE III

Structures de contrôle

Présentation des conditions et des opérateurs de comparaison

Dans cette nouvelle partie, nous allons étudier et comprendre l'intérêt des structures de contrôle en PHP. Une structure de contrôle est un ensemble d'instructions qui permet de contrôler l'exécution du code.

Il existe différents types de structures de contrôle. Les deux types les plus connus et les plus utilisés sont les structures de contrôle conditionnelles qui permettent d'exécuter un bloc de code si une certaine condition est vérifiée et les structures de contrôle de boucle qui permettent d'exécuter un bloc de code en boucle tant qu'une condition est vérifiée.

Nous allons déjà commencer avec l'étude des structures de contrôle conditionnelles.

Présentation des conditions en PHP

Les structures de contrôle conditionnelles (ou plus simplement conditions) vont nous permettre d'exécuter différents blocs de code selon qu'une condition spécifique soit vérifiée ou pas.

Par exemple, on va pouvoir utiliser les conditions pour afficher un message de bienvenue différent en PHP sur notre site selon que l'utilisateur soit connu ou un simple visiteur qui ne s'est jamais inscrit sur notre site.

Nous allons très souvent utiliser les conditions avec des variables : selon la valeur stockée dans une variable, nous allons vouloir exécuter un bloc de code plutôt qu'un autre.

Les conditions vont ainsi être un passage incontournable pour rentre un site dynamique puisqu'elles vont nous permettre d'exécuter différents codes et ainsi afficher différents résultats selon le contexte.

En PHP, nous allons pouvoir utiliser les conditions suivantes :

- La condition `if` (si) ;
- La condition `if... else` (si… sinon) ;
- La condition `if... elseif... else` (si… sinon si… sinon).

Nous allons étudier chacune de ces conditions dans la suite de cette partie.

Présentation des opérateurs de comparaison

Comme je l'ai précisé plus haut, nous allons souvent construire nos conditions autour de variables : selon la valeur d'une variable, nous allons exécuter tel bloc de code ou pas.

En pratique, nous allons donc comparer la valeur d'une variable à une certaine autre valeur donnée et selon le résultat de la comparaison exécuter un bloc de code ou pas. Pour comparer des valeurs, nous allons devoir utiliser des opérateurs de comparaison.

Voici ci-dessous les différents opérateurs de comparaison disponibles en PHP ainsi que leur signification :

Opérateur	Définition
==	Permet de tester l'égalité sur les valeurs
===	Permet de tester l'égalité en termes de valeurs et de types
!=	Permet de tester la différence en valeurs
<>	Permet également de tester la différence en valeurs
!==	Permet de tester la différence en valeurs ou en types
<	Permet de tester si une valeur est strictement inférieure à une autre
>	Permet de tester si une valeur est strictement supérieure à une autre
<=	Permet de tester si une valeur est inférieure ou égale à une autre
>=	Permet de tester si une valeur est supérieure ou égale à une autre

Certain de ces opérateurs nécessitent certainement une précision de ma part. Avant tout, vous devez bien comprendre que lorsqu'on utilise un opérateur de comparaison en PHP, on n'indique pas au PHP que telle valeur est supérieure, inférieure, égale ou différente de telle autre.

Au contraire, les opérateurs de comparaisons nous servent à demander au PHP de comparer deux opérandes (les « opérandes » sont les valeurs situées de chaque côté d'un opérateur), c'est-à-dire d'évaluer si telle valeur est bien supérieure, inférieure, égale ou différente (selon l'opérateur utilisé) à telle autre valeur. Le PHP va ensuite renvoyer une valeur selon le résultat de la comparaison.

Revenons à nos opérateurs. Tout d'abord, notez que notre « égal » mathématique (l'égalité en termes de valeurs) se traduit en PHP par le double signe égal `==`.

Ensuite, certains d'entre vous doivent certainement se demander ce que signifie le triple égal. Lorsqu'on utilise un triple égal `===`, on cherche à effectuer une comparaison non seulement sur la valeur mais également sur le type des deux opérandes.

Prenons un exemple simple pour illustrer cela. Imaginons que l'on possède une variable `$x` dans laquelle on stocke le chiffre 4. On veut ensuite comparer la valeur stockée dans notre variable à la chaîne de caractères « 4 ».

Si on utilise le double signe égal pour effectuer la comparaison, l'égalité va être validée par le PHP car celui-ci ne va tester que les valeurs, et 4 est bien égal à « 4 » en termes de valeurs.

En revanche, si on utilise le triple signe égal, alors l'égalité ne va pas être validée car nous comparons un nombre à une chaine de caractères (donc des types différents de valeurs). On va suivre exactement le même raisonnement pour les deux opérateurs != et !== qui vont nous permettre de tester respectivement la différence en termes de valeurs simplement et la différence en termes de valeurs ou de type.

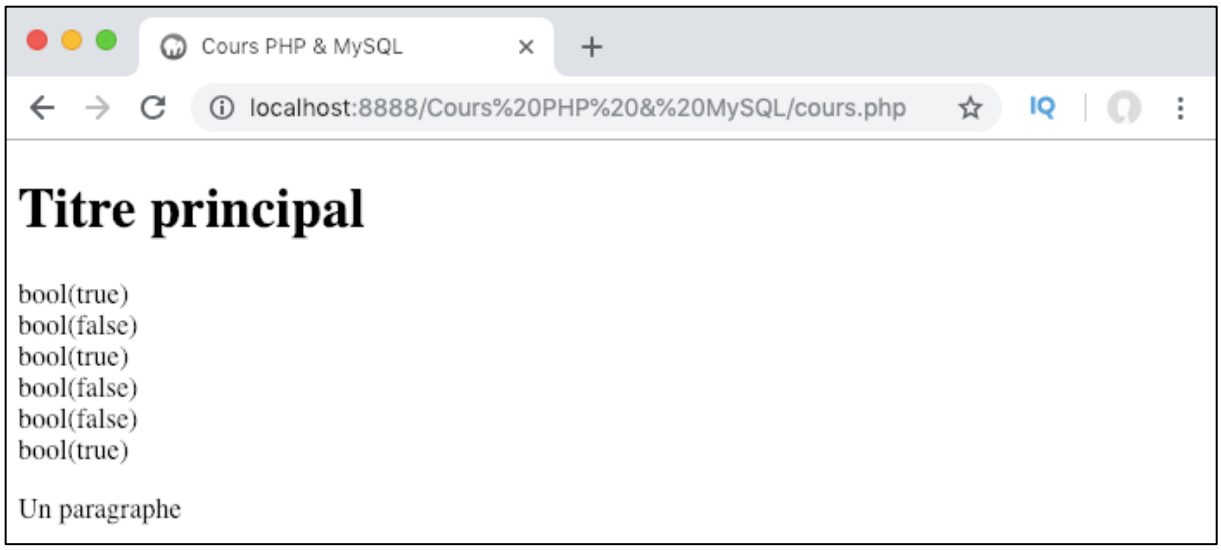
Utiliser les opérateurs de comparaison

Il y a différentes choses que vous devez savoir et comprendre pour bien utiliser les opérateurs de comparaison.

La première chose à savoir est que lorsqu'on utilise un opérateur de comparaison, le PHP va comparer la valeur à gauche de l'opérateur à celle à droite. On dit également qu'il évalue la comparaison.

Si la comparaison est vérifiée ou validée, alors le PHP renvoie la valeur booléenne true. Si le test de comparaison échoue, alors PHP renvoie la valeur booléenne false. Cela est très important à comprendre car nos conditions vont s'appuyer sur cette valeur booléenne pour décider du code à exécuter ou pas.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4; //On affecte la valeur 4 à $x

            /*On compare la valeur contenue dans $x à 4 en valeur.
             *On renvoie le résultat de la comparaison grâce à var_dump()*/
            var_dump($x == 4);
            echo '<br>';

            var_dump($x > 7);
            echo '<br>';

            /*On compare la valeur de $x à la chaine de caractères "4" en
             *valeur simplement*/
            var_dump($x == "4");
            echo '<br>';

            /*On compare la valeur de $x à la chaine de caractères "4" en
             *termes de valeur et de type*/
            var_dump($x === "4");
            echo '<br>';

            var_dump($x != "4");
            echo '<br>';

            var_dump($x !== "4");
            echo '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Expliquons les résultats obtenus. Tout d'abord, vous remarquez qu'on a utilisé la fonction `var_dump()` pour afficher les résultats plutôt qu'un `echo`.

Cela a été fait pour une raison simple : `echo` va transformer toute valeur en chaîne de caractères avant de l'afficher. Or, l'équivalent de la valeur booléenne `true` en chaîne de caractères est la chaine de caractères « 1 » tandis que `false` devient « 0 » ou une chaine de caractères vide.

Comme je voulais absolument vous montrer que le PHP renvoyait bien une valeur booléenne après l'utilisation d'opérateurs de comparaison, j'ai donc été obligé d'utiliser la fonction `var_dump()` à la place de `echo`.

Le rôle de la fonction PHP `var_dump()` est en effet d'afficher les informations structurées d'une variable ou d'une expression et notamment son type et sa valeur. C'est pour cela que le navigateur nous affiche des `bool(true)` ou `bool(false)`.

Ceci étant dit, analysons maintenant chacune de nos opérations en détail et expliquons les résultats en soi.

Pour commencer, on compare la valeur contenue dans `$x` au chiffre 4. Si le PHP considère l'égalité vérifiée, il renvoie le booléen `true`. Dans le cas contraire, il renverra code>false. Comme vous pouvez le voir, c'est la valeur `true` qui est renvoyée. En effet, la valeur de notre variable est bien égale en valeur au chiffre 4.

On compare ensuite la valeur contenue dans `$x` au chiffre 7 avec l'opérateur de supériorité absolue. Si le PHP détermine que la valeur contenue dans `$x` est strictement supérieure au chiffre 7, il renvoie `true`. Dans le cas contraire, il renvoie code>false. Dans le cas présent, `$x` contient le chiffre 4, qui n'est pas strictement supérieur au chiffre 7. C'est donc la valeur code>false qui est renvoyée.

Pour notre troisième comparaison, on décide de comparer la valeur de `$x` à la chaine de caractères « 4 ». On utilise le double signe égal pour cela, on ne va donc comparer que les valeurs. Comme la valeur 4 est bien égale en valeur à la chaine de caractères « 4 », PHP renvoie `true`.

Ensuite, on compare à nouveau la valeur contenue dans notre variable `$x` à la chaine de caractères « 4 ». Cependant, cette fois-ci, on utilise le triple signe égal. En faisant cela, on signifie que l'on veut comparer les valeurs mais également les types de chacune des deux valeurs. Comme un nombre et une chaine de caractères n'ont pas le même type, PHP renvoie cette fois-ci `false`.

Dans notre cinquième opération, on demande à PHP de déterminer si la valeur contenue dans `$x` différente (en valeur) de la chaine de caractères « 4 ». Comme le chiffre 4 n'est pas différent en valeur de la chaine de caractères « 4 », le PHP renvoie `false` (car rappelez-vous qu'on teste ici la différence).

Enfin, on demande à PHP de déterminer si la valeur contenue dans `$x` est différente en valeur ou en type de la chaine de caractères « 4 ». Le chiffre 4 est bien d'un type différent de la chaine de caractères « 4 », et donc le PHP renvoie `true`.

Les opérateurs de comparaison ternaire, spaceship et fusion null

La dernière version majeure du PHP, le PHP 7 a introduit deux nouveaux opérateurs de comparaison qui se comportent différemment des précédents puisque le PHP ne va pas renvoyer `true` ou `false` à l'issue de la comparaison.

Le premier de ces opérateurs est l'opérateur « spaceship » `<=>`. A l'issue de la comparaison des deux opérandes, le PHP va ici renvoyer :

- 0 dans le cas où les deux opérandes sont égaux ;
- -1 si l'opérande à gauche de l'opérateur est plus petit que celui de droite ;
- 1 si l'opérande à gauche de l'opérateur est plus grand que celui de droite.

Le deuxième opérateur est l'opérateur « fusion null » `??`. Cet opérateur va principalement nous permettre de comparer le contenu de deux variables. Le PHP va ici renvoyer :

- La valeur de l'opérande à droite de l'opérateur si la valeur de l'opérande à gauche est `NULL` ;
- La valeur de l'opérande à gauche de l'opérateur dans tous les autres cas.

Le dernier opérateur est l'opérateur ternaire `? :`. Nous allons étudier en détail ces trois opérateurs dans une prochaine leçon.

Les conditions if, if…else et if…elseif…else PHP

Nous savons désormais utiliser les opérateurs de comparaison. Il est donc temps d'entrer dans le vif du sujet et d'apprendre à créer nos premières structures conditionnelles.

Dans cette nouvelle leçon, nous allons étudier les structures de contrôle conditionnelles `if`, `if...else` et `if... elseif... else`.

La condition if en PHP

La structure de contrôle conditionnelle, ou plus simplement « condition » `if` est l'une des plus importantes et des plus utilisées dans l'ensemble des langages de programmation utilisant les structures de contrôle et en particulier en PHP.

La condition `if` est également la plus simple, puisqu'elle va nous permettre d'exécuter un bloc de code si et seulement si le résultat d'un test vaut `true`.

Créons immédiatement nos premières conditions `if` :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4; //On affecte la valeur 4 à $x
            $y = 2; //On affecte la valeur 2 à $y

            if($x > 1){
                echo '$x contient une valeur supérieure à 1';
            }

            if($x == $y){
                echo '$x et $y contiennent la même valeur';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, j'ai créé deux conditions `if` avec une comparaison qui va être évaluée à `true` et une autre qui va être évaluée à `false`.

Commencez par noter la syntaxe de nos conditions qui est la suivante : `if(test){code à exécuter si le test est validé}`. Il convient de différencier ici « test » et « comparaison » : le test correspond à l'ensemble du code écrit dans les parenthèses et peut être composé de plusieurs comparaisons comme on va le voir par la suite.

Dans notre première condition, le résultat de la comparaison renvoyé par le PHP est `true` puisque notre variable `$x` stocke le chiffre 4 qui est bien supérieur à 1. Le code dans la condition est alors exécuté.

Dans notre deuxième condition, la comparaison est cette fois-ci évaluée à `false` car la valeur contenue dans `$x` n'est pas égale en valeur à la valeur contenue dans `$y`. Le code contenu dans la condition ne sera donc pas lu ni exécuté.

Inverser la valeur logique d'une condition

Dans les exemples ci-dessus, le code placé dans notre condition n'est exécuté que si le résultat de la comparaison est `true`.

Dans certaines situations, nous préférerons créer nos conditions de telle sorte à ce que le code dans la condition soit exécuté si le résultat de la comparaison est `false`.

Nous allons pouvoir faire cela de deux manières : soit en utilisant l'opérateur logique inverse `!` que nous étudierons dans la leçon suivante, soit en comparant le résultat de notre comparaison à `false`.

Concentrons-nous pour le moment sur cette deuxième méthode qui demande de bien connaitre l'ordre de priorité des opérateurs et de bien comprendre l'associativité de ceux-ci.

Il y a deux choses à savoir ici :

- Les opérateurs `<`, `<=`, `>` et `>=` ont une priorité plus importante que les opérateurs `==`, `===`, `!=`, `!==`, `<>` et `<>` qui ont eux-mêmes une priorité plus importante que l'opérateur `??`;
- Les opérateurs de comparaison sont non-associatifs (à l'exception de l'opérateur `??` dans l'associativité se fait par la droite), ce qui signifie que si ils ont

une priorité équivalente on ne pourra pas les utiliser entre eux. Cela signifie qu'on n'aura pas le droit d'écrire la comparaison `2 < 4 > 2` par exemple en PHP car les opérateurs `<` et `>` ont le même ordre de priorité et sont non associatifs.

Ainsi, si on écrit par exemple `2 < 4 == false`, PHP va d'abord effectuer la comparaison `2 < 4` puis comparer le résultat de cette comparaison (qui est ici `true` car 2 est bien inférieur à 4) à `false`. Ici, `true` est différent de `false` et donc le test va finalement échouer.

Pour inverser la valeur logique d'une condition, c'est-à-dire pour exécuter le code de la condition uniquement lorsque notre première comparaison est évaluée à `false`, il suffit donc de comparer le résultat de cette première comparaison à la valeur `false`.

Si notre première comparaison n'est pas vérifiée et est évaluée à `false`, alors le test de notre condition va devenir `if(false == false)` ce qui va être finalement évalué à `true` et donc le code de notre condition va bien être exécuté !

Notez qu'on va également pouvoir utiliser les parenthèses pour forcer la priorité d'un opérateur sur un autre, ce qui peut être utile lorsqu'on souhaite utiliser plusieurs opérateurs de comparaison qui ont une priorité équivalente par défaut.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4; //On affecte la valeur 4 à $x
            $y = 2; //On affecte la valeur 2 à $y

            if(($x <= 1) == false){
                echo '$x contient une valeur supérieure à 1';
            }

            if(($x != $y) == false){
                echo '$x et $y contiennent la même valeur';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, notre variable `$x` stocke la valeur 4 qui est supérieure à 1. La partie `$x <= 1` de notre première condition va donc être évaluée à `false`. On compare ensuite le résultat de la comparaison à `false`. On obtient donc ici `if(false == false)` qui va être évalué à `true` puisque c'est le cas et donc le code dans notre condition va bien être exécuté.

On utilise le même procédé pour notre deuxième condition que je vous laisse décortiquer seul pour que vous puissiez vous entrainer.

La condition if…else en PHP

Avec la condition `if`, nous restons relativement limités puisque cette condition nous permet seulement d'exécuter un bloc de code selon que le résultat d'un test soit évalué à `true`.

La structure conditionnelle `if...else` (« si… sinon » en français) va être plus complète que la condition `if` puisqu'elle va nous permettre d'exécuter un premier bloc de code si un test renvoie `true` ou un autre bloc de code dans le cas contraire.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4; //On affecte la valeur 4 à $x
            $y = 2; //On affecte la valeur 2 à $y

            if($x > 1){
                echo '$x contient une valeur stric. supérieure à 1 <br>';
            }else{
                echo '$x contient une valeur inférieure ou égale à 1 <br>';
            }

            if($x == $y){
                echo '$x et $y contiennent la même valeur <br>';
            }else{
                echo '$x et $y contiennent des valeurs différentes <br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Notez la syntaxe de la condition `if...else` : on place notre comparaison et on effectue notre test dans le `if` mais pas dans le `else`.

En effet, la structure `else` est une structure qui a été créée pour prendre en charge tous les cas nos gérés précédemment. Ainsi, on ne précisera jamais de condition au sein d'un `else` puisque par défaut cette structure prend en charge tous les autres cas (tous les cas non gérés par le `if` ici).

Si le test de notre condition est validé, le code dans le `if` va s'exécuter et le code dans le `else` va alors être ignoré.

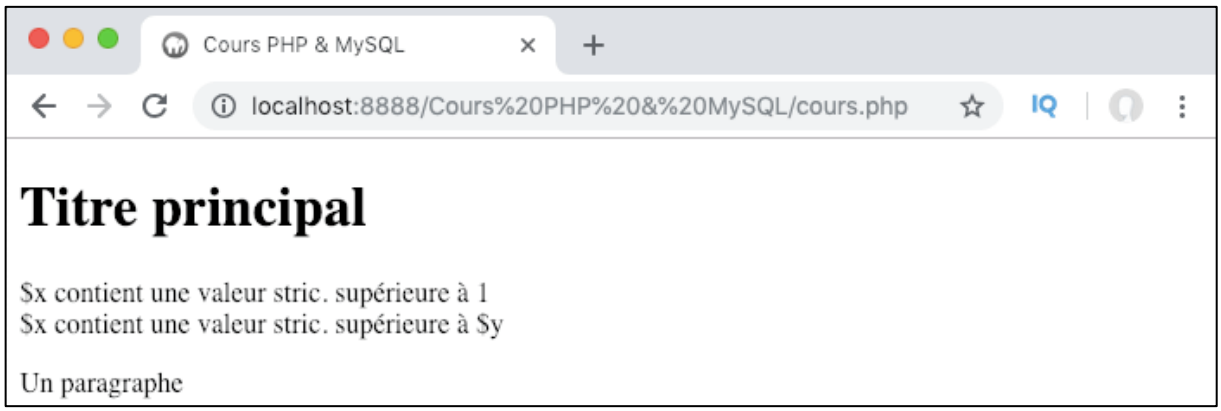
Si au contraire le test n'est pas validé alors le code dans le `if` va être ignoré et c'est le cas dans le `else` qui va être exécuté.

La condition if…elseif…else en PHP

La condition `if...elseif...else` (« si…sinon si…sinon ») est une structure conditionnelle encore plus complète que la condition `if...else` puisqu'elle va nous permettre cette fois-ci de générer autant de cas que l'on souhaite.

En effet, nous allons pouvoir écrire autant de `elseif` (en un seul mot, attention) que l'on veut dans notre condition `if...elseif...else`, chacun avec un test différent.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4; //On affecte la valeur 4 à $x
            $y = 2; //On affecte la valeur 2 à $y

            if($x > 1){
                echo '$x contient une valeur stric. supérieure à 1 <br>';
            }elseif($x == 1){
                echo '$x contient la valeur 1 <br>';
            }else{
                echo '$x contient une valeur stric. inférieure à 1 <br>';
            }

            if($x == $y){
                echo '$x et $y contiennent la même valeur <br>';
            }elseif($x < $y){
                echo '$x contient une valeur stric. inférieure à $y <br>';
            }elseif($x > $y){
                echo '$x contient une valeur stric. supérieure à $y <br>';
            }else{
                echo 'Les valeurs de $x et $y n\'ont pas pu être comparées';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Comme vous pouvez le constater, les `elseif` occupent un rôle similaire au `if` de départ puisque chacun d'entre eux va posséder son propre test (qui est obligatoire).

Notez que dans le cas où plusieurs `elseif` possèdent un test qui va être évalué à `true`, seul le code du premier `elseif` rencontré sera exécuté. En effet, dès qu'un test va être validé, le PHP va ignorer les tests suivants.

Notez également qu'on devra toujours obligatoirement terminer notre condition `if...elseif...else` avec un `else` qui servira à gérer toutes les issues (ou les cas) non pris en charge par le `if` ou par les `elseif`.

Utiliser les opérateurs logiques pour créer des conditions robustes

Dans cette nouvelle leçon, nous allons apprendre à créer des conditions complexes qui vont utiliser plusieurs comparaisons grâce aux opérateurs logiques.

Imbriquer plusieurs conditions

Souvent, nous allons vouloir comparer plusieurs valeurs au sein d'une même condition, c'est-à-dire n'exécuter son code que si plusieurs conditions sont vérifiées.

Pour faire cela, nous allons pouvoir soit utiliser les opérateurs logiques, soit imbriquer plusieurs conditions les unes dans les autres.

Les opérateurs logiques vont nous permettre de créer des conditions plus puissantes mais dans certains cas il sera plus intéressant et plus rapide d'imbriquer des conditions.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $inscrit = true;
            $age = 21;

            if($inscrit){
                if($age >= 18){
                    echo 'Utilisateur inscrit et majeur, accès autorisé';
                }else{
                    echo 'Utilisateur inscrit et mineur, accès restreint';
                }
            }else{
                echo 'Utilisateur non inscrit, accès refusé';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Dans cet exemple, on veut tester les critères « utilisateur inscrit ou non » et « utilisateur majeur ou non » et répondre de façon différente en fonction de chaque situation.

La façon la plus simple de faire cela est de créer une première condition `if...else` et d'imbriquer une deuxième condition `if...else` à l'intérieur du premier `if`.

Le PHP va d'abord évaluer la comparaison du premier `if`. Si le test est validé, on entre alors dans le deuxième `if` et on évalue la condition de celui-ci. Si le test du premier `if` échoue, alors on passe directement au `else` sans jamais rentrer dans le deuxième `if...else`.

Notez par ailleurs que comme dans notre cas notre variable `$inscrit` stocke une valeur booléenne on n'a pas besoin de faire de comparaison explicite puisque la valeur `true` est elle-même évaluée à `true` tandis que `false` est évaluée à `false` par le PHP.

Présentation des opérateurs logiques

Les opérateurs logiques vont être principalement utilisés avec les conditions puisqu'ils vont nous permettre d'écrire plusieurs comparaisons au sein d'une même condition ou encore d'inverser la valeur logique d'un test.

En PHP, nous pouvons utiliser les opérateurs logiques suivants :

Opérateur	Définition
AND	Renvoie `true` si toutes les comparaisons valent `true`
&&	Renvoie `true` si toutes les comparaisons valent `true`
OR	Renvoie `true` si une des comparaisons vaut `true`
||	Renvoie `true` si une des comparaisons vaut `true`
XOR	Renvoie `true` si une des comparaisons exactement vaut `true`
!	Renvoie `true` si la comparaison vaut `false` (et inversement)

Comme vous pouvez le constater, les opérateurs logiques « ET » et « OU » peuvent s'écrire de deux façons différentes : soit avec les mots clefs `AND` et `OR`, soit avec les signes `&&` et `||`.

Ces deux écritures ne sont pas tout à fait équivalentes : la différence réside dans l'ordre des priorités de traitement des opérations. En effet, l'écriture avec des signes a une priorité plus importante que l'écriture avec des mots clefs.

Attention également à ne pas confondre les opérateurs logiques `OR` et `XOR` en PHP : si on utilise l'opérateur `OR`, le PHP va renvoyer `true` si au moins une des comparaisons vaut `true` et renverra donc `true` si plusieurs comparaisons valent `true` tandis qu'en utilisant l'opérateur `XOR` le PHP ne renverra `true` que si une seule comparaison vaut `true` (et reverra `false` si plusieurs comparaisons valent `true`).

Récapitulatif : l'ordre de priorité des opérateurs

Au cours des leçons précédentes, nous avons vu les types d'opérateurs les plus communs : opérateurs arithmétiques, d'affectation, de chaine, de comparaison et maintenant logiques.

Les seuls types d'opérateurs communs que nous n'avons pas encore étudié sont les opérateurs d'incrémentation et de décrémentation ainsi que l'opérateur ternaire.

A ce niveau du cours, il me semble donc important de faire un récapitulatif global de la priorité de tous ces opérateurs et de leur associativité afin de pouvoir les utiliser de la meilleure manière.

La tableau suivant liste les différents opérateurs vus jusqu'ici ainsi que les opérateurs d'incrémentation et de décrémentation et le ternaire qu'on étudiera plus tard selon leur priorité (la première ligne du tableau contient les opérateurs avec la plus grande priorité et etc. jusqu'à la dernière ligne contenant les opérateurs avec la plus petite priorité).

Opérateurs	**Associativité**
`**`	droite
`++` (incrémentation), `--` (décrémentation)	droite
`!`	droite
`*`, `/`, `%`	gauche
`+`, `-`, `.`	gauche
`<`, `<=`, `>`, `>=`	non-associatif
`==`, `===`, `!=`, `!==`, `<>`, `<=>`	non-associatif
`&&`	gauche

Opérateurs	Associativité
`||`	gauche
`??`	droite
`? :` (ternaire)	gauche
`=`, `+=`, `-=`, `*=`, `/=`, `%=`, `**=`, `.=`	droite
`AND`	gauche
`XOR`	gauche
`OR`	gauche

Je sais que ça fait beaucoup de choses à retenir et que ce genre de choses ne fait pas partie des choses les plus marrantes ou intéressantes à apprendre à priori. Cependant, connaitre la priorité des opérateurs est indispensable pour bien les utiliser et nous allons beaucoup utiliser les opérateurs en PHP.

Je vous demande donc ici de faire votre maximum pour ne pas directement « passer à la suite » mais plutôt pour prendre le temps de comprendre et de retenir ces différentes priorités et l'associativité des différents opérateurs.

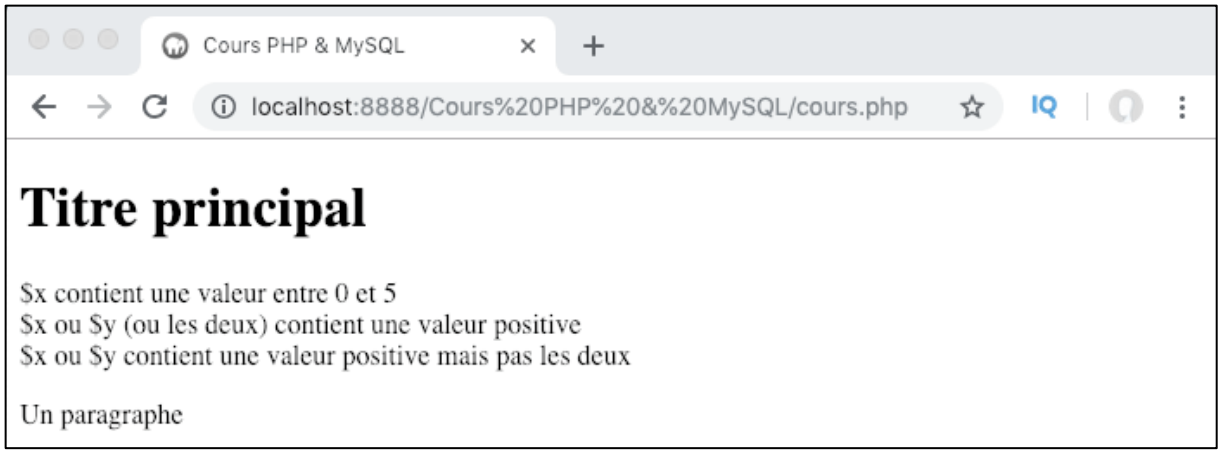
Utilisation des opérateurs logiques avec les conditions

Une condition avec plusieurs comparaisons

Les opérateurs logiques `AND`, `&&`, `OR`, `||` et `XOR` vont nous permettre de créer des conditions avec plusieurs comparaisons.

On va ainsi pouvoir choisir d'exécuter un code uniquement si toutes les comparaisons valent `true`, ou si au moins l'une d'entre elles vaut `true`, ou encore si seulement l'une d'entre elles vaut `true`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4;
            $y = -12;

            if($x >= 0 AND $x <= 5){
                echo '$x contient une valeur entre 0 et 5 <br>';
            }

            if($x >= 0 AND $y >= 0){
                echo '$x et $y contiennent une valeur positive <br>';
            }

            if($x >= 0 OR $y >= 0){
                echo '$x ou $y (ou les deux) contient une valeur positive <br>';
            }

            if($x >= 0 XOR $y >= 0){
                echo '$x ou $y contient une valeur positive mais pas les deux';
            }

        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



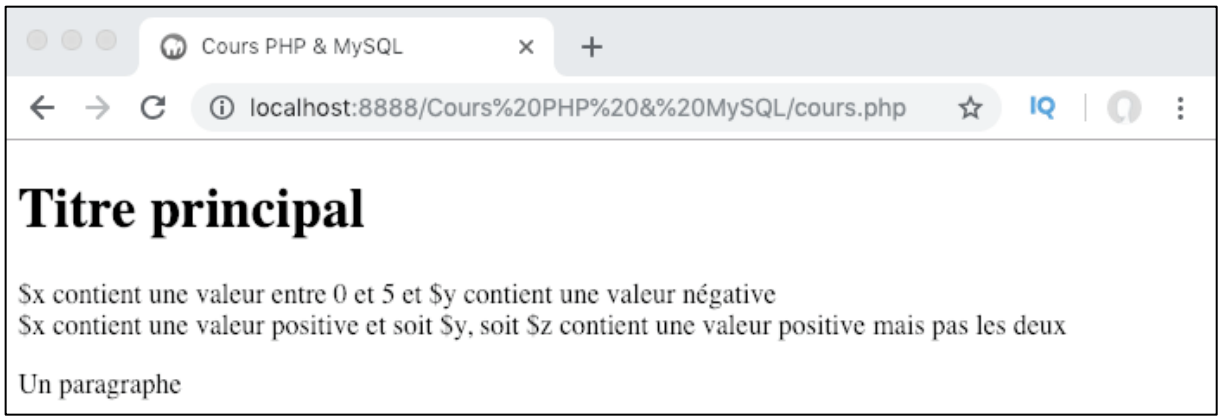
Dans l'exemple ci-dessus, notre première condition utilise l'opérateur `AND`. Chacune des deux comparaisons va devoir être évaluée à `true` pour que le code de la condition soit exécuté.

Notre deuxième condition est équivalente dans sa structure à la première. La seule différence est qu'on effectue deux comparaisons avec deux variables différentes.

Notre troisième condition utilise l'opérateur `OR`. Dans ce cas, il suffit qu'une comparaison soit évaluée à `true` pour que le code de notre condition soit exécuté.
Notre dernière condition utilise elle l'opérateur `XOR`. Pour que le code de la condition soit exécuté, il va falloir ici qu'une comparaison exactement soit évaluée à `true`.

De plus, on va tout à fait pouvoir utiliser plusieurs opérateurs logiques dans une même condition pour ajouter des règles et des comparaisons à celle-ci. Il faudra ici faire bien attention à la priorité des différents opérateurs pour obtenir le résultat voulu ou utiliser des parenthèses pour forcer la priorité de certains opérateurs par rapport à d'autres.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4;
            $y = -12;
            $z = 1;

            if($x >= 0 AND $x <= 5 AND $y <= 0){
                echo '$x contient une valeur entre 0 et 5 et $y contient une valeur
                négative <br>';
            }else

            if($x >= 0 AND $y >= 0 XOR $z >= 0){
                echo '$x et $y contiennent tous les deux une valeur positive et $z
                une valeur stric. négative ou le contraire<br>';
            }

            if($x >= 0 AND ($y >= 0 XOR $z >= 0)){
                echo '$x contient une valeur positive et soit $y, soit $z contient
                une valeur positive mais pas les deux <br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Pour comprendre comment fonctionnent les conditions ci-dessus, il faut savoir que l'opérateur `AND` possède une priorité plus importante que `XOR`.

Notre première condition utilise deux opérateurs `AND`. Ici, c'est relativement simple : il va falloir que chacune des trois comparaisons soit évaluée à `true` pour que le code dans notre condition soit exécuté.

Notre deuxième condition utilise les opérateurs `AND` et `XOR`. L'opérateur `AND` a la priorité sur `XOR` et donc le test va être analysé comme suivant : on va d'abord tester si `$x` et `$y` contiennent toutes les deux une valeur positive puis on va tester si `$z` contient une valeur négative.

Si l'un de ces deux tests et seulement l'un d'entre eux renvoie `true`, alors le code de la condition est exécuté.

Dans notre troisième condition, on utilise cette fois-ci des parenthèses pour imposer la priorité des opérations. Ici, on va tester d'un côté si `$x` contient une valeur positive et renvoyer `true` si c'est le cas puis si `$y` ou `$z` possèdent une valeur positive et renvoyer `true` si c'est le cas pour seulement l'une des deux variables. Si nos deux tests renvoient `true` et `true`, le code dans la condition sera exécuté.

Inverser la valeur logique d'une variable, d'une condition ou d'un test

L'opérateur logique inverse `!` va nous permettre d'inverser la valeur logique d'une variable, d'une condition ou d'un test. Grosso modo, cela signifie qu'à chaque fois que le PHP va évaluer quelque chose à `true` à `false`, l'opérateur `!` va inverser cette valeur (`true` va devenir `false` et inversement).

Cet opérateur va donc très simplement nous permettre d'exécuter le code de nos conditions lorsque le test de la condition a échoué (c'est-à-dire lorsque PHP renvoie `false`).

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4;
            $y = -12;

            if(!$x >= 0 AND !$y >= 0){
                echo 'Ce texte est toujours affiché !<br>';
            }

            if(!($x >= 0) AND !($y >= 0)){
                echo 'Ce texte s\'affiche uniquement si $x et $y contiennent
                toutes les deux une valeur stric. négative <br>';
            }

             if(!($x >= 0 AND $y >= 0)){
                echo 'Soit $x contient une valeur stric. négative, soit $y
                contient une valeur stric. négative, soit les deux variables
                contiennent une valeur stric. négative';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Pour comprendre les résultats ci-dessus, il convient à nouveau de bien connaitre les priorités des opérateurs.

Dans le cas présent, l'opérateur ! a une priorité plus importante que les autres opérateurs logiques et que les opérateurs de comparaison.

Le résultat de notre première condition est peut-être le moins évident à comprendre car de nombreux mécanismes entrent en jeu ici. Essayons d'expliquer ce qu'il se passe.

Dans notre première condition, nous plaçons l'opérateur `!` devant nos deux variables `$x` et `$y` sans utiliser de parenthèses. Ici, il faut bien comprendre que PHP va commencer par évaluer `!$x` et `!$y` et renvoyer un booléen (`true` ou `false`).

L'opérateur logique `!` inverse une valeur logique, c'est-à-dire transforme `true` en `false` et `false` en `true`.

Le PHP va donc déjà effectuer un transtypage et commencer par évaluer les valeurs de `$x` et `$y` en termes booléen pour ensuite pouvoir inverser la valeur obtenue.

Ici, vous devez savoir que lors d'une conversion en booléen, seules les valeurs suivantes sont évaluées à false :

- Le booléen `false` lui-même ;
- Le chiffre 0 ;
- La chaine de caractères vide et la chaine de caractères « 0 » ;
- Le type `NULL` ;
- Un tableau vide.

Toute autre valeur est évaluée à `true`.

La variable `$x` contient 4 tandis que `$y` contient -12. En termes booléen, les deux variables vont être évaluées à `true` et donc `!$x` et `!$y` vont renvoyer `false`.

Ensuite, on va comparer les valeurs obtenues à 0 avec l'opérateur `>=` pour nos deux variables. Ici, le PHP va effectuer un nouveau transtypage et convertir les valeurs booléennes obtenues en entier. La booléen `false` évalué comme entier est égal à 0 (`true` est égal à 1).

Or, 0 est bien supérieur ou égal à 0 et donc `!$x >= 0` et `!$y >= 0` sont évalués à `true` et donc notre test renvoie finalement `true`.

Dans ce premier cas, le test ne peut finalement jamais renvoyer `false` puisque quelles que soient les valeurs de `$x` et `$y` ces variables vont d'abord être évaluées en termes booléen et donc l'inverse (qui sera l'autre booléen) sera toujours soit `true` soit `false` qui, une fois transtypées en entiers, donneront 1 ou 0 qui est bien dans tous les cas supérieur ou égal à 0.

Ce premier exemple, je vous rassure, était complexe et particulièrement à votre niveau mais il permet de bien comprendre comment l'opérateur `!` fonctionne et comment PHP évalue les variables selon les opérateurs.

Dans notre deuxième condition, on utilise cette fois-ci des parenthèses pour indiquer que les comparaisons `$x >= 0` et `$y >= 0` doivent être évaluées avant d'inverser la valeur logique obtenue.

Ici, `$x >= 0` renvoie `true` puisque 4 est bien supérieur à 0 et `$y >= 0` renvoie `false` puisque -12 est strictement inférieur à 0. Ces deux valeurs sont ensuite inversées et on obtient `if(false AND true)` qui est évalué à `false`.

Dans notre troisième et dernière condition, on inverse le résultat final des comparaisons. Ici, on a toujours `$x >= 0` qui renvoie `true` et `$y >= 0` qui renvoie `false` et donc `$x >= 0 AND $y >= 0` qui renvoie `false`. On inverse ensuite la valeur logique et on obtient finalement `true`.

Les opérateurs ternaire et fusion null

Dans cette nouvelle leçon, nous allons étudier en détail deux opérateurs de comparaison qu'on a jusqu'à présent laissé volontairement de côté : l'opérateur ternaire `?:` et l'opérateur fusion null `??`.

Ces deux opérateurs vont nous permettre d'écrire des conditions plus condensées et donc d'alléger nos scripts et de gagner du temps en développement.

L'opérateur ternaire et les structures conditionnelles ternaires

Les structures conditionnelles ternaires (souvent simplement abrégées "ternaires") correspondent à une autre façon d'écrire nos conditions en utilisant une syntaxe basée sur l'opérateur ternaire `?:` qui est un opérateur de comparaison.

Les ternaires vont utiliser une syntaxe très condensée et nous allons ainsi pouvoir écrire toute une condition sur une ligne et accélérer la vitesse d'exécution de notre code.

Avant de vous montrer les écritures ternaires, je dois vous prévenir : beaucoup de développeurs n'aiment pas les ternaires car elles ont la réputation d'être très peu lisibles et très peu compréhensibles.

Ce reproche est à moitié justifié : d'un côté, on peut vite ne pas comprendre une ternaire si on est un développeur moyen ou si le code qui nous est présenté n'est pas ou mal commenté. De l'autre côté, si vous indentez et commentez bien votre code, vous ne devriez pas avoir de problème à comprendre une structure ternaire.

Il reste cependant fortement recommandé de ne pas imbriquer les structures ternaires les unes dans les autres tout simplement car votre code deviendrait vite illisible et car le comportement de PHP lors de l'utilisation de plus d'un opérateur ternaire dans une seule instruction n'est pas évident.

Les structures ternaires vont se présenter sous la forme suivante : `test ? code à exécuter si true : code à exécuter si false`.

Illustrons immédiatement cela avec un exemple :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4; //On affecte la valeur 4 à $x

            //Ecrire ceci :
            if($x >= 0){
                echo '$x stocke un nombre positif<br>';
            }else{
                echo '$x stocke un nombre négatif<br>';
            }

            //Est equivalent à cela :
            echo $x >= 0 ? '$x stocke un nb positif' : '$x stocke un nb négatif';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Notez également qu'on va pouvoir abréger encore plus nos ternaires en omettant la partie centrale de l'opérateur ternaire, c'est-à-dire le code entre le point d'interrogation et les deux points.

En utilisant cette écriture, c'est la partie située avant l'opérateur qui sera renvoyée si le test est évalué à `true` et la partie après l'opérateur sinon.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4; //On affecte la valeur 4 à $x

            //Ecrire ceci :
            if($x >= 0){
                echo ($x >= 0).'<br>';
            }else{
                echo '$x stocke un nombre négatif<br>';
            }

            //Est equivalent à cela :
            echo $x >= 0 ?: '$x stocke un nb négatif';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, il faut bien comprendre que PHP ne va pas renvoyer le contenu de `$x` mais le résultat de la comparaison `$x >= 0`, c'est-à-dire `true`, convertie sous forme d'entier (c'est-à-dire 1).

On est ici obligés d'utiliser un couple de parenthèses dans notre instruction `if` pour que le point soit bien considéré comme un opérateur de concaténation et pas comme appartenant à 0.

L'opérateur fusion null

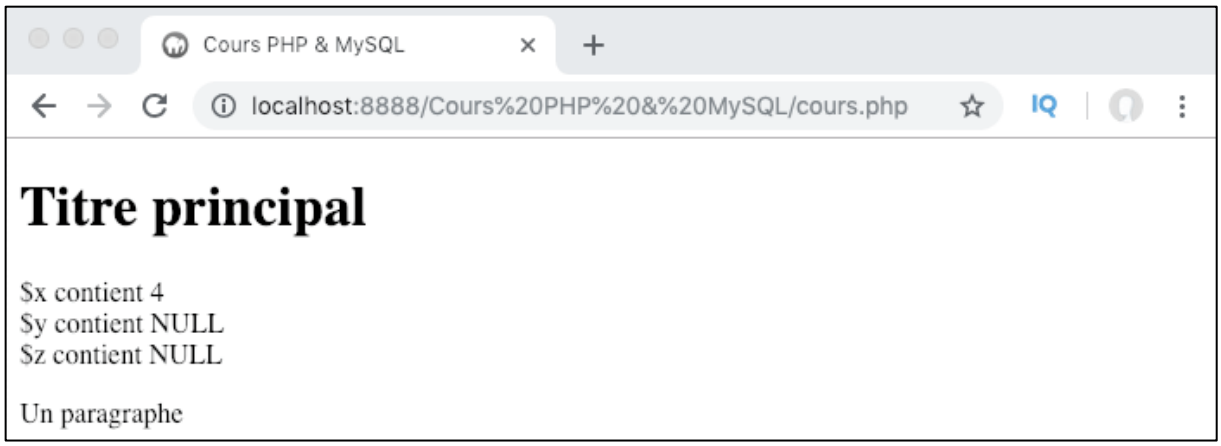
L'opérateur fusion null `??` ressemble dans sa syntaxe et dans son fonctionnement à l'opérateur ternaire.

Cet opérateur utilise la syntaxe suivante : `test ?? code à renvoyer si le résultat du test est NULL`.

Si l'expression à gauche de l'opérateur vaut `NULL`, alors l'expression à droite de l'opérateur sera renvoyée. Dans tous les autres cas, c'est l'expression à gauche de l'opérateur qui sera renvoyée.

On va généralement utiliser cet opérateur pour tester si une variable contient quelque chose ou pas et, dans le cas où la variable est vide, lui faire stocker une valeur.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 4; //On affecte la valeur 4 à $x
            $y = NULL;
            $z;

            $resultatx = $x ?? 'NULL';
            $resultaty = $y ?? 'NULL';
            $resultatz = $z ?? 'NULL';

            echo '$x contient ' .$resultatx. '<br>
                  $y contient ' .$resultaty. '<br>
                  $z contient ' .$resultatz;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on teste nos variables `$x`, `$y` et `$z` en utilisant l'opérateur fusion null. Si la variable est vide ou contient `NULL`, la valeur `NULL` sera renvoyée et stockée dans `$resultatx`, `$resultaty` ou `$resultatz`. Dans le cas contraire, c'est le contenu de la variable qui sera stocké.

On affiche ensuite le contenu de nos variables stockant les résultats suite à l'utilisation de l'opérateur fusion null en utilisant `echo`.

L’instruction switch en PHP

L’instruction `switch` va nous permettre d’exécuter un code en fonction de la valeur d’une variable. On va pouvoir gérer autant de situations ou de « cas » que l’on souhaite.

En cela, l’instruction `switch` représente une alternative à l’utilisation d’un `if...elseif...else` mais qui est plus limitante puisque le `switch` ne va gérer que l’égalité comme type de comparaison.

En dehors de cela, on va pouvoir utiliser un `switch` ou une condition « classique » de manière équivalente en PHP. Il n’y a donc pas de réel intérêt à utiliser le `switch` plutôt qu’un `if...elseif...else`.

Cependant, dans certains cas, utiliser un `switch` peut rendre le code plus clair et légèrement plus rapide dans son exécution.

La syntaxe d’une instruction `switch` va être différente de celle des conditions vues jusqu’ici. Regardez plutôt l’exemple ci-dessous :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 2;

            switch($x){
                case 0:
                    echo '$x stocke la valeur 0';
                    break;
                case 1:
                    echo '$x stocke la valeur 1';
                    break;
                case 2:
                    echo '$x stocke la valeur 2';
                    break;
                case 3:
                    echo '$x stocke la valeur 3';
                    break;
                case 4:
                    echo '$x stocke la valeur 4';
                    break;
                default:
                    echo '$x ne stocke pas de valeur entre 0 et 4';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

La première chose à noter ici est qu'on doit fournir une variable sur laquelle on va « switcher ».

Ensuite, l'instruction switch va s'articuler autour de case qui sont des cas. Si la valeur de notre variable est égale à celle du case, alors on exécute le code qui est à l'intérieur.

Par exemple, le code contenu dans case 0: va être exécuté si la valeur contenue dans notre variable est 0, le code contenu dans case 1: va être exécuté si la valeur contenue dans notre variable est 1, etc.

Chaque case d'un switch doit se terminer par une instruction break. Cette instruction indique au PHP qu'il doit sortir du switch.

Sans break, le PHP continuerait à tester les différents autres case du switch même si un case égal à la valeur de la variable a été trouvé, ce qui ralentirait inutilement le code et pourrait produire des comportements non voulus.

Enfin, à la fin de chaque switch, il convient d'indiquer une instruction default. Le default est l'équivalent du else des conditions vues précédemment : il sert à gérer tous les autres cas et son code ne sera exécuté que si aucun case ne correspond à la valeur de la variable.

Pas la peine d'utiliser une instruction break au sein de default puisque default sera toujours placée en fin de switch. Si le PHP arrive jusqu'au default, alors il sortira ensuite naturellement du switch puisque celui-ci ne contient plus aucun code après default.

Encore une fois, le switch ne présente pas en PHP de réel intérêt par rapport à l'utilisation d'une condition classique en dehors du fait qu'utiliser un switch peut dans certains cas réduire le temps d'exécution d'un script et que cette structure est parfois plus claire qu'un if...elseif...else contenant des dizaines de elseif.

Pour le moment, je vous conseille tout de même de vous entrainer avec tous les outils que je vous présente. Vous pourrez par la suite décider de ne pas utiliser ceci ou cela, mais pour le moment il est essentiel que vous ayez une vue d'ensemble des fonctionnalités de base proposées par PHP.

Présentation des boucles et des opérateurs d'incrémentation et de décrémentation

Les boucles, tout comme les conditions, représentent l'une des fondations du PHP et il est donc essentiel de comprendre comment elles fonctionnent et de savoir les utiliser.

Présentation des boucles

Les boucles vont nous permettre d'exécuter plusieurs fois un bloc de code, c'est-à-dire d'exécuter un code « en boucle » tant qu'une condition donnée est vérifiée.

Lorsqu'on code, on va en effet souvent devoir exécuter plusieurs fois un même code. Utiliser une boucle nous permet de n'écrire le code qu'on doit exécuter plusieurs fois qu'une seule fois.

Nous allons ainsi pouvoir utiliser les boucles pour parcourir et afficher plusieurs valeurs un une seule instruction, comme par exemple récupérer la liste des produits achetés dans une commande, afficher le prénom de tous les utilisateurs de notre site, récupérer des valeurs jusqu'à un certain point donné dans une variable tableau, etc.

Nous disposons de quatre boucles différentes en PHP :

- La boucle `while` (« tant que ») ;
- La boucle `do... while` (« faire… tant que ») ;
- La boucle `for` (« pour ») ;
- La boucle `foreach` (« pour chaque »).

Le fonctionnement général des boucles sera toujours le même : on pose une condition qui sera généralement liée à la valeur d'une variable et on exécute le code de la boucle « en boucle » tant que la condition est vérifiée.

Pour éviter de rester bloqué à l'infini dans une boucle, vous pouvez donc déjà noter qu'il faudra que la condition donnée soit fausse à un moment donné (pour pouvoir sortir de la boucle).

Pour que notre condition devienne fausse à un moment, on incrémentera ou on décrémentera la valeur de notre variable à chaque nouveau passage dans la boucle. Voyons immédiatement ce que tout cela signifie.

Les opérateurs d'incrémentation et de décrémentation

Incrémenter une valeur signifie ajouter 1 à cette valeur tandis que décrémenter signifie enlever 1.

Les opérations d'incrémentation et de décrémentation vont principalement être utilisées avec les boucles en PHP. Elles vont pouvoir être réalisées grâce aux opérateurs d'incrémentation `++` et de décrémentation `--`.

Retenez déjà qu'il y a deux façons d'incrémenter ou de décrémenter une variable : on peut soit incrémenter / décrémenter la valeur de la variable puis retourner la valeur de la variable incrémentée ou décrémentée (on parle alors de pré-incrémentation et de pré-décrémentation), soit retourner la valeur de la variable avant incrémentation ou décrémentation puis ensuite l'incrémenter ou la décrémenter (on parle alors de post-incrémentation et de post-décrémentation).

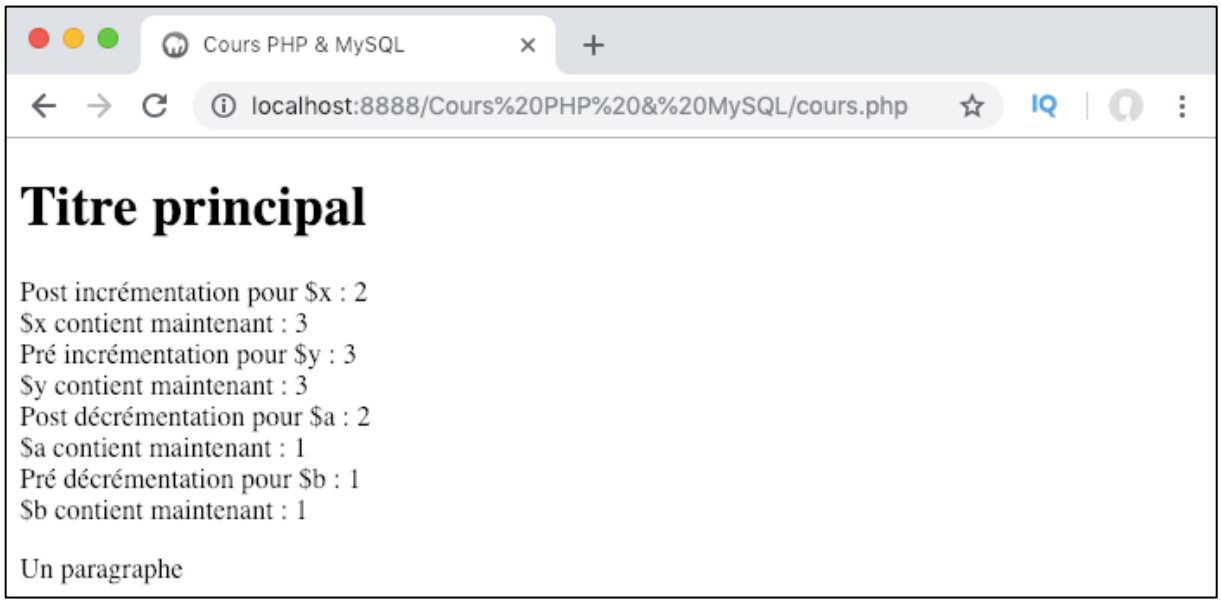
Cette différence d'ordre de traitement des opérations va influer sur le résultat de nombreux codes et notamment lorsqu'on voudra en même temps incrémenter ou décrémenter la valeur d'une variable et l'afficher avec une instruction `echo` ou la manipuler d'une quelconque façon. Tenez-en donc bien compte à chaque fois que vous utilisez les opérateurs d'incrémentation ou de décrémentation.

Le tableau ci-dessous présente les différentes façons d'utiliser les opérateurs d'incrémentation et de décrémentation ainsi que le résultat associé :

Exemple	Résultat
`++$x`	Pré-incrémentation : incrémente la valeur contenue dans la variable $x, puis retourne la valeur incrémentée
`$x++`	Post-incrémentation : retourne la valeur contenue dans $x avant incrémentation, puis incrémente la valeur de $x
`--$x`	Pré-décrémentation : décrémente la valeur contenue dans la variable $x, puis retourne la valeur décrémentée
`$x--`	Post-décrémentation : retourne la valeur contenue dans $x avant décrémentation, puis décrémente la valeur de $x

Prenons immédiatement un exemple concret pour illustrer les différences entre pré et post incrémentation ou décrémentation.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 2; $y = 2; $a = 2; $b = 2;

            echo 'Post incrémentation pour $x : ' .$x++. '<br>';
            echo '$x contient maintenant : ' .$x. '<br>';

            echo 'Pré incrémentation pour $y : ' .++$y. '<br>';
            echo '$y contient maintenant : ' .$y. '<br>';

            echo 'Post décrémentation pour $a : ' .$a--. '<br>';
            echo '$a contient maintenant : ' .$a. '<br>';

            echo 'Pré décrémentation pour $b : ' .--$b. '<br>';
            echo '$b contient maintenant : ' .$b. '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Cet exemple montre les priorités de traitement du PHP selon que l'on place les signes `++` et `--` avant ou après le nom de notre variable.

Dans le premier cas, on place l'opérateur d'incrémentation ++ après notre variable et on incrémente au sein d'un echo. On utilise ici la post-incrémentation : la valeur originale de $x va être affiché puis cette valeur va ensuite être incrémentée.

On voit que si on echo à nouveau la valeur dans $x ensuite, la valeur est bien incrémentée.

Dans le second cas, en revanche, on place l'opérateur ++ avant le nom de notre variable $y ce qui signifie qu'on va ici la pré-incrémenter. Ici, l'incrémentation va se faire avant toute autre opérateur et c'est donc la valeur après incrémentation qui va être affichée.

On réitère ensuite exactement les mêmes opérations mais en utilisant cette fois-ci des opérateurs de décrémentation avec nos variables â et $b.

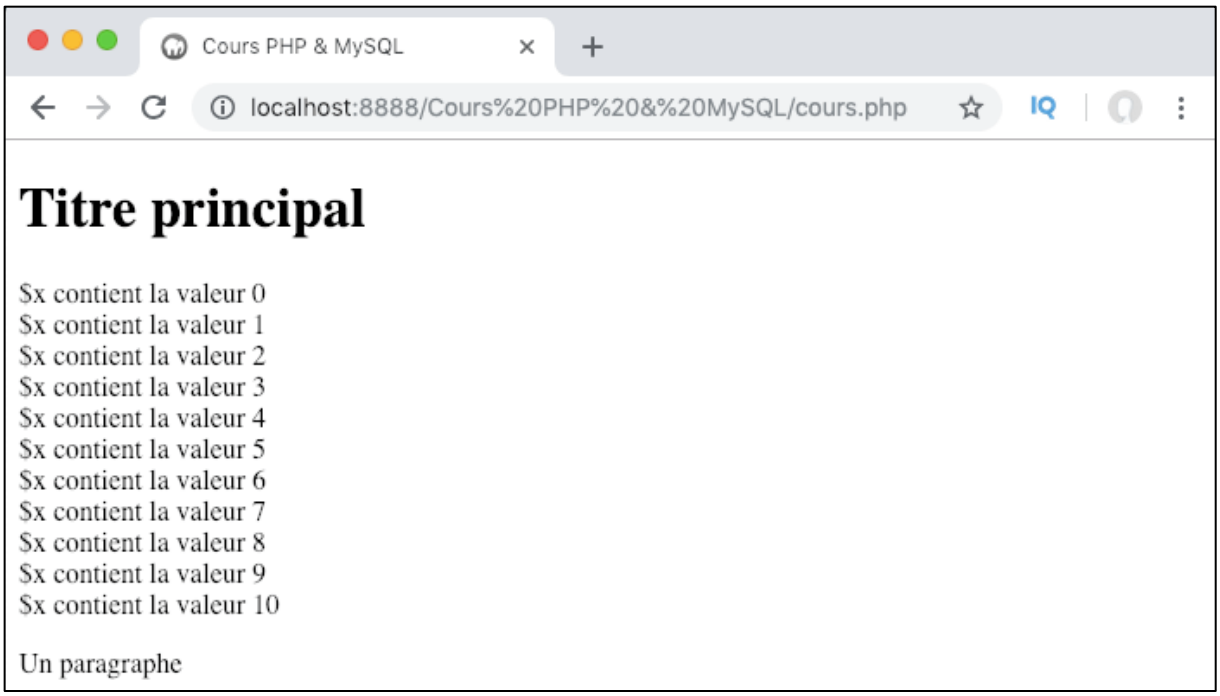
La boucle PHP while

La boucle while (« tant que » en français) est la boucle PHP la plus simple à appréhender.

La boucle while va nous permettre d'exécuter un certain bloc de code « tant qu'une » condition donnée est vérifiée.

Voyons immédiatement la syntaxe de cette boucle que je vais détailler par la suite :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 0;

            while($x <= 10){
                echo '$x contient la valeur ' .$x. '<br>';
                $x++;
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Dans l'exemple ci-dessus, on commence par stocker la valeur 0 dans notre variable $x. On dit également qu'on « initialise » notre variable.

Ensuite, on crée notre boucle while, qui va réutiliser $x puisque nous allons boucler sur les valeurs successives stockées dans $x jusqu'à ce que $x stocke une certaine valeur. Que se passe-t-il exactement dans cette boucle ? Dans le cas présent, on demande à notre boucle d'afficher la phrase $x contient la valeur : suivie de la valeur de $x tant que la valeur stockée dans $x ne dépasse pas 10.

A la fin de chaque passage dans la boucle, on ajoute 1 à la valeur précédente contenue dans $x grâce à l'opérateur d'incrémentation ++. Cette étape d'incrémentation est indispensable. Sans celle-ci, $x contiendrait toujours la valeur 0 (sa valeur de départ) et on ne pourrait jamais sortir de la boucle.

Comment le PHP procède-t-il exactement avec une boucle while ? Ici, le PHP va commencer par vérifier la valeur de $x pour s'assurer que la condition est vérifiée. Si c'est le cas, alors il exécutera une première fois le code dans notre boucle avant de se replacer au départ de notre boucle pour évaluer à nouveau notre condition.

Au début de la boucle, $x contient 0. La condition est donc vérifiée puisque 0 est bien inférieur à 10. PHP exécuté donc le code à l'intérieur de la boucle, c'est-à-dire afficher la phrase $x contient la valeur 0 et ajouter 1 à la valeur de $x.

Le PHP se replace ensuite au début de notre boucle et réévalue notre condition. Lors de ce deuxième passage dans la boucle, notre variable $x contient désormais la valeur 1 (0 + 1). Comme 1 est toujours inférieur à 10, la condition de notre boucle est toujours vérifiée. Le PHP va donc à nouveau exécuter le code dans la boucle et se replacer au début de la boucle pour évaluer une troisième fois notre condition.

Lors de ce deuxième passage, c'est la phrase $x contient la valeur 1 qui est affichée puisque $x contient désormais 1. A la fin de la boucle, on incrémente également à nouveau $x. $x contient donc désormais la valeur 2 (1 + 1).

Le PHP va continuer à effectuer des passages dans la boucle jusqu'à ce que la valeur contenue dans `$x` soit strictement supérieure à 10, c'est-à-dire jusqu'à ce que la condition ne soit plus vérifiée. Dès que ce sera le cas, nous sortirons alors de la boucle.

Notez qu'il faut toujours bien penser à initialiser la variable utilisée dans la boucle en dehors de la boucle. En effet, ici, si j'avais initialisé `$x` dans ma boucle, sa valeur aurait été réinitialisée à chaque nouveau passage de boucle !

Pensez bien également une nouvelle fois qu'il faudra toujours s'arranger pour qu'à un moment ou un autre la condition de la boucle ne soit plus vérifiée, afin de pouvoir en sortir. Dans le cas contraire, le PHP ne pourra jamais sortir de la boucle et le code de celle-ci sera exécuté en boucle jusqu'à ce que le temps d'exécution du script dépasse le temps maximum autorisé par votre serveur et qu'une erreur soit renvoyée.

Évidemment, vous n'êtes pas obligés d'incrémenter la valeur de votre variable. Vous pouvez tout aussi bien décrémenter, multiplier par 2, etc. Le tout est qu'à un certain moment donné la condition ne soit plus vérifiée pour pouvoir sortir de la boucle.

La boucle PHP do…while

La boucle PHP `do... while` (« faire… tant que » en français) ressemble à priori à la boucle `while` mais va fonctionner « en sens inverse » par rapport à `while`.

En effet, la boucle PHP `do... while` va également nous permettre d'exécuter un code tant qu'une condition donnée est vraie, mais cette fois-ci le code dans la condition sera exécuté avant que la condition soit vérifiée.

Ainsi, même si une condition est fausse dès le départ, on effectuera toujours au moins un passage au sein d'une boucle `do...while`, ce qui n'est pas le cas avec une boucle `while`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 0;
            $y = 20;

            do{
                echo '$x contient la valeur ' .$x. '<br>';
                $x++;
            }while($x <= 5);

            do{
                echo '$y contient la valeur ' .$y. '<br>';
                $y++;
            }while($y <= 5);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Faites bien attention : le `while` est bien en dehors des accolades.

Dans l'exemple ci-dessus, nous avons créé deux boucles `do...while` afin de bien illustrer ce que j'ai dit précédemment.

Dans la première boucle, $x contient la valeur 0 au départ. La phrase est affichée et la valeur de $x est incrémentée puis la comparaison est évaluée.

La deuxième boucle est strictement la même que la première à part qu'on utilise cette fois-ci une variable $y.

Notre variable $y contient la valeur 20 au départ et ainsi la condition de la boucle est fausse dès le départ. Cependant, dans le cadre d'une boucle do...while, la condition n'est vérifiée qu'en fin de boucle, c'est-à-dire après le passage dans la boucle.

Ainsi, même si la condition est fausse dès le départ, on effectue tout de même un premier passage dans la boucle (et on exécute donc le code à l'intérieur).

La boucle PHP for

La boucle PHP for (« pour » en français) est plus complexe à appréhender à priori que les boucles précédentes.

Cependant, cette boucle est très puissante et c'est celle qui sera majoritairement utilisée dans nos scripts PHP, faites donc l'effort de comprendre comment elle fonctionne.

Nous pouvons décomposer le fonctionnement d'une boucle for selon trois phases :

- Une phase d'initialisation ;
- Une phase de test ;
- Une phase d'incrémentation.

Dans la phase d'initialisation, nous allons tout simplement initialiser la variable que nous allons utiliser dans la boucle.

Dans la phase de test, nous allons préciser la condition qui doit être vérifiée pour rester dans la boucle.

Dans la phase d'incrémentation, nous allons pouvoir incrémenter ou décrémenter notre variable afin que notre condition soit fausse à un moment donné.

Voyons immédiatement comment cette boucle va fonctionner en prenant un exemple concret :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            for($x = 0; $x <= 5; $x++){
                echo '$x contient la valeur ' .$x. '<br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Comme vous pouvez le voir, on initialise cette fois-ci directement notre variable à l'intérieur de notre boucle. Vous pouvez également noter que chaque phase doit être séparée par un point-virgule.

Le fonctionnement général de la boucle `for` est ensuite similaire à celui des autres boucles. Dans l'exemple ci-dessus, on décrémente cette fois notre variable à chaque nouveau passage dans la boucle pour changer des exemples précédents.

La boucle PHP foreach

La boucle PHP `foreach` est un peu particulière puisqu'elle a été créée pour fonctionner avec des variables tableaux, qui sont des variables spéciales que nous n'avons pas encore étudiées.

Nous étudierons donc le fonctionnement de cette boucle en même temps que le type de valeur PHP `array` (« tableau » en français).

Déterminer la boucle à utiliser selon la situation

A ce stade, vous pourriez vous demander quand utiliser une boucle PHP plutôt qu'un autre. Il va être très simple de déterminer quelle boucle PHP utiliser dans chaque situation. Tout d'abord, si vous travaillez avec des tableaux, vous utiliserez une boucle PHP `for` ou `foreach` selon le type de tableau sur lequel vous travaillez. Nous aurons le temps de détailler cela lorsqu'on étudiera les tableaux.

Ensuite, dans les autres cas, on choisira d'utiliser une boucle `for` lorsqu'on sait à priori combien de passages nous allons effectuer dans notre boucle. En effet, la boucle `for` permet de définir la valeur de base d'une variable ainsi que la condition de sortie de la boucle. C'est donc une convention d'utiliser une boucle `for` lorsqu'on sait combien de passage on va effectuer dans une boucle.

Si vous ne savez pas à priori combien de passages vous allez effectuer dans votre boucle, mais que votre code a besoin d'effectuer toujours au moins un passage dans la boucle, vous utiliserez une boucle PHP `do...while`.

Finalement, si vous ne savez pas combien de passages vous allez effectuer dans votre boucle et que vous ne souhaitez entrer dans la boucle que si la condition est vraie à un moment donné, vous utiliserez une boucle PHP `while`.

Inclure des fichiers dans d'autres en PHP avec include et require

Dans cette nouvelle leçon, nous allons étudier 4 nouvelles structures de contrôle qui vont se révéler très utiles et pratiques lorsqu'on voudra créer un site : `include`, `require`, `include_once` et `require_once`.

Présentation de include et de require et cas d'utilisation

Les instructions PHP `include` et `require` vont nous permettre toutes deux d'inclure des fichiers de code (ou plus exactement le contenu de ces fichiers) à l'intérieur d'autres fichiers de code.

Ces deux instructions sont très puissantes et vont s'avérer extrêmement utiles lors de la création d'un site web.

En effet, imaginions que vous vouliez créer un site personnel ou un site comme le mien. Sur votre site, il y a de fortes chances que l'en-tête, le menu et le pied de page soient identiques pour toutes les pages.

Jusqu'à présent, en HTML, nous étions obligés de réécrire le même code correspondant à l'en-tête, au menu et au pied de page sur chacune des pages de notre site.

Cela, en plus de ne pas être optimisé d'un point de vue performance, nous fait perdre beaucoup de temps et va poser de sérieux problèmes le jour où l'on souhaite modifier le texte dans notre pied de page par exemple pour tout notre site.

Plutôt que de réécrire tout le code relatif à ces différentes parties sur chacune de nos pages, pourquoi ne pas plutôt enregistrer le code de notre menu dans un fichier séparé que l'on appellera par exemple `menu.php`, et faire de même pour le code relatif à notre en-tête et à notre pied de page puis ensuite inclure directement ces fichiers sur chacune de nos pages ?

Nous allons pouvoir faire cela grâce aux instructions `include` et `require`.

Ainsi, nous n'avons plus qu'à écrire le code relatif à notre menu une bonne fois pour toutes et à l'inclure dans chaque page par exemple. Cela représente une considérable simplification pour la maintenance de notre code puisque nous n'avons qu'à modifier le code de notre menu dans le fichier `menu.php` plutôt que de le modifier dans toutes les pages si nous avions copié-collé le code du menu dans chaque page.

Notez déjà que pour inclure un fichier dans un autre fichier, il faudra préciser son emplacement par rapport au fichier qui contient l'instruction `include` ou `require` de la même façon qu'on pourrait le faire pour faire un lien en ou pour inclure une image en HTML.

Les différences entre include, include_once, require et require_once

La seule et unique différence entre les instructions include et require va se situer dans la réponse du PHP dans le cas où le fichier ne peut pas être inclus pour une quelconque raison (fichier introuvable, indisponible, etc.).

Dans ce cas-là, si l'inclusion a été tentée avec include, le PHP renverra un simple avertissement et le reste du script s'exécutera quand même tandis que si la même chose se produit avec require, une erreur fatale sera retournée par PHP et l'exécution du script s'arrêtera immédiatement.

L'instruction require est donc plus « stricte » que include.

La différence entre les instructions include et require et leurs variantes include_once et require_once est qu'on va pouvoir inclure plusieurs fois un même fichier avec include et require tandis qu'en utilisant include_once et require_once cela ne sera pas possible : un même fichier ne pourra être inclus qu'une seule fois dans un autre fichier.

Exemples d'utilisation de include, include_once, require et require_once

Les 4 structures de contrôle include, require, include_once et require_once vont fonctionner de manière similaire.

Pour les faire fonctionner, il va nous falloir un fichier à inclure. On va donc commencer par créer un fichier menu.php qui va contenir un menu de navigation que je souhaite afficher dans les différentes pages de mon site. J'en profite également pour placer les styles CSS de mon menu dans un fichier cours.css.

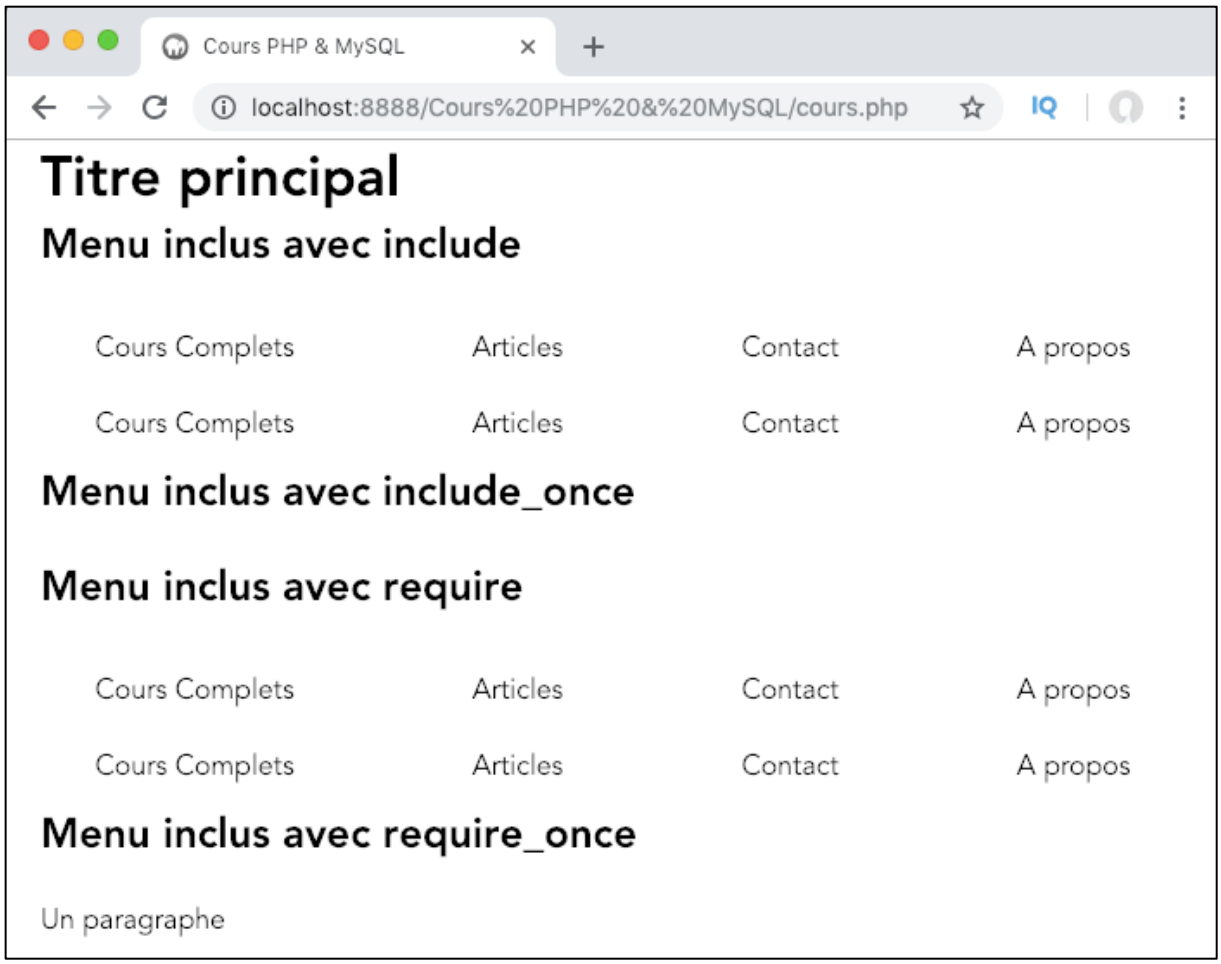
Vous pouvez trouver les codes HTML et CSS du menu.php ci-dessous :

```
<nav>
    <ul>
        <li><a href="#">Cours Complets</a></li>
        <li><a href="#">Articles</a></li>
        <li><a href="#">Contact</a></li>
        <li><a href="#">A propos</a></li>
    </ul>
</nav>
```

```css
*{
    margin: 0px;
    padding: 0px;
    font-family: Avenir, sans-serif;
}
nav{
    background-color: white;
    position: sticky;
    top: 0px;
}
nav ul{
    display: flex;
    flex-flow: nowrap;
    list-style-type: none;
}
nav li{
    flex: 1 1 auto;
    text-align: center;
}
nav a{
    display: block;
    text-decoration: none;
    color: black;
    border-bottom: 2px solid transparent;
    margin: 10px;
}
nav a:hover{
    color: orange;
    border-bottom: 2px solid gold;
}
```

Ensuite, nous allons tenter d'inclure le code de notre fichier de menu dans notre fichier principal qui s'appelle dans mon cas `cours.php` plusieurs fois en utilisant nos différentes instructions.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo '<h2>Menu inclus avec include</h2> <br>';
            include 'menu.php';
            include 'menu.php';

            echo '<h2>Menu inclus avec include_once</h2> <br>';
            include_once 'menu.php';

            echo '<h2>Menu inclus avec require</h2> <br>';
            require 'menu.php';
            require 'menu.php';

            echo '<h2>Menu inclus avec require_once</h2> <br>';
            require_once 'menu.php';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on inclut deux fois notre menu en utilisant `include`, puis on tente de l'inclure à nouveau en utilisant `include_once` et on répète la même opération avec `require` et `require_once`.

Comme vous pouvez le constater, les instructions `include_once` et `require_once` ne vont pas inclure le menu ici car le fichier a déjà été inclus dans la page une première fois (avec notre première instruction `include` en l'occurrence).

N'hésitez pas également à tester la réponse du PHP en passant un nom de fichier qui n'existe pas à `include` puis en réitérant l'opération avec `require`.

PARTIE IV

Les fonctions en PHP

Introduction aux fonctions PHP

Les fonctions sont l'un des outils les plus puissants et les plus pratiques du PHP et vous devez absolument savoir les manipuler.

Dans cette nouvelle leçon, nous allons donc définir ce que sont les fonctions, comprendre comment elles fonctionnent et apprendre à créer nos propres fonctions.

Définition des fonctions et fonctions internes ou prêtes à l'emploi

Une fonction correspond à une série cohérente d'instructions qui ont été créées pour effectuer une tâche précise. Pour exécuter le code contenu dans une fonction, il va falloir appeler la fonction.

Nous avons déjà croisé des fonctions dans ce cours avec notamment la fonction `gettype()` dont le rôle est de renvoyer le type d'une variable.

La fonction `gettype()` fait partie des fonctions dites « internes » au PHP ou « prêtes à l'emploi » tout simplement car sa définition fait partie du langage PHP.

En effet, vous devez bien comprendre ici que la fonction `gettype()` contient une série d'instructions qui permettent de définir le type d'une variable.

Comme cette fonction a été créée et définie par le PHP lui-même, nous n'avons pas à nous soucier des instructions qu'elle contient : nous n'avons qu'à appeler `gettype()` en précisant son nom et le nom d'une variable dont on souhaite connaitre le type dans le couple de parenthèses pour que la série d'instructions qu'elle contient soient exécutées et pour obtenir le type de notre variable.

Les fonctions prêtes à l'emploi sont l'une des plus grandes forces du PHP puisqu'il en existe plus de 1000 qui vont couvrir quasiment tous nos besoins. Nous allons en voir une bonne partie dans la suite de ce cours.

Les fonctions définies par l'utilisateur en PHP

En plus des fonctions internes, le PHP nous laisse la possibilité de définir nos propres fonctions. Cela va s'avérer très pratique dans le cas où nous travaillons sur un projet avec des besoins très spécifiques et pour lequel on va souvent devoir répéter les mêmes opérations.

Pour utiliser nos propres fonctions, nous allons déjà devoir les définir, c'est-à-dire préciser une première fois la série d'instructions que chaque fonction devra exécuter lors de son appel.

Pour déclarer une fonction, il faut déjà commencer par préciser le mot clef `function` qui indique au PHP qu'on va définir une fonction personnalisée.

Ensuite, nous allons devoir préciser le nom de notre fonction (sauf dans le cas des fonctions anonymes que nous étudierons plus tard). Le nom des fonctions va suivre les mêmes règles que celui des variables, à savoir qu'il devra commencer par une lettre ou un underscore et ne devra pas être un nom déjà pris par le langage PHP (comme le nom d'une fonction déjà existante par exemple).

En plus de cela, notez qu'à la différence des variables le nom des fonctions est insensible à la casse. Cela signifie que l'utilisation de majuscules et des minuscules ne servent pas à différencier une fonction d'une autre.

Par exemple, si on crée une fonction `bonjour()`, les notations `BONJOUR()`, `bonJOUR()`, etc. feront référence à la même fonction.

Après le nom de notre fonction, nous devons obligatoirement mentionner un couple de parenthèses puis ensuite placer les instructions que la fonction devra exécuter lors de son appel entre des crochets.

Créons immédiatement une première fonction très simple pour voir comment cela fonctionne en pratique :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function bonjour(){
                echo 'Bonjour à tous';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, nous créons une fonction qu'on appelle `bonjour()` dont le rôle est de renvoyer le message « Bonjour à tous » en utilisant un `echo`. Bien évidemment, cette fonction n'a pas grand intérêt ici mais il faut bien commencer par quelque chose pour un jour créer des choses complexes !

Une fonction ne va pas s'exécuter toute seule ou « automatiquement » lors du chargement de la page. Au contraire, il va falloir l'appeler quelque part dans le code pour déclencher son exécution. C'est un autre point fort des fonctions puisque cela nous permet de n'exécuter certains codes que lorsqu'on le souhaite dans un script.

Pour appeler une fonction à un point dans notre script, il suffit d'écrire son nom suivi d'un couple de parenthèses. Notez qu'on va pouvoir appeler une fonction autant de fois qu'on le veut dans un script et c'est ce qui fait le plus grand intérêt des fonctions.

En effet, plutôt que d'avoir à réécrire les instructions dans une fonction à chaque fois, il suffit de créer une fonction et de l'appeler autant de fois qu'on le souhaite. Cela peut faire gagner beaucoup de temps de développement et rend le code bien plus facilement maintenable puisque nous n'avons qu'à éditer la définition de notre fonction si on souhaite changer quelque chose dans son fonctionnement.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function bonjour(){
                echo 'Bonjour à tous <br>';
            }
            bonjour();
            bonjour();
            bonjour();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Les paramètres et arguments des fonctions PHP

Souvent, les fonctions vont avoir besoin d'informations qui leurs sont externes, c'est-à-dire d'informations qui ne se situent pas dans la définition de la fonction pour fonctionner correctement.

Par exemple, notre fonction `gettype()` va avoir besoin qu'on lui fournisse ou qu'on lui « passe » une variable pour pouvoir déterminer son type.

Les informations qu'on va passer à une fonction sont appelées des arguments. Nous allons toujours passer les arguments dans les parenthèses de notre fonction. Certaines fonctions ne vont pas avoir besoin d'argument pour fonctionner, d'autres vont avoir besoin d'un argument, d'autres encore de deux, etc. Par ailleurs, certains arguments vont parfois être facultatifs tandis que d'autres seront obligatoires.

Quelles différences entre un paramètre et un argument ? On parlera de paramètre lors de la définition d'une fonction. Un paramètre sert à indiquer qu'une fonction va avoir besoin d'un argument pour fonctionner mais ne correspond pas à une valeur effective : ce n'est qu'un prête nom. Un argument en revanche correspond à la valeur effective passée à une fonction.

Vous pouvez retenir ici qu'un paramètre correspond à la valeur générale définie dans la définition d'une fonction tandis qu'un argument correspond à la valeur effective qu'on va passer à la fonction.

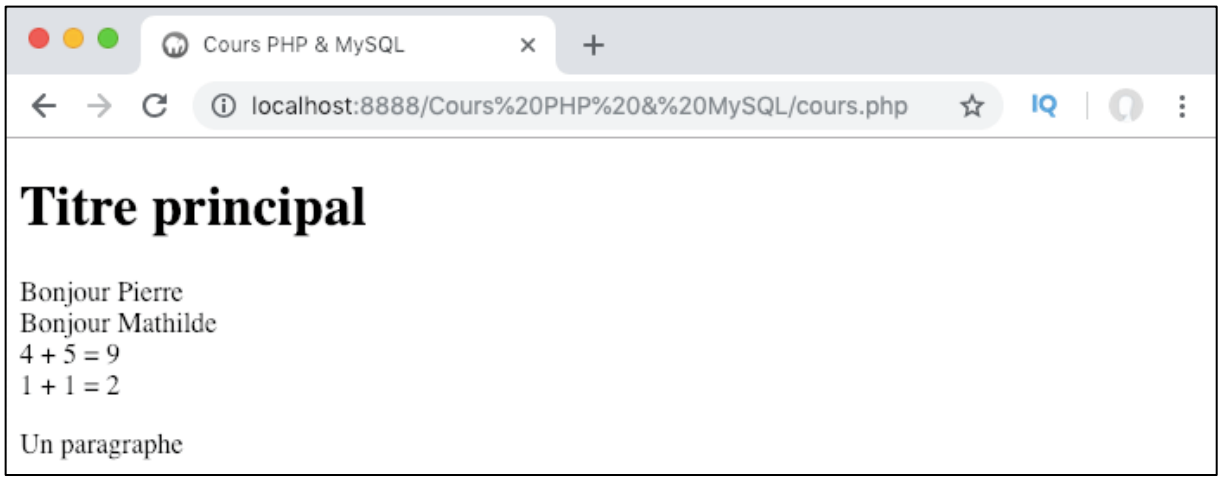
Imaginons par exemple qu'on définisse une fonction dont le rôle est de renvoyer « Bonjour + un prénom ». Cette fonction va avoir besoin qu'on lui passe un prénom pour fonctionner correctement.

Dans sa définition, on va donc indiquer le fait que notre fonction va avoir besoin d'une donnée pour fonctionner en précisant un paramètre entre les parenthèses. On peut fournir n'importe quel nom ici. La chose importante va être de réutiliser le même nom dans le code de notre fonction à l'endroit où la fonction s'en sert.

Ensuite, lorsqu'on appelle notre fonction, nous allons lui passer un prénom effectif, soit en lui passant une variable qui contient un prénom, soit en lui passant directement une chaine de caractères. La valeur effective va alors ici remplacer le paramètre de la définition de la fonction et on parle d'argument.

Regardez plutôt l'exemple suivant :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = 'Pierre';
            $x = 4;
            $y = 5;

            function bonjour($p){
                echo 'Bonjour ' .$p. '<br>';
            }

            function addition($p1, $p2){
                echo $p1. ' + ' .$p2. ' = ' .($p1 + $p2). '<br>';
            }

            bonjour($prenom);
            bonjour('Mathilde');
            addition($x, $y);
            addition(1, 1);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on crée deux fonctions : une première fonction `bonjour()` qui va avoir besoin qu'on lui passe un argument pour fonctionner et une deuxième fonction `addition()` dont le rôle va être d'additionner deux nombres et de renvoyer le résultat et qui va donc avoir besoin qu'on lui fournisse deux nombres pour fonctionner.

Notez qu'on est obligé d'utiliser des parenthèses ici pour calculer la somme de nos deux variables à cause de la priorité des opérateurs et des règles d'associativité.

En effet ici les opérateurs `.` et `+` ont la même priorité et l'associativité se fait par la gauche. Sans parenthèses, ce serait tout le bloc de gauche qui serait additionné à la valeur à droite du `+`. Comme le bloc de gauche est une chaine de caractères, PHP va essayer de la convertir en nombre pour pouvoir l'additionner.

Ici, il faut savoir que PHP procède de la manière suivante pour la conversion : si la chaine de caractères commence par un nombre, alors c'est ce nombre qui sera utilisé. Dans le cas contraire, la valeur 0 sera utilisée.

Un point sur le cours jusqu'ici

Jusqu'à présent, nous avons parlé de variables, d'opérateur, de conditions, de boucles, de fonctions… Si vous êtes nouveau dans la programmation et n'avez jamais étudié d'autres langages qui s'exécutent côté serveur, il est fortement probable que vous soyez un peu perdu à ce niveau du cours.

Je vous rassure : cela est tout à fait normal. J'essaie d'expliquer un maximum et le plus simplement possible chaque nouvelle notion et de donner le plus d'exemples possibles concrets mais cela n'est pas toujours évident car la plupart des outils du PHP nécessitent de connaitre d'autres outils pour être utilisés intelligemment.

L'idée principale à retenir ici est que dans tous les cas personne n'apprend à coder en quelques jours car il ne suffit pas d'apprendre les différentes notions d'un langage mais il faut surtout comprendre comment elles fonctionnent indépendamment et entre elles.

Pas de panique donc : faites l'effort de vous concentrer à chaque leçon, de refaire tous les exemples que je vous propose et vous allez voir qu'au fur et à mesure qu'on avancera dans le cours vous allez comprendre de mieux en mieux les notions précédentes.

Pour cette raison, je vous conseille fortement de faire le cours dans son entièreté au moins deux fois : le deuxième passage vous permettra de voir chaque notion sous un nouveau jour puisque vous aurez déjà abordé les notions connexes lors de votre première lecture.

Par ailleurs, il est possible que vous ne compreniez pas bien l'intérêt des conditions, des boucles ou des fonctions pour le moment car on ne fait que passer des valeurs pour obtenir d'autres valeurs. C'est à nouveau tout à fait normal puisque pour le moment nous définissons nous-mêmes la valeur de nos variables, ce qui enlève tout l'intérêt de la plupart des exercices.

Cependant, vous devez bien comprendre et vous imaginer que nous allons par la suite travailler avec des variables dont nous ne connaissons pas les valeurs à priori (comme par exemple dans le cas où on récupère les données envoyées par nos utilisateurs via un formulaire). C'est à ce moment là où tout ce que nous avons vu jusqu'à présent va devenir vraiment intéressant.

Contrôler le passage des arguments

Dans la leçon précédente, nous avons défini ce qu'était une fonction et vu comment utiliser les fonctions internes au PHP ou comment créer et utiliser nos propres fonctions.

Nous avons également compris la différence entre paramètre et argument et pourquoi certaines fonctions avaient besoin qu'on leur passe des arguments pour fonctionner correctement.

Dans cette leçon, nous allons aller plus loin et apprendre notamment à définir une valeur par défaut pour nos arguments, à passer nos arguments par référence et à faire des déclarations de type.

Passer des arguments par référence

Jusqu'à présent, nous avons passé nos arguments par valeur à nos fonctions ce qui correspond au passage par défaut en PHP.

Lorsqu'on passe une variable comme argument par valeur à une fonction, le fait de modifier la valeur de la variable à l'intérieur de la fonction ne va pas modifier sa valeur à l'extérieur de la fonction.

Imaginons par exemple qu'on crée une fonction donc le rôle est d'ajouter 3 à une valeur passée en argument. Notre fonction va ressembler à cela :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 0;

            function plus3($p){
                $p = $p + 3;
                echo 'Valeur dans la fonction : ' .$p;
            }

            plus3($x);
            echo '<br>Valeur en dehors de la fonction : ' .$x;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on a défini une variable `$x = 0` en dehors de notre fonction puis on a passé `$x` en argument de notre fonction.

Notre fonction va ajouter 3 à la valeur de `$x` et `echo` le résultat.

Cependant, à l'extérieur de la fonction, notre variable `$x` va toujours stocker 0. On peut s'en assurer en affichant la valeur de la variable hors fonction avec `echo`.

Parfois, on voudra cependant que nos fonctions puissent modifier la valeur des variables qu'on leur passe en argument. Pour cela, il va falloir passer ces arguments par référence.

Pour indiquer qu'on souhaite passer un argument par référence à une fonction, il suffit d'ajouter le signe & devant le paramètre en question dans la définition de la liste des paramètres de la fonction.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 0;

            function plus3(&$p){
                $p = $p + 3;
                echo 'Valeur dans la fonction : ' .$p;
            }

            plus3($x);
            echo '<br>Valeur en dehors de la fonction : ' .$x;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Comme vous pouvez le voir, en passant notre argument par référence, la fonction peut modifier la valeur de celui-ci et cette valeur sera conservée dans le reste du script.

Définir des valeurs par défaut pour les paramètres de nos fonctions

Le PHP va également nous permettre de définir une valeur par défaut pour un paramètre d'une fonction. Cette valeur sera utilisée si aucun argument n'est fourni lors de l'appel de la fonction.

Notez ici que la valeur par défaut doit obligatoirement être une valeur constante et ne peut pas être une variable.

Notez également que si on définit une fonction avec plusieurs paramètres et qu'on choisit de donner des valeurs par défaut à seulement certains d'entre eux, alors il faudra placer les paramètres qui ont une valeur par défaut après ceux qui n'en possèdent pas dans la définition de la fonction.

Dans le cas contraire, le PHP renverra une erreur et notre fonction ne pourra pas être exécutée.

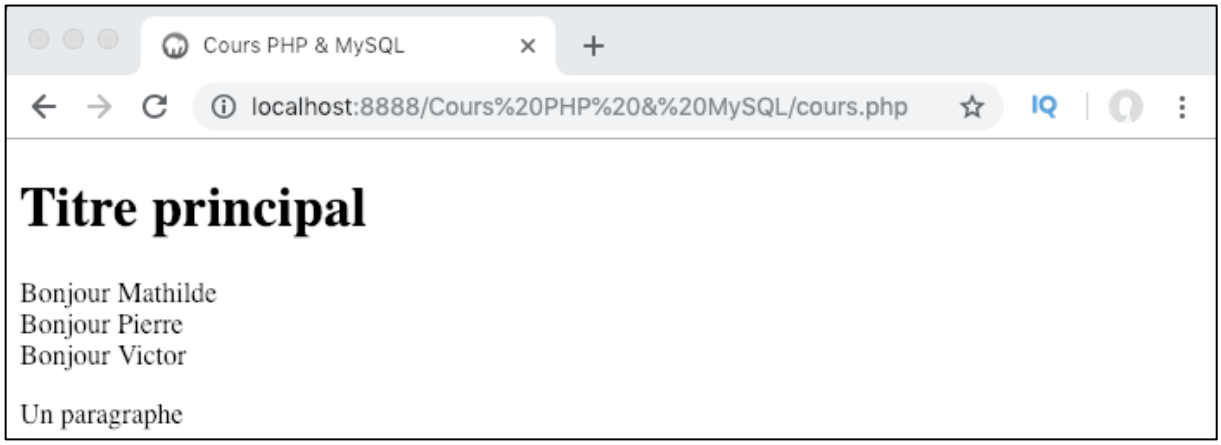
```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function bonjour($prenom, $role='abonné(e)'){
                echo 'Bonjour ' .$prenom. ' vous êtes un(e) ' .$role. '.<br>';
            }

            bonjour('Mathilde');
            bonjour('Pierre', 'administrateur');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Créer une fonction avec un nombre de paramètres variables

Nous allons encore pouvoir définir des fonctions qui vont pouvoir accepter un nombre variable d'arguments en valeurs.

Pour cela, il suffit d'ajouter ... avant la liste des paramètres dans la définition de la fonction pour indiquer que la fonction pourra recevoir un nombre d'arguments variable.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function bonjour(...$prenoms){
                foreach($prenoms as $p){
                    echo 'Bonjour ' .$p. '<br>';
                }
            }

            bonjour('Mathilde', 'Pierre', 'Victor');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Cette syntaxe va créer un tableau avec nos différents arguments. Je dois donc utiliser une boucle foreach pour parcourir mon tableau. Nous étudierons cela plus tard. Pour le moment, contentez-vous de retenir cette syntaxe avec ... avant la définition du paramètre dans notre fonction.

Le PHP, un langage au typage faible

Une autre chose importante que vous devez savoir à propos du langage PHP est que le PHP est un langage qui ne possède pas un typage fort (on parle de « loosely typed language » en anglais).

Cela signifie de manière concrète qu'on n'a pas besoin de spécifier le type de données attendues lorsqu'on définit des paramètres pour une fonction car le PHP va déterminer lui-même le type des données passées à notre fonction en fonction de leur valeur.

Une conséquence de cela est qu'il va être possible de passer des arguments qui n'ont aucun sens à une fonction sans déclencher d'erreur.

On va par exemple pouvoir parfaitement passer deux chaines de caractères à notre fonction `addition()` créée dans la leçon précédente sans déclencher d'erreur.

Le typage des arguments

Depuis sa dernière version majeure (PHP7), le PHP nous offre néanmoins la possibilité de préciser le type de données attendues lorsqu'on définit une fonction. Si une donnée passée ne correspond pas au type attendu, le PHP essaiera de la convertir dans le bon type et s'il n'y arrive pas une erreur sera cette fois-ci renvoyée.

Cela va permettre à nos fonctions de ne s'exécuter que si les valeurs passées en argument sont valides et donc d'obtenir toujours un résultat cohérent par rapport à nos attentes.

Les types valides sont les suivants :

Type	Définition
string	L'argument passé doit être de type string (chaine de caractères)
int	L'argument passé doit être de type integer (nombre entier)
float	L'argument passé doit être de type float ou double (nombre décimal)
bool	L'argument passé doit être de type boolean (booléen)
array	L'argument passé doit être de type array (tableau)
iterable	L'argument passé doit être de type array (tableau) ou une instance de l'interface Traversable
callable	L'argument passé doit être de type callable (fonction de rappel)

Type	Définition
Nom de classe / d'interface	L'argument passé doit être une instance de la classe ou de l'interface donnée
self	L'argument passé doit être une instance de la même classe qui a défini la méthode
object	L'argument passé doit être de type object (objet)

Certains types nous sont encore inconnus, nous les étudierons plus tard dans ce cours. Concentrez-vous pour le moment sur les types connus, à savoir string, int, float et bool. Voici comment on va pouvoir utiliser les déclarations de type concrètement avec nos fonctions :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function test($a, $b){
                echo $a. ' + ' .$b. ' = ' .($a + $b). '<br>';
            }

            function addition(float $a, float $b){
                echo $a. ' + ' .$b. ' = ' .($a + $b). '<br>';
            }

            test(3, 4);
            test(3, '4Pierre');//'4Pierre' est converti en 4 par PHP
            test(3, 'Pierre');//'Pierre' est converti en 0 par PHP
            addition(3, 4);
            addition(3, 4.5);
            addition(3.5, 4.2);
            addition(3, '4Pierre');//'4Pierre' est converti en 4 par PHP
            addition(3, 'Pierre');//Renvoie une erreur qui termine l'exécution

            //Nous étudierons les erreurs plus tard dans le cours
            echo 'Ce message ne s\'affichera pas ';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, l'utilisation de l'opérateur arithmétique `+` fait que PHP va convertir les valeurs à gauche et à droite de l'opérateur en nombres. Si le PHP doit convertir une chaine de caractères, il va regarder si un nombre se situe au début de celle-ci. Si c'est le cas, il conserve le nombre. Sinon, la chaine sera évaluée à 0.

Notre deuxième fonction utilise le typage : on demande que les arguments fournis soient de type `float` (nombres décimaux). Ici, si le PHP n'arrive pas à convertir les arguments passés vers le type attendu, une erreur va être retournée.

Le typage strict

Nous allons pouvoir aller entre plus loin et activer un typage strict pour nos fonctions. En utilisant le mode strict, les fonctions ne vont plus accepter que des arguments dont le type correspond exactement au type demandé dans leur définition.

Notez ici que l'ensemble des nombres entiers fait partie de l'ensemble des nombres décimaux. Ainsi, si on passe un nombre entier en argument d'une fonction qui attend d'après sa définition un nombre décimal, la fonction s'exécutera normalement même avec le mode strict activé.

Pour activer le typage strict, nous allons utiliser la structure de contrôle `declare` qui sert à ajouter des directives d'exécution dans un bloc de code.

Nous allons pouvoir passer trois directives différentes à `declare` :

- La directive `ticks` ;
- La directive `encoding` ;
- La directive `strict_types`.

La directive qui va nous intéresser ici est `strict_types`. Pour activer le mode strict, nous écrirons `declare(strict_types= 1)`. Afin que le typage strict soit activé, il faudra que `declare(strict_types= 1)` soit la première déclaration de notre fichier.

```php
<?php declare(strict_types= 1);?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function addition(float $a, float $b){
                echo $a. ' + ' .$b. ' = ' .($a + $b). '<br>';
            }

            addition(3, 4);
            addition(3, 4.5);
            addition(3.5, 4.2);
            addition(3, '4Pierre');//Ne fonctionne pas car typage strict activé
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Faites bien attention ici : le typage strict ne va s'appliquer que s'il est activé dans le fichier depuis lequel la fonction est appelée. Ainsi, si vous définissez une fonction dans un premier fichier qui possède le typage strict activé puis que vous appelez cette fonction dans un autre fichier qui ne possède pas le typage strict activé, le typage strict ne s'appliquera pas.

Notez également que le typage strict ne s'applique que pour les déclarations de type scalaire, c'est-à-dire pour les types `string`, `int`, `float` et `bool`.

Contrôler les valeurs de retour d'une fonction

Dans cette leçon, nous allons comprendre l'intérêt de la structure de contrôle `return` et apprendre à l'utiliser pour retourner le résultat d'une fonction.

Avantages et spécificités de l'instruction return

Jusqu'à présent, le seul moyen que nous avions d'avoir accès au résultat d'une fonction qu'on avait créée était d'afficher ce résultat en utilisant un `echo` dans la définition de la fonction lors de sa création.

Le problème de cette méthode est que nous n'avons aucun contrôle sur le résultat de la fonction : dès que l'on appelle la fonction en question, le `echo` à l'intérieur est exécuté et le résultat est immédiatement affiché.

La structure de contrôle `return` va nous permettre de demander à une fonction de retourner un résultat qu'on va ensuite pouvoir stocker dans une variable ou autre pour le manipuler.

Cela va être utile dans de nombreux cas puisqu'on voudra souvent récupérer le résultat d'une fonction pour effectuer d'autres calculs ou d'autres opérations avec celui-ci dans un script plutôt que simplement l'afficher.

Attention cependant : l'instruction `return` va terminer l'exécution d'une fonction, ce qui signifie qu'on placera généralement cette instruction en fin de fonction puisque le code suivant une instruction `return` dans une fonction ne sera jamais lu ni exécuté.

Pour illustrer cela, créons deux fonctions dont le rôle est de multiplier deux nombres entre eux et de renvoyer le résultat de la multiplication. La première fonction va utiliser un `echo`, la seconde utilisera plutôt `return`.

```php
<?php declare(strict_types= 1);?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*Note : l'opérateur "*" a une priorité plus grande que "."
             *pas besoin de () ici donc pour faire le calcul*/
            function multecho(float $a, float $b){
                echo $a. ' * ' .$b. ' = ' .$a * $b. '<br>';
            }

            function multreturn(float $a, float $b){
                return $a. ' * ' .$b. ' = ' .$a * $b. '<br>';
            }

            multecho(2, 3);
            multreturn(4, 5);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on utilise un `echo` pour afficher le résultat dans notre fonction `multecho()`. Le `echo` s'exécute lors de l'appel à la fonction et résultat est donc affiché immédiatement après l'appel à notre fonction. Ici, nous n'avons aucun contrôle sur le résultat puisque celui-ci est affiché automatiquement.

En revanche, on utilise `return` pour simplement retourner le résultat de notre fonction `multreturn()`. Ici, le résultat n'est pas affiché, il est simplement retourné par la fonction.

Cela explique le fait qu'aucun résultat ne s'affiche lorsqu'on appelle notre fonction `multreturn()`. Le résultat a bien été calculé et a bien été retourné, mais il attend qu'on en fasse quelque chose.

On va alors pouvoir placer ce résultat dans une variable pour ensuite l'utiliser pour effectuer différents calculs ailleurs dans notre script.

```php
<?php declare(strict_types= 1);?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*Note : l'opérateur "*" a une priorité plus grande que "."
             *pas besoin de () ici donc pour faire le calcul*/
            function multecho(float $a, float $b){
                echo $a. ' * ' .$b. ' = ' .$a * $b. '<br>';
            }

            function multreturn(float $a, float $b){
                return $a * $b;
            }

            multecho(2, 3);

            //On exécute multreturn() et on place le résultat dans $res
            $res = multreturn(4, 5);

            echo $res+= 2;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, notre fonction `multreturn ()` retourne le résultat de la multiplication de deux nombres. On place la valeur retournée dans une variable `$res`. On ajoute 2 à cette variable et on affiche le résultat final. Les opérations ne sont bien évidemment pas très intéressantes ici mais en pratique il va être très utile de pouvoir se servir d'une valeur retournée par une fonction.

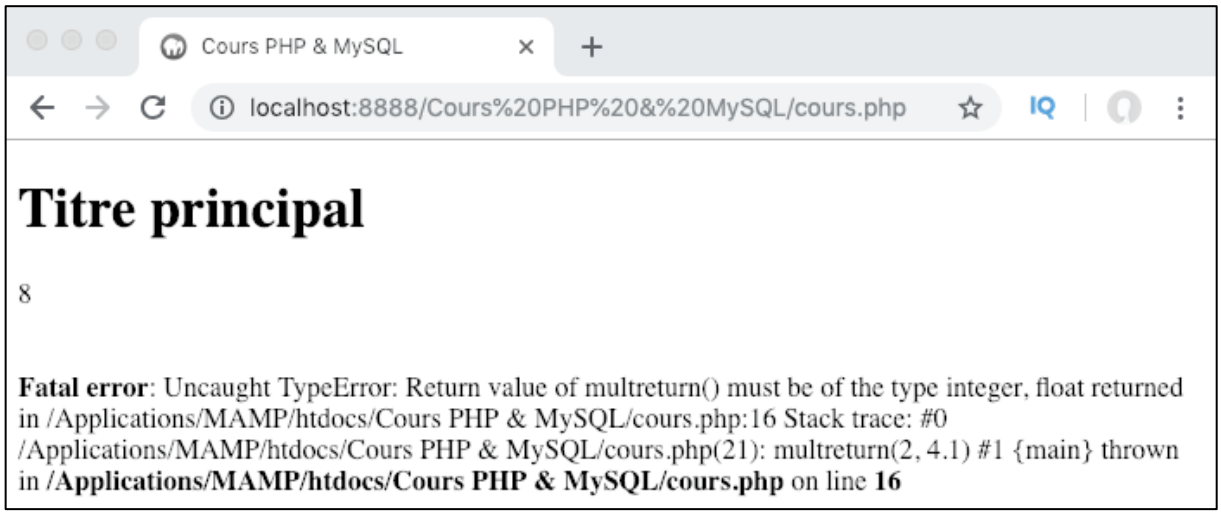
La déclaration des types de valeurs de retour

Depuis PHP7, nous allons de façon similaire à la déclaration de type des arguments pouvoir déclarer les types de retour.

Les déclarations du type de retour vont permettre de spécifier le type de la valeur qui sera retournée par une fonction. On va ici pouvoir utiliser les mêmes types que les types pour les arguments vus dans la leçon précédente.

Notez que le typage strict affecte également les types de retour : si celui-ci est activé, alors la valeur retournée doit être du type attendu. Dans le cas contraire, une exception sera levée par le PHP. Nous étudierons les exceptions bien plus tard dans ce cours. Pour faire simple, vous pouvez pour le moment retenir qu'une exception est une erreur.

```php
<?php declare(strict_types= 1);?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function multreturn($a, $b): int{
                return $a * $b;
            }

            echo multreturn(2, 4);
            echo '<br><br>';
            echo multreturn(2, 4.1);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Dans le cas où le typage strict n'est pas activé (qui est le cas par défaut), si les valeurs retournées ne sont pas du type attendu alors elles seront transtypées, ce qui signifie que le PHP va essayer de transformer leur type pour qu'il corresponde au type attendu.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function multreturn($a, $b): int{
                return $a * $b;
            }

            echo multreturn(2, 4);
            echo '<br><br>';

            /*Sans typage strict, la valeur décimale est arrondie pour pouvoir
             *être trantypée en int (entier)*/
            echo multreturn(2, 4.1);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

La portée des variables

Dans cette nouvelle leçon, nous allons étudier un concept très important concernant les variables et leur utilisation qui est la notion de portée des variables en PHP.

Définition : la portée des variables en PHP

En PHP, nous pouvons déclarer des variables n'importe où dans notre script : au début du script, à l'intérieur de boucles, au sein de nos fonctions, etc.

L'idée principale à retenir ici est que l'endroit dans le script où on déclare une variable va déterminer l'endroit où la variable va être accessible c'est-à-dire utilisable. La « portée » d'une variable désigne justement la partie du script où la variable va être accessible.

Pour faire très simple, vous pouvez considérer que les variables peuvent avoir deux portées différentes : soit une portée globale, soit une portée locale.

Toute variable définie en dehors d'une fonction a une portée globale. Par définition, une variable qui a une portée globale est accessible « globalement », c'est-à-dire dans tout le script sauf dans les espaces locaux d'un script.

Au contraire, toute variable définie à l'intérieur d'une fonction va avoir une portée locale à la fonction. Cela signifie que la variable ne sera accessible qu'au sein de la fonction et notre variable sera par ailleurs par défaut détruite dès la fin de l'exécution de la fonction.

Regardez plutôt les exemples suivants pour bien comprendre la notion de portée des variables :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 10;

            function portee1(){
                echo 'La valeur de $x globale est : ' .$x. '<br>';
            }
            function portee2(){
                $x = 5;
                echo 'La valeur de $x locale est : ' .$x. '<br>';
            }
            function portee3(){
                $y = 0;
                $y++;
                echo '$y contient la valeur : ' .$y. '<br>';
            }
            function portee4(){
                $z = 1;
            }
            portee1();
            portee2();
            portee3();
            portee3();
            portee4();
            echo 'La variable locale $z contient : ' .$z;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on commence par déclarer une variable `$x` en dehors de toute fonction. Notre variable possède donc une portée globale.

Dans notre première fonction `portee1()`, on tente d'afficher le contenu de notre variable `$x` déclarée globalement. Cela ne va pas fonctionner puisqu'une variable globale n'est par défaut pas accessible dans un espace local.

Notre deuxième fonction `portee2()` définit sa propre variable `$x` et a pour but d'afficher son contenu. Ici, vous devez bien comprendre que les deux variables `$x` globale et `$x` locale sont différentes pour le PHP. On le voit bien lorsqu'on affiche ensuite le contenu de notre variable `$x` globale qui n'a pas été modifié par son homologue locale.

Notre troisième fonction `portee3()` définit elle une variable `$y = 0` et son but est d'incrémenter la valeur de notre variable puis de la renvoyer. Si on appelle plusieurs fois `portee3()`, on se rend compte que le résultat est toujours 1. Cela s'explique par le fait que la variable est détruite à la fin de l'exécution de chaque fonction et est donc réinitialisée sur sa valeur `$y = 0` à chaque fois qu'on appelle la fonction.

Finalement, notre quatrième fonction `portee4()` définit une variable `$z`. Lorsqu'on essaie d'afficher le contenu de `$z` depuis l'espace global, aucune valeur n'est renvoyée puisqu'une variable définie localement n'est accessible que dans l'espace dans laquelle elle a été définie par défaut.

Accéder à une variable de portée globale depuis un espace local

Parfois, nous voudrons nous servir de variables possédant une portée globale (c'est-à-dire définies en dehors d'une fonction) à l'intérieur d'une fonction.

Pour cela, on va pouvoir utiliser le mot clef `global` avant la déclaration des variables qu'on souhaite utiliser dans notre fonction. Cela va nous permettre d'indiquer que les variables déclarées dans la fonction sont en fait nos variables globales. Pour être tout à fait précis, on dit que les variables globales sont importées dans le contexte local par référence.

On va ainsi pouvoir utiliser nos variables globales localement. Attention ici : si on modifie la valeur de ces variables dans notre fonction, la valeur des variables globales sera également modifiée puisque ce sont essentiellement les mêmes variables qu'on manipule à l'intérieur de notre fonction.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 10;

            function portee(){
                global $x;
                echo 'La valeur de $x globale est : ' .$x. '<br>';
                $x = $x + 5; //On ajoute 5 à la valeur de $x
            }

            portee();
            echo '$x contient maintenant : ' .$x;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

On va également pouvoir utiliser la variable super globale `$GLOBALS` pour accéder localement à de variables de portée globale. Nous verrons comment faire cela lorsqu'on étudiera les variables super globales.

Accéder à une variable définie localement depuis l'espace global

Il n'y a aucun moyen d'accéder à une variable définie localement depuis l'espace global. Cependant, on va tout de même pouvoir récupérer la valeur finale d'une variable définie localement et la stocker dans une nouvelle variable globale en utilisant l'instruction `return`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function portee(){
                $y = 5;
                echo 'Valeur de $y (depuis la fonction) :' .$y. '<br>';
            }

            function portee2(){
                $z = 5;
                return $z;
            }

            portee();
            echo 'Valeur de $y (depuis l\'espace global) : ' .$y. '<br>';

            $a = portee2(); //On stocke la valeur renvoyée par portee2() dans $a
            echo '$z contient la valeur : ' .$a;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Notez cependant bien ici qu'une variable locale n'aura toujours qu'une portée locale et que sa portée ne pourra pas être étendue dans l'espace global.

Le mot clef static

Une variable définie localement va être supprimée ou détruite dès la fin de l'exécution de la fonction dans laquelle elle a été définie.

Parfois, nous voudrons pouvoir conserver la valeur finale d'une variable locale pour pouvoir s'en resservir lors d'un prochain appel à la fonction. Cela va notamment être le cas pour des fonctions dont le but va être de compter quelque chose.

Pour qu'une fonction de « souvienne » de la dernière valeur d'une variable définie dans la fonction, nous allons pouvoir utiliser le mot clef `static` devant la déclaration initiale de la variable.

La portée de la variable sera toujours statique, mais la variable ne sera pas détruite lors de la fin de l'exécution de la fonction mais plutôt conservée pour pouvoir être réutilisée lors d'une prochaine exécution.

Notez par ailleurs que lorsque nous initialisons une variable en utilisant `static`, la variable ne sera initialisée que lors du premier appel de la fonction (si ce n'était pas le cas, le mot clef `static` n'aurait pas grand intérêt).

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function compteur(){
                static $x = 0;
                echo '$x contient la valeur : ' .$x. '<br>';
                $x++;
            }

            compteur();
            compteur();
            compteur();
            compteur();
            compteur();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, notre fonction commence par initialiser une variable `$x`, affiche sa valeur puis l'incrémente. Lors du premier appel, `$x` contient la valeur 0. Lors du deuxième appel, la fonction va réutiliser la dernière valeur de `$x` qui n'a pas été détruite grâce à l'utilisation du mot clef `static` et va à nouveau afficher la valeur contenue dans notre variable (1 donc) puis l'incrémenter à nouveau. Notez qu'ici la fonction ne réinitialise pas notre variable.

Sans le mot clef `static`, chaque nouvel appel de notre fonction renverrait la valeur 0 puisqu'à la fin de chaque exécution de la fonction les variables utilisées seraient détruites et seraient donc réinitialisées à chaque nouvel appel.

Les constantes et constantes magiques PHP

Dans cette leçon, nous allons définir ce que sont les constantes en PHP, présenter les constantes magiques qui sont des constantes prédéfinies ou internes au PHP et apprendre à créer nos propres constantes.

Définition des constantes PHP

Une constante est un identifiant ou un nom qui représente une valeur simple. Les constantes, tout comme les variables, vont donc être des conteneurs qui vont nous servir à stocker une valeur.

Cependant, à la différence des variables, la valeur stockée dans une constante ne va pas pouvoir être modifiée : elle va être « constante » (sauf dans le cas des constantes magiques dont nous allons reparler plus tard).

Notez également que les constantes vont par défaut être toutes accessibles dans tout le script : on va pouvoir les définir n'importe où dans le script et pouvoir y accéder depuis n'importe quel autre endroit du script.

Notez de plus que par convention, les constantes seront toujours écrites en majuscules pour bien les différencier des autres objets du langage PHP.

Créer ou définir une constante en PHP

Pour définir une constante en PHP, nous allons pouvoir utiliser la fonction `define()` ou le mot clef `const`.

Pour créer une constante en utilisant `define()`, nous allons devoir passer le nom de la constante que l'on souhaite créer et la valeur que doit stocker la constante en arguments de cette fonction. Une fois qu'une constante est définie, elle ne peut jamais être modifiée ni détruite.

Si vous souhaitez créer une constante en utilisant le mot clef `const`, alors vous devez noter que seules des données de type `string`, `integer`, `float`, `boolean` et `array` (dans les dernières versions de PHP) vont pouvoir être stockées dans notre constante.

En dehors de cela, le nom des constantes suit les mêmes règles que n'importe quel nom en PHP, à savoir qu'il doit obligatoirement commencer par une lettre ou éventuellement par un underscore. Notez également que par défaut le nom des constantes est sensible à la casse mais cela ne devrait pas importer puisque vous devriez toujours déclarer vos constantes en majuscules.

De plus, à la différence des variables, nous ne devrons pas préfixer le nom d'une constante avec le signe `$`.

Pour ensuite utiliser la valeur d'une constante ou l'afficher, il suffit d'appeler la constante par son nom comme on en a l'habitude avec les variables.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            define('DIX', 10);

            echo 'La constante DIX stocke la valeur ' .DIX;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Les constantes prédéfinies et les constantes magiques en PHP

Le PHP nous fournit également un certain nombre de constantes prédéfinies qui vont généralement nous donner des informations de type « meta » comme la version de PHP actuellement utilisée, le plus petit entier supporté par PHP, des constantes de rapport d'erreur etc.

Ces constantes prédéfinies ne vont pas toujours être toutes disponibles car certaines sont définies par des extensions PHP. Une extension PHP est une librairie (un ensemble de feuilles de codes) qui permettent d'ajouter des fonctionnalités au PHP de base.

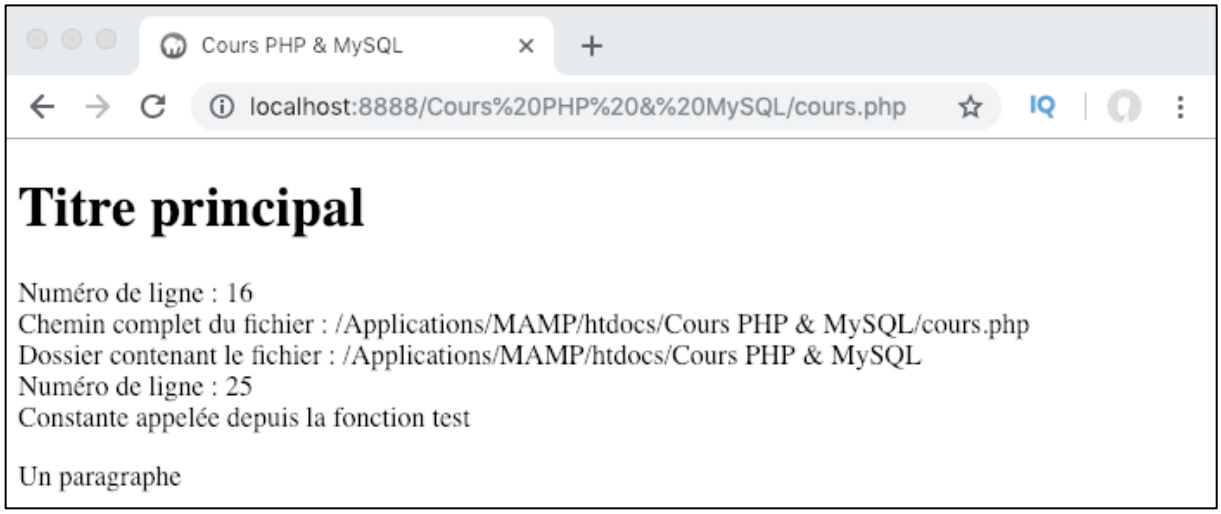
Parmi ces constantes prédéfinies, il y en a neuf qui vont retenir notre attention et qu'on appelle des constantes « magiques ». Ces constantes se distinguent des autres puisque leur valeur va changer en fonction de l'endroit dans le script où elles vont être utilisées.

Les constantes magiques sont reconnaissables par le fait que leur nom commence par deux underscores et se termine de la même manière (excepté pour une). Ce sont les suivantes :

Nom de la constante	Description
__FILE__	Contient le chemin complet et le nom du fichier
__DIR__	Contient le nom du dossier dans lequel est le fichier
__LINE__	Contient le numéro de la ligne courante dans le fichier
__FUNCTION__	Contient le nom de la fonction actuellement définie ou {closure} pour les fonctions anonymes
__CLASS__	Contient le nom de la classe actuellement définie
__METHOD__	Contient le nom de la méthode actuellement utilisée
__NAMESPACE__	Contient le nom de l'espace de noms (« namespace ») courant
__TRAIT__	Contient le nom du trait (incluant le nom de l'espace de noms dans lequel il a été déclaré)
ClassName::class	Contient le nom entièrement qualifié de la classe

Pour le moment, nous allons nous concentrer sur les quatre premières constantes citées, car vous ne disposez pas encore de connaissances suffisantes pour bien comprendre les quatre dernières.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*Affiche le numéro de la ligne où la
             *constante a été appelée (dans le code)*/
            echo 'Numéro de ligne : ' .__LINE__. '<br>';

            //Affiche le chemin du ficher et son nom
            echo 'Chemin complet du fichier : ' .__FILE__. '<br>';

            //Affiche le nom du dossier qui contient le fichier
            echo 'Dossier contenant le fichier : ' .__DIR__. '<br>';

            //Affiche à nouveau la ligne où la constante a été appelée
            echo 'Numéro de ligne : ' .__LINE__. '<br>';

            /*Renvoie le nom de la fonction depuis
             *laquelle la constante est appelée*/
            function test(){
                echo 'Constante appelée depuis la fonction ' .__FUNCTION__;
            }
            /*Ne pas oublier d'exécuter la fonction pour
             *qu'elle echo la valeur de la constante magique !*/
            test();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Les constantes magiques vont notamment être utiles pour récupérer des adresses de fichiers et les utiliser dans des fichiers de configuration ou lors d'appels à nos bases de données SQL ou encore pour effectuer des actions de débogage.

PARTIE V

Variables tableaux PHP

Présentation des tableaux et tableaux numérotés PHP

Dans cette nouvelle partie, nous allons étudier les variables tableau (type `array` en anglais).

Les tableaux vont nous servir à stocker plusieurs valeurs en même temps et vont donc se révéler très utiles pour stocker les résultats successifs d'une boucle par exemple ou n'importe quelle liste de valeurs pour laquelle il serait long de créer une variable pour chaque valeur.

Présentation des tableaux en PHP

Les tableaux en PHP sont des variables spéciales qui peuvent stocker plusieurs valeurs en même temps.

Dans un tableau, chaque valeur va être associée à une clef unique. Cette clef va nous permettre notamment de récupérer la valeur associée. Nous allons pouvoir définir les différentes clefs ou laisser le PHP générer automatiquement les clefs pour les différentes valeurs d'un tableau.

On va pouvoir créer trois types de tableaux différents en PHP :

- Des tableaux numérotés ou indexés (les clefs vont être des nombres) ;
- Des tableaux associatifs (nous allons définir la valeur que l'on souhaite pour chaque clef) ;
- Des tableaux multidimensionnels (tableaux qui stockent d'autres tableaux en valeur).

Nous allons bien évidemment étudier chacun de ces trois types de tableaux dans la suite de ce cours et présenter leurs spécificités.

Pour créer un tableau, on peut soit utiliser la structure de langage `array()`, soit le nouvelle syntaxe plus courte `[]`.

On va pouvoir passer autant d'argument qu'on souhaite stocker de valeurs dans notre tableau à `array()` ou dans `[]`. Les arguments vont pouvoir être soit des valeurs simples (auquel cas les clefs seront des entiers générés automatiquement), soit des paires `clef => valeur`.

Création d'un tableau numéroté ou indexé en PHP

Les tableaux numérotés sont le type de tableaux le plus simple à créer en PHP puisque les clefs vont être générées automatiquement par le PHP.

Pour créer un tableau numéroté en PHP, il suffit en fait d'indiquer une série de valeurs et le PHP associera automatiquement une clef unique à chaque valeur, en commençant avec

la clef 0 pour la première valeur, la clef 1 pour la deuxième valeur, la clef 2 pour la troisième valeur, etc.

Regardez plutôt le code ci-dessous. On crée ici deux tableaux numérotés `$prenoms` et `$ages` en utilisant les deux syntaxes vues précédemment.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenoms = array('Mathilde', 'Pierre', 'Amandine', 'Florian');
            $ages = [27, 29, 21, 29];
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on crée deux variables tableau `$prenoms` et `$ages`. Notez bien qu'on va toujours stocker nos tableaux dans des variables. Notre premier tableau stocke 4 valeurs qui sont ici des chaines de caractères. Notre deuxième tableau stocke 4 chiffres.

On peut tout à fait créer des tableaux qui vont stocker des chaines de caractères, des nombres, des booléens, etc. Comme d'habitude, seules les chaines de caractères doivent être entourées d'apostrophes ou de guillemets droits.

Ici, le PHP va automatiquement créer les clefs qu'on appelle également indices ou index et les associer aux différentes valeurs. La valeur Mathilde de notre premier tableau va avoir la clef 0, la valeur Pierre la clef 1, etc. De même, la clef 0 va être associée à la valeur 27 de notre deuxième tableau, la deuxième valeur (le premier 29) va être associée à la clef 1 et etc.

On va également pouvoir créer un tableau indice par indice et valeur par valeur en précisant ici les indices à associer à chaque valeur. Dans ce cas-là, il faudra utiliser la syntaxe suivante :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenoms[0] = 'Mathilde';
            $prenoms[1] = 'Pierre';
            $prenoms[2] = 'Amandine';
            $prenoms[3] = 'Florian';

            $ages[0] = 27;
            $ages[1] = 29;
            $ages[2] = 21;
            $ages[3] = 29;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Afficher les valeurs d'un tableau numéroté

Pour afficher les valeurs d'un tableau numéroté une à une, il suffit d'`echo` notre variable tableau en précisant l'indice (entre crochets) correspondant à la valeur que l'on souhaite afficher.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenoms = ['Mathilde', 'Pierre', 'Amandine', 'Florian'];

            echo $prenoms[0]. '<br>';
            echo $prenoms[2];
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Pour afficher toutes les valeurs d'un tableau numéroté d'un coup, nous allons cette fois-ci devoir utiliser une boucle `for`.

Nous allons boucler sur les indices de notre tableau et nous allons récupérer la valeur associée à l'indice en question à chaque nouveau passage dans la boucle.

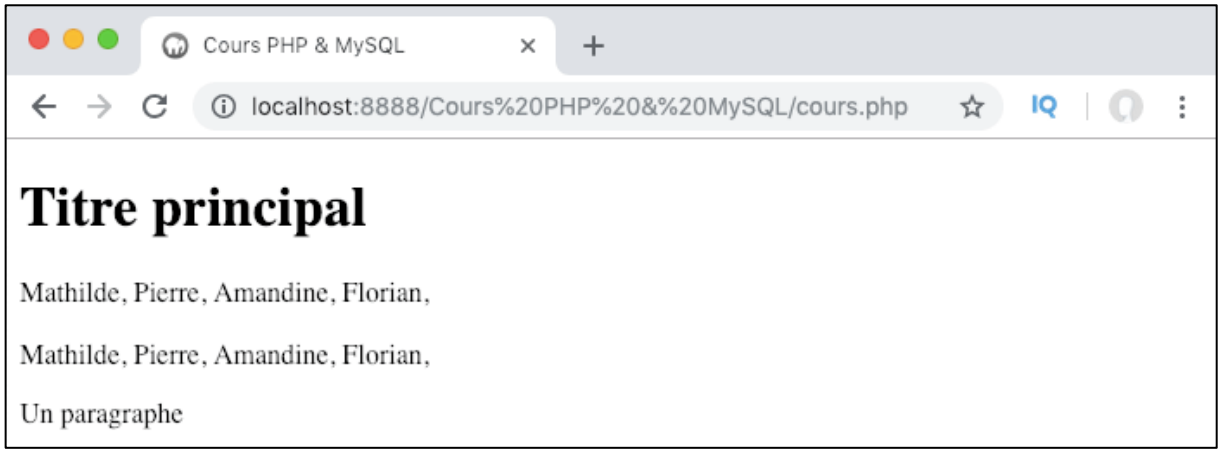
Il va donc avant tout falloir déterminer la taille de notre tableau, c'est à dire le nombre de valeurs contenues dans notre tableau. Bien évidemment, nous travaillons pour le moment avec des tableaux qu'on crée manuellement et on peut donc ici compter le nombre de valeurs à la main mais notez bien ici qu'en pratique on ne saura pas forcément à priori combien de valeurs sont stockées dans un tableau.

La façon la plus simple de déterminer la taille d'un tableau est d'utiliser la fonction `count()` qui va tout simplement prendre un tableau en argument et va retourner le nombre d'éléments de ce tableau.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenoms = ['Mathilde', 'Pierre', 'Amandine', 'Florian'];

            echo 'Le tableau $prenoms contient ' .count($prenoms). ' valeurs';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Une fois qu'on a déterminé la taille de notre tableau, il va être très simple de créer notre boucle for.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenoms = ['Mathilde', 'Pierre', 'Amandine', 'Florian'];

            //On récupère la taille du tableau et on la stocke dans $taille
            $taille = count($prenoms);

            //On peut soit parcourir le tableau et afficher les valeurs une à une
            for($i = 0; $i < $taille; $i++){
                echo $prenoms[$i]. ', ';
            }

            echo '<br><br>';

            //...soit les stocker dans une autre variable et echo cette variable
            for($i = 0; $i < $taille; $i++){
                $p .= $prenoms[$i]. ', ';
            }
            echo $p;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on utilise la fonction `count()` pour déterminer le nombre de valeurs dans notre tableau puis on boucle sur les indices en partant de l'indice 0. Lors du premier passage dans la boucle, on va donc récupérer la valeur associée à l'indice 0 du tableau ; lors du deuxième passage, nous allons récupérer la valeur associée à l'indice 1 et etc.

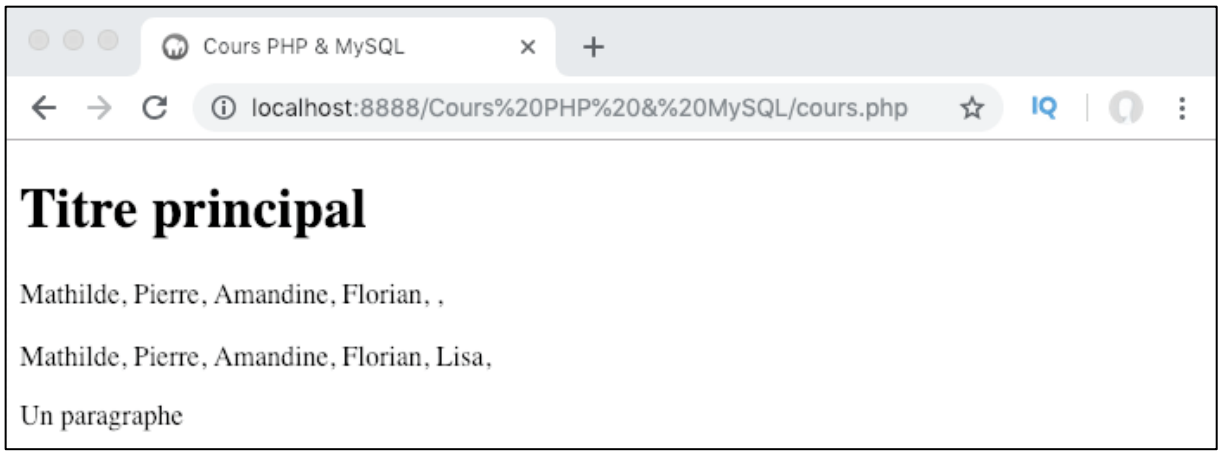
Notre première boucle `for` va `echo` la valeur associée à la clef à chaque passage de boucle tandis que notre deuxième boucle `for` va stocker les valeurs liées à chaque clef dans la variable `$p` grâce à l'opérateur d'affectation `+=`.

Cette technique va très bien fonctionner tant que nos tableaux numérotés vont avoir des indices « naturels ». En effet, il est tout à fait possible d'attribuer les indices manuellement et de sauter certains indices pour stocker nos valeurs dans nos variables tableaux.

Dans ce cas-là, on ne pourra pas récupérer toutes les valeurs en bouclant sur les indices comme ci-dessus mais on utilisera plutôt une boucle `foreach` qui est une boucle spécialement créée pour les tableaux.

Nous allons utiliser cette syntaxe :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenoms = ['Mathilde', 'Pierre', 'Amandine', 'Florian'];

            //On rajoute une valeur à $prenoms et on lui associe la clef 8
            $prenoms[8] = 'Lisa';

            $taille = count($prenoms);

            /*On tente d'afficher les valeurs de notre tableau en utilisant le
             *résultat de count() et en bouclant sur les indices*/
            for($i = 0; $i < $taille; $i++){
                $p .= $prenoms[$i]. ', ';
            }
            echo $p. '<br><br>';

            //On utilise une boucle foreach ($tableau as $valeurs)
            foreach ($prenoms as $valeurs){
                $resultat .= $valeurs. ', ';
            }
            echo $resultat;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, boucler sur les indices avec une boucle `for` ne permet pas de renvoyer toutes les valeurs du tableau tout simplement car notre boucle va s'arrêter avant d'atteindre l'indice 8. En effet, `count()` sert à compter le nombre d'éléments d'un tableau. Ici, notre fonction renvoie donc 5. On va donc ensuite boucler sur les indices de 0 à 4, ce qui ne nous apporte pas le résultat attendu.

La boucle `foreach`, au contraire, va nous permettre de parcourir des tableaux élément par élément. A chaque nouveau passage dans la boucle, la valeur d'un élément du tableau va être placée dans la variable qu'on a ici appelé `$valeurs` et la boucle va aller chercher la valeur suivante jusqu'à arriver à la fin du tableau.

Les tableaux associatifs PHP

Dans cette nouvelle leçon, nous allons voir ce que sont les tableaux associatifs et leurs différences avec les tableaux numérotés. Nous allons également apprendre à créer des tableaux associatifs et à les parcourir et à afficher leurs valeurs.

Présentation des tableaux associatifs en PHP

Un tableau associatif est un tableau qui va utiliser des clefs textuelles qu'on va associer à chaque valeur.

Les tableaux associatifs vont s'avérer intéressant lorsqu'on voudra donner du sens à nos clefs, c'est-à-dire créer une association forte entre les clefs et les valeurs d'un tableau.

Imaginons par exemple qu'on souhaite stocker les âges de nos différents utilisateurs dans un tableau. Ici, plutôt que d'utiliser un tableau numéroté dans lequel il serait difficile de dire à qui appartient chaque âge, il serait judicieux d'utiliser un tableau associatif en utilisant par exemple les pseudonymes de nos membres comme clefs.

Créer un tableau associatif en PHP

Les tableaux associatifs vont être différents des tableaux numérotés au sens où nous allons devoir définir chacune des clefs : le PHP ne va pas ici pouvoir nommer automatiquement nos clefs.

Tout comme pour les tableaux numérotés, on va pouvoir créer notre tableau en une fois en utilisant la structure de langage `array()` ou la syntaxe `[]` ou le construire clef par clef et valeur par valeur.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $ages = ['Mathilde' => 27, 'Pierre' => 29, 'Amandine' => 21];

            /*Identique à
             *$ages = array('Mathilde' => 27, 'Pierre' => 29, 'Amandine' => 21);
             */

            $mails['Mathilde'] = 'math@gmail.com';
            $mails['Pierre'] = 'pierre.giraud@edhec.com';
            $mails['Amandine'] = 'amandine@lp.fr';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

On crée notre premier tableau `$ages` d'un seul coup en utilisant la syntaxe `[]`. Ici, « Mathilde », « Pierre » et « Amandine » sont les clefs ou indices du tableau et 27, 29 et 21 sont les valeurs associées. Notez bien l'utilisation du signe `=>` qui sert à associer une clef à une valeur.

Récupérer et afficher les valeurs d'un tableau associatif

On va pouvoir afficher une valeur en particulier d'un tableau associatif très simplement de la même façon que pour les tableaux numérotés :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $ages = ['Mathilde' => 27, 'Pierre' => 29, 'Amandine' => 21];

            /*Identique à
             *$ages = array('Mathilde' => 27, 'Pierre' => 29, 'Amandine' => 21);
             */

            $mails['Mathilde'] = 'math@gmail.com';
            $mails['Pierre'] = 'pierre.giraud@edhec.com';
            $mails['Amandine'] = 'amandine@lp.fr';

            echo 'Pierre a ' .$ages['Pierre']. ' ans';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Pour parcourir un tableau associatif et par exemple afficher les valeurs les unes après les autres, nous allons en revanche être obligés d'utiliser une boucle `foreach` qui est une boucle créée spécialement pour les tableaux.

Ici, nous allons utiliser la boucle `foreach` de la manière suivante :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $ages = ['Mathilde' => 27, 'Pierre' => 29, 'Amandine' => 21];

            /*Identique à
             *$ages = array('Mathilde' => 27, 'Pierre' => 29, 'Amandine' => 21);
             */

            $mails['Mathilde'] = 'math@gmail.com';
            $mails['Pierre'] = 'pierre.giraud@edhec.com';
            $mails['Amandine'] = 'amandine@lp.fr';

            foreach($ages as $clef => $valeur){
                echo $clef. ' a ' .$valeur. ' ans<br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, nous utilisons une syntaxe de type `foreach($tableau as $clef => $valeur)`. Cette syntaxe nous permet de récupérer à la fois les valeurs du tableau qui vont être stockées dans la variable qu'on a ici appelée `$valeur` ainsi que les clefs associées à chaque valeur.

Lors du premier passage dans la boucle, la première paire clef => valeur du tableau va être récupérée et affichée grâce à `echo` puis `foreach` va nous permettre de passer à la paire suivante clef => valeur du tableau qu'on va afficher lors du deuxième passage dans la boucle et etc. jusqu'à la fin de notre tableau.

Les tableaux multidimensionnels PHP

Dans cette nouvelle leçon, nous allons étudier un nouveau type de tableaux PHP : les tableaux multidimensionnels.

Définition des tableaux multidimensionnels en PHP

Un tableau multidimensionnel est un tableau qui va lui-même contenir d'autres tableaux en valeurs.

On appelle ainsi tableau à deux dimensions un tableau qui contient un ou plusieurs tableaux en valeurs, tableau à trois dimensions un tableau qui contient un ou plusieurs tableaux en valeurs qui contiennent eux-mêmes d'autres tableaux en valeurs et etc.

Les « sous » tableaux vont pouvoir être des tableaux numérotés ou des tableaux associatifs ou un mélange des deux.

Nous ne sommes pas limités dans le nombre de dimensions d'un tableau : le PHP sait tout à fait travailler avec des tableaux à 2, 3, 4, 5… dimensions. Cependant, il est généralement déconseillé de créer trop de dimensions de tableaux tout simplement car cela rend le code très vite très peu lisible et très peu compréhensible pour nous autres développeurs.

Vous pouvez déjà noter que le nombre de dimensions d'un tableau va indiquer le nombre d'indices nécessaires pour accéder à une valeur du tableau (nous allons illustrer cela par la suite).

Créer un tableau multidimensionnel en PHP

Un tableau multidimensionnel est un tableau dont les valeurs peuvent elles-mêmes être des tableaux qui vont à nouveau pouvoir contenir d'autres tableaux et etc.

Commençons déjà par créer des tableaux à deux dimensions. Vous pourrez ensuite créer des tableaux à 3, 4, 5… dimensions en suivant le même modèle.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*Tableau multidimensionnel numéroté stockant
             *des tableaux numérotés*/
            $suite = [
                [1, 2, 4, 8, 16],
                [1, 3, 9, 27, 81]
            ];

            /*Tableau multidimensionnel numéroté stockant
             *des tableaux associatifs et une valeur simple*/
            $utilisateurs = [
                ['nom' => 'Mathilde', 'mail' => 'math@gmail.com'],
                ['nom' => 'Pierre', 'mail' => 'pierre.giraud@edhec.com'],
                ['nom' => 'Amandine', 'mail' => 'amandine@lp.fr'],
                'Florian'
            ];

            /*Tableau multidimensionnel associatif stockant
             *des tableaux associatifs*/
            $produits = [
                'Livre' => ['poids' => 200, 'quantite' => 10, 'prix' => 15],
                'Stickers' => ['poids' => 10, 'quantite' => 100, 'prix' => 1.5]
            ]
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on a créé trois tableaux `$suite`, `$utilisateurs` et `$produits` à deux dimensions.

Notre premier tableau, `$suite`, est un tableau multidimensionnel numéroté qui contient deux valeurs qui vont elles-mêmes être des tableaux numérotés : les valeurs `[1, 2, 4, 8, 16]` et `[1, 3, 9, 27, 81]`.

Notre deuxième tableau est à nouveau un tableau multidimensionnel numéroté qui contient cette fois-ci 4 valeurs : trois tableaux associatifs et la valeur « florian ». En effet, chaque valeur d'un tableau multidimensionnel ne doit pas forcément être elle-même un tableau : il suffit au contraire qu'une valeur d'un tableau soit elle-même un tableau pour que le tableau de départ soit multidimensionnel.

Finalement, notre dernier tableau est un tableau multidimensionnel associatif qui stocke deux valeurs qui sont elles-mêmes des tableaux associatifs. Ici, les clefs de notre tableau multidimensionnel sont `Livre` et `Stickers` et les valeurs associées sont les tableaux `['poids' => 200, 'quantite' => 10, 'prix' => 15]` et `['poids' => 10, 'quantite' => 100, 'prix' => 1.5]`.

Récupérer ou afficher une valeur en particulier d'un tableau multidimensionnel

Pour récupérer une valeur en particulier dans un tableau numéroté ou associatif à une dimension, il suffisait simplement d'indiquer la clef associée à la valeur en question.

Dans le cas d'un tableau multidimensionnel qui ne contient que des tableaux en valeur, nous allons donc accéder aux différents sous tableaux si on ne précise qu'une seule clef. Regardez plutôt l'exemple suivant :

```php
<body>
    <h1>Titre principal</h1>
    <?php
        /*Tableau multidimensionnel numéroté stockant
         *des tableaux numérotés*/
        $suite = [
            [1, 2, 4, 8, 16],
            [1, 3, 9, 27, 81]
        ];

        /*Tableau multidimensionnel numéroté stockant
         *des tableaux associatifs et une valeur simple*/
        $utilisateurs = [
            ['nom' => 'Mathilde', 'mail' => 'math@gmail.com'],
            ['nom' => 'Pierre', 'mail' => 'pierre.giraud@edhec.com'],
            ['nom' => 'Amandine', 'mail' => 'amandine@lp.fr'],
            'Florian'
        ];

        /*Tableau multidimensionnel associatif stockant
         *des tableaux associatifs*/
        $produits = [
            'Livre' => ['poids' => 200, 'quantite' => 10, 'prix' => 15],
            'Stickers' => ['poids' => 10, 'quantite' => 100, 'prix' => 1.5]
        ];

        //$sous_suite = [1, 2, 4, 8, 16]
        $sous_suite = $suite[0];
        echo $sous_suite[0]. '<br>'.$sous_suite[2]. '<br>';

        //$sous_util = ['nom' => 'Amandine', 'mail' => 'amandine@lp.fr']
        $sous_util = $utilisateurs[2];
        echo $sous_util['nom']. '<br>';

        //$sous_produits = ['poids' => 200, 'quantite' => 10, 'prix' => 15]
        $sous_produits = $produits['Livre'];
        echo $sous_produits['prix'];
    ?>
    <p>Un paragraphe</p>
</body>
```

Ici, on procède en deux étapes à chaque fois pour bien comprendre ce qu'il se passe. Notre but est d'afficher certaines valeurs finales de nos tableaux multidimensionnels à deux dimensions.

A chaque fois, on commence par récupérer l'une des valeurs de nos tableaux multidimensionnels qui sont elles-mêmes des tableaux.

On récupère la valeur liée à l'indice 0 de notre tableau `$suite` c'est-à-dire le tableau `[1, 2, 4, 8, 16]` qu'on place dans une variable `$sous_suite` qui devient de fait une variable tableau. On affiche ensuite les valeurs liées aux clefs 0 et 2 de notre tableau sous-suite, c'est-à-dire 1 et 4.

On effectue le même type d'opérations avec nos deux autres tableaux multidimensionnels, en faisant bien attention à préciser les bonnes clefs textuelles lorsque nos tableaux et / ou sous tableaux sont des tableaux associatifs.

La chose à retenir ici est qu'il nous faut donc deux indices pour récupérer l'une des valeurs finales d'un tableau à deux dimensions qui ne contient que des tableaux en valeurs : un premier indice qui va nous permettre d'accéder à une valeur-tableau de notre tableau multidimensionnel et un deuxième indice qui va nous permettre d'accéder à une valeur effective dans notre valeur-tableau.

Nous allons alors pouvoir abréger l'écriture de la façon suivante :

```php
<body>
    <h1>Titre principal</h1>
    <?php
        /*Tableau multidimensionnel numéroté stockant
         *des tableaux numérotés*/
        $suite = [
            [1, 2, 4, 8, 16],
            [1, 3, 9, 27, 81]
        ];

        /*Tableau multidimensionnel numéroté stockant
         *des tableaux associatifs et une valeur simple*/
        $utilisateurs = [
            ['nom' => 'Mathilde', 'mail' => 'math@gmail.com'],
            ['nom' => 'Pierre', 'mail' => 'pierre.giraud@edhec.com'],
            ['nom' => 'Amandine', 'mail' => 'amandine@lp.fr'],
            'Florian'
        ];

        /*Tableau multidimensionnel associatif stockant
         *des tableaux associatifs*/
        $produits = [
            'Livre' => ['poids' => 200, 'quantite' => 10, 'prix' => 15],
            'Stickers' => ['poids' => 10, 'quantite' => 100, 'prix' => 1.5]
        ];

        echo $suite[0][0]. '<br>'.$suite[0][2]. '<br>';
        echo $utilisateurs[2]['nom']. '<br>';
        //Affichage d'une valeur simple contenue directement dans $utilisateurs
        echo $utilisateurs[3]. '<br>';
        echo $produits['Livre']['prix'];
    ?>
    <p>Un paragraphe</p>
</body>
```

Pour accéder aux valeurs finales d'un tableau à 2 dimensions, nous allons devoir préciser deux indices : le premier indice permet d'accéder à une valeur (qui est un tableau) du

tableau multidimensionnel et le deuxième indice sert à accéder à une valeur en particulier dans ce sous tableau.

Nous allons suivre exactement le même schéma pour les tableaux à 3, 4, 5… dimensions en précisant autant de clefs que notre tableau possède de dimensions.

Notez toutefois ici que dans le cas où notre tableau multidimensionnel contient à la fois des tableaux et des valeurs simples, alors on accèdera aux valeurs simples de manière « classique », c'est-à-dire en précisant seulement le nombre d'indices nous permettant d'accéder à la valeur en question. Il est cependant très rare d'avoir des tableaux multidimensionnels composés de valeurs-tableaux et de valeurs simples.

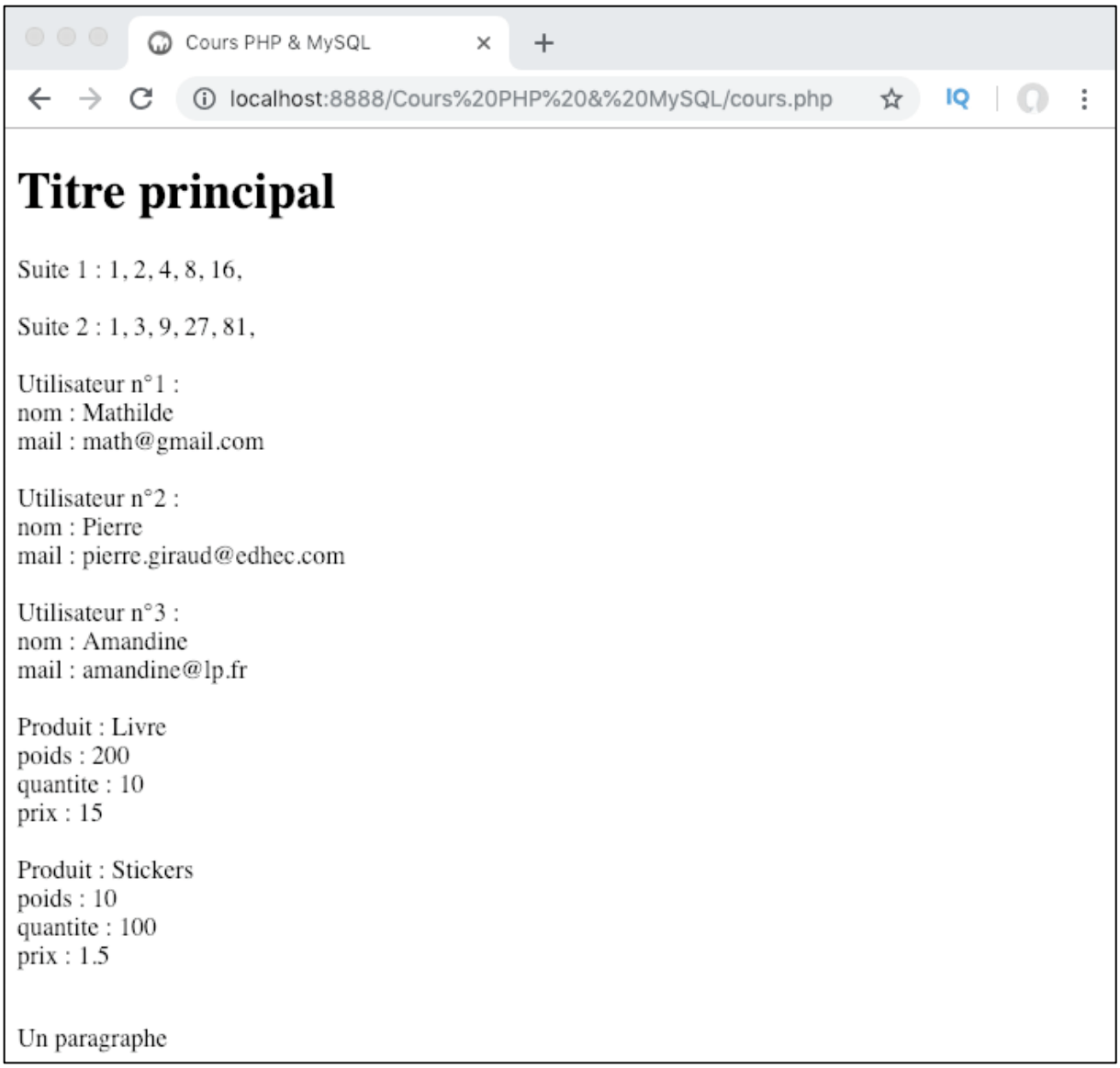
Parcourir et afficher les valeurs d'un tableau multidimensionnel

Pour parcourir toutes les valeurs d'un tableau multidimensionnel (et éventuellement les afficher ou effectuer d'autres opérations dessus), la meilleure manière de faire va être d'utiliser plusieurs boucles `foreach` imbriquées.

On va ici utiliser autant de boucles `foreach` qu'on a de dimensions dans le tableau qu'on souhaite parcourir. La première boucle `foreach` va nous permettre de parcourir les valeurs de notre tableau multidimensionnel de base, puis la deuxième boucle `foreach` va nous permettre de parcourir les valeurs des tableaux contenus directement dans le tableau multidimensionnel et etc.

Essayons d'afficher toutes les valeurs de nos tableaux précédents (note : j'ai enlevé la valeur simple « Florian » de mon tableau utilisateur car elle aurait été complexe à traiter ici).

```php
<body>
    <h1>Titre principal</h1>
    <?php
        $suite = [
            [1, 2, 4, 8, 16],
            [1, 3, 9, 27, 81]
        ];

        $utilisateurs = [
            ['nom' => 'Mathilde', 'mail' => 'math@gmail.com'],
            ['nom' => 'Pierre', 'mail' => 'pierre.giraud@edhec.com'],
            ['nom' => 'Amandine', 'mail' => 'amandine@lp.fr'],
        ];

        $produits = [
            'Livre' => ['poids' => 200, 'quantite' => 10, 'prix' => 15],
            'Stickers' => ['poids' => 10, 'quantite' => 100, 'prix' => 1.5]
        ];
        foreach ($suite as $suitenb => $n){
            echo 'Suite ' .($suitenb + 1). ' : ';
            foreach($n as $ni => $nn){
                echo $nn. ', ';
            }
            echo '<br><br>';
        }
        foreach($utilisateurs as $nb => $infos){
            echo 'Utilisateur n°' .($nb + 1). ' :<br>';
            foreach ($infos as $c => $v){
                echo $c. ' : ' .$v. '<br>';
            }
            echo '<br>';
        }
        foreach ($produits as $clef => $produit){
            echo 'Produit : ' .$clef. '<br>';
            foreach($produit as $caracteristique => $valeur){
                echo $caracteristique. ' : ' .$valeur. '<br>';
            }
            echo '<br>';
        }
    ?>
    <p>Un paragraphe</p>
</body>
```



Ici, nos trois tableaux sont trois tableaux à deux dimensions. Nous allons donc utiliser deux boucles `foreach` à chaque fois. La première boucle `foreach` notre premier tableau `$suite` va nous servir à accéder aux éléments de ce tableau multidimensionnel, c'est-à-dire aux deux clefs numérotées et aux deux valeurs qui sont des tableaux. On `echo` déjà à partir de cette première boucle le numéro de la suite qui va être affichée en rajoutant 1 à la valeur de son index (puisque les index numérotés commencent à 0).

Lors du premier passage dans cette première boucle `foreach`, on va donc accéder à notre première suite et on va `echo` « Suite 1 : » et rentrer dans notre deuxième boucle `foreach`. Cette deuxième boucle `foreach` va parcourir le sous tableau `[1, 2, 4, 8, 16]` et `echo` les différentes valeurs du tableau à chaque fois.

Dès qu'on arrive à la fin de ce premier sous tableau, on retourne dans notre première boucle `foreach` pour un deuxième passage et ce sont cette fois-ci les valeurs de notre deuxième sous tableau qui vont être affichées.

La chose à bien comprendre dans ce code est que notre boucle `foreach` interne ou imbriquée va renvoyer toutes les valeurs d'un sous tableau puis on va ensuite retourner dans notre première boucle pour effectuer un autre passage.

Afficher rapidement la structure d'un tableau en PHP

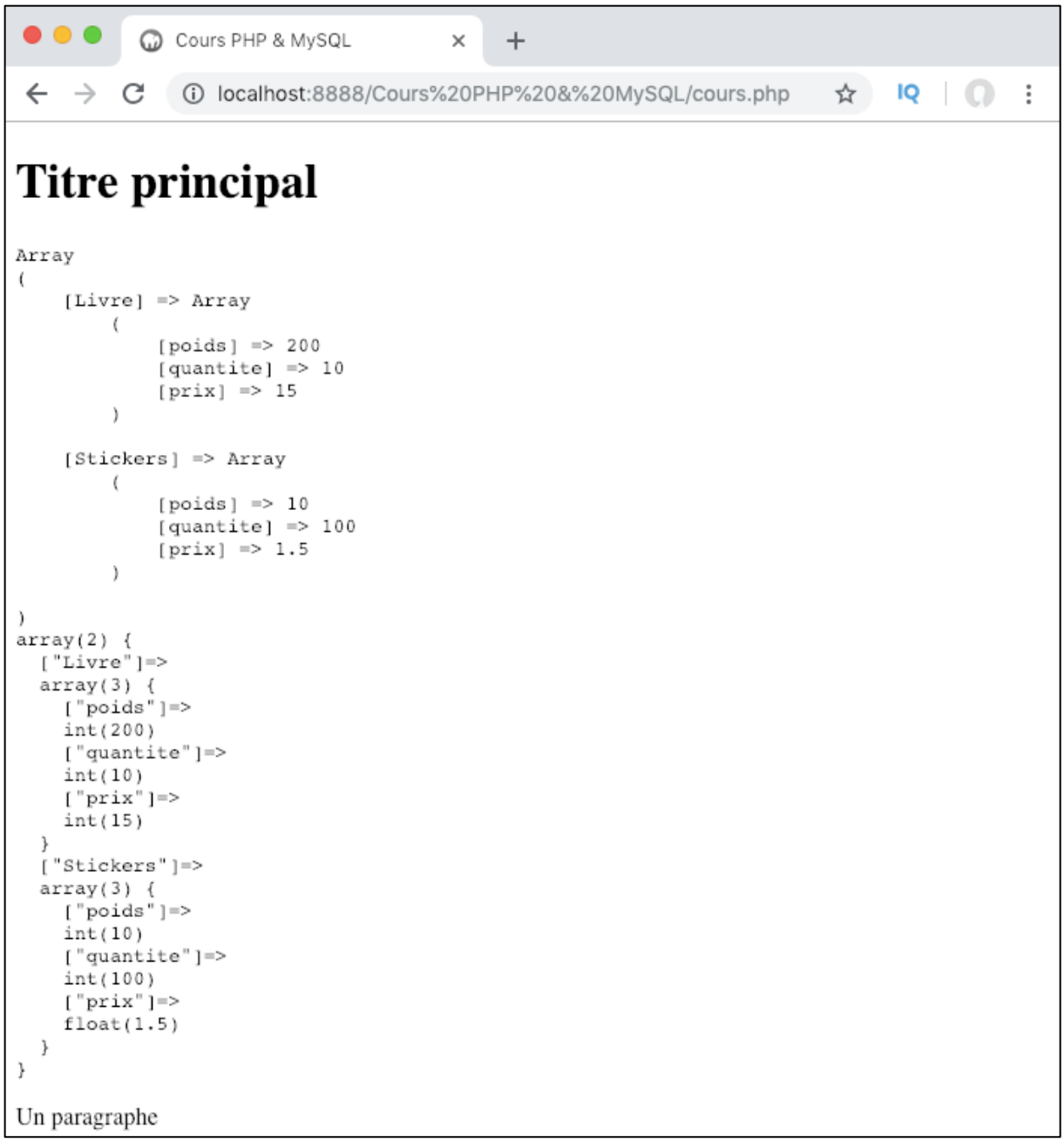
Parfois, on voudra simplement afficher la structure d'un tableau PHP sans mise en forme pour vérifier ce qu'il contient ou pour des questions de débogage.

Le PHP nous fournit deux possibilités de faire cela : on va pouvoir soit utiliser la fonction `print_r()`, soit la fonction `var_dump()` que nous connaissons déjà pour afficher n'importe quel type de tableaux (numérotés, associatifs ou multidimensionnels).

Notez que `var_dump()` va nous fournir davantage d'informations que `print_r()`.

On va généralement utiliser cette fonction avec l'élément HTML `pre` pour avoir un meilleur affichage de la structure du tableau qu'on souhaite afficher (je vous rappelle que `pre` va permettre de conserver la mise en forme de notre code).

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $suite = [
                [1, 2, 4, 8, 16],
                [1, 3, 9, 27, 81]
            ];

            $utilisateurs = [
                ['nom' => 'Mathilde', 'mail' => 'math@gmail.com'],
                ['nom' => 'Pierre', 'mail' => 'pierre.giraud@edhec.com'],
                ['nom' => 'Amandine', 'mail' => 'amandine@lp.fr'],
            ];

            $produits = [
                'Livre' => ['poids' => 200, 'quantite' => 10, 'prix' => 15],
                'Stickers' => ['poids' => 10, 'quantite' => 100, 'prix' => 1.5]
            ];

            echo '<pre>';
            print_r($produits);
            var_dump($produits);
            echo '</pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```


Cours PHP & MySQL
localhost:8888/Cours%20PHP%20&%20MySQL/cours.php
Titre principal
Array
(
[Livre] => Array
(
[poids] => 200
[quantite] => 10
[prix] => 15
)
[Stickers] => Array
(
[poids] => 10
[quantite] => 100
[prix] => 1.5
)
)
array(2) {
["Livre"]=>
array(3) {
["poids"]=>
int(200)
["quantite"]=>
int(10)
["prix"]=>
int(15)
}
["Stickers"]=>
array(3) {
["poids"]=>
int(10)
["quantite"]=>
int(100)
["prix"]=>
float(1.5)
}
}
Un paragraphe

PARTIE VI

Manipuler des dates en PHP

Le Timestamp UNIX et la date en PHP

Dans cette nouvelle partie, nous allons nous intéresser aux dates et aux fonctions PHP qui vont nous permettre de les manipuler.

Avant tout, nous allons définir ce qu'est le Timestamp UNIX et comprendre son intérêt.

Le Timestamp UNIX

Le format Timestamp UNIX est un format de mesure de date très utilisé en programmation. Le Timestamp UNIX représente le nombre de secondes écoulées depuis le 1er janvier 1970 à minuit (heure GMT) et jusqu'à une date donnée.

Ce format de date est difficile à manier pour un humain mais comme le Timestamp UNIX est un nombre on va très facilement pouvoir le manipuler en PHP et l'utiliser pour par exemple comparer deux dates.

Un autre intérêt majeur du Timestamp est qu'il va être le même pour un moment donné quel que soit le fuseau horaire puisque ce nombre représente le nombre de secondes écoulées depuis un point précis dans le temps. Cela va s'avérer très pratique lorsqu'on aura besoin de représenter le temps pour l'ensemble de la planète d'un coup.

Notez finalement que le PHP met bien évidemment à notre disposition certaines fonctions qui vont nous permettre d'exprimer une date sous un format dont on a l'habitude (jour-mois-année par exemple) à partir d'un Timestamp.

Obtenir un Timestamp et le Timestamp actuel en PHP

Obtenir le Timestamp relatif à la date actuelle en PHP

Pour obtenir le Timestamp actuel en PHP, c'est-à-dire le nombre de secondes écoulées entre le 1er janvier 1970 à minuit GMT et le moment actuel, nous allons utiliser la fonction `time()`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo 'Timestamp du 25/01/19 10h00 : ' .time(). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Si vous testez ce code chez vous, il est tout à fait normal que vous n'ayez pas les mêmes valeurs que moi puisque nous n'exécutons pas le code au même moment et donc qu'un nombre différent de secondes se sont écoulées depuis le 1er janvier 1970 pour vous et moi.

Obtenir le Timestamp d'une date donnée en PHP

Pour obtenir le Timestamp UNIX lié à une date donnée, nous allons pouvoir utiliser la fonction `mktime()` qui retourne le Timestamp UNIX d'une date ou la fonction `gmmktime()` qui retourne le Timestamp UNIX d'une date GMT.

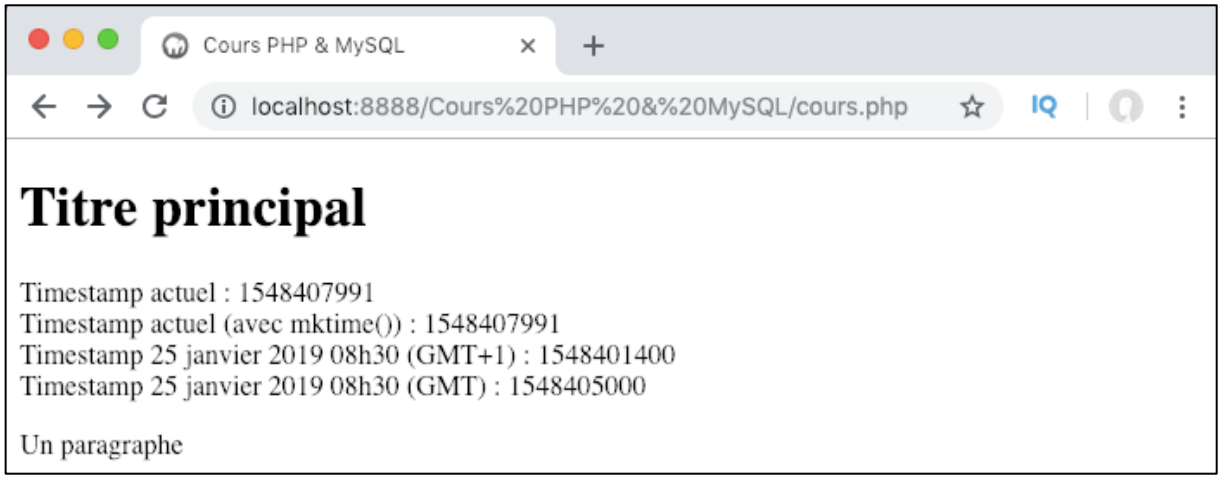
Ces deux fonctions vont s'utiliser de la même façon. Nous allons pouvoir leur passer une série de nombres en arguments. Ces nombres vont constituer une date et vont représenter (dans l'ordre) les parties suivantes de la date dont on souhaite obtenir le Timestamp :

1. L'heure ;
2. Les minutes ;
3. Les secondes ;
4. Le jour ;
5. Le mois ;
6. L'année.

Les arguments sont tous facultatifs et si certains sont omis ce sera la valeur courante ou actuelle de la date qui sera utilisée et donc le Timestamp actuel qui sera renvoyé. Attention cependant : lorsque des arguments sont omis, la fonction considère que ce sont les derniers car elle n'a aucun moyen de savoir quel argument est omis.

Ainsi, il est impossible d'omettre juste le jour par exemple puisque dans ce cas la fonction considérerait que la valeur du mois correspond au jour et celle de l'année au mois. Pour omettre l'argument « jour », il faudra également omettre le mois et l'année.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo 'Timestamp actuel : ' .time(). '<br>';
            echo 'Timestamp actuel (avec mktime()) : '.mktime(). '<br>';
            $t1 = mktime(8, 30, 0, 1, 25, 2019);
            $gmt1 = gmmktime(8, 30, 0, 1, 25, 2019);
            echo 'Timestamp 25 janvier 2019 08h30 (GMT+1) : ' .$t1. '<br>';
            echo 'Timestamp 25 janvier 2019 08h30 (GMT) : ' .$gmt1. '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Lorsqu'on travaille avec les dates, la chose la plus difficile à comprendre est généralement la façon dont vont être gérés les fuseaux horaires.

Ici, la fonction `mktime()` calcule le Timestamp d'une date donnée en fonction du fuseau horaire du serveur. Pour moi, en France et en hiver, je suis à GMT+1. Quand il est 8h30

chez moi, il est donc 7h30 sur un fuseau GMT. La fonction va donc transformer mon heure en heure GMT et se baser sur 7h30 pour calculer le Timestamp.

La fonction gmmktime(), en revanche, considère que l'heure passée est une heure GMT. Cette fonction va donc calculer le Timestamp de l'heure exacte donnée. C'est la raison pour laquelle j'ai un écart de 3600 secondes entre mes deux Timestamps renvoyées par mktime() et gmmktime() : car ces deux Timestamps ne représentent pas la même date.

Obtenir un Timestamp à partir d'une chaine de caractères

Finalement, nous allons également pouvoir obtenir un Timestamp à partir d'une chaine de caractères en utilisant la fonction strtotime() qui transforme une chaine de caractères de format date ou temps en Timestamp.

Pour que cette fonction fonctionne correctement, il va falloir lui passer une chaine de caractères en argument sous un format qu'elle comprend. Notez déjà que cette fonction se base sur le fuseau horaire de votre serveur.

Il y a de nombreuses façons d'écrire une date en PHP. Si vous souhaitez connaitre la liste complète, je vous invite à aller sur la documentation officielle ici : http://php.net/manual/fr/datetime.formats.php.

Les dates littérales devront bien évidemment toujours être écrites en anglais puisque l'anglais est la langue prise comme standard en développement informatique. Les formats de date les plus courants acceptés sont les suivants :

- mm/dd/y (mois/jour/année). Ex : 1/25/19, 1/25/2019 ;
- y-mm-dd (année-mois-jour). Ex : 19-01-25, 2019-01-25 ;
- dd-mm-yy (jour-mois-année). Ex : 25-01-2019 ;
- dd-mois y (jour-mois textuel année). Ex : 25-January 2019 ;
- mois dd, y (mois textuel jour, année). Ex : January 25, 2019 ;

On va ensuite pouvoir ajouter à ces dates un temps qu'on va également pouvoir préciser sous différents formats. Le format de loin le plus utilisé pour les temps est le format heures:minutes:secondes comme par exemple 12:30:15.

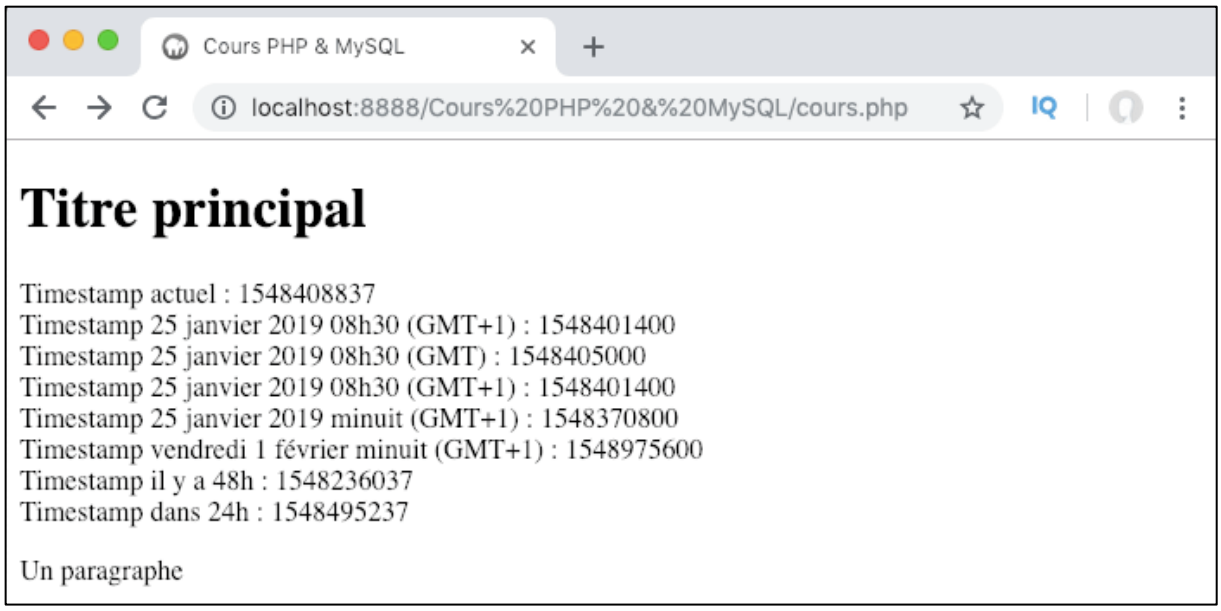
Notez que la fonction strtotime() va également accepter des formats de date relatifs comme :

- Le nom d'un jour (« sunday », « monday », etc.) ;
- Une notation basée sur le jour comme yesterday (hier à minuit), today (aujourd'hui à minuit), now (maintenant), tomorrow (demain à minuit), etc. ;
- Une notation ordinale comme « first fri of January 2019 » (premier vendredi de janvier en 2019) ;
- Une notation avec « ago » comme « 2 days ago » (avant-hier) ou « 3 months 2 days ago » (il y a trois mois et deux jours) ;
- Une notation avec des « + » et des « – » comme « +1 day » (demain) ou « – 3 weeks » (il y a 3 semaines).

Nous allons de plus pouvoir combiner ces formats de dates ensemble en respectant certaines règles pour créer des formats de dates complexes. Dans ce cours, nous allons

cependant nous contenter de manipuler des dates simples car elles vont se révéler suffisantes dans l'immense majorité des cas.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo 'Timestamp actuel : ' .time(). '<br>';
            $t1 = mktime(8, 30, 0, 1, 25, 2019);
            $gmt1 = gmmktime(8, 30, 0, 1, 25, 2019);
            echo 'Timestamp 25 janvier 2019 08h30 (GMT+1) : ' .$t1. '<br>';
            echo 'Timestamp 25 janvier 2019 08h30 (GMT) : ' .$gmt1. '<br>';

            $stt1 = strtotime('1/25/19 08:30:00');
            $stt2 = strtotime('2019-01-25');
            $stt3 = strtotime('next friday');
            $stt4 = strtotime('2 days ago');
            $stt5 = strtotime('+1 day');

            echo 'Timestamp 25 janvier 2019 08h30 (GMT+1) : ' .$stt1. '<br>';
            echo 'Timestamp 25 janvier 2019 minuit (GMT+1) : ' .$stt2. '<br>';
            echo 'Timestamp vendredi 1 février minuit (GMT+1) : ' .$stt3. '<br>';
            echo 'Timestamp il y a 48h : ' .$stt4. '<br>';
            echo 'Timestamp dans 24h : ' .$stt5;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Obtenir une date à partir d'un Timestamp en PHP

Pour obtenir une date en PHP, on va pouvoir utiliser la fonction `getdate()`.
Cette fonction va accepter un Timestamp en argument et retourner un tableau associatif contenant les différentes informations relatives à la date liée au Timestamp.

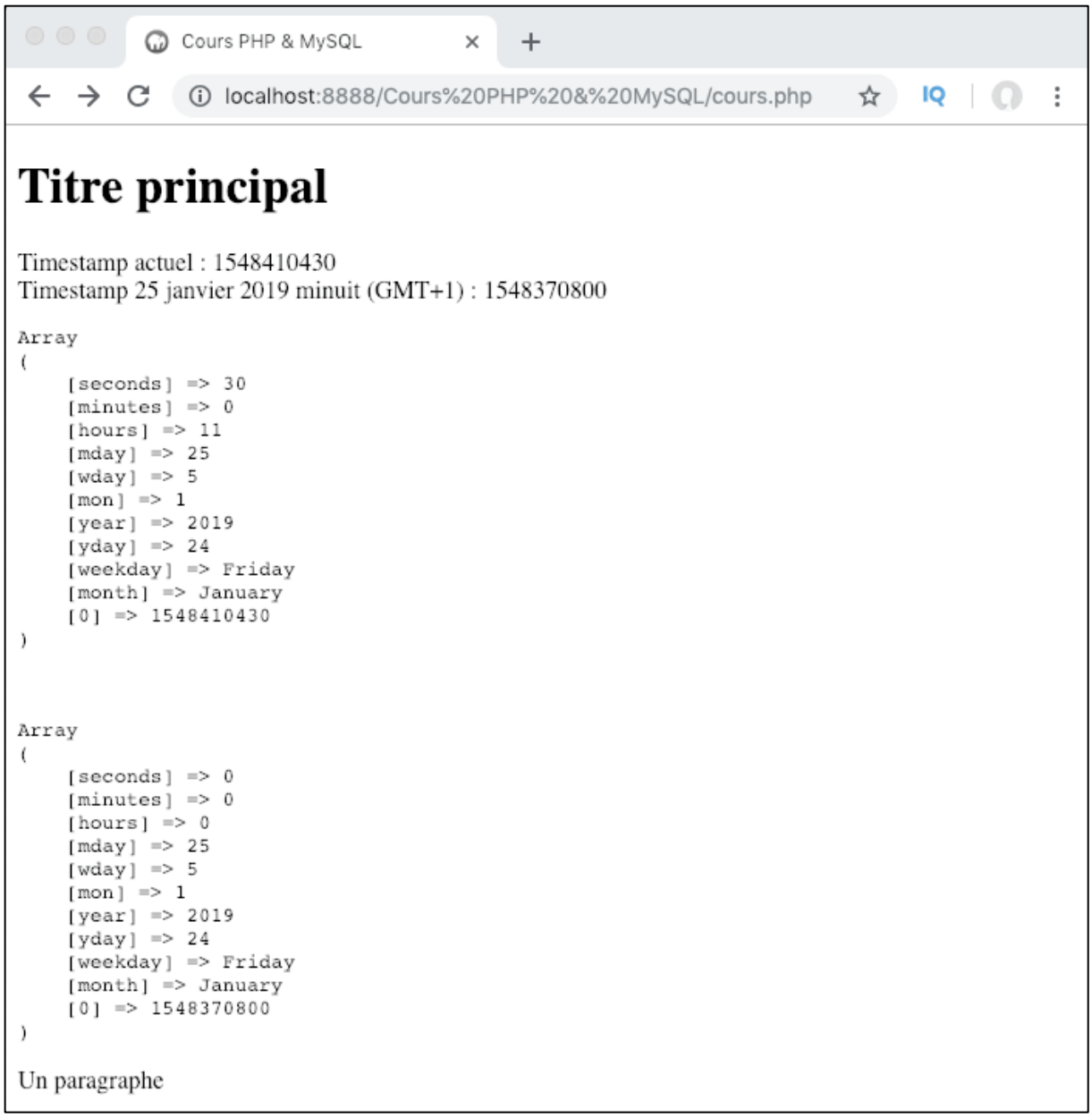
Si aucun argument n'est passé à `getdate()`, alors la fonction utilisera le Timestamp relatif à la date courante et retournera donc la date actuelle locale.

Le tableau associatif renvoyé par `getdate()` est de la forme suivante :

Clef	Valeur associée
seconds	Les secondes
minutes	Les minutes
hours	L'heure
mday	Le numéro du jour dans le mois
wday	Le numéro du jour de la semaine (dimanche = 0, lundi = 1, samedi = 6 pour nous)
mon	Le numéro du mois (de 1 à 12)
year	L'année complète
yday	Le jour de l'année (1er janvier = 0, 2 janvier = 1, etc.)
weekday	Le jour de la semaine sous forme de texte (en anglais)

Clef	Valeur associée
month	Le mois de l'année écrit en toutes lettres (en anglais)
0	Le Timestamp relative à la date renvoyée

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo 'Timestamp actuel : ' .time(). '<br>';

            $stt = strtotime('2019-01-25');
            echo 'Timestamp 25 janvier 2019 minuit (GMT+1) : ' .$stt. '<br>';

            echo '<pre>';
            print_r(getdate());
            echo '</pre><br>';

            echo '<pre>';
            print_r(getdate($stt));
            echo '</pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```


Cours PHP & MySQL
localhost:8888/Cours%20PHP%20&%20MySQL/cours.php
Titre principal
Timestamp actuel : 1548410430
Timestamp 25 janvier 2019 minuit (GMT+1) : 1548370800
Array
(
[seconds] => 30
[minutes] => 0
[hours] => 11
[mday] => 25
[wday] => 5
[mon] => 1
[year] => 2019
[yday] => 24
[weekday] => Friday
[month] => January
[0] => 1548410430
)
Array
(
[seconds] => 0
[minutes] => 0
[hours] => 0
[mday] => 25
[wday] => 5
[mon] => 1
[year] => 2019
[yday] => 24
[weekday] => Friday
[month] => January
[0] => 1548370800
)
Un paragraphe

Obtenir et formater une date en PHP

Dans cette nouvelle leçon, nous allons apprendre à récupérer une date en PHP et à la formater pour la rendre lisible pour nous et nos visiteurs.

La fonction PHP date() et les formats de date

La fonction PHP `date()` va nous permettre d'obtenir une date selon le format de notre choix.

Cette fonction va pouvoir prendre deux arguments. Le premier argument correspond au format de date souhaité et est obligatoire. Le deuxième argument est facultatif et va être un Timestamp relatif à la date qu'on souhaite retourner.

Si le Timestamp est omis, alors la fonction `date()` se basera sur la date courante du serveur.

Pour indiquer le format de date qu'on souhaite voir renvoyer à la fonction `date()`, nous allons lui passer une série de lettres qui vont avoir une signification spéciale.

Les caractères les plus couramment utilisés pour formater une date sont les suivants :

Caractère	Signification
d	Représente le jour du mois en deux chiffres (entre 01 et 31)
j	Représente le jour du mois en chiffres sans le zéro initial (de 1 à 31)
D	Représente le jour de la semaine en 3 lettres (en anglais)
l (L minuscule)	Représente le jour de la semaine en toutes lettres (en anglais)
N	Représente le jour de la semaine en chiffre au format ISO-8601 (lundi = 1, dimanche = 7)
w	Représente le jour de la semaine en chiffre (dimanche = 0, samedi = 6)
z	Représente le jour de l'année de 0 (1er janvier) à 365
W	Représente le numéro de la semaine au format ISO-8601 (les semaines commencent le lundi)
m	Représente le mois de l'année en chiffres avec le zéro initial (de 01 à 12)
n	Représente le mois de l'année de chiffres sans le zéro initial (de 1 à 12)
M	Représente le mois en trois lettres en anglais (Jan, Feb…)

Caractère	Signification
F	Représente le mois en toutes lettres en anglais
t	Représente le nombre de jours contenus dans le mois (de 28 à 31)
Y	Représente l'année sur 4 chiffres (ex : 2019)
y	Représente l'année sur 2 chiffres (ex : 19 pour 2019)
L	Renvoie 1 si l'année est bissextile, 0 sinon
a et A	Ajoute « am » ou « pm » (pour a) ou « AM » ou « PM » (pour A) à la date
h	Représente l'heure au format 12h avec le zéro initial
g	Représente l'heure au format 12h sans zéro initial
H	Représente l'heure au format 24h avec le zéro initial
G	Représente l'heure au format 24h sans le zéro initial
i	Représente les minutes avec le zéro initial
s	Représente les seconds avec le zéro initial
v	Représente les millisecondes avec le zéro initial
O et P	Indique la différence d'heures avec l'heure GMT sans deux points (pour O, ex : +0100) ou avec deux points (pour P, ex : +01:00)
I (i majuscule)	Renvoie 1 si l'heure d'été est activée, 0 sinon
c	Représente la date complète au format ISO 8601 (ex : 2019-01-25T12:00:00+01:00)
r	Représente la date complète au format RFC 2822 (ex : Fri, 25 Jan 2019 12 :00 :00 +0100)
Z	Représente le décalage horaire en secondes par rapport à l'heure GMT

Cela fait beaucoup de signes différents et bien évidemment personne ne vous demande de tous les retenir d'un coup. Contentez-vous de retenir et de comprendre que nous allons pouvoir grâce à ces caractères récupérer une date selon différents formats ce qui va être très pratique pour ensuite pouvoir travailler avec les dates.

Essayons d'utiliser ces caractères avec `date()` afin de renvoyer la date sous différents formats :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo date('d/m/Y'). '<br>';
            echo date('l d m Y h:i:s'). '<br>';
            echo date('c'). '<br>';
            echo date('r'). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Il y a deux choses principales à noter ici : déjà, on va pouvoir séparer nos différents caractères de dates avec des tirets, des points, des slashs ou des espaces pour rendre une date plus lisible.

Ensuite, notez également qu'il faudra faire bien attention à la casse lorsqu'on définit un format d'heure puisque la plupart des caractères ont deux significations totalement différentes selon qu'on les écrit en minuscule ou en majuscule.

Formater une date en PHP : la gestion du décalage horaire

Notez que la fonction `date()` formate une date localement, ce qui signifie que la date renvoyée va être la date locale (avec le décalage horaire). Si on souhaite retourner une

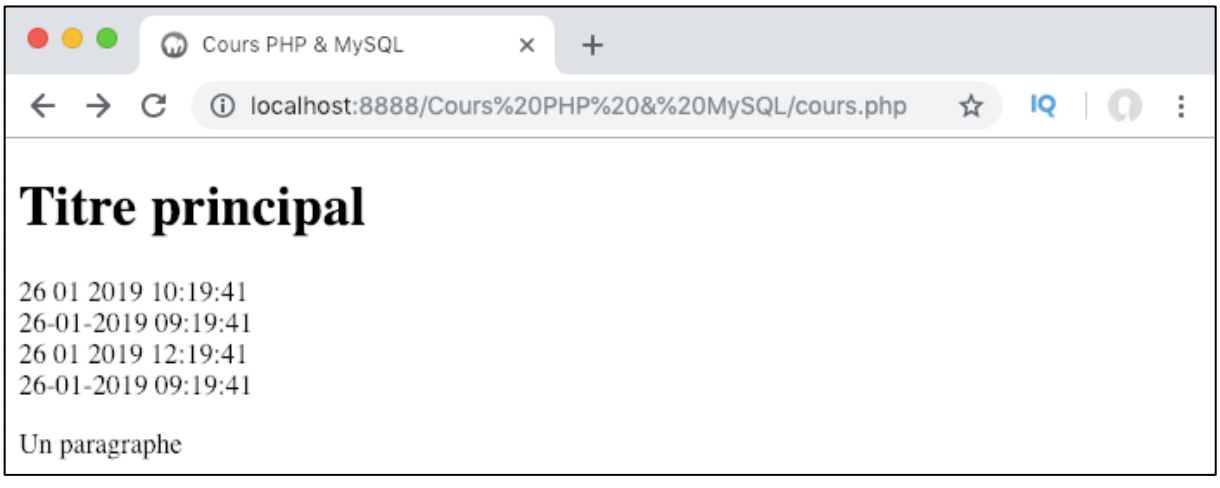
date GMT, alors on utilisera plutôt la fonction gmdate() qui va s'utiliser exactement de la même manière.

Par ailleurs, il est également possible que la date renvoyée ne corresponde pas à celle de l'endroit où vous vous trouvez si le serveur qui héberge votre site se situe sur un autre fuseau horaire ou s'il a été paramétré sur un fuseau horaire différent du vôtre.

Pour régler ce problème, on peut utiliser la fonction date_default_timezone_set() en lui passant un fuseau horaire sous un format valide pour définir le fuseau horaire qui devra être utilisé pour les fonctions relatives à la date utilisées dans le script.

Le fuseau horaire qui va particulièrement nous intéresser va être Europe/Paris. Pour le liste complète des fuseaux horaires valides, je vous invite à lire la documentation officielle : http://php.net/manual/fr/timezones.php.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo date('d m Y h:i:s'). '<br>';
            echo gmdate('d-m-Y h:i:s'). '<br>';
            date_default_timezone_set('Europe/Moscow');//Moscou = GMT+3
            echo date('d m Y h:i:s'). '<br>';
            echo gmdate('d-m-Y h:i:s'). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on modifie notre fuseau horaire de référence pour prendre celui de Moscou au milieu de notre code. Le résultat de la fonction date() appelée ensuite va donc être différent de précédemment.

Rappelez-vous bien ici que le code est lu linéairement dans notre script, c'est-à-dire ligne après ligne. Il faudra donc bien faire attention à l'endroit où on définit les fonctions comme date_default_timezone_set() afin d'avoir les résultats attendus.

Transformer une date en français

Par défaut, les dates vont être renvoyées en anglais par la plupart des serveurs. Pour transformer une date en anglais vers du français, nous avons plusieurs solutions mais une est recommandée par les éditeurs du PHP : l'utilisation des fonctions setlocale() et strftime().

La fonction setlocale() va nous permettre de modifier et de définir de nouvelles informations de localisation. On va déjà pouvoir passer une constante à cette fonction qui va définir les données qui doivent être définies localement : la comparaison de chaines de caractères, la monnaie, les chiffres et le séparateur décimal, les dates ou tout cela à la fois.

Dans notre cas, nous allons nous contenter de modifier les informations de localisation pour le format de date et d'heure et allons pour cela utiliser la constante LC_TIME.

En plus de la constante, il va falloir passer un tableau en deuxième argument de setlocale() qui va nous permettre de choisir la langue souhaitée pour nos informations de localisation.

Notez que la valeur « correcte » du deuxième argument censé déterminer la langue va pouvoir être différente d'un système d'opérations à l'autre car celle-ci n'est pas standardisée. Pour le Français par exemple certains systèmes vont utiliser fr, d'autres fr_FR ou d'autres encore fra.

Par sécurité, on va donc indiquer un maximum de valeurs dans ce tableau : la fonction setlocale() sélectionnera ensuite la première qui est reconnue.

Avec setlocale(), nous avons défini des informations de localisation. Cependant, les fonctions date() et strtotime() par exemple vont ignorer ces informations et continuer de travailler uniquement en anglais.

La seule fonction relative à la date qui supporte la localisation en PHP est la fonction strftime() et c'est donc celle que nous allons utiliser avec setlocale().

Nous allons passer à cette fonction des caractères qui vont représenter des parties de date, de la même façon qu'on avait pu le faire avec la fonction date().

Attention cependant : les caractères ne vont pas forcément être les mêmes et signifier la même chose que pour date() et nous allons cette fois-ci devoir les préfixer avec un %.

Caractère	Signification
%a	Représente le jour de la semaine en trois lettres en anglais
%A	Représente le jour de la semaine en toutes lettres en anglais
%u	Représente le jour de la semaine en chiffre au format ISO-8601 (lundi 1, dimanche = 7)
%w	Représente le jour de la semaine en chiffre (dimanche = 0, samedi = 6)
%d	Représente le jour du mois en deux chiffres (entre 01 et 31)
%j	Représente le jour de l'année avec les zéros de 001 à 366
%U	Représente le numéro de la semaine de l'année en ne comptant que les semaines pleines
%V	Représente le numéro de la semaine de l'année en suivant la norme ISO-8601 (si au moins 4 jours d'une semaine se situent dans l'année alors la semaine compte)
%m	Représente le mois sur deux chiffres de 01 à 12
%b	Représente le nom du mois en lettres en abrégé
%B	Représente le nom complet du mois
%y	Représente l'année sur deux chiffres
%Y	Représente l'année sur 4 chiffres
%H	Représente l'heure, de 00 à 23
%k	Représente l'heure de 0 à 23
%I (i majuscule)	Représente l'heure de 01 à 12
%M	Représente les minutes de 00 à 59
%S	Représente les secondes de 00 à 59
%T	Identique à %H:%M:%S
%D	Identique à %m/%d/%y
%x	Représente la date sans l'heure au format préféré en se basant sur la constant locale

Caractère	Signification
%c	Affiche la date et l'heure basées sur la constant locale

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo strftime('%A %d %B %Y %I:%M:%S'). '<br>';
            setlocale(LC_TIME, ['fr', 'fra', 'fr_FR']);
            echo strftime('%A %d %B %Y %I:%M:%S'). '<br>';
            echo strftime('%x'). '<br>';
            echo strftime('%T'). '<br>';
            echo strftime('%c'). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Comparer des dates et tester la validité d'une date en PHP

Dans cette nouvelle leçon, nous allons comprendre les défis de la comparaison de deux dates et allons voir comment les contourner. Nous apprendrons également à tester la validité d'une date.

La comparaison de dates en PHP

La comparaison de dates est une chose difficile en informatique et particulièrement en PHP.

La raison principale à cela est qu'une date peut être écrite sous de multiples formats : soit tout en chiffres, soit tout en lettres, soit avec un mélange de chiffres et de lettres, avec les mois avant ou après les jours, etc.

Lorsqu'on utilise un opérateur de comparaison, les deux opérandes (ce qui se situe d'un côté et de l'autre de l'opérateur) vont être comparés caractère par caractère. Cela rend impossible la comparaison de dates dont le format n'est pas strictement identique ainsi que la comparaison de dates écrites avec des lettres.

Nous allons donc ici généralement privilégier la comparaison des Timestamp liés aux dates puisque les Timestamp sont des nombres et qu'il est très facile de comparer deux nombres en PHP.

On va donc procéder en deux étapes : on va déjà commencer par récupérer les Timestamp liés aux dates qu'on souhaite comparer avec une fonction comme `strtotime()` par exemple puis on va comparer les Timestamp.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            setlocale(LC_TIME, ['fr', 'fra', 'fr_FR']);

            $d1 = '25-01-2019';
            $d2 = '30-June 2018';

            $tmstp1 = strtotime($d1);
            $tmstp2 = strtotime($d2);

            $dfr1 = strftime('%A %d %B %Y', $tmstp1);
            $dfr2 = strftime('%A %d %B %Y', $tmstp2);

            if($tmstp1 < $tmstp2){
                echo 'Le ' .$dfr1. ' est avant le ' .$dfr2;
            }elseif($tmstp1 == $tmstp2){
                echo 'Les deux dates sont les mêmes';
            }else{
                 echo 'Le ' .$dfr2. ' est avant le ' .$dfr1;
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

La validation de dates : en quoi est-ce utile ?

Jusqu'à présent, nous n'avons travaillé qu'en local, c'est-à-dire sur nos propres machines et pour nous-mêmes. Comme nous créons et testons nos codes nous-mêmes, nous n'avons aucun problème de sécurité / validité.

Cependant, l'objectif reste de pouvoir par la suite créer de véritables sites dynamiques, avec des formulaires, des espaces clients, etc. Dans ces cas-là, bien souvent, vous allez demander aux utilisateurs de vous envoyer des données.

Cela va être par exemple le cas si vous créez un formulaire d'inscription sur votre site, ou si vous possédez un site marchand. Vous demanderez alors certainement les nom, prénom, adresse mail, numéro de téléphone, date de naissance, etc. des utilisateurs.

Il va alors falloir être très précautionneux car vous ne devez jamais vous attendre à recevoir des données valides de la part de vos utilisateurs.

En effet, certains vous faire des fautes, d'autres seront étourdis, d'autres encore vont refuser de remplir certaines données et un petit pourcentage va même potentiellement être composé d'utilisateurs mal intentionnés qui essaieront d'exploiter des possibles failles dans votre site, en tenant de vous envoyer (ou "injecter") des bouts de code par exemple. Pour ces raisons, nous testerons et traiterons toujours les informations envoyées par nos utilisateurs, et notamment les dates.

Tester la validité d'une date

La manière la plus robuste de vérifier la validité d'une date en PHP est d'utiliser la fonction `checkdate()` qui a été créée dans ce but précis.

Cette fonction va accepter trois chiffres en arguments : le premier représente le mois de la date à tester, le deuxième représente le jour et le troisième représente l'année.

Pour que la date soit considérée valide, elle devra remplir les critères suivants :

- Le mois doit être compris entre 1 et 12 ;
- Le jour doit être un jour autorisé par le mois donné (le 30 février, le 31 avril ou le 45 juin seront considérés invalides par exemple). Notez que les années bissextiles sont prises en compte ;
- L'année doit être comprise entre l'an 1 et l'an 32767.

Si `checkdate()` valide la date passée, alors elle renverra le booléen `true`. Dans le cas contraire, elle renverra `false`.

Notez bien ici que cette fonction va nous permettre d'écarter les valeurs aberrantes de dates, c'est-à-dire les dates qui n'existent pas dans le calendrier mais ne va bien évidemment pas nous permettre de vérifier la cohérence d'une date.

Si vous avez un formulaire demandant la date de naissance des utilisateurs sur votre site, par exemple, ce sera à vous d'ajouter des contraintes sur la date de naissance pour écarter par exemple les dates de naissance fantaisistes comme les dates dans le futur ou les dates avant 1900.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            setlocale(LC_TIME, ['fr', 'fra', 'fr_FR']);

            if(checkdate(1,25,2019)){
                echo 'Le 25 janvier 2019 est une date valide <br>';
            }
            if(checkdate(1,35,2019)){
                echo 'Le 35 janvier 2019 est une date valide <br>';
            }
            if(checkdate(2,29,2015)){
                echo 'Le 29 février 2015 est une date valide <br>';
            }
            if(checkdate(2,29,2016)){
                echo 'Le 29 février 2016 est une date valide <br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on teste différentes dates avec `checkdate()`. Si la date passée à `checkdate()` est validée par la fonction, alors la fonction renvoie la date et le test de notre condition va être validé et on va donc `echo` la date formatée localement grâce à `setlocale()`.

Tester la validité d'un format de date locale

On va également pouvoir vérifier si un format de date locale nous convient, c'est-à-dire vérifier la validité d'une date générée par la fonction `strftime()` en utilisant cette fois-ci la fonction `strptime()`.

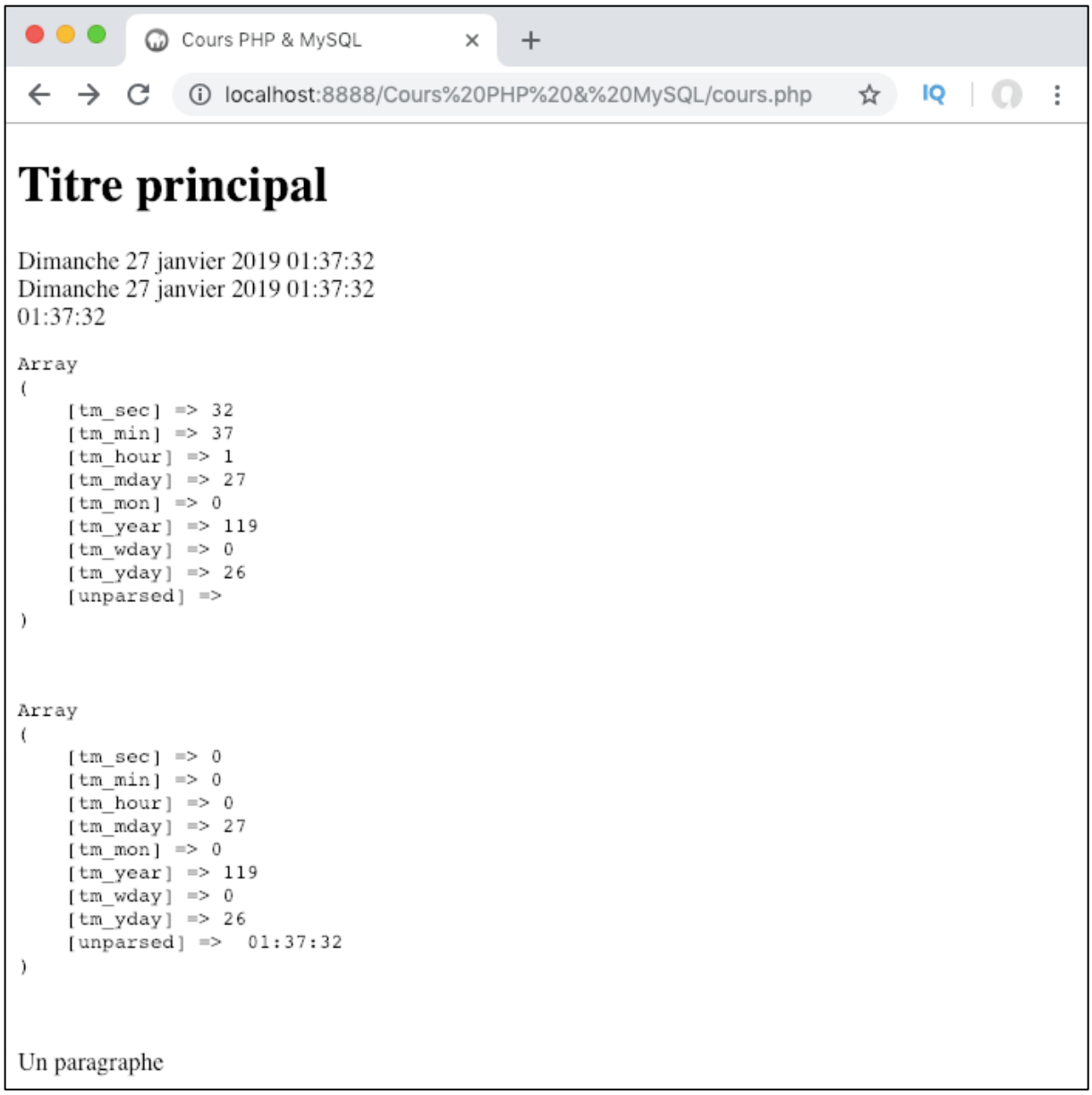
On va passer la chaine de type date renvoyée par `strftime()` à la fonction `strptime()` ainsi que le format de date utilisé par `strftime()`.

La fonction `strptime()` va alors soit retourner un tableau contenant les différentes informations liées à la date si celle-ci a été jugée valide, soit la valeur `false` en cas d'erreur sur la date.

Si la date est validée, le tableau renvoyé sera de la forme suivante :

Type de données	**Signification**
tm_sec	Représente les secondes
tm_min	Représente les minutes
tm_hour	Représente l'heure de 0 à 23
tm_mday	Le jour du mois en chiffres sans le zéro initial
tm_mon	Représente le mois sous forme de chiffres (janvier = 0, décembre = 11)
tm_year	Le nombre d'années écoulées depuis 1900
tm_wday	Le numéro du jour de la semaine (Dimanche = 0, Samedi= 6)
tm_yday	Le jour de l'année en chiffres (1er janvier = 0)
unparsed	La partie de la date qui n'a pas été reconnue

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            setlocale(LC_TIME, ['fr', 'fra', 'fr_FR']);

            $format1 = '%A %d %B %Y %H:%M:%S';
            $format2 = '%H:%M:%S';

            $date1 = strftime($format1);
            $date2 = strftime($format1);
            $date3 = strftime($format2);

            echo $date1. '<br>' .$date2. '<br>' .$date3. '<br>';

            if(strptime($date1, $format1)){
                echo '<pre>';
                print_r(strptime($date1, $format1));
                echo '</pre><br>';
            }
            if(strptime($date2, '%A %d %B %Y')){
                echo '<pre>';
                print_r(strptime($date2, '%A %d %B %Y'));
                echo '</pre><br>';
            }
            if(strptime($date3, '%A %d %B %Y')){
                echo '<pre>';
                print_r(strptime($date3, '%A %d %B %Y'));
                echo '</pre>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on commence par définir une constante de date locale avec `setlocale()` puis on définit les formats de date `%A %d %B %Y %H:%M:%S` et `%H:%M:%S` qu'on va passer à `strftime()`.

On demande ensuite à `strftime()` de nous renvoyer trois dates en utilisant nos différents formats et on place les résultats dans trois variables `$date1`, `$date2` et `$date3` pour bien voir avec quelles dates on travaille.

On crée ensuite trois conditions. Dans notre première condition, on appelle `strptime()` en lui passant notre variable `$date1` et le même format que celui utilisé avec `strftime()`.

La date contenue dans `$date1` est la date courante et les formats utilisés sont les mêmes pour `strftime()` et `strptime()`. Toutes les informations de date vont bien être affichées.

Dans notre deuxième condition, en revanche, on passe un format de date différent à `strptime()`. Ici, seuls les éléments communs aux formats passés à `strftime()` et

à strptime() seront renvoyés dans les « bons » éléments du tableau créé par strptime(). Le reste de la date sera associé à la clef unparsed car non reconnu.

Finalement, on utilise un format totalement différent pour tester notre troisième date. Ici, une erreur va être renvoyée et donc strptime() va renvoyer false et le test de notre condition va échouer.

PARTIE VII

Variables superglobales

Présentation des variables PHP superglobales

Dans cette nouvelle partie, nous allons étudier des variables très spéciales et très importantes en PHP : les variables superglobales.

Nous allons en particulier nous attarder sur certaines d'entre elles qu'il est indispensable de connaitre et de savoir utiliser.

Présentation des variables superglobales

Les variables superglobales sont des variables internes au PHP, ce qui signifie que ce sont des variables créées automatiquement par le PHP.

Ces variables vont être accessibles n'importe où dans le script et quel que soit le contexte, qu'il soit local ou global. C'est d'ailleurs la raison pour laquelle on appelle ces variables « superglobales ».

Il existe 9 superglobales en PHP. Ces variables vont toutes être des variables tableaux qui vont contenir des groupes de variables très différentes. La plupart des scripts PHP vont utiliser les variables superglobales car ces dernières vont s'avérer très souvent très utiles. Il est donc indispensable de bien les connaitre et de savoir les manipuler.

Les variables superglobales PHP sont les suivantes :

- `$GLOBALS` ;
- `$_SERVER` ;
- `$_REQUEST` ;
- `$_GET` ;
- `$_POST` ;
- `$_FILES` ;
- `$_ENV` ;
- `$_COOKIE` ;
- `$_SESSION`.

On écrira toujours les superglobales en majuscules. Cela est une convention qui permet de les différencier des variables « classiques » que nous créons nous-mêmes. Par ailleurs, vous pouvez remarquer que toutes les superglobales à l'exception de `$GLOBALS` commencent par un underscore `_`.

Dans la suite de cette leçon, nous allons présenter brièvement chacune de ces variables superglobales. Nous allons étudier la plupart d'entre elles plus en détail dans les leçons ou les parties suivantes.

La variable superglobale $GLOBALS

La variable superglobale `$GLOBALS` est un peu différente des autres superglobales puisque c'est la première superglobale qui a été créée en PHP, bien avant les autres (en réalité, `$GLOBALS` a toujours été disponible en PHP).

Cette superglobale va nous permettre d'accéder à des variables définies dans l'espace global depuis n'importe où dans le script et notamment depuis un espace local (dans une fonction).

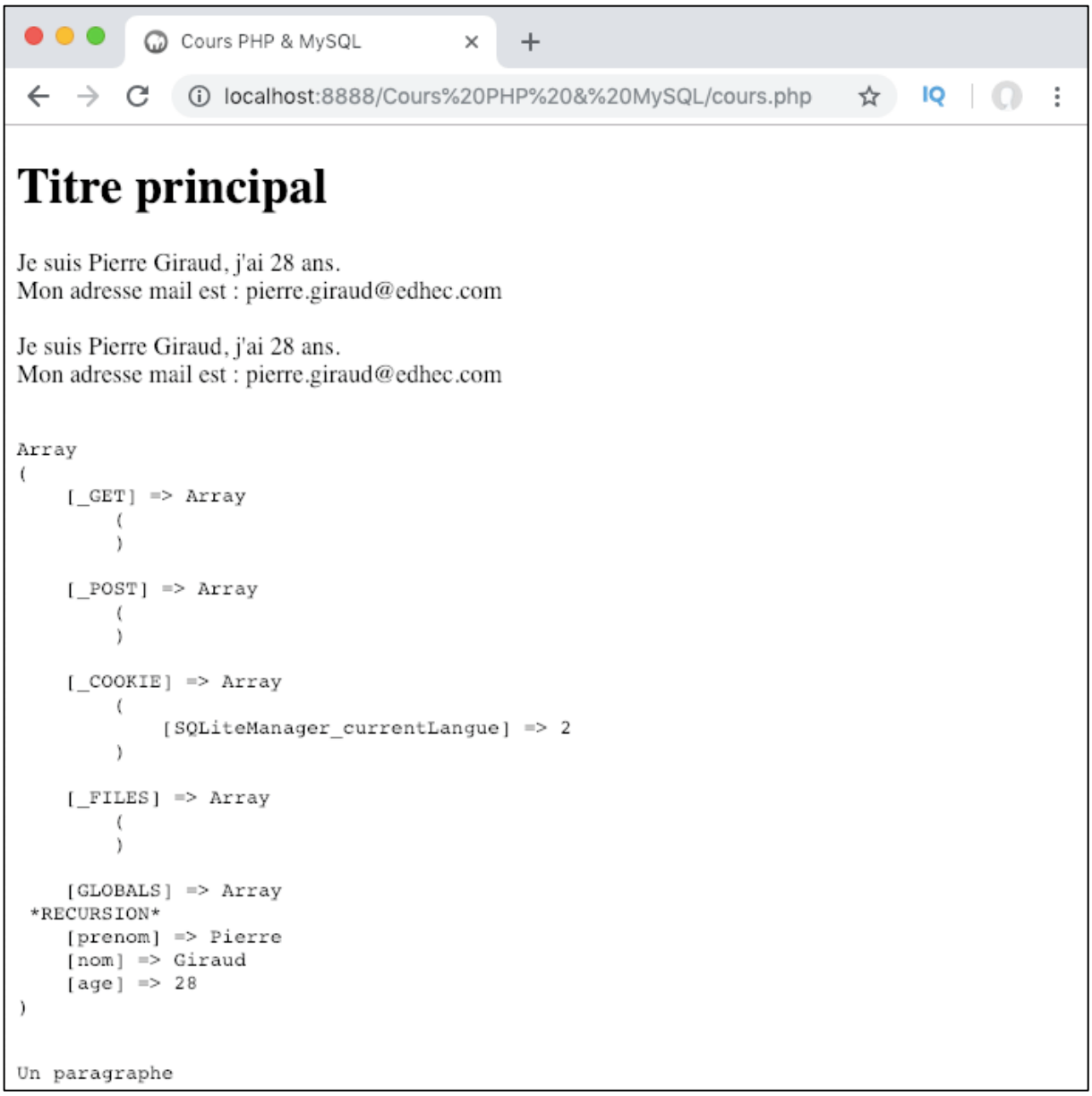
La variable superglobale `$GLOBALS` est une variable tableau qui stocke en fait automatiquement toutes les variables globales déclarées dans le script.

Ce tableau est un tableau associatif qui contient les noms des variables créées dans l'espace global en index et leurs valeurs en valeurs du tableau.

Si vous avez lu le chapitre sur la portée des variables, vous devriez vous rappeler du mot clef `global` qui sert justement à accéder à une variable définie dans l'espace global depuis un espace local.

Ce mot clef est lié au contenu de `$GLOBALS` et il va souvent être équivalent d'utiliser `global` ou `$GLOBALS` pour accéder à une variable en particulier depuis un espace local.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = 'Pierre';
            $nom = 'Giraud';
            $age = 28;

            //On utilise le mot clef global
            function prez(){
                $mail = 'pierre.giraud@edhec.com';
                global $prenom, $nom, $age;
                echo 'Je suis ' .$prenom. ' ' .$nom. ', j\'ai ' .$age. ' ans.<br>
                Mon adresse mail est : ' .$mail;
            }

            //On utilise la superglobale $GLOBALS
            function prez2(){
                $mail = 'pierre.giraud@edhec.com';
                echo 'Je suis ' .$GLOBALS[prenom]. ' ' .$GLOBALS[nom]. ', j\'ai '
                .$GLOBALS[age]. ' ans.<br>Mon adresse mail est : ' .$mail;
            }
            prez();
            echo '<br><br>';
            prez2();
            echo '<br><br>';
            echo '<pre>';
            print_r($GLOBALS);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on commence par définir 3 variables dans l'espace global : `$prenom`, `$nom` et `$age`. On crée ensuite deux fonctions qui vont utiliser ces variables.

Pour accéder à nos variables globales dans notre première fonction `prez()`, on utilise le mot clef `global`.

Pour accéder à nos variables globales depuis notre deuxième fonction `prez2()`, on va plutôt cette fois-ci utiliser notre superglobale `$GLOBALS`. On va lui passer les noms des variables auxquelles on souhaite accepter en indice afin que le tableau nous retourne les valeurs associées.

Comme vous pouvez le voir, les deux méthodes sont ici équivalentes. Je vous conseille dans cette situation d'utiliser plutôt le mot clef `global`.

La variable superglobale `$GLOBALS` va pouvoir être intéressante lorsqu'on voudra parcourir rapidement l'ensemble des variables globales définies dans un script. Comme cette variable est un tableau associatif, on peut utiliser une boucle `foreach` pour afficher

toutes ses valeurs ou tout simplement la fonction `print_r()` pour afficher rapidement sa structure.

La variable superglobale $_SERVER

La superglobale `$_SERVER` contient des variables définies par le serveur utilisé ainsi que des informations relatives au script.

Cette superglobale est à nouveau un tableau associatif dont les clefs sont les noms des variables qu'elle stocke et les valeurs sont les valeurs des variables liées.

Voici quelques clefs qu'il peut être intéressant de connaitre :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //Renvoie le nom du fichier contenant le script
            echo $_SERVER['PHP_SELF']. '<br>';

            //Renvoie le nom du serveur qui héberge le fichier
            echo $_SERVER['SERVER_NAME']. '<br>';

            //Renvoie l'adresse IP du serveur qui héberge le fichier
            echo $_SERVER['SERVER_ADDR']. '<br>';

            //Retourne l'IP du visiteur demandant la page
            echo $_SERVER['REMOTE_ADDR']. '<br>';

            /*Renvoie une valeur non vide si le script a
             *été appelé via le protocole HTTPS*/
            echo $_SERVER['HTTPS']. '<br>';

            //Retourne le temps Unix du début de la requête
            echo $_SERVER['REQUEST_TIME'];
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Vu que j'exécute ce script localement, ces valeurs ne sont évidemment pas très intéressantes dans mon cas. Cependant, elles vont pouvoir s'avérer très utiles dans le cas d'un « vrai » site hébergé sur un serveur.

Notez qu'il est possible que vous n'ayez pas les mêmes valeurs que moi renvoyées par `$_SERVER['SERVER_ADDR']` et `$_SERVER['REMOTE_ADDR']`. En effet, selon que vous utilisiez une adresse ipv6 ou ipv4, vous pouvez avoir soit la valeur `::1` soit la valeur `127.0.0.1`

La variable superglobale $_REQUEST

La variable superglobale `$_REQUEST` va contenir toutes les variables envoyées via HTTP GET, HTTP POST et par les cookies HTTP.

Cette variable, qui est un tableau associatif, va ainsi contenir les variables de `$_GET`, `$_POST` et `$_COOKIE`.

Nous n'avons pas pour le moment les connaissances nécessaires pour illustrer le fonctionnement de cette variable, nous en reparlerons plus tard.

La variable superglobale $_ENV

La superglobale `$_ENV` va contenir des informations liées à l'environnement dans lequel s'exécute le script.

Cette variable est une nouvelle fois une variable tableau. Celle-ci va pouvoir contenir, par exemple, le nom de l'utilisateur qui exécute le script si celui-ci est accessible.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo $_ENV['USER']. ' exécute actuellement ce script';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

La variable superglobale $_FILES

La superglobale `$_FILES` va contenir des informations sur un fichier téléchargé, comme le type du fichier, sa taille, son nom, etc.

On pourra donc utiliser cette superglobale lorsqu'on offre la possibilité à nos utilisateurs de nous envoyer des fichiers, afin d'obtenir des informations sur les fichiers envoyés ou même pour filtrer et interdire l'envoi de certains fichiers.

Les variables superglobales $_GET et $_POST

Les superglobales `$_GET` et `$_POST` vont être utilisées pour manipuler les informations envoyées via un formulaire HTML.

En effet, ces deux superglobales vont stocker les différentes valeurs envoyées par un utilisateur via un formulaire selon la méthode d'envoi : `$_GET` stockera les valeurs lorsque le formulaire sera envoyé via la méthode `GET` tandis que `$_POST` stockera les valeurs lorsque le formulaire sera envoyé via la méthode `POST`.

Nous allons pouvoir revoir en détail le fonctionnement de ces variables lorsqu'on va étudier la gestion des données des formulaires en PHP. Cependant, on peut déjà prendre un premier exemple simple :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <form method="post" action="cours.php">
            <label for="Prenom">Entrez votre prénom</label>
            <input type="text" id="prenom" name="prenom">

            <input type="submit" value="Envoyer">
        </form>

        <?php
            //Si notre variable $_POST['prenom'] est bien définie, echo...
            if(isset($_POST['prenom'])){
                echo 'Bonjour, tu t\'appelles ' .$_POST['prenom'];
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on définit la méthode `post` comme méthode d'envoi du formulaire et la page `cours.php` comme page de traitement des données (la page où les données du formulaire devront être envoyées).

Dans le cas présent, cette page est la même que la page dans laquelle se situe le formulaire, ce qui signifie que les données seront envoyées vers notre page courante. On va donc les récupérer dans notre page.

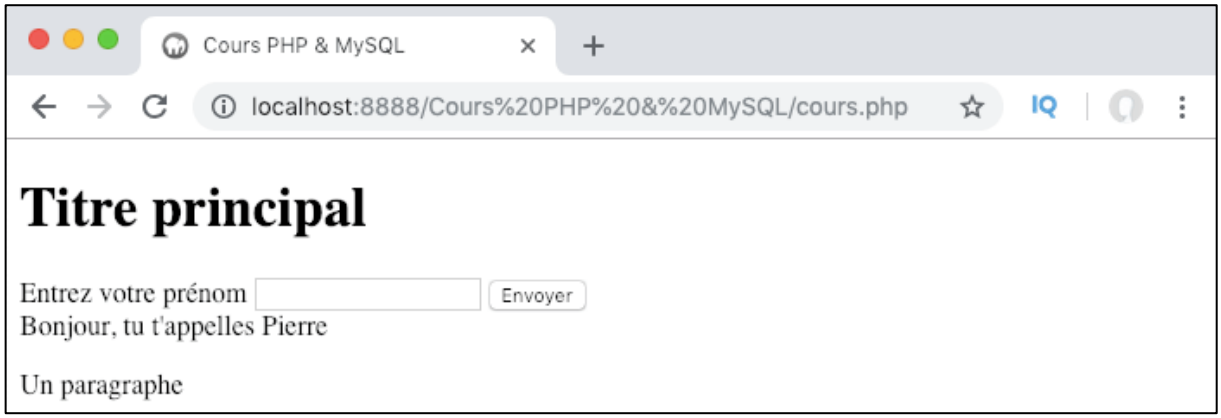
Notre formulaire en soi ne contient qu'un champ `input` qui va demander un prénom ainsi qu'un bouton d'envoi.

Pour récupérer les données envoyées dans notre script PHP, nous allons utiliser la superglobale `$_POST`. Cette superglobale est un tableau associatif qui prend en clefs les noms passés via les attributs `name` de nos formulaires.

On va commencer dans notre script PHP par créer une condition pour vérifier qu'un prénom a bien été envoyé en vérifiant que `$_POST['prenom']` ait bien une valeur associée grâce à la fonction `isset()` qui sert à vérifier si une variable a bien été définie et est différente de `NULL`.

Si la variable existe bien, `isset()` retourne `true` et le test de notre condition est vérifié. Dans le cas contraire, `isset()` renvoie `false` et le test de notre condition échoue et le code dans la condition n'est donc pas exécuté.

Ici, le code de notre condition sert simplement à afficher la valeur envoyée via le formulaire. Si je tape « Pierre » dans mon navigateur et valide, j'obtiens donc le résultat suivant :



La variable superglobale $_COOKIE

La superglobale `$_COOKIE` est un tableau associatif qui contient toutes les variables passées via des cookies HTTP. Nous allons étudier cette variable superglobale en détail dans la suite de ce cours.

La variable superglobale $_SESSION

La superglobale `$_SESSION` est un tableau associatif qui contient toutes les variables de session. Nous allons étudier cette variable superglobale en détail dans la suite de ce cours.

Création et gestion de cookies en PHP

Dans cette nouvelle leçon, nous allons découvrir ce que sont les cookies et comprendre tout leur intérêt. En particulier, nous allons apprendre à définir, lire et à supprimer un cookie.

Présentation des cookies

Un cookie est un petit fichier texte qui ne peut contenir qu'une quantité limitée de données.

Les cookies vont être stockés sur les ordinateurs de vos visiteurs. Ainsi, à tout moment, un utilisateur peut lui même supprimer les cookies de son ordinateur.

De plus, les cookies vont toujours avoir une durée de vie limitée. On pourra définir la date d'expiration d'un cookie.

Généralement, nous allons utiliser les cookies pour faciliter la vie des utilisateurs en préenregistrant des données les concernant comme un nom d'utilisateur par exemple.

Ainsi, dès qu'un utilisateur connu demande à accéder à une page de notre site, les cookies vont également automatiquement être envoyées dans la requête de l'utilisateur. Cela va nous permettre de l'identifier et de lui proposer une page personnalisée.

Les cookies ne sont donc pas dangereux en soi même s'ils continuent d'avoir mauvaise réputation. En revanche, on évitera toujours de stocker des informations sensibles dans les cookies comme des mots de passe par exemple car les cookies sont stockés sur l'ordinateur des visiteurs et nous n'avons donc aucune maitrise ni aucun moyen de les sécuriser après le stockage.

Créer un cookie en PHP

Pour créer un cookie en PHP, nous allons utiliser la fonction `setcookie()`.

Une particularité notable de cette fonction est qu'il va falloir l'appeler avant d'écrire tout code HTML pour qu'elle fonctionne puisque les cookies doivent être envoyés avant toute autre sortie. Pour information, cette restriction provient du protocole HTTP et non pas de PHP.

Cette fonction peut accepter jusqu'à sept valeurs en arguments. Cependant, seul la première (le nom du cookie créé) est obligatoire.

La syntaxe de base de `setcookie()` est la suivante < code>setcookie(name, value, expire, path, domain, secure, httponly). Les paramètres ont la signification suivante :

Paramètre	Signification
name	Le nom du cookie. Le nom d'un cookie est soumis aux mêmes règles que les noms des variables.
value	La valeur du cookie. Comme cette valeur est stockée sur l'ordinateur d'un utilisateur, on évitera de stocker des informations sensibles.
expires	La date d'expiration du cookie sous forme d'un timestamp UNIX (nombre de secondes écoulées depuis le 1er janvier 1970). Si aucune valeur n'est passée en argument, le cookie expirera à la fin de la session (lorsque le navigateur sera fermé).
path	Le chemin sur le serveur sur lequel le cookie sera disponible. Si la valeur est '/', le cookie sera disponible sur l'ensemble du domaine. Si la valeur est '/cours/', le cookie ne sera disponible que dans le répertoire (le dossier) /cours/ (et dans tous les sous-répertoires qu'il contient).
domain	Indique le domaine ou le sous domaine pour lequel le cookie est disponible.
secure	Indique si le cookie doit uniquement être transmis à travers une connexion sécurisée HTTPS depuis le client. Si la valeur passée est `true`, le cookie ne sera envoyé que si la connexion est sécurisée.
httponly	Indique si le cookie ne doit être accessible que par le protocole HTTP. Pour que le cookie ne soit accessible que par le protocole http, on indiquera la valeur `true`. Cela permet d'interdire l'accès au cookie aux langages de scripts comme le JavaScript par exemple, pour se protéger potentiellement d'une attaque de type XSS.

Créons immédiatement deux premiers cookies pour bien comprendre comment fonctionne `setcookie()`.

```php
<?php
    setcookie('user_id', '1234');
    setcookie('user_pref', 'dark_theme', time()+3600*24, '/', '', true, true);
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php

        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on crée deux cookies avec `setcookie()` : un premier cookie nommé `user_id` qui stockera l'ID d'un visiteur pour la session actuelle par exemple ce qui nous permettra de l'identifier pendant sa navigation sur notre site et un deuxième cookie qu'on appelle `user_pref` qui stockera les préférences mentionnés par l'utilisateur pour notre site (utilisation d'un thème sombre par exemple).

Bien évidemment, dans le cas présent, il faut imaginer qu'on possède un système nous permettant de créer des ID de session et que notre site propose aux utilisateurs de choisir une apparence personnalisée pour celui-ci car ce n'est pas l'objet de la leçon.

Comme je vous l'ai précisé précédemment, il faut appeler cette fonction avant d'écrire un quelconque code HTML. Nous appelons donc `setcookie()` dans une balise PHP, avant même d'écrire notre élément `html`.

Pour notre premier cookie `user_id`, nous ne précisons qu'un nom et une valeur et laissons le PHP utiliser les valeurs par défaut pour les autres arguments de `setcookie()`.

Nous utilisons ensuite la fonction `time()` sans lui passer d'argument pour récupérer la valeur du timestamp actuel. Nous allons nous servir de cette valeur en lui ajoutant un certain nombre de secondes pour définir la date d'expiration de notre deuxième cookie `user_pref`. Dans le cas présent, on définit une durée de vie de 24h pour le cookie (3600 secondes * 24 à partir de sa date de création).

Toujours pour notre deuxième cookie, nous utilisons la valeur par défaut pour le chemin du serveur sur lequel le serveur est accessible, c'est à dire la valeur `/` qui signifie que le cookie sera accessible sur l'ensemble d'un domaine ou d'un sous-domaine (c'est-à-dire dans tous ses répertoires).

On ne précise pas de domaine de validité ici car nous travaillons en local. Si j'avais voulu rendre mon cookie disponible pour tout mon site, j'aurais précisé `pierre-giraud.com`.

Finalement, nous précisons les valeurs `true` pour les arguments « secure » (passage par une connexion sécurisée pour transmettre le cookie) et « httponly » (obligation d'utiliser le protocole HTTP pour accéder au cookie).

Récupérer la valeur d'un cookie

Pour récupérer la valeur d'un cookie, nous allons utiliser la variable superglobale `$_COOKIE`.

Cette superglobale est un tableau associatif qui utilise les noms des cookies en clefs et associe leurs valeurs en valeurs du tableau.

On va donc pouvoir accéder à la valeur d'un cookie en particulier en renseignant le nom du cookie en clef de ce tableau.

```php
<?php
    setcookie('user_id', '1234');
    setcookie('user_pref', 'dark_theme', time()+3600*24, '/', '', true, true);
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            if(isset($_COOKIE['user_id'])){
                echo 'Votre ID de session est le ' .$_COOKIE['user_id'];
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on commence par vérifier qu'un cookie `user_id` existe et a bien été défini et stocké dans `$_COOKIE` avec la fonction `isset()`. Si c'est le cas, on `echo` la valeur du cookie.

Ici, il faut bien noter que la variable `$_COOKIE` stocke la liste des cookies renvoyés par le navigateur. Lorsqu'un utilisateur demande à accéder à notre page pour la première fois, le cookie `user_id` est créé côté serveur et est renvoyé au navigateur afin qu'il soit stocké sur la machine du visiteur.

Ainsi, la première fois qu'un utilisateur demande notre page, la variable `$_COOKIE` ne stocke pas encore notre cookie puisque celui-ci n'a pas encore été créé et donc le navigateur du visiteur ne peut rien renvoyer. Le test de notre condition `if` échoue donc lors du premier affichage de la page.

Si on actualise ensuite la page, en revanche, le navigateur renvoie bien cette fois-ci la valeur de notre cookie et son nom et celui-ci est bien stocké dans `$_COOKIE`. Le test de notre condition va alors être vérifié et on va pouvoir `echo` la valeur de ce cookie.

Modifier la valeur d'un cookie ou supprimer un cookie

Pour modifier la valeur d'un cookie, nous allons appeler à nouveau la fonction `setcookie()` en lui passant le nom du cookie dont on souhaite changer la valeur et changer l'argument de type valeur passé à la fonction avec la nouvelle valeur souhaitée.

Pour supprimer un cookie, nous allons encore appeler `setcookie()` en lui passant le nom du cookie qu'on souhaite supprimer et allons cette fois-ci définir une date d'expiration se situant dans le passé pour le cookie en question.

```php
<?php
    //On définit deux cookies
    setcookie('user_id', '1234');
    setcookie('user_pref', 'dark_theme', time()+3600*24, '/', '', false, false);

    //On modifie la valeur du cookie user_id
    setcookie('user_id', '5678');

    //On supprime le cookie user_pref
     setcookie('user_pref', '', time()-3600, '/', '', false, false);
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            if(isset($_COOKIE['user_id'])){
                echo 'Votre ID de session est le ' .$_COOKIE['user_id']. '<br>';
            }
            if(isset($_COOKIE['user_pref'])){
                echo 'Votre thème préféré est ' .$_COOKIE['user_pref'];
            }else{
                echo 'Pas de thème préféré défini';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on commence par définir deux cookies `user_id` et `user_pref`. On modifie ensuite la valeur de notre cookie `user_id` et on passe une date d'expiration passée à notre cookie `user_pref` pour le supprimer.

Bien évidemment, encore une fois, cela n'a pas l'air très intéressant dans le cas présent car nous définissons nous-mêmes nos cookies, leurs valeurs et leurs durées de vie manuellement.

Cependant, vous devez à chaque fois imaginer que toutes ces notions vont être utiles dans un contexte dynamique où il faudra changer la valeur d'un cookie en fonction du changement de préférence d'un utilisateur par exemple.

Définir et utiliser des sessions en PHP

Dans cette leçon, nous allons découvrir ce qu'est une session en PHP et apprendre à utiliser une nouvelle variable superglobale : la superglobale `$_SESSION`.

Présentation des sessions PHP

Une session en PHP correspond à une façon de stocker des données différentes pour chaque utilisateur en utilisant un identifiant de session unique.

Les identifiants de session vont généralement être envoyés au navigateur via des cookies de session et vont être utilisés pour récupérer les données existantes de la session.

Un des grands intérêts des sessions est qu'on va pouvoir conserver des informations pour un utilisateur lorsqu'il navigue d'une page à une autre. De plus, les informations de session ne vont cette fois-ci pas être stockées sur les ordinateurs de vos visiteurs à la différence des cookies mais plutôt côté serveur ce qui fait que les sessions vont pouvoir être beaucoup plus sûres que les cookies.

Notez toutefois que le but des sessions n'est pas de conserver des informations indéfiniment mais simplement durant une « session ». Une session démarre dès que la fonction `session_start()` est appelée et se termine en général dès que la fenêtre courante du navigateur est fermée (à moins qu'on appelle une fonction pour terminer la session de manière anticipée ou qu'un cookie de session avec une durée de vie plus longues ait été défini).

La superglobale `$_SESSION` est un tableau associatif qui va contenir toutes les données de session une fois la session démarrée.

Démarrer une session en PHP

Pour pouvoir utiliser les variables de session, il va avant tout falloir qu'une session soit démarrée à un moment ou à un autre.

Pour démarrer une session en PHP, on va utiliser la fonction `session_start()`. Cette fonction va se charger de vérifier si une session a déjà été démarrée en recherchant la présence d'un identifiant de session et, si ce n'est pas le cas, va démarrer une nouvelle session et générer un identifiant de session unique pour un utilisateur.

Il va falloir appeler `session_start()` avant toute autre opération dans nos pages, c'est-à-dire au début de celles-ci de la même façon qu'on a déjà pu le faire avec la fonction `setcookie()`. Par ailleurs, notez qu'il va falloir appeler `session_start()` dans chaque page où on souhaite pouvoir accéder aux variables de session. En pratique, on créera généralement une page `header.php` qui va contenir notre fonction `session_start()` et qu'on va inclure à l'aide de `include` ou `require` dans les pages voulues d'un site.

Lorsqu'une session est démarrée, c'est-à-dire lorsqu'un utilisateur qui ne possède pas encore d'identifiant de session demande à accéder à une page contenant `session_start()`,

cette fonction va générer un identifiant de session unique qui va généralement être envoyé au navigateur sous forme de cookie sous le nom PHPSESSID.

Pour être tout à fait précis, le PHP supporte deux méthodes pour garder la trace des sessions : via des cookies ou via l'URL. Si les cookies sont activés, le PHP va préférer leur utilisation. C'est le comportement recommandé. Dans le cas contraire, les informations de session vont être passées via l'URL.

```php
<?php
    //On démarre une nouvelle session
    session_start();

    /*On utilise session_id() pour récupérer l'id de session s'il existe.
     *Si l'id de session n'existe  pas, session_id() rnevoie une chaine
     *de caractères vide*/
    $id_session = session_id();
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            if($id_session){
                echo 'ID de session (récupéré via session_id()) : <br>'
                .$id_session. '<br>';
            }
            echo '<br><br>';
            if(isset($_COOKIE['PHPSESSID'])){
                echo 'ID de session (récupéré via $_COOKIE) : <br>'
                .$_COOKIE['PHPSESSID'];
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Notez que dès qu'une session est lancée, le PHP va créer automatiquement un petit fichier de session qui va contenir les informations liées à la session durant le temps de celle-ci.

Définir et récupérer des variables de session

Pour définir et récupérer les valeurs des variables de session, nous allons pouvoir utiliser la variable superglobale `$_SESSION`.

Cette superglobale est un tableau associatif qui stocke les différentes variables de sessions avec leurs noms en index du tableau et leurs valeurs en valeurs du tableau.

Nous allons donc très simplement pouvoir à la fois définir de nouvelles variables de session et modifier ou récupérer les valeurs de nos variables de session.

Une fois une variable de session définie, celle-ci va pouvoir être accessible durant la durée de la session à partir de toutes les pages du site pour lesquelles les sessions ont été activées. Pour illustrer cela, on peut créer une autre page `session.php` en plus de notre page `cours.php`.

On va déjà démarrer une nouvelle session et créer quelques variables de session manuellement dans notre page `cours.php` :

```php
<?php
    //On démarre une nouvelle session
    session_start();

    //On définit des variables de session
    $_SESSION['prenom'] = 'Pierre';
    $_SESSION['age'] = 29;
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //Du code PHP
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

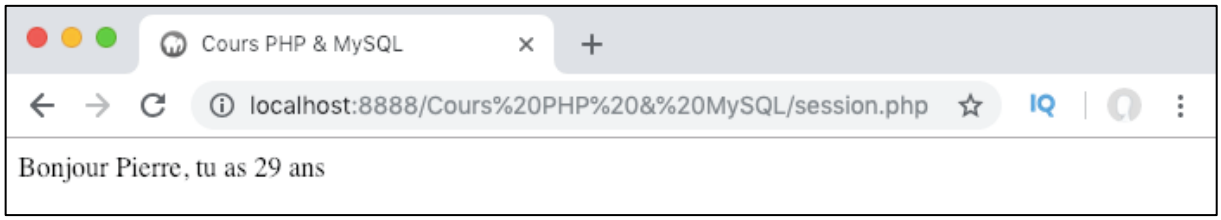
Ensuite, nous allons également utiliser `session_start()` dans notre page `session.php` pour activer les sessions. En effet, je vous rappelle que cette fonction permet de démarrer une session si aucun identifiant de session n'existe ou de reprendre une session existante dans le cas contraire.

Une fois les sessions activées sur notre page, nous allons pouvoir récupérer les valeurs des variables de session définies dans notre page précédente et les afficher ou les manipuler d'une quelconque façon :

cours.php × cours.css × menu.php × session.php ×

... MAMP > htdocs > Cours PHP & MySQL > session.php > Ln: 22 Col: 1 UTF-8

```php
<?php
    session_start();
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <?php
            echo 'Bonjour ' .$_SESSION['prenom']. ',
            tu as ' .$_SESSION['age']. ' ans';
        ?>
    </body>
</html>
```



Terminer une session et détruire les variables de session

Une session PHP se termine généralement automatiquement lorsqu'un utilisateur ferme la fenêtre de son navigateur.

Il peut être cependant parfois souhaitable de terminer une session avant. Pour faire cela, nous allons pouvoir utiliser les fonctions `session_destroy()` qui détruit toutes les données associées à la session courante et `session_unset()` qui détruit toutes les variables d'une session.

La fonction `session_destroy()` va supprimer le fichier de session dans lequel sont stockées toutes les informations de session. Cependant, cette fonction ne détruit pas les variables globales associées à la session (c'est-à-dire le contenu du tableau `$_SESSION`) ni le cookie de session.

Pour détruire totalement une session, il va également falloir supprimer l'identifiant de session. Généralement, cet identifiant est contenu dans le cookie `PHPSESSID` qu'on pourra effacer en utilisant `setcookie()` en définissant une date d'expiration passée pour le cookie.

Il va cependant être très rare d'avoir besoin de détruire les données associées à une session et donc d'appeler `session_destroy()`. On préférera généralement modifier le tableau `$_SESSION` manuellement pour supprimer des données en particulier.

Notez qu'on va également pouvoir utiliser la fonction `session_unset()` (sans lui passer d'argument) pour détruire toutes les variables de la session courante. Cette fonction va également nous permettre de détruire une variable de session en particulier en lui passant sa valeur de la manière suivante : `unset($_SESSION['nom-de-la-variable-de-session-a-detruire'])`.

```php
<?php
    //On démarre une nouvelle session
    session_start();

    //On définit des variables de session
    $_SESSION['prenom'] = 'Pierre';
    $_SESSION['age'] = 29;
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*Si la variable de session age est définie, on echo sa valeur
             *puis on la détruit avec unset()*/
            if(isset($_SESSION['age'])){
                echo 'Tu as ' .$_SESSION['age']. ' ans<br>';
                unset($_SESSION['age']);
            }

            /*On détruit les données de session*/
            session_destroy();

            //On tente d'afficher les valeurs des variables age et prenom
            echo 'Contenu de $_SESSION[\'age\'] : ' .$_SESSION['age']. '<br>';
            echo 'Contenu de $_SESSION[\'prenom\'] : ' .$_SESSION['prenom'];
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on commence par démarrer une session ou par reprendre une session existante avec `session_start()`. Cette étape est essentielle si on souhaite supprimer des informations de session.

Ensuite, dans notre script, on vérifie que la variable `$_SESSION['age']` ait bien été définie et, si c'est le cas, on affiche sa valeur puis on la détruit avec `unset()`. A la fin du script, on détruit les informations associées à la session avec `session_destroy()`.

On essaie alors d'afficher le contenu de nos variables de session en utilisant le tableau `$_SESSION`. Ici, `$_SESSION['age']` ne renvoie aucune valeur puisqu'on l'a détruite avec `unset()`. En revanche, `$_SESSION['prenom']` renvoie bien toujours une valeur.

En effet, je vous rappelle ici que `session_destroy()` ne va pas détruire les variables globales de session. Cependant, comme les informations de session sont détruites, les variables de session ne vont plus être accessibles que dans le script courant.

PARTIE VIII

Manipuler des fichiers en PHP

Introduction à la manipulation de fichiers en PHP

Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers en PHP. Nous allons ainsi apprendre à ouvrir et lire un fichier déjà existant ou à créer des fichiers de différents formats (fichiers texte, etc.) grâce aux fonctions PHP et à écrire des informations dedans.

Le PHP met à notre disposition de nombreuses fonctions nous permettant de travailler avec les fichiers (quasiment une centaine !) car la manipulation de fichiers est un sujet complexe et pour lequel les problématiques peuvent être très différentes et très précises.

Nous n'allons pas dans ce cours explorer le sujet à 100% car cela serait très indigeste et rajouterait beaucoup de complexité « inutilement » mais allons plutôt présenter les cas de manipulation de fichiers les plus généraux et les fonctions communes liées.

La manipulation de fichiers en PHP

En PHP, nous allons pouvoir manipuler différents types de fichiers comme des fichiers texte (au format `.txt`) ou des fichiers image par exemple. Bien évidemment, nous n'allons pas pouvoir effectuer les mêmes opérations sur chaque type de fichier.

Dans ce cours, nous allons particulièrement nous intéresser à la manipulation de fichiers texte puisque ce sont les fichiers qu'on manipule le plus souvent en pratique et car nous allons pouvoir stocker du texte dans ce type de fichier.

En effet, jusqu'à présent, nous n'avons appris à stocker des informations que de manière temporaire en utilisant les variables « classiques », les cookies ou les sessions. L'utilisation de fichiers en PHP va nous permettre entre autres de stocker de façon définitive des informations.

Les fichiers vont donc nous offrir une alternative aux bases de données qui servent également à stocker des informations de manière organisée et définitive et que nous étudierons plus tard dans le cours même s'il faut bien admettre qu'on préférera dans la majorité des cas utiliser les bases de données plutôt que les fichiers pour enregistrer les données.

Il reste néanmoins intéressant d'apprendre à stocker des données dans des fichiers car on préfèrera dans certains cas stocker des données dans des fichiers plutôt que dans des bases de données (par exemple lorsqu'on laissera la possibilité aux utilisateurs de télécharger ensuite le fichier avec ses différentes informations à l'intérieur).

Par ailleurs, la manipulation de fichiers ne se limite pas au stockage de données : nous allons pouvoir effectuer toutes sortes d'opérations sur nos fichiers ce qui représente un avantage indéniable.

Lire un fichier entier en PHP

L'une des opérations de base relative aux fichiers en PHP va tout simplement être de lire le contenu d'un fichier.

Il existe deux façons de faire cela : on peut soit lire le contenu d'un fichier morceau par morceau, chose que nous apprendrons à faire dans la prochaine leçon, soit lire un fichier entièrement c'est-à-dire afficher tout son contenu d'un coup.

Pour faire cela, on va pouvoir utiliser l'une des fonctions suivantes :

- La fonction `file_get_contents()` ;
- La fonction `file()` ;
- La fonction `readfile ()` ;

On va devoir passer le chemin relatif menant au fichier qu'on souhaite lire en argument de chacune de ces fonctions.

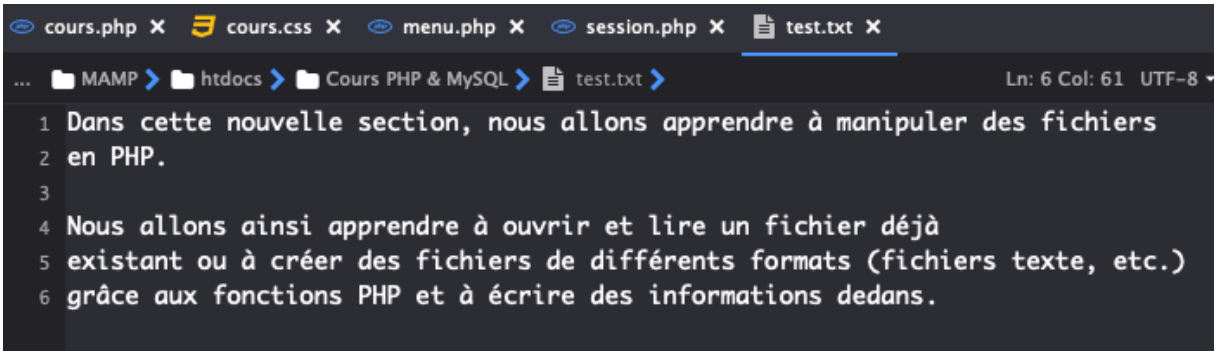
La fonction `file_get_contents()` va retourner le contenu du fichier dans une chaine de caractères qu'il faudra `echo` pour afficher ou la valeur booléenne `false` en cas d'erreur.

La fonction `file()` est identique à `file_get_contents()` à la seule différence que le contenu du fichier sera renvoyé dans un tableau numéroté. Chaque ligne de notre fichier sera un nouvel élément de tableau.

La fonction `readfile()` va elle directement lire et afficher le contenu de notre fichier.

Pour illustrer le fonctionnement de ces trois fonctions, je vous invite à créer un premier fichier de texte, c'est-à-dire un fichier enregistré avec l'extension `.txt`.

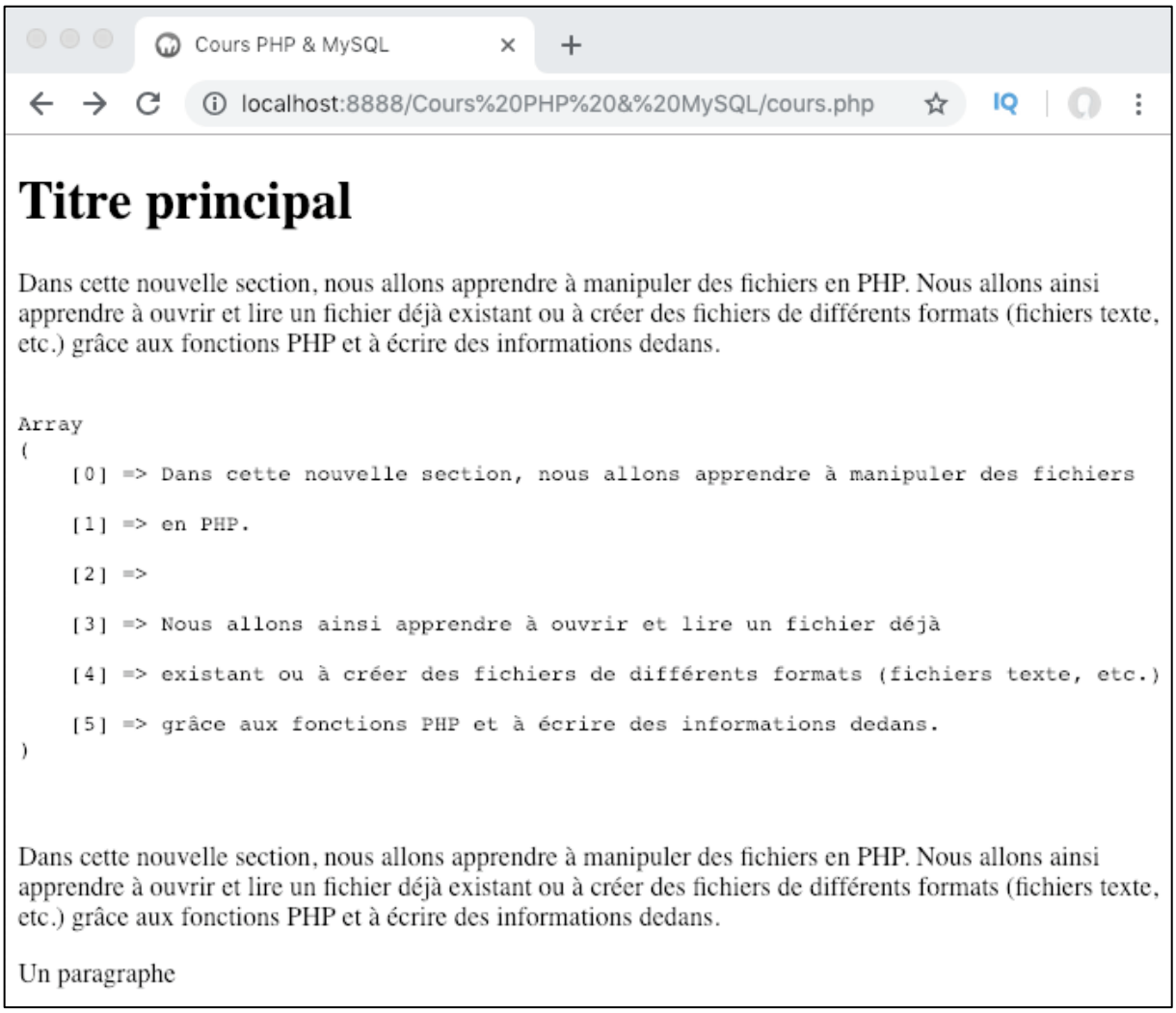
Personnellement, j'utiliserai un fichier que j'ai appelé `exemple.txt` et qui contient le texte suivant :



Pour créer ce fichier texte, vous pouvez par exemple vous servir de votre éditeur. Je vous conseille ici de l'enregistrer dans le même dossier que notre fichier `cours.php` pour plus de simplicité.

Dès que le fichier est créé et enregistré, nous allons tenter de l'afficher dans notre script :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo file_get_contents('test.txt');
            echo '<br><br>';

            echo '<pre>';
            print_r(file('test.txt'));
            echo '</pre>';
            echo '<br><br>';

            readfile('test.txt');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on utilise nos trois fonctions `file_get_contents()`, `file()` et `readfile()` à la suite pour lire et afficher notre fichier. Comme vous pouvez le voir, `file_get_contents()` et `readfile()` produisent un résultat similaire mais il faut `echo` le résultat renvoyé par `file_get_contents()` pour l'afficher tandis que `readfile()` affiche directement le résultat.

La fonction `file_get_contents()` nous donne donc davantage de contrôle sur la valeur renvoyée puisqu'on va pouvoir l'enfermer dans une variable pour la manipuler à loisir.

Conserver la mise en forme d'un fichier avant affichage

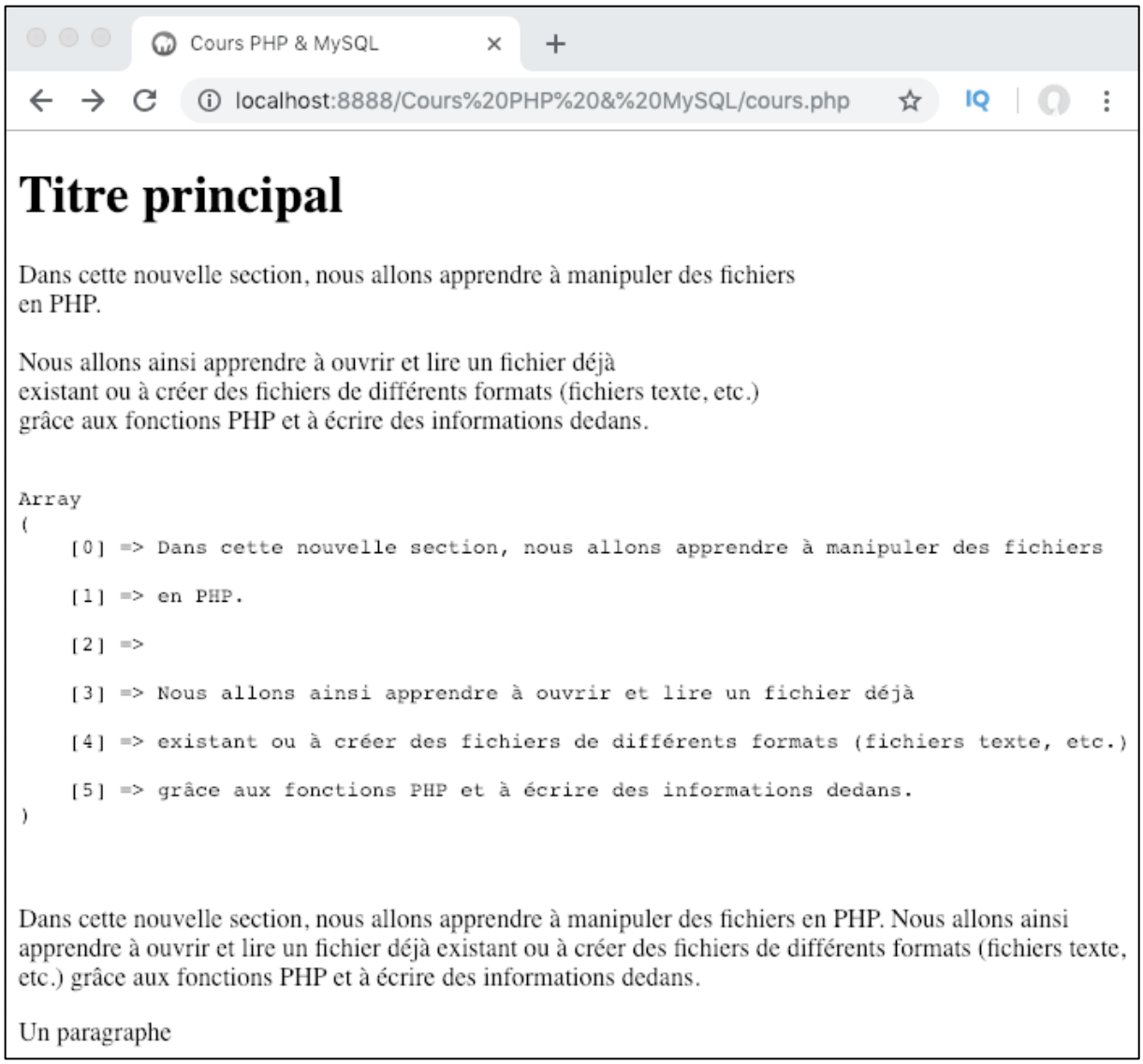
Dans l'exemple ci-dessus, on s'aperçoit que les retours à la ligne et les sauts de ligne contenus dans notre fichier texte ne sont pas conservés lors de l'affichage du texte dans le navigateur.

Cela est normal puisque nos fonctions vont renvoyer le texte tel quel sans y ajouter des éléments HTML indiquant de nouvelles lignes ou des sauts de ligne et donc le navigateur va afficher le texte d'une traite.

Pour conserver la mise en forme du texte, on va pouvoir utiliser la fonction `nl2br()` qui va se charger d'ajouter des éléments `br` devant chaque nouvelle ligne de notre texte. On va passer le texte à mettre en forme en argument de cette fonction.

On va par exemple pouvoir utiliser cette fonction sur le résultat renvoyé par `file_get_contents()` avant de l'afficher. En revanche, on ne va pas pouvoir l'utiliser avec `readfile()` puisque cette fonction va directement afficher le résultat sans que nous puissions exercer un quelconque contrôle dessus.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo nl2br(file_get_contents('test.txt'));
            echo '<br><br>';

            echo '<pre>';
            print_r(file('test.txt'));
            echo '</pre>';
            echo '<br><br>';

            readfile('test.txt');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on effectue plusieurs opérations en une seule ligne. Comme d'habitude, lorsqu'on utilise plusieurs fonctions ensemble, c'est la fonction la plus interne dans notre code qui va s'exécuter en premier puis la fonction qui l'englobe va s'exécuter ensuite sur le résultat de la première fonction.

Dans notre cas, `file_get_contents()` s'exécute donc en premier et renvoie le contenu de notre fichier puis `nl2br()` s'exécute ensuite sur le résultat renvoyé par `file_get_contents()` (c'est-à-dire le texte de notre fichier) et ajoute des éléments `br` à chaque nouveau retour à la ligne. Finalement, la structure de langage `echo` permet d'afficher le résultat (le texte avec nos éléments `br`.

Ouvrir, lire et fermer un fichier en PHP

Dans la leçon précédente, nous avons appris à lire un fichier texte entièrement. Souvent, cependant, nous ne voudrons lire et récupérer qu'une partie d'un fichier. Nous allons voir comment faire cela dans cette nouvelle leçon.

Ouvrir un fichier en PHP

Pour lire un fichier partie par partie, nous allons avant tout devoir l'ouvrir. Pour ouvrir un fichier en PHP, nous allons utiliser la fonction `fopen()`, abréviation de « file open » ou « ouverture de fichier » en français.

On va devoir passer le chemin relatif menant à notre fichier ainsi qu'un « mode » en arguments à cette fonction qui va alors retourner en cas de succès une ressource de pointeur de fichier, c'est-à-dire pour le dire très simplement une ressource qui va nous permettre de manipuler notre fichier.

Le mode choisi détermine le type d'accès au fichier et donc les opérations qu'on va pouvoir effectuer sur le fichier. On va pouvoir choisir parmi les modes suivants :

Mode d'ouverture	Description
r	Ouvre un fichier en lecture seule. Il est impossible de modifier le fichier.
r+	Ouvre un fichier en lecture et en écriture.
a	Ouvre un fichier en écriture seule en conservant les données existantes. Si le fichier n'existe pas, le PHP tente de le créer.
a+	Ouvre un fichier en lecture et en écriture en conservant les données existantes. Si le fichier n'existe pas, le PHP tente de le créer.
w	Ouvre un fichier en écriture seule. Si le fichier existe, les informations existantes seront supprimées. S'il n'existe pas, crée un fichier.
w+	Ouvre un fichier en lecture et en écriture. Si le fichier existe, les informations existantes seront supprimées. S'il n'existe pas, crée un fichier.
x	Crée un nouveau fichier accessible en écriture seulement. Retourne false et une erreur si le fichier existe déjà.
x+	Crée un nouveau fichier accessible en lecture et en écriture. Retourne false et une erreur si le fichier existe déjà.

Mode d'ouverture	Description
c	Ouvre un fichier pour écriture seulement. Si le fichier n'existe pas, il sera créé. Si il existe, les informations seront conservées.
c+	Ouvre un fichier pour lecture et écriture. Si le fichier n'existe pas, il sera créé. Si il existe, les informations seront conservées.
e	Le mode e est particulier et n'est pas toujours disponible. Nous n'en parlerons pas ici.

Comme vous pouvez le constater, le choix du mode influe fortement sur les opérations que nous allons pouvoir réaliser sur le fichier en question.

Par exemple lorsqu'on ouvre un fichier en lecture seulement toute modification de ce fichier est impossible, ce qui n'est pas le cas si on choisit un mode permettant l'écriture dans ce fichier.

Notez par ailleurs qu'on ajoutera généralement la lettre `b` au paramètre « mode » de `fopen()`. Cela permet une meilleure compatibilité et évite les erreurs pour les systèmes qui différencient les fichiers textes et binaires comme Windows par exemple.

Lire un fichier partie par partie

PHP met à notre disposition plusieurs fonctions qui vont nous permettre de lire un fichier partie par partie :

- La fonction `fread()` ;
- La fonction `fgets()` ;
- La fonction `fgetc()`.

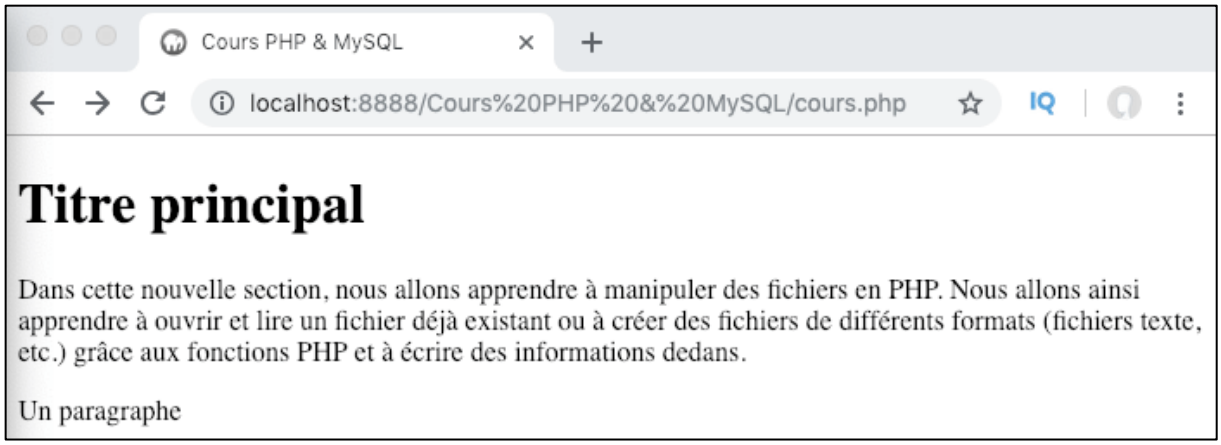
Lire un fichier jusqu'à un certain point avec fread()

La fonction `fread()` (abréviation de « file read » ou « lecture de fichier » en français) va prendre en arguments le fichier renvoyé par `fopen()` ainsi qu'un nombre qui correspond aux nombres d'octets du fichier qui doivent être lus.

Pour lire le fichier entièrement, on peut utiliser la fonction `filesize()` en lui passant le nom du fichier qu'on souhaite lire en deuxième argument de `fread()`. En effet, `filesize()` renvoie la taille d'un fichier. Cela va donc nous permettre de lire entièrement un fichier à condition de commencer la lecture au début.

Illustrons cela avec un exemple concret. Pour cet exemple, je réutilise le fichier `test.txt` créé dans la leçon précédente.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $ressource = fopen('test.txt', 'rb');
            echo fread($ressource, filesize('test.txt'));
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on commence par utiliser `fopen()` pour lire notre fichier et on récupère la ressource renvoyée par la fonction dans une variable `$ressource`.

On passe ensuite le contenu de notre variable à `fread()` et on lui demande de lire le fichier jusqu'au bout en lui passant la taille exacte du fichier obtenue avec `filesize()`.

Finalement, on affiche le résultat renvoyé par `fread()` grâce à une structure `echo`.

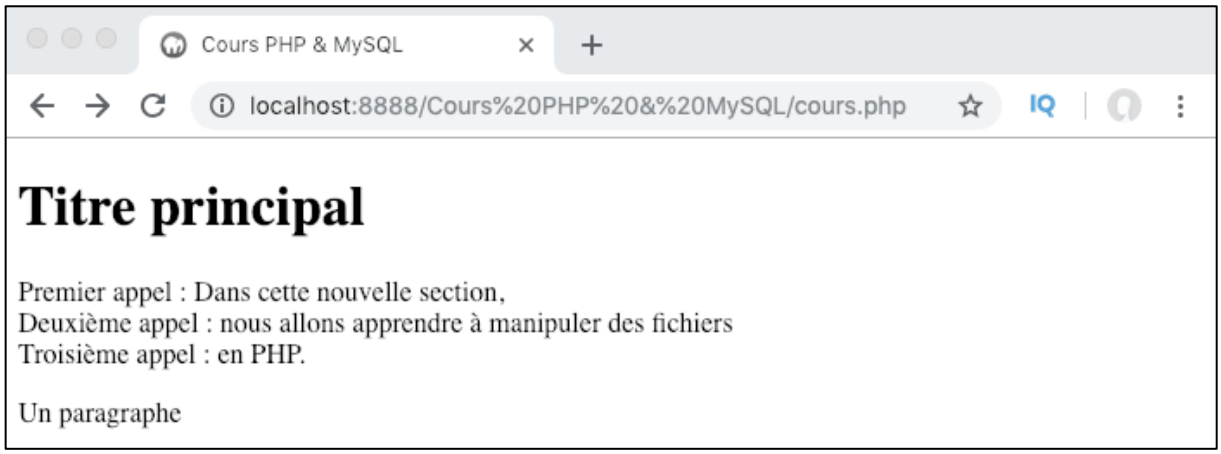
Lire un fichier ligne par ligne avec fgets()

La fonction `fgets()` va nous permettre de lire un fichier ligne par ligne. On va passer le résultat renvoyé par `fopen()` en argument de `fgets()` et à chaque nouvel appel de la fonction, une nouvelle ligne du fichier va pouvoir être lue.

On peut également préciser de manière facultative un nombre en deuxième argument de `fgets()` qui représente un nombre d'octets. La fonction lira alors soit le nombre d'octets précisé, soit jusqu'à la fin du fichier, soit jusqu'à arriver à une nouvelle ligne (le premier des trois cas qui va se présenter). Si aucun nombre n'est précisé, `fgets()` lira jusqu'à la fin de la ligne.

Notez que si on précise un nombre d'octets maximum à lire inférieur à la taille de notre ligne et qu'on appelle successivement `fgets()`, alors la fin de la ligne courante sera lue lors du deuxième appel de `fgets()`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $ressource = fopen('test.txt', 'rb');

            //Lit les 30 premiers caractères du fichier
            echo 'Premier appel : ' .fgets($ressource, 30). '<br>';

            //Lit le reste de la première ligne
            echo 'Deuxième  appel : ' .fgets($ressource). '<br>';

            //Lit la deuxième ligne du fichier
            echo 'Troisième appel : ' .fgets($ressource). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

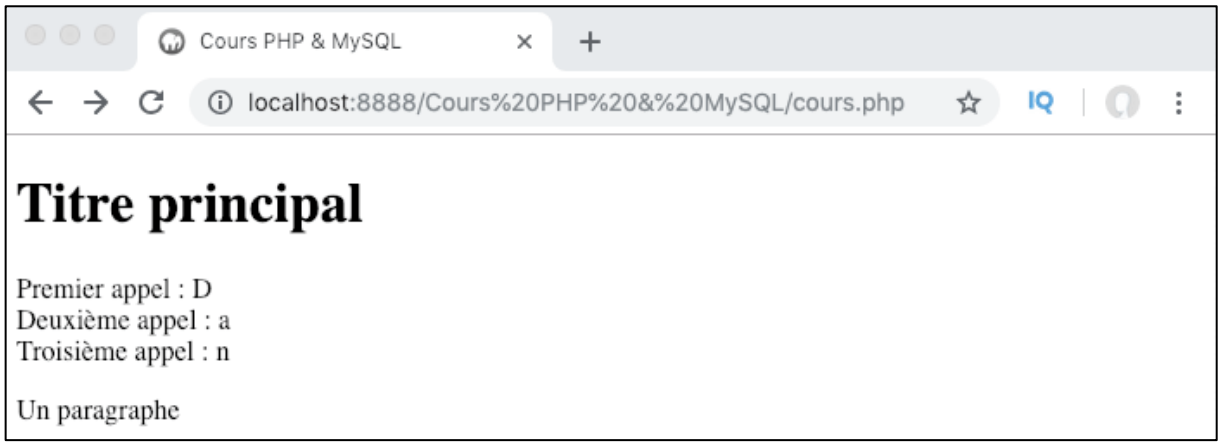


Lire un fichier caractère par caractère avec getc()

Dans certains cas, il va également être intéressant de lire un fichier caractère par caractère notamment pour récupérer un caractère en particulier ou pour arrêter la lecture lorsqu'on arrive à un certain caractère.

Pour faire cela, nous allons cette fois-ci utiliser la fonction fgetc(). Cette fonction va s'utiliser exactement comme fgets(), et chaque nouvel appel à la fonction va nous permettre de lire un nouveau caractère de notre fichier. Notez que les espaces sont bien entendus considérés comme des caractères.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $ressource = fopen('test.txt', 'rb');

            //Lit le premier caractère du fichier
            echo 'Premier appel : ' .fgetc($ressource). '<br>';

            //Lit le deuxième caractère
            echo 'Deuxième  appel : ' .fgetc($ressource). '<br>';

            //Lit le troisième caractère
            echo 'Troisième appel : ' .fgetc($ressource). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Trouver la fin d'un fichier de taille inconnue

Dans ce cours, nous travaillons pour l'instant sur des exemples simples. Ici, par exemple, nous avons nous mêmes créé un fichier pour effectuer différentes opérations dessus.

En pratique, cependant, nous utiliserons généralement les fichiers pour stocker des informations non connues à l'avance. Dans ce cas-là, il est impossible de prévoir à l'avance la taille de ces fichiers et on risque donc de ne pas pouvoir utiliser les fonctions fgets() et fgetc() de manière optimale.

Il existe plusieurs moyens de déterminer la taille ou la fin d'un fichier. La fonction filesize(), par exemple, va lire la taille d'un fichier. Dans le cas présent, cependant, nous cherchons plutôt à déterminer où se situe la fin d'un fichier (ce qui n'est pas forcément équivalent à la taille d'un fichier à cause de la place du curseur, notion que nous allons voir en détail par la suite).

La fonction PHP feof() (« eof » signifie « end of the file » ou « fin du fichier ») va nous permettre de savoir si la fin d'un fichier a été atteinte ou pas. Dès que la fin d'un fichier est atteinte, cette fonction va renvoyer la valeur true. Avant cela, elle renverra la valeur false. On va donc pouvoir utiliser cette fonction pour boucler dans un fichier de taille inconnue.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $res = fopen('test.txt', 'rb');

            /*Tant que la fin du fichier n'est pas atteninte, c'est-à-dire
             *tant que feof() renvoie FALSE (= tant que !feof() renvoie TRUE)
             *on echo une nouvelle ligne du fichier*/
            while(!feof($res)){
                $ligne = fgets($res);
                echo 'La ligne "' .$ligne. '" contient
                ' .strlen($ligne). ' caractères <br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, tant que la fin du fichier n'est pas atteinte, on affiche une nouvelle ligne de notre fichier et on calcule le nombre de caractères de la ligne grâce à la fonction `strlen()`, abréviation de « string length » ou « longueur de la chaine » en français.

Notez qu'ici le fait de retourner à la ligne compte comme un caractère et que la fonctions `fgets()` s'arrête après ce passage à la ligne. C'est la raison pour laquelle lorsqu'on compte le nombre de caractères de nos lignes, on a un caractère de plus que ce à quoi on pouvait s'attendre (sauf pour la dernière).

La place du curseur interne ou pointeur de fichier

La position du curseur (ou « pointeur de fichier ») va impacter le résultat de la plupart des manipulations qu'on va pouvoir effectuer sur les fichiers. Il est donc essentiel de toujours savoir où se situe ce pointeur et également de savoir comment le bouger.

Le curseur ou pointeur est l'endroit dans un fichier à partir duquel une opération va être faite. Pour donner un exemple concret, le curseur dans un document Word, dans un champ de formulaire ou lorsque vous effectuez une recherche Google ou tapez une URL dans votre navigateur correspond à la barre clignotante.

Ce curseur indique l'emplacement à partir duquel vous allez écrire votre requête ou supprimer un caractère, etc. Le curseur dans les fichiers va être exactement la même chose à la différence qu'ici on ne peut pas le voir visuellement.

Le mode d'ouverture choisi va être la première chose qui va influer sur la position du pointeur. En effet, selon le mode choisi, le pointeur de fichier va se situer à une place différente et va pouvoir ou ne va pas pouvoir être déplacé :

Mode utilisé	Position du pointeur de fichier
r / r+	Début du fichier

Mode utilisé	Position du pointeur de fichier
a / a+	Fin du fichier
w / w+	Début du fichier
x / x+	Début du fichier
c / c+	Début du fichier

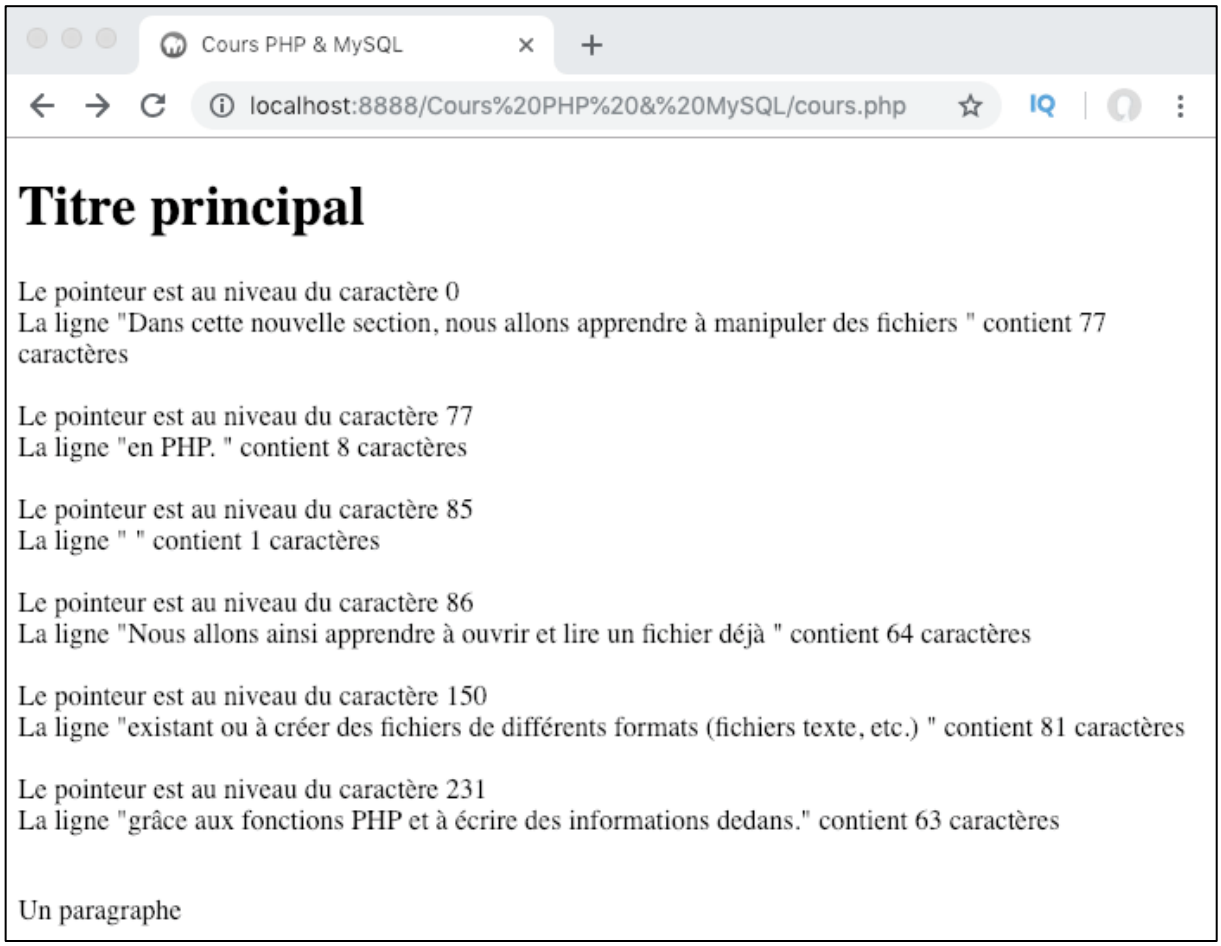
Ensuite, vous devez également savoir que certaines fonctions vont modifier la place du curseur à chaque exécution. Cela va par exemple être le cas des fonctions `fgets()` et `fgetc()` qui servent à lire un fichier ligne par ligne ou caractère par caractère.

En effet, la première fois qu'on appelle `fgets()` par exemple, le pointeur est généralement au début de notre fichier et c'est donc la première ligne de notre fichier est lue par défaut. Cependant, lors du deuxième appel à cette fonction, c'est bien la deuxième ligne de notre fichier qui va être lue.

Ce comportement est justement dû au fait que la fonction `fgets()` déplace le pointeur de fichier du début de la première ligne au début de la seconde ligne dans ce cas précis.

Pour savoir où se situe notre pointeur de fichier, on peut utiliser la fonction `ftell()` qui renvoie la position courante du pointeur. Nous allons devoir lui passer la valeur renvoyée par `fopen()` pour qu'elle fonctionne correctement.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $res = fopen('test.txt', 'rb');

            while(!feof($res)){
              echo 'Le pointeur est au niveau du caractère ' .ftell($res). '<br>';
              $ligne = fgets($res);
              echo 'La ligne "' .$ligne. '" contient
              ' .strlen($ligne). ' caractères <br><br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Déplacer le curseur manuellement

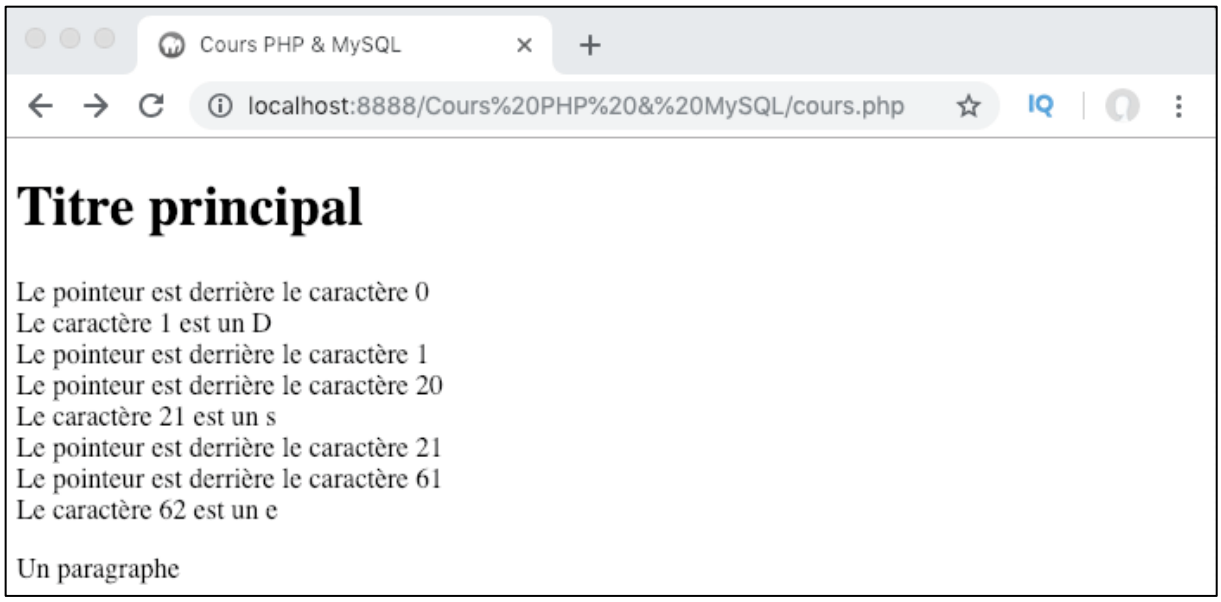
Pour commencer la lecture d'un fichier à partir d'un certain point, ou pour écrire dans un fichier à partir d'un endroit précis ou pour toute autre manipulation de ce type, nous allons avoir besoin de contrôler la position du curseur. Pour cela, nous allons pouvoir utiliser la fonction `fseek()`.

Cette fonction va prendre en arguments l'information renvoyée par `fopen()` ainsi qu'un nombre correspondant à la nouvelle position en octets du pointeur.

La nouvelle position du pointeur sera par défaut calculée par rapport au début du fichier). Pour modifier ce comportement et faire en sorte que le nombre passé s'ajoute à la position courante du curseur, on peut ajouter la constante `SEEK_CUR` en troisième argument de `fseek()`.

Notez cependant que si vous utilisez les modes `a` et `a+` pour ouvrir votre fichier, utiliser la fonction `fseek()` ne produira aucun effet et votre curseur se placera toujours en fin de fichier.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
          $res = fopen('test.txt', 'rb');

          echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
          echo 'Le caractère ' .(ftell($res) + 1). ' est un ' .fgetc($res). '<br>';
          echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
          fseek($res, 20);
          echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
          echo 'Le caractère ' .(ftell($res) + 1). ' est un ' .fgetc($res). '<br>';
          echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
          fseek($res, 40, SEEK_CUR);
          echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
          echo 'Le caractère ' .(ftell($res) + 1). ' est un ' .fgetc($res). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



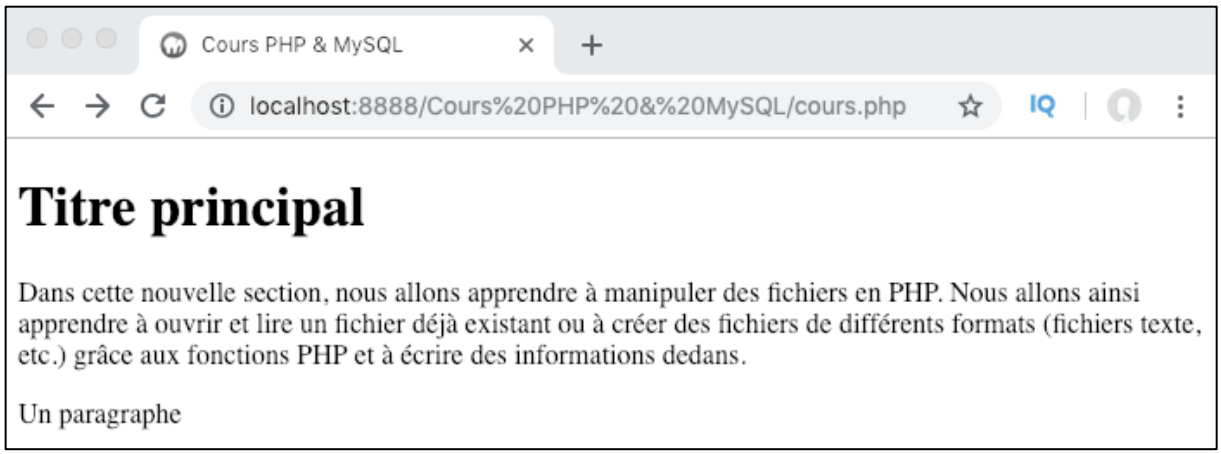
Fermer un fichier

Pour fermer un fichier en PHP, nous utiliserons cette fois la fonction `fclose()`.

On va une nouvelle fois passer le résultat renvoyé par `fopen()` en argument de cette fonction.

Notez que la fermeture d'un fichier n'est pas strictement obligatoire. Cependant, cela est considéré comme une bonne pratique : cela évite d'user inutilement les ressources de votre serveur.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
          $res = fopen('test.txt', 'rb');

          echo fread($res, filesize('test.txt'));
          fclose($res);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```


Cours PHP & MySQL
localhost:8888/Cours%20PHP%20&%20MySQL/cours.php
Titre principal
Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers en PHP. Nous allons ainsi apprendre à ouvrir et lire un fichier déjà existant ou à créer des fichiers de différents formats (fichiers texte, etc.) grâce aux fonctions PHP et à écrire des informations dedans.
Un paragraphe

Créer et écrire dans un fichier en PHP

Dans la leçon précédente, nous avons appris à ouvrir un fichier et avons découvert l'importance du mode d'ouverture qui va conditionner les opérations qu'on va pouvoir effectuer sur le fichier ouvert.

Nous avons également appris à gérer la place du pointeur, ce qui va se révéler essentiel pour écrire dans un fichier en PHP contenant déjà du texte.

Dans cette nouvelle leçon, nous allons apprendre à créer un fichier et à écrire dans un fichier vierge ou contenant déjà du texte.

Créer et écrire dans un fichier en PHP : les méthodes disponibles

Il existe différentes façons de créer un fichier et d'écrire dans un fichier déjà existant ou pas et contenant déjà du texte ou pas en PHP.

Les deux façons les plus simples vont être d'utiliser soit la fonction `file_put_contents()`, soit les fonctions `fopen()` et `fwrite()` ensemble.

Notez déjà que l'utilisation des fonctions `fopen()` et `fwrite()` va nous donner plus de contrôle sur notre écriture, en nous permettant de choisir un mode d'ouverture, d'écrire à un endroit du fichier, etc.

Écrire dans un fichier avec file_put_contents()

La fonction `file_put_contents()` va nous permettre d'écrire simplement des données dans un fichier.

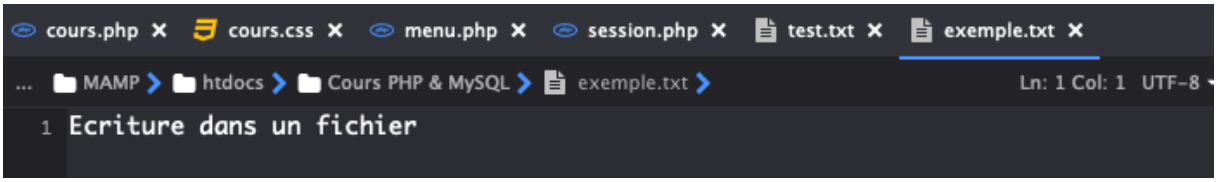
Cette fonction va accepter en argument le chemin vers le fichier dans lequel on doit écrire les données, les données à écrire (qui peuvent être une chaine de caractères ou un tableau) ainsi qu'un drapeau (nous reviendrons là-dessus plus tard).

Si le fichier spécifié dans le chemin du fichier n'existe pas, alors il sera créé. S'il existe, il sera par défaut écrasé, ce qui signifie que ses données seront supprimées.

Vous pouvez déjà noter qu'appeler `file_put_contents()` correspond à appeler successivement les fonctions `fopen()`, `fwrite()` et `fclose()`.

Utilisons immédiatement cette fonction pour écrire dans un premier fichier :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
          file_put_contents('exemple.txt', 'Ecriture dans un fichier');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



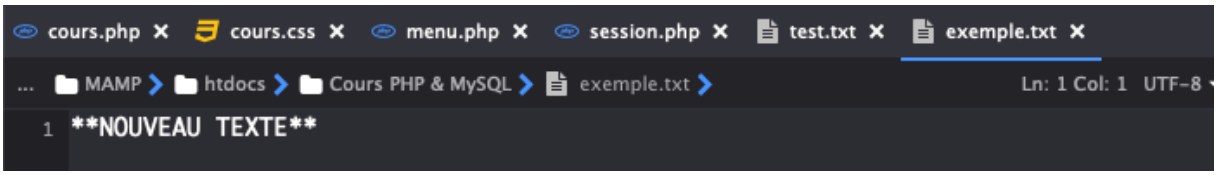
Ici, on passe le chemin de fichier `exemple.txt` à notre fonction `file_put_contents()`. On va donc chercher un fichier qui s'appelle `exemple.txt` et qui se situe dans le même répertoire que notre script.

Comme ce fichier n'existe pas, il va être créé et le texte précisé en deuxième argument de notre fonction va être inséré dedans.

Note : Pensez bien à exécuter votre script en rechargeant l'URL de la page dans votre navigateur à chaque fois pour que les différentes opérations dans le script s'exécutent bien !

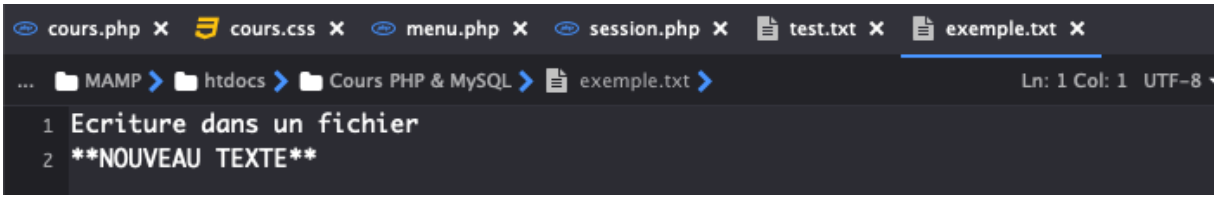
Si on essaie maintenant de répéter l'opération et d'écrire un texte différent dans notre fichier, on s'aperçoit que le nouveau texte remplace l'ancien qui est écrasé. C'est le comportement par défaut :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
          file_put_contents('exemple.txt', 'Ecriture dans un fichier');
          file_put_contents('exemple.txt', '**NOUVEAU TEXTE**');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Pour « ajouter » du texte à notre fichier, une astuce simple consiste ici à récupérer le texte d'origine de notre fichier dans une variable, puis à le concaténer avec le texte qu'on souhaite écrire dans notre fichier, puis à passer le nouveau texte à `file_put_contents()`. L'ancien contenu du fichier sera alors remplacé par le nouveau comme précédemment.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //On écrit un premier texte dans notre fichier
            file_put_contents('exemple.txt', 'Ecriture dans un fichier');

            //On récupère le contenu du fichier
            $texte = file_get_contents('exemple.txt');

            //On ajoute notre nouveau texte à l'ancien
            $texte .= "\n**NOUVEAU TEXTE**";

            //On écrit tout le texte dans notre fichier
            file_put_contents('exemple.txt', $texte);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



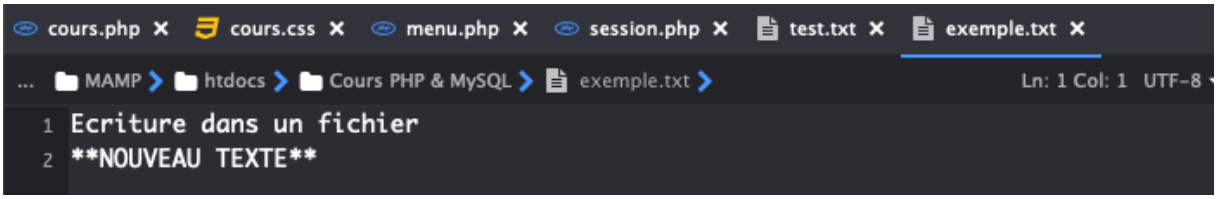
Ici, vous pouvez noter que j'utilise le caractère `\n`. Celui-ci sert à créer un retour à la ligne en PHP. Ici, on l'utilise donc que notre nouveau texte soit inséré dans la ligne suivante de notre fichier. On est obligés d'utiliser des guillemets plutôt que des apostrophes ici pour que le `\n` soit bien interprété comme un retour à la ligne par le PHP.

Cette astuce pour rajouter du texte dans un fichier fonctionne mais nous force finalement à faire plusieurs opérations et à écraser le contenu de base du fichier pour placer le nouveau (qui contient le contenu original).

Pour maintenant véritablement conserver les données de base de notre fichier et lui ajouter de nouvelles données à la suite des données déjà précédentes, on va également plus simplement pouvoir passer le drapeau `FILE_APPEND` en troisième argument de notre fonction `file_put_contents()`.

Un drapeau est une constante qui va correspondre à un nombre. De nombreuses fonctions utilisent des drapeaux différents. Le drapeau ou la constante `FILE_APPEND` va ici nous permettre d'ajouter des données en fin de fichier (c'est-à-dire à la suite du texte déjà existant) plutôt que d'écraser les données déjà existantes.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //On écrit un premier texte dans notre fichier
            file_put_contents('exemple.txt', 'Ecriture dans un fichier');

            //On rajoute du texte dans notre fichier
            file_put_contents('exemple.txt', "\n**NOUVEAU TEXTE**", FILE_APPEND);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on utilise une première fois `file_put_contents()` sans le drapeau `FILE_APPEND` pour écrire « Ecriture dans un fichier » dans notre fichier. Ce texte va donc écraser les données déjà présentes dans notre fichier. Ensuite, on rajoute un autre texte en utilisant notre drapeau.

Créer un fichier PHP avec fopen()

Pour créer un nouveau fichier en PHP (sans forcément écrire dedans), nous allons à nouveau utiliser la fonction `fopen()`. En effet, rappelez-vous que cette fonction va pouvoir créer un fichier si celui-ci n'existe pas à condition qu'on utilise un mode adapté.

Pour créer un fichier avec `fopen()`, nous allons devoir lui passer en arguments un nom de fichier qui sera le nom du fichier créé ainsi qu'un mode d'ouverture adapté. En mentionnant simplement cela, le fichier sera par défaut créé dans le même dossier que notre page PHP.

Essayons immédiatement de créer un fichier qu'on va appeler `exemple2.txt`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $fichier = fopen('exemple2.txt', 'c+b');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, nous créons un nouveau fichier avec le mode `c+`. Le fichier est donc accessible en lecture et en écriture et si celui-ci existait et contenait déjà des informations, celles-ci ne seraient pas effacées à la différence du mode `w+`.

On pense bien également à rajouter l'option `b` (pour binaire) afin de maximiser la compatibilité pour les différents systèmes.

Bien évidemment, afin de véritablement créer le fichier, le code de votre page doit être exécuté. Pensez donc bien toujours à ouvrir votre fichier `.php` dans votre navigateur pour exécuter le script au moins une fois.

Une fois cela fait, vous pouvez aller vérifier dans le dossier contenant votre page PHP qu'un fichier nommé `exemple2.txt` a bien été créé.

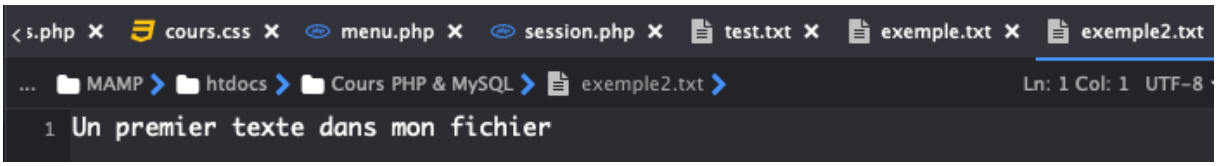
Écrire dans un fichier avec fwrite()

Une fois un fichier ouvert ou créé avec `fopen()`, on va pouvoir écrire dedans en utilisant la fonction `fwrite()`.

Cette fonction va prendre la valeur retournée par `fopen()` ainsi que la chaine de caractères à écrire dans le fichier en arguments.

Si notre fichier est vide, on va très simplement pouvoir écrire du texte dedans :

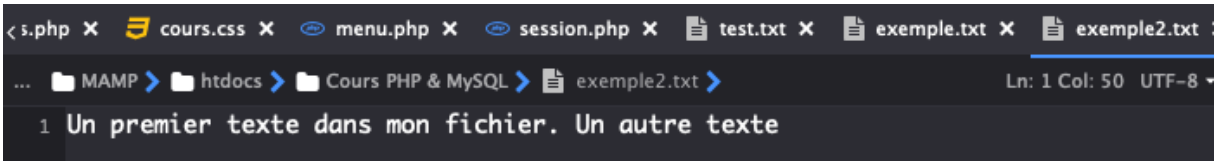
```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $fichier = fopen('exemple2.txt', 'c+b');
            fwrite($fichier, 'Un premier texte dans mon fichier');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Dans le cas où le fichier ouvert contient déjà du texte, cela va être un peu plus complexe. En effet, il va déjà falloir se poser la question d'où se situe notre pointeur.

Si on utilise la fonction `fwrite()` plusieurs fois de suite, le texte va être ajouté à la suite car `fwrite()` va déplacer le pointeur après le texte inséré après chaque utilisation.

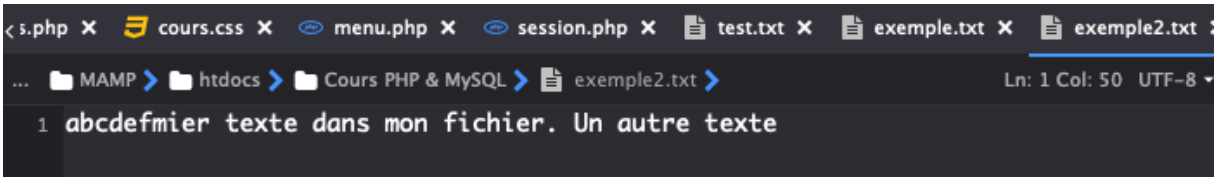
```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $fichier = fopen('exemple2.txt', 'c+b');
            fwrite($fichier, 'Un premier texte dans mon fichier');
            fwrite($fichier, '. Un autre texte');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Le problème va si situer lors de la première utilisation de fwrite() dans un fichier qui contient déjà du texte. En effet, la plupart des modes de fopen() vont placer le curseur en début de fichier. Les informations vont donc être écrites par-dessus les anciennes.

Regardez plutôt l'exemple suivant.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*Notre fichier contient toujours le texte :
             *"Un premier texte dans mon fichier. Un autre texte"
             *ajouté précédemment*/
            $fichier = fopen('exemple2.txt', 'c+b');
            fwrite($fichier, 'abc');
            fwrite($fichier, 'def');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



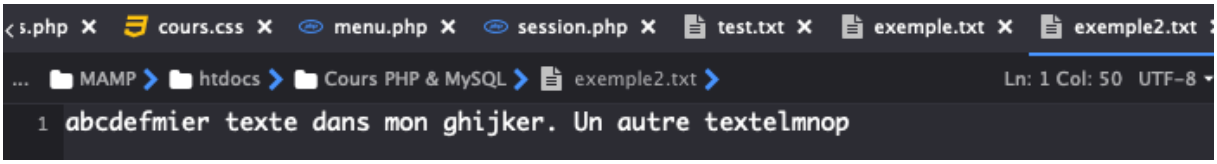
Ici, notre fichier contient le texte « Un premier texte dans mon fichier. Un autre texte » lors de son ouverture. Lors du premier appel à fwrite(), le curseur se situe au début du fichier. Les informations « abc » vont alors être écrites par-dessus celles déjà présentes.

Après sa première utilisation, fwrite() déplace le curseur à la fin du texte ajouté, c'est-à-dire juste derrière le caractère « c ». Si on appelle fwrite() à nouveau immédiatement après, les nouvelles informations vont être insérées après le « c ».

On va ici pouvoir utiliser la fonction fseek() pour modifier la position du pointeur et écrire à partir d'un autre endroit dans le fichier. En faisant cela, le nouveau texte sera écrit à partir

d’un certain point dans le fichier. Si on tente d’écrire du texte au milieu du fichier, cependant, les données déjà présentes à cet endroit continueront d’être écrasées.

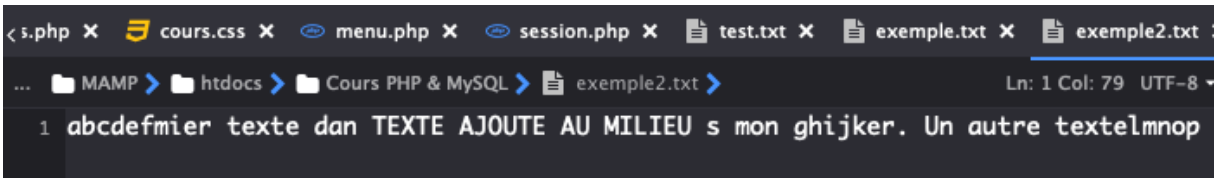
```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*Notre fichier contient toujours le texte :
             *"abcdefmier texte dans mon fichier. Un autre texte"
             *ajouté précédemment*/
            $fichier = fopen('exemple2.txt', 'c+b');
            fwrite($fichier, 'abc');
            fwrite($fichier, 'def');

            //On déplace le curseur de 20 octets
            fseek($fichier, 20, SEEK_CUR);
            fwrite($fichier, 'ghijk');

            //On place le curseur en fin de fichier
            fseek($fichier, filesize('exemple2.txt'));
            fwrite($fichier, 'lmnop');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Si on souhaite rajouter du texte au milieu d’un fichier tout en conservant le texte précédent, nous allons devoir procéder en plusieurs étapes.

L’idée va être ici de récupérer la première partie du texte de notre fichier jusqu’au point d’insertion du nouveau contenu, puis de concaténer le nouveau contenu à ce texte, puis de de récupérer la deuxième partie du texte de notre fichier et de la concaténer au reste avant de finalement écrire le tout dans notre fichier.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*On appelle fopen() avec le mode c : le pointeur se situe
             *au début du fichier. Le fichier contient le texte :
             *"abcdefmier texte dans mon ghijker. Un autre textelmnop"*/
            $fichier = fopen('exemple2.txt', 'c+b');

            /*On lit les 20 premiers octets du fichier avec fread(), le
             *pointeur se situe là où fread() arrête sa lecture*/
            $texte = fread($fichier, 20);

            //On ajoute du texte dans notre variable
            $texte .= ' TEXTE AJOUTE AU MILIEU ';

            /*On lit la suite du fichier (fread() reprend sa lecture là où se
             *trouve le pointeur) et on ajoute le texte lu dans $texte*/
            $texte .= fread($fichier, filesize('exemple2.txt'));

            /*On replace le pointeur en début de fichier et on écrase l'ancien
             *texte avec le nouveau (qui est plus long)*/
            fseek($fichier, 0);
            fwrite($fichier, $texte);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



La chose importante dans le script ci-dessus est de bien toujours suivre le pointeur, qui va être déplacé par certaines de nos fonctions liées aux fichiers.

Autres opérations sur les fichiers en PHP

Dans les leçons précédentes, nous avons découvert comment manipuler les fichiers et notamment comme les ouvrir, les lire et écrire dans nos fichiers en PHP.

Il existe d'autres types de manipulations moins courantes sur les fichiers comme le renommage de fichier ou la suppression que nous allons rapidement voir dans cette nouvelle leçon.

Nous allons également dans cette leçon discuter des niveaux de permission des fichiers, un concept qu'il va être essentiel de comprendre et de maitriser pour travailler avec des fichiers sur un « vrai » site hébergé sur serveur distant.

Tester l'existence d'un fichier

Le PHP met à notre disposition deux fonctions qui vont nous permettre de vérifier si un fichier existe et si un fichier est un véritable fichier.

La fonction `file_exists()` vérifie si un fichier ou si un dossier existe. On va devoir lui passer le chemin du fichier ou du dossier dont on souhaite vérifier l'existence en argument. Si le fichier ou le dossier existe bien, la fonction renverra le booléen `true`. Dans le cas contraire, elle renverra `false`.

La fonction `is_file()` indique si le fichier est un véritable fichier (et non pas un répertoire par exemple). Nous allons également devoir lui passer le chemin du fichier supposé en argument. Si le fichier existe et que c'est bien un fichier régulier, alors `is_file()` renverra le booléen `true`. Dans le cas contraire, elle renverra `false`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //Nous avons créé le fichier exemple.txt précédemment
            $f = 'exemple.txt';
            $d = 'jenexistepas.txt';

            if(file_exists($f)){
                if(is_file($f)){
                    echo 'Le fichier ' .$f. ' existe et est bien un fichier';
                }
                else{
                    echo $f. ' existe mais n\'est pas un fichier régulier';
                }
            }else{
                echo $f. ' n\'existe pas';
            }
            echo '<br><br>';
            if(file_exists($d)){
                if(is_file($d)){
                    echo 'Le fichier ' .$d. ' existe et est bien un fichier';
                }
                else{
                    echo $d. ' existe mais n\'est pas un fichier régulier';
                }
            }else{
                echo $d. ' n\'existe pas ';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Renommer un fichier

La fonction `rename()` permet de renommer un fichier ou un dossier. On va devoir lui passer le nom d'origine du fichier ou du dossier et le nouveau nom en arguments.

Dans le cas où le nouveau nom choisi est le nom d'un fichier qui existe déjà, alors ce fichier sera écrasé et remplacé par notre fichier renommé.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //Nous avons créé le fichier exemple.txt précédemment
            $f = 'exemple.txt';
            $d = 'jenexistepas.txt';

            if(file_exists($f)){
                if(is_file($f)){
                    $newf = 'fichier.txt';
                    rename($f, $newf);
                    echo 'Le fichier ' .$f. ' a été renommé en ' .$newf;
                }
                else{
                    echo $f. ' existe mais n\'est pas un fichier régulier';
                }
            }else{
                echo $f. ' n\'existe pas';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on commence par vérifier que notre fichier existe et est bien un fichier régulier. Si c'est le cas, on le renomme en utilisant la fonction `rename()`.

Effacer un fichier

La fonction unlink() permet d'effacer un fichier. On va lui passer le chemin du fichier à effacer en argument. Cette fonction va retourner true si le fichier a bien été effacé ou false en cas d'erreur.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $f = 'exemple2.txt';

            /*On fait deux opérations en une : on exécute unlink() et on effectue
             *notre test sur la valeur renvoyée. Si la fonctoiin renvoie true,
             *on indique que le fichier a bien été effacé*/
            if(unlink($f)){
                echo 'Le fichier ' .$f. ' a bien été effacé';
            }else{
                echo 'Le fichier ' .$f. ' n\'a pas pu être effacé';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Introduction aux permissions des fichiers et au Chmod

Le sujet des permissions liés aux fichiers (et aux dossiers) est un sujet très vaste et complexe et nous n'allons ici que l'aborder en surface et dans les grandes lignes afin que vous puissiez comprendre la relation avec les fichiers en PHP.

Le système Linux (utilisés par la plupart des hébergeurs) définit différents types d'utilisateurs pouvant interagir avec les fichiers (et les dossiers) :

- Le propriétaire ;
- Les membres du groupe ;
- Les autres utilisateurs.

La façon dont est créé le système Linux permet à plusieurs utilisateurs d'avoir accès au système en même temps et va nous permettre de définir des groupes d'utilisateurs. Cependant, pour que le système fonctionne toujours bien, il a fallu définir différents niveaux de permission d'accès aux différents fichiers pour les différents utilisateurs.

Pour pouvoir manipuler des fichiers (ou le contenu de dossiers), c'est-à-dire effectuer des opérations dessus, nous allons donc avant tout avoir besoin de permissions. Différentes opérations (lecture du fichier, écriture, etc.) vont pouvoir nécessiter différents niveaux de permission.

Lorsqu'on travaille en local et sur nos propres fichiers on ne doit normalement pas avoir de problème de permissions puisque par défaut le système attribue généralement les permissions maximales au propriétaire du fichier.

Cependant, cela se complique lorsqu'on héberge notre site sur serveur distant et qu'on doit donner un accès aux différents utilisateurs à certains fichiers et c'est là qu'il faut bien faire attention aux différentes permissions accordées.

Les permissions d'accès accordées pour chaque groupe d'utilisateurs à un fichier (ou à un dossier) sont symbolisées par 3 chiffres allant de 0 à 7 ou par 3 lettres. Le premier caractère indique les droits accordés au propriétaire du fichier, le deuxième caractère indique les droits accordés au groupe et le troisième caractère indique les droits accordés aux autres utilisateurs.

Voici les permissions liées à chaque caractère :

Droit	**Valeur alphanumérique**	**Valeur octale**
aucun droit	`---`	0
exécution seulement	`--x`	1
écriture seulement	`-w-`	2
écriture et exécution	`-wx`	3
lecture seulement	`r--`	4
lecture et exécution	`r-x`	5
lecture et écriture	`rw-`	6
tous les droits (lecture, écriture et exécution)	`rwx`	7

Si un fichier ou un dossier possède les permissions `754` par exemple, cela signifie qu'on accorde tous les droits (lecture, écriture et exécution) sur les fichiers contenus dans ce dossier à l'auteur, des droits de lecture et d'exécution aux membres du groupe et des droits de lecture uniquement aux autres utilisateurs.

Vérifier et modifier les permissions d'un fichier

Pour vérifier les permissions d'un fichier d'un fichier ou d'un dossier manuellement, on va déjà pouvoir tout simplement effectuer un clic droit dessus et afficher les informations liées à celui-ci.

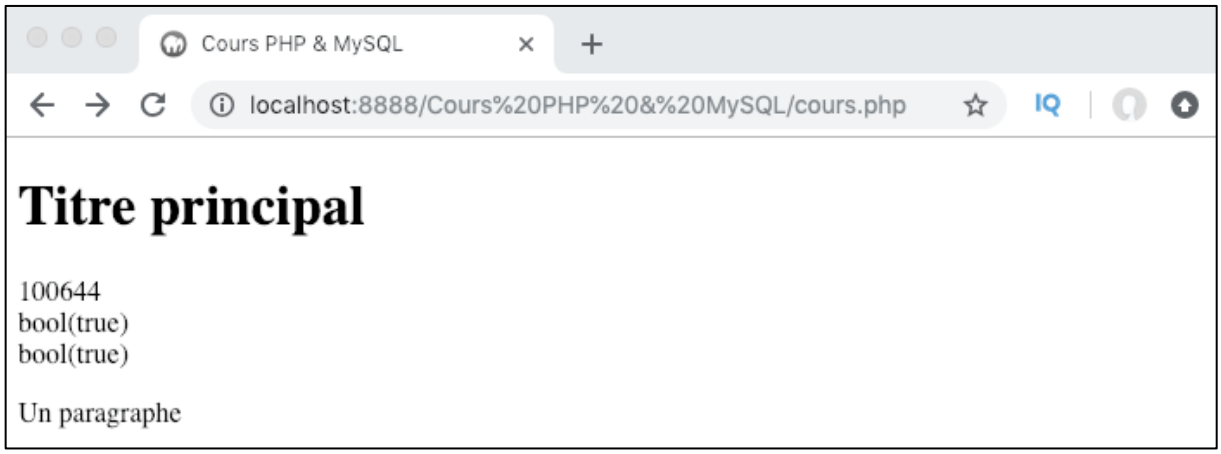
Pour vérifier les permissions d'un fichier via un script PHP, c'est-à-dire dynamiquement, on va pouvoir utiliser la fonction PHP `fileperms()` qui renvoie les permissions pour un fichier.

Cette fonction va renvoyer les permissions d'un fichier sous forme numérique. On pourra ensuite convertir le résultat sous forme octale avec la fonction `decoct()` pour obtenir la représentation des permissions selon nos trois chiffres.

Notez que `fileperms()` peut renvoyer des informations supplémentaires selon le système utilisé. Les trois derniers chiffres renvoyés correspondent aux permissions.

On va également pouvoir utiliser les fonctions `is_readable()` et `is_writable()` qui vont respectivement déterminer si le fichier peut être lu et si on peut écrire dedans. Ces fonctions vont renvoyer `true` si c'est le cas ou `false` dans le cas contraire ce qui les rend très pratiques d'utilisation au sein d'une condition.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*On teste les permissions de notre
             *fichier fichier.txt créé précédemment*/

            echo decoct(fileperms('fichier.txt')). '<br>';
            echo var_dump(is_readable('fichier.txt')). '<br>';
            echo var_dump(is_writable('fichier.txt')). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Pour modifier les permissions d'un fichier en PHP, on va devoir utiliser la fonction `chmod()` qui va prendre en arguments le fichier dont on souhaite modifier les permissions ainsi que les nouvelles permissions du fichier en notation octale (on placera un zéro devant nos trois chiffres).

Le réglage des permissions concernant les membres du groupe et les autres utilisateurs va être particulier à chaque cas selon votre site et la sensibilité des informations stockées : vos utilisateurs doivent ils pouvoir lire les fichiers ? Les exécuter ? Les modifier ?

Notez que la fonction `chmod()` tire son nom de chmod ou « change mode » qui est une commande Unix permettant de changer les permissions d'accès d'un fichier ou d'un répertoire.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo 'Permissions d\'origine : '
            .decoct(fileperms('fichier.txt')). '<br>';

            //chmod() renvoie truc en cas de succès
            if(chmod('fichier.txt', 0755)){
                echo 'Permissions du fichier bien modifiées';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on change les permissions de notre fichier « fichier.txt » qu'on règle sur `755` grâce à la fonction `chmod()`. On tire parti du fait que `chmod()` renvoie `true` en cas de succès (permissions bien modifiées) ou `false` en cas d'échec pour l'utiliser au sein d'une condition.

PARTIE IX

Expressions rationnelles

Introduction aux expressions rationnelles ou expressions régulières

Dans cette nouvelle partie, nous allons nous intéresser aux expressions régulières qu'on appelle également expressions rationnelles.

Avant tout, vous devez bien comprendre que les expressions régulières ne font pas partie du langage PHP en soi mais que PHP a intégré un support pour les expressions régulières dans son langage car ces dernières vont s'avérer très pratiques, notamment pour vérifier la conformité formelle des données envoyées par des utilisateurs via des formulaires.

Présentation des expressions régulières

Une expression régulière (aussi abrégé en « regex ») est une séquence de caractères qu'on va définir et qui va nous servir de schéma de recherche.

Les expressions régulières, en les utilisant de concert avec certains fonctions PHP, vont nous permettre de vérifier la présence de certains caractères dans une chaine de caractères en évaluant la chaine de caractères selon l'expression régulière passée.

Nous allons très souvent utiliser les expressions régulières pour filtrer et vérifier la validité des données envoyées par les utilisateurs via des formulaires par exemple.

Notez que les expressions régulières n'appartiennent pas au PHP mais constituent un langage en soi.

Cependant, le PHP supporte et reconnait les expressions régulières et nous fournit des fonctions qui vont nous permettre d'exploiter toute la puissance de celles-ci.

Regex POSIX contre regex PCRE

Il existe deux types d'expressions régulières possédant des syntaxes et des possibilités légèrement différentes : les expressions régulières POSIX et PCRE.

L'acronyme POSIX signifie « Portable Operating System Interface for Unix ».

L'acronyme PCRE signifie lui Perl Compatible Regular Expression.

Ces deux types de regex vont posséder des syntaxes différentes, mais cela va nous importer peu puisque depuis la version 5.3 du PHP l'extension correspondant aux regex POSIX a été rendue obsolète.

Nous allons donc utiliser les PCRE, qui sont un type de regex dont la syntaxe est tirée du langage Perl.

Création de premières expressions régulières

Les expressions régulières vont être formées d'un assemblage de caractères qui vont former ensemble un schéma de recherche ainsi que de délimiteurs. L'ensemble « schéma de recherche + délimiteurs » est également appelé « masque ».

Les caractères vont pouvoir être des caractères simples ou des caractères spéciaux qui vont avoir une signification particulière.

Un délimiteur peut être n'importe quel caractère, tant qu'il n'est pas alphanumérique, un caractère blanc, l'antislash (« \ ») ou le caractère nul. De plus, si le délimiteur choisi est réutilisé dans notre expression régulière, alors il faudra échapper ou « protéger » le caractère dans la regex en le précédant d'un antislash. Pour le moment, je vous conseille d'utiliser le caractère slash (« / ») comme délimiteur.

En PHP, nous enfermerons généralement nos regex dans des variables pour pouvoir les manipuler facilement.

Commençons par créer une première expression régulière ensemble afin de voir en pratique à quoi ça ressemble.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque = '/pierre/';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, notre regex contient le schéma de recherche pierre et nous avons utilisé, comme convenu, des slashs pour entourer ce schéma de recherche. Ce schéma de recherche va nous permettre de rechercher la présence de la séquence « pierre » dans une chaine de caractères.

En soi, ici, notre regex ne nous sert pas à grand-chose. Cependant, nous allons ensuite pouvoir utiliser des fonctions PHP pour par exemple valider la présence de notre schéma de recherche dans une chaîne de caractères.

Le grand intérêt des expressions régulières est qu'elles vont nous permettre d'effectuer des recherches très puissantes.

En effet, dans le langage des expressions régulières, beaucoup de caractères possèdent un sens spécial, ce qui va nous permettre d'effectuer des recherches très précises.

Par exemple, les regex PCRE possèdent ce qu'on appelle des « options ». Ces options vont nous permettre d'ajouter des critères supplémentaires à nos recherches et vont être représentées par des lettres.

La lettre i, par exemple, va nous permettre de rendre notre regex insensible à la casse, ce qui signifie que notre regex ne fera pas de distinction entre majuscules et minuscules (on peut donc en déduire que les regex sont sensibles à la casse par défaut).

Les options doivent être placées en fin de regex, après le délimiteur, comme ceci :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque = '/pierre/i';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Dans les chapitres qui vont suivre, nous allons créer des regex de plus en plus complexes et découvrir les fonctions PHP nous permettant d'exploiter toute la puissance des expressions régulières.

Les fonctions PCRE PHP

Dans cette nouvelle leçon, nous allons découvrir les différentes fonctions internes au PHP qui vont nous permettre d'exploiter toute la puissance des expressions régulières. Cela devrait rendre l'intérêt des expressions régulières beaucoup plus concret pour vous.

Référence : liste des fonctions PCRE PHP

Le PHP dispose de fonctions internes qui vont nous permettre d'utiliser nos masques pour par exemple rechercher une expression dans une chaine, la remplacer par une autre, etc. Les fonctions PHP relatives aux regex commencent toutes par `preg_`).

Voici la liste de ces fonctions, que nous allons étudier par la suite, ainsi qu'une courte description de leur action.

Fonction	Description
`preg_filter()`	Recherche et remplace
`preg_grep()`	Recherche et retourne un tableau avec les résultats
`preg_last_error()`	Retourne le code d'erreur de la dernière regex exécutée
`preg_match()`	Compare une regex à une chaine de caractères
`preg_match_all()`	Compare une regex à une chaine de caractères et renvoie tous les résultats
`preg_quote()`	Échappe les caractères spéciaux dans une chaine
`preg_replace()`	Recherche et remplace
`preg_replace_callback()`	Recherche et remplace en utilisant une fonction de rappel
`preg_replace_callback_array()`	Recherche et remplace en utilisant une fonction de rappel
`preg_split()`	Découpe une chaine

Dans cette partie, nous allons déjà voir comment utiliser chacune de ces fonctions. Cela vous permettra ainsi de voir immédiatement l'utilité des regex. Nous approfondirons par la suite le sujet des expressions régulières en soi.

Les fonctions PHP preg_match() et preg_match_all()

Les fonctions PHP `preg_match()` et `preg_match_all()` vont être les fonctions qu'on va le plus utiliser avec nos expressions régulières.

Ces deux fonctions vont nous permettre de rechercher un schéma dans une chaine de caractères. Elles vont donc nous permettre de vérifier la présence d'une certaine séquence de caractères dans une chaine de caractères.

La fonction `preg_match()` va renvoyer la valeur 1 si le schéma recherché est trouvé dans la chaine de caractères ou 0 dans le cas contraire.

La fonction `preg_match_all()` va renvoyer le nombre total de fois où le schéma de recherche a été trouvé dans la chaîne de caractères sous forme de tableau.

Chacune de ces deux fonctions va pouvoir accepter jusqu'à 5 arguments mais seuls 2 arguments sont obligatoires à leur fonctionnement. Ces deux arguments sont le masque ou schéma de recherche passé sous forme de chaine de caractères ainsi que la chaine de caractères dans laquelle effectuer la recherche.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque = '/r/';
            $chaine = 'Je suis Pierre Giraud';

            if(preg_match($masque, $chaine)){
                echo 'Le caractère "r" a été trouvé '
                .preg_match_all($masque, $chaine).
                ' fois dans "' .$chaine. '"<br>';
            }else{
                'Aucun "r" dans "' .$chaine. '"<br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Dans l'exemple précédent, nous utilisons le schéma de recherche `/r/` puis on demande à `preg_match()` de rechercher le présence de notre schéma de recherche `r` dans la chaine de caractères « Je suis Pierre Giraud ».

Si `preg_match()` trouve effectivement un `r`, elle renvoie 1 (qui va être évalué à `true`) et on rentre dans le `if`. Dans le cas contraire, notre fonction renvoie 0 (qui est évalué à `false`, rappelons-le) et on entre dans le `else`.

Au sein de contre condition `if`, on utilise également la fonction `preg_match_all()` pour compter le nombre de fois que le schéma `r` est rencontré dans notre chaine de caractères. On va ensuite pouvoir passer d'autres arguments à nos fonctions `preg_match()` et `preg_match_all()` qui vont nous permettre d'effectuer des recherches plus ciblées ou d'obtenir des informations supplémentaires par rapport à notre recherche.

Le premier argument facultatif qu'on va pouvoir passer à ces deux fonctions va être une variable dans laquelle vont être stockés les résultats de la recherche sous forme d'un tableau.

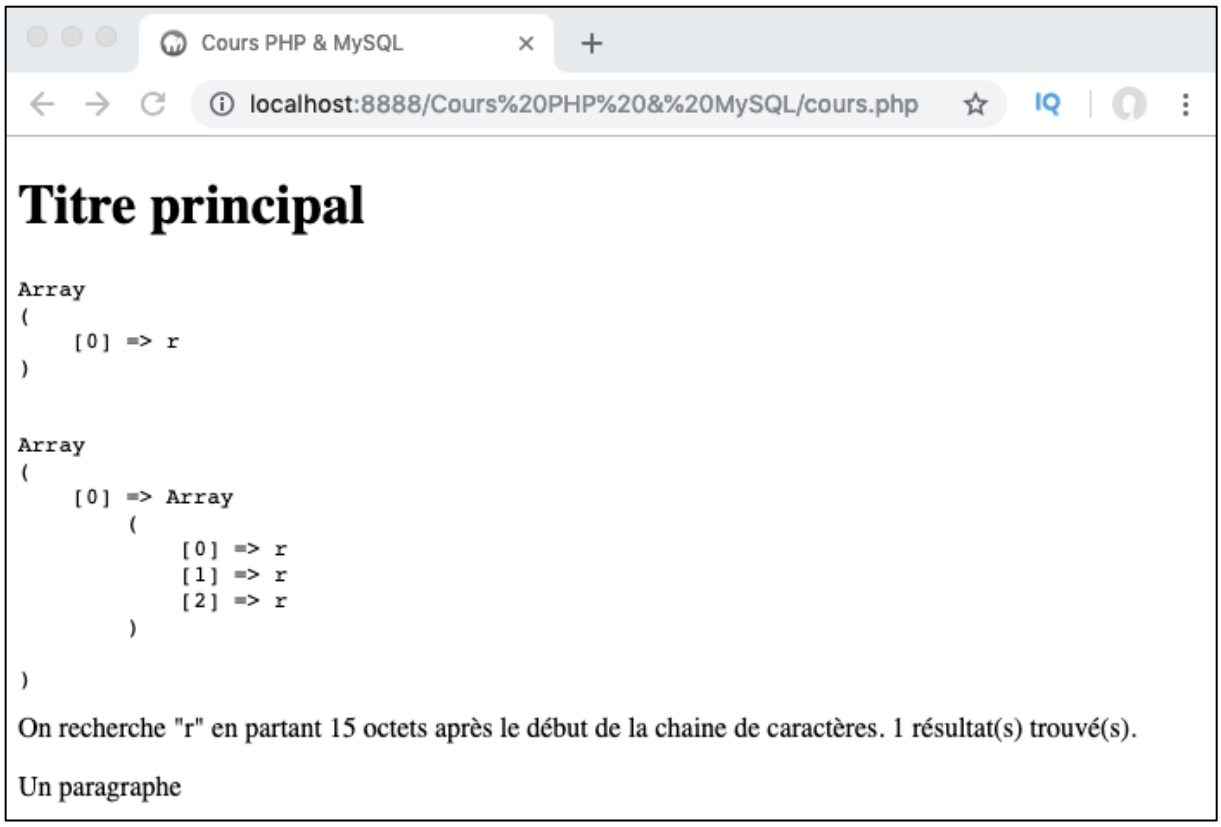
Dans le cas de `preg_match()`, le tableau sera un tableau numéroté. La première valeur du tableau sera le texte qui satisfait le masque complet, la deuxième valeur sera le texte qui satisfait la première parenthèse capturante de notre masque et etc. Nous allons voir plus tard comment utiliser des parenthèses dans nos regex.

Dans le cas de `preg_match_all()`, le tableau sera un tableau multidimensionnel ordonné. Par défaut, notre tableau multidimensionnel principal va contenir en première valeur un tableau qui va lui-même contenir les résultats qui satisfont le masque complet, puis va contenir en deuxième valeur un tableau qui contient les résultats qui satisfont la première parenthèse capturante et etc. Une nouvelle fois, nous illustrerons cela lorsque nous aurons une plus grande connaissance des regex.

Le deuxième argument facultatif de nos fonctions va être un drapeau (une constante) qui va nous permettre de modifier la façon dont notre tableau passé en argument précédent va être créé. Nous ne rentrerons pas dans un tel niveau de précision ici.

Finalement, le dernier argument facultatif permet de préciser à partir de quel endroit dans la chaine de caractères passé la recherche doit commencer. Cet argument est très pratique pour n'effectuer une recherche que sur une partie de chaine. On va ici passer une valeur en octets.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque = '/r/';
            $chaine = 'Je suis Pierre Giraud';
            $match = [];//On ititilise $match et $match_all (non obligatoire)
            $match_all = [];

            preg_match($masque, $chaine, $match);
            preg_match_all($masque, $chaine, $match_all);

            echo '<pre>';
            print_r($match);
            echo '<br><br>';
            print_r($match_all);
            echo '</pre>';

            $m2 = [];//On initialise $m2 (non obligatoire)
            $res = preg_match_all($masque, $chaine, $m2, PREG_PATTERN_ORDER, 15);
            echo 'On recherche "' .$match_all[0][0]. '" en partant 15 octets
            après le début de la chaine de caractères. ' .$res. ' résultat(s)
            trouvé(s).';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, j'utilise le drapeau `PREG_PATTERN_ORDER` qui est la constante utilisée par défaut pour `preg_match_all()` (et qui importe donc peu) et on demande à `preg_match_all()` de rechercher notre schéma de recherche à partir du 15è octet de notre chaine de caractères.

Pour le moment, nos recherches sont très simples et ont donc peu d'intérêt. Cependant, nous allons ensuite apprendre à créer des schémas de recherche complexes qui vont nous permettre de valider des formats de données : on va par exemple pouvoir vérifier qu'une chaine donnée a bien la forme d'une adresse mail ou d'une URL ou encore d'un numéro de téléphone par exemple.

Les fonctions PHP preg_filter(), preg_replace(), preg_replace_callback() et preg_replace_callback_array()

La fonction `preg_filter()` va nous permettre d'effectuer une recherche dans une chaine de caractères selon un schéma de recherche et de remplacer les correspondances par une autre chaine.

On va passer trois arguments à cette fonction : un schéma de recherche, une chaine de remplacement et la chaine dans laquelle faire la recherche.

La fonction `preg_filter()` va ensuite renvoyer la chaine transformée. Attention : la chaine de départ ne sera pas modifiée.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque = '/jour/';
            $chaine = 'Bonjour, je suis Pierre';

            $res = preg_filter($masque, 'soir', $chaine);
            echo 'Chaine transformée : ' .$res. '<br>
                  Chaine de départ : ' .$chaine;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on se sert de `preg_filter()` pour rechercher le schéma `jour` dans notre chaine de caractères. Dans le cas où il est trouvé, on le remplace par la chaine `soir` dans le résultat retourné.

Encore une fois, notre chaine de départ n'est pas modifiée en soi. On peut le voir lorsqu'on `echo` le contenu de notre variable `$filter_res` après avoir utilisé `preg_filter()`.

La fonction `preg_replace()` va fonctionner exactement comme `preg_filter()`. La seule différence entre ces deux fonctions va être dans la valeur retournée si le schéma de recherche n'est pas trouvé dans la chaine de caractères.

En effet, dans ce cas-là, la fonction `preg_filter()` va renvoyer la valeur `null` (correspondant à l'absence de valeur) tandis que `preg_replace()` va renvoyer la chaine de caractères de départ.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque = '/jour/';
            $masque2 = '/azerty/';
            $chaine = 'Bonjour, je suis Pierre';

            $filter_res = preg_filter($masque, 'soir', $chaine);
            echo 'Chaine transformée : ' .$filter_res. '<br>
                  Chaine de départ : ' .$chaine. '<br><br>';

            $replace_res = preg_filter($masque, 'ne année', $chaine);
            echo 'Chaine transformée : ' .$replace_res. '<br>
                  Chaine de départ : ' .$chaine. '<br><br>';

            //Essayons avec un schéma de recherche qui ne sera pas trouvé
            $filter_res2 = preg_filter($masque2, 'soir', $chaine);
            $replace_res2 = preg_replace($masque2, 'soir', $chaine);
            echo 'Résultat de preg_filter() : ' .$filter_res2. '<br>
                  Résultat de preg_replace() : ' .$replace_res2;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Finalement, les fonctions `preg_replace_callback()` et `preg_replace_callback_array()` vont fonctionner selon le même principe général que `preg_replace()` à la différence qu'il faudra préciser une fonction de rappel plutôt qu'une valeur de remplacement.

Ce sujet est un peu complexe à votre niveau et je ne veux pas vous embrouiller davantage pour le moment, nous laisserons donc cette fonction de côté pour l'instant.

La fonction PHP preg_grep()

La fonction `preg_grep()` va nous permettre de rechercher un certain schéma dans un tableau. Les résultats trouvés (les correspondances) seront renvoyés dans un nouveau tableau en conservant les indices du premier tableau.

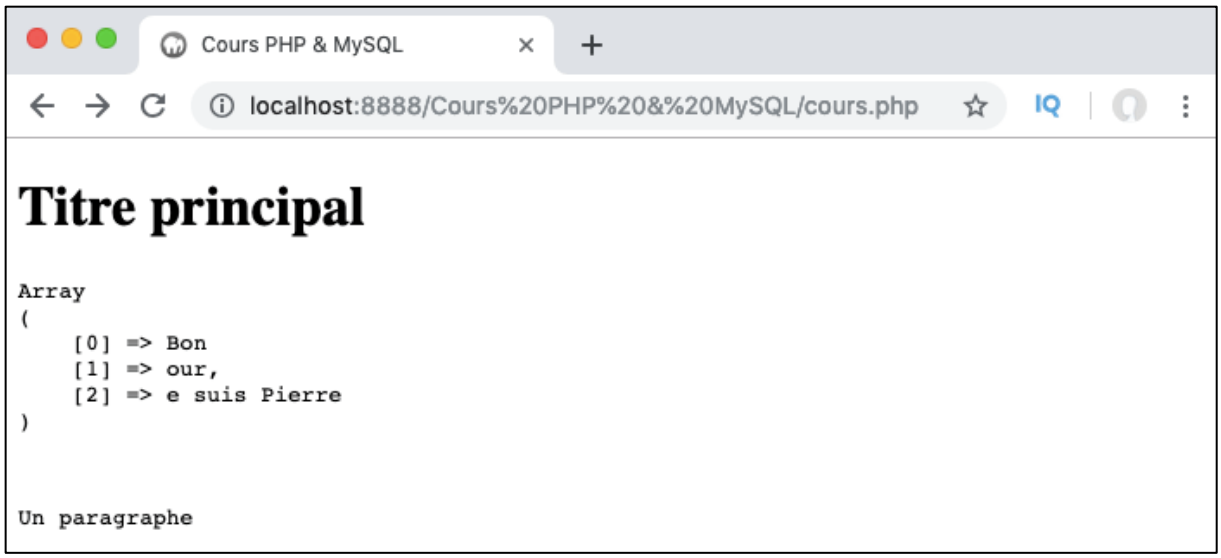
```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque = '/Pierre/';
            $tb = ['Pierre Gr', 'Mathilde Ml', 'Pierre Dp', 'Florian Dc'];

            $grep_res = preg_grep($masque, $tb);

            echo '<pre>';
            print_r($grep_res);
            echo '</pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

La fonction PHP preg_split()

La fonction `preg_split()` va éclater une chaine de caractères en fonction d'un schéma de recherche et renvoyer un tableau.

A chaque fois que le schéma de recherche est trouvé dans la chaine de départ, `preg_split()` crée un nouvel élément dans le tableau renvoyé.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque = '/j/';
            $chaine = 'Bonjour, je suis Pierre';

            $split_res = preg_split($masque, $chaine);
            echo '<pre>';
            print_r($split_res);
            echo '<pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



La fonction PHP preg_quote()

La fonction `preg_quote()` va nous permettre d'échapper certains caractères spéciaux pour les regex.

Utiliser `preg_quote()` correspond à placer un antislash (le caractère d'échappement ou de protection) devant chaque caractère spécial.

Cette fonction peut s'avérer utile lorsque notre schéma de recherche possède beaucoup de caractères spéciaux dont on veut échapper le sens.

Nous reparlerons des caractères spéciaux et de l'échappement des caractères plus tard dans cette partie.

La fonction PHP preg_last_error()

La fonction `preg_last_error()` va être surtout utilisée pour du débogage.

En effet, celle-ci va retourner le code d'erreur correspondant à la dernière regex utilisée. On pourra donc utiliser cette fonction lorsqu'une de nos regex ne fonctionne pas, afin d'avoir plus d'informations sur la nature du problème.

Les classes de caractères des regex

Dans cette nouvelle leçon, nous allons découvrir les classes de caractères et commencer à créer des masques relativement complexes et intéressants pour nos expressions régulières.

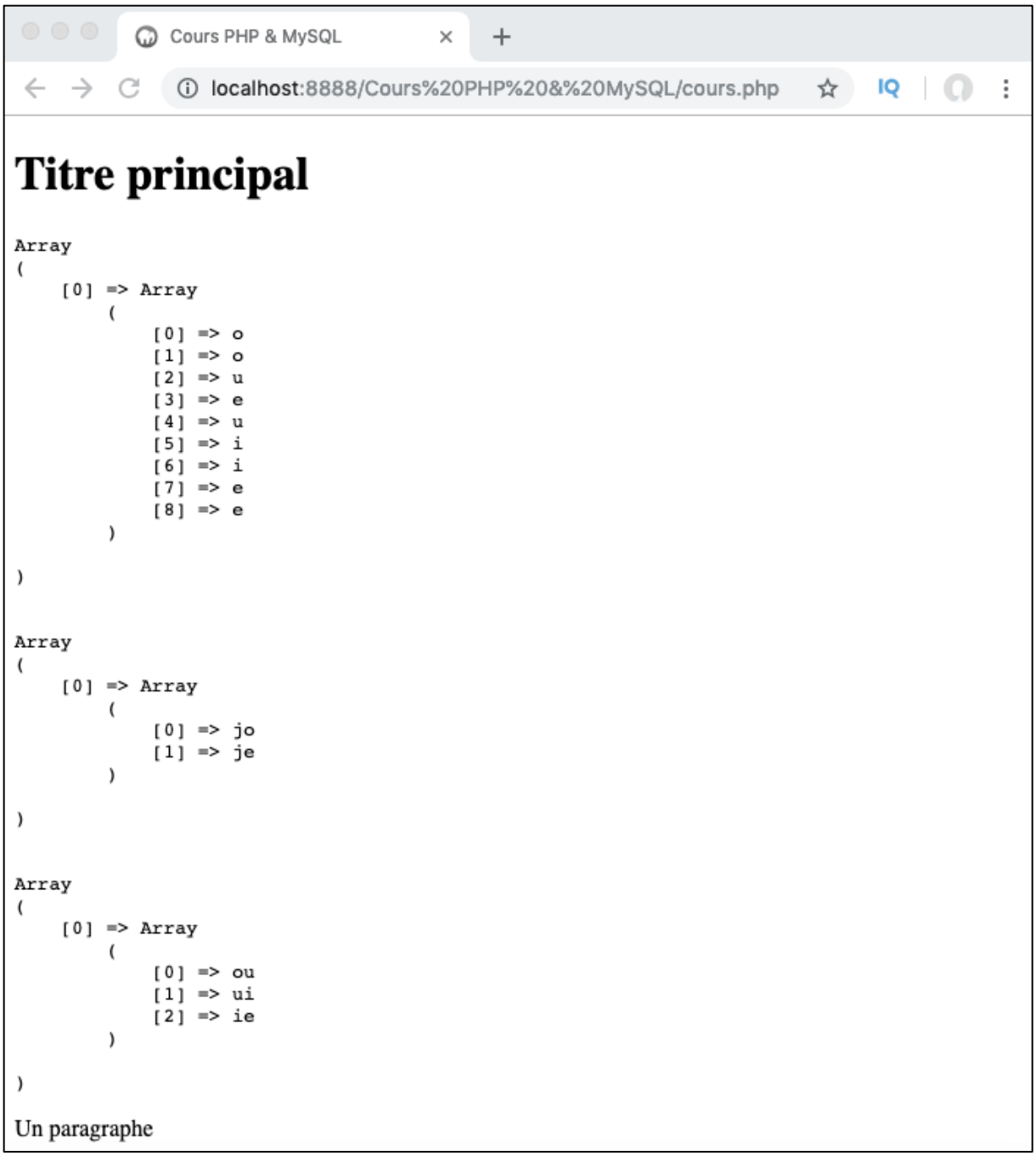
Les classes de caractères

Les classes de caractères vont nous permettre de fournir différents choix de correspondance pour un caractère en spécifiant un ensemble de caractères qui vont pouvoir être trouvés. En d'autres termes, elles vont nous permettre de rechercher n'importe quel caractère d'une chaine qui fait partie de la classe de caractères fournie dans le masque.

Pour déclarer une classe de caractères dans notre masque, nous allons utiliser une paire de crochets [ ] qui vont nous permettre de délimiter la classe en question.

Prenons immédiatement un exemple concret en utilisant des classes de caractères simples :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque1 = '/[aeiouy]/';
            $masque2 = '/j[aeiouy]/';
            $masque3 = '/[aeiouy][aeiouy]/';
            $chaine = 'Bonjour, je suis Pierre';

            preg_match_all($masque1,$chaine,$tb1);
            preg_match_all($masque2,$chaine,$tb2);
            preg_match_all($masque3,$chaine,$tb3);
            echo '<pre>';
            print_r($tb1);
            echo '<br><br>';
            print_r($tb2);
            echo '<br><br>';
            print_r($tb3);
            echo '</pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on utilise différentes classes de caractères dans nos masques pour rechercher différents caractères ou séquences de caractères dans nos chaines.

Notre premier masque est très simple : il contient uniquement la classe de caractères [aeiouy]. Cette classe de caractères va nous permettre de chercher tous les caractères de notre chaine qui sont des voyelles (soit un « a », soit « e », soit « i », soit « o », soit « u », soit « y »).

Notre fonction preg_match_all() va donc ici renvoyer toutes les voyelles de notre chaine une à une.

Notre deuxième masque permet de chercher la séquence « j suivi d'un voyelle ». En effet, ici, on place le caractère « j » en dehors de notre classe de caractères. Le masque va

donc nous permettre de chercher des séquences de deux caractères dont le premier est un « j » et le deuxième fait partie de la classe [aeiouy].

Dans notre troisième masque, nous utilisons cette fois-ci deux classes de caractères d'affilée. Ici, les deux classes de caractères sont identiques (on aurait tout-à-fait pu spécifier deux classes de caractères différentes) et vont toutes les deux nous permettre de rechercher une voyelle. On va donc ici pouvoir chercher deux voyelles à la suite.

Les classes de caractères et les méta caractères

Dans le langage des expressions régulières, de nombreux caractères vont avoir une signification spéciale et vont nous permettre de signifier qu'on recherche tel caractères ou telle séquence de caractères un certain nombre de fois ou à une certaine place dans une chaine.

Ces caractères spéciaux qu'on appelle des métacaractères vont nous permettre de créer des schémas ou masques de recherche très puissants et donc d'exploiter toute la puissance des expressions régulières. On a pu en voir un premier exemple avec les caractères crochets [ ] qui permettent de définir une classe de caractères.

Au sein des classes de caractères, nous n'avons accès qu'à 3 métacaractères, c'est-à-dire qu'il n'existe que trois caractères qui possèdent un sens spécial lorsqu'ils sont placés tels quels dans une classe de caractères. Ces métacaractères sont les suivants :

Métacaractère	Description
\	Caractère de protection qui va avoir plusieurs usages (on va pouvoir s'en servir pour donner un sens spécial à des caractères qui n'en possèdent pas ou au contraire pour neutraliser le sens spécial des métacaractères).
^	Si placé au tout début d'une classe, permet de nier la classe c'est-à-dire de chercher tout caractère qui n'appartient pas à la classe.
–	Entre deux caractères, permet d'indiquer un intervalle de caractères.

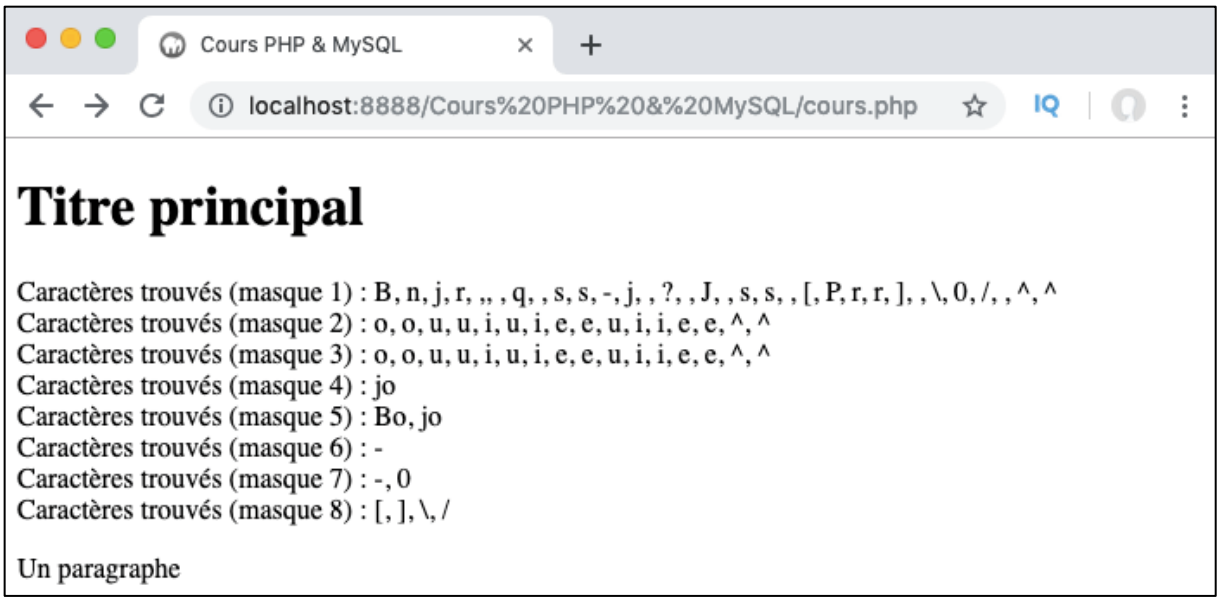
Si on souhaite rechercher le caractère représenté par un métacaractère et qu'on ne souhaite pas utiliser son sens spécial (par exemple si on souhaite rechercher le signe moins), il faudra alors le protéger avec un antislash.

Attention ici : pour rechercher le caractère antislash avec une expression rationnelle (c'est-à-dire à partir d'un masque stocké sous forme de chaine de caractères), il faudra préciser 4 antislashs d'affilée. En effet, l'analyseur PHP va ici considérer les 1er et 3è antislash comme des caractères d'échappement pour les deux autres et ne conserver donc que les 2è et 4è antislash puis la regex va considérer le 1er des deux antislashs restants comme un caractère de protection et va donc rechercher le caractère antislash placé derrière.

Notez qu'il faudra également protéger les signes crochets fermants ainsi que le délimiteur de masque choisi si on souhaite les inclure pour les rechercher dans une classe de

caractères car dans le cas contraire le PHP penserait qu'on termine une classe de caractères ou notre masque.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
      <h1>Titre principal</h1>
      <?php
        $masque1 = '/[^aeiouy]/';
        $masque2 = '/[\^aeiouy]/';
        $masque3 = '/[aei^ouy]/';
        $masque4 = '/[a-z]o/';
        $masque5 = '/[a-zA-Z]o/';
        $masque6 = '/[a\-z]/';
        $masque7 = '/[0-9az-]/';
        $masque8 = '/[\/[\]\\\\]/';
        $chaine = 'Bonjour, qui suis-je ? Je suis [Pierre] \0/ ^^';

        preg_match_all($masque1,$chaine,$tb1);
        preg_match_all($masque2,$chaine,$tb2);
        preg_match_all($masque3,$chaine,$tb3);
        preg_match_all($masque4,$chaine,$tb4);
        preg_match_all($masque5,$chaine,$tb5);
        preg_match_all($masque6,$chaine,$tb6);
        preg_match_all($masque7,$chaine,$tb7);
        preg_match_all($masque8,$chaine,$tb8);

        /*Ici, je sais que mes tableaux multidimensionnels ne contiennent
         *qu'un tableau à chaque fois. J'utilise la fonction implode()
         *qui renvoie les éléments d'un tableau sous forme de chaine de
         *caractères séparés par un séparateur choisi (ici la virgule)*/
        echo 'Caractères trouvés (masque 1) : ' .implode(', ', $tb1[0]). '<br>';
        echo 'Caractères trouvés (masque 2) : ' .implode(', ', $tb2[0]). '<br>';
        echo 'Caractères trouvés (masque 3) : ' .implode(', ', $tb3[0]). '<br>';
        echo 'Caractères trouvés (masque 4) : ' .implode(', ', $tb4[0]). '<br>';
        echo 'Caractères trouvés (masque 5) : ' .implode(', ', $tb5[0]). '<br>';
        echo 'Caractères trouvés (masque 6) : ' .implode(', ', $tb6[0]). '<br>';
        echo 'Caractères trouvés (masque 7) : ' .implode(', ', $tb7[0]). '<br>';
        echo 'Caractères trouvés (masque 8) : ' .implode(', ', $tb8[0]). '<br>';
      ?>
      <p>Un paragraphe</p>
    </body>
</html>
```



Ici, nous avons créé 8 masques différents. Le premier masque utilise le caractère ^ en début de classe de caractère. Ce caractère va donc être interprété selon son sens de métacaractère et va nier la classe. Notre masque va donc nous permettre de chercher tous les caractères d'une chaine qui ne sont pas des voyelles minuscules. Notez que les espaces sont également des caractères et vont être trouvés ici.

Dans notre deuxième masque, on protège le métacaractère ^ avec un antislash. Notre masque va donc nous permettre de trouver toutes les voyelles de notre chaine plus le caractère « ^ ».

Dans notre troisième masque, on utilise le caractère « ^ » au milieu de la classe. Celui-ci ne possède donc pas son sens de métacaractère et nous n'avons pas besoin ici de le protéger. Ce troisième masque va nous permettre de chercher les mêmes choses que le précédent.

Notre quatrième masque utilise le métacaractère -. Dans le cas présent, il indique que notre classe de caractère contient toutes les lettres minuscules de a à z, c'est-à-dire tout l'alphabet. Notre masque va donc trouver toutes les séquences contenant une lettre de l'alphabet minuscule suivie d'un « o ».

Notez bien ici que les lettres qui ne font pas partie strictement de l'alphabet anglais commun (c'est-à-dire les lettres accentuées, les lettres avec cédilles, etc.) ne seront pas ici trouvées.

Dans notre cinquième masque, on définit deux plages ou intervalles de caractères grâce au métacaractère -. Ici, toutes les lettres de l'alphabet minuscules ou majuscules vont correspondre aux critères de la classe. Le masque va donc nous permettre de chercher toutes les séquences contenant une lettre de l'alphabet minuscule ou majuscule suivie d'un « o ».

Dans notre sixième masque, on protège cette fois-ci le caractère « – » . Notre masque va donc nous permettre de trouver les caractères « a », « – » et « z ».

Dans notre septième masque, on utilise cette fois-ci le métacaractère - pour définir une place numérique (les regex vont nous permettre de trouver n'importe quel caractère, que

ce soit une lettre, un chiffre, un signe, etc.). Notre masque va donc trouver n'importe quel chiffre (de 0 à 9), la lettre « a », le lettre « z » et le caractère « – ». En effet, le caractère est ici également mentionné en fin de classe et ne possède donc pas de sens spécial et n'a pas besoin d'être protégé.

Finalement, notre dernier masque va nous permettre de trouver les caractères « / », « [», «] » et « \ ». Notez ici qu'il est nécessaire de protéger le caractère crochet fermant mais pas le crochet ouvrant qui sera recherché en tant que tel.

Dans cet exemple, on utilise `preg_match_all()` qui va renvoyer toutes les correspondances sous forme de tableau multidimensionnel. Nos masques restent pour le moment relativement simples et je sais ici que les différents tableaux renvoyés ne vont comporter qu'un seul tableau.
J'utilise donc ensuite la fonction `implode()` qui renvoie les différents éléments d'un tableau sous forme de chaine de caractères en les séparant avec un séparateur de notre choix. Je passe ici le séparateur virgule à cette fonction et lui passe le tableau contenu dans mes différents tableaux multidimensionnels pour qu'elle me renvoie tous les résultats.

Notez que j'aurais aussi bien pu me contenter d'utiliser `print_r()` comme précédemment mais comme on renvoie beaucoup de choses ce coup-ci, je voulais un rendu sous forme de chaine afin qu'il soit plus compacte et plus présentable.

Les classes de caractères abrégées ou prédéfinies

Le caractère d'échappement ou de protection antislash va pouvoir avoir plusieurs rôles ou plusieurs sens dans un contexte d'utilisation au sein d'expressions régulières. On a déjà vu que l'antislash nous permettait de protéger certains métacaractères, c'est-à-dire que le métacaractères ne prendra pas sa signification spéciale mais pourra être cherché en tant que caractère simple.

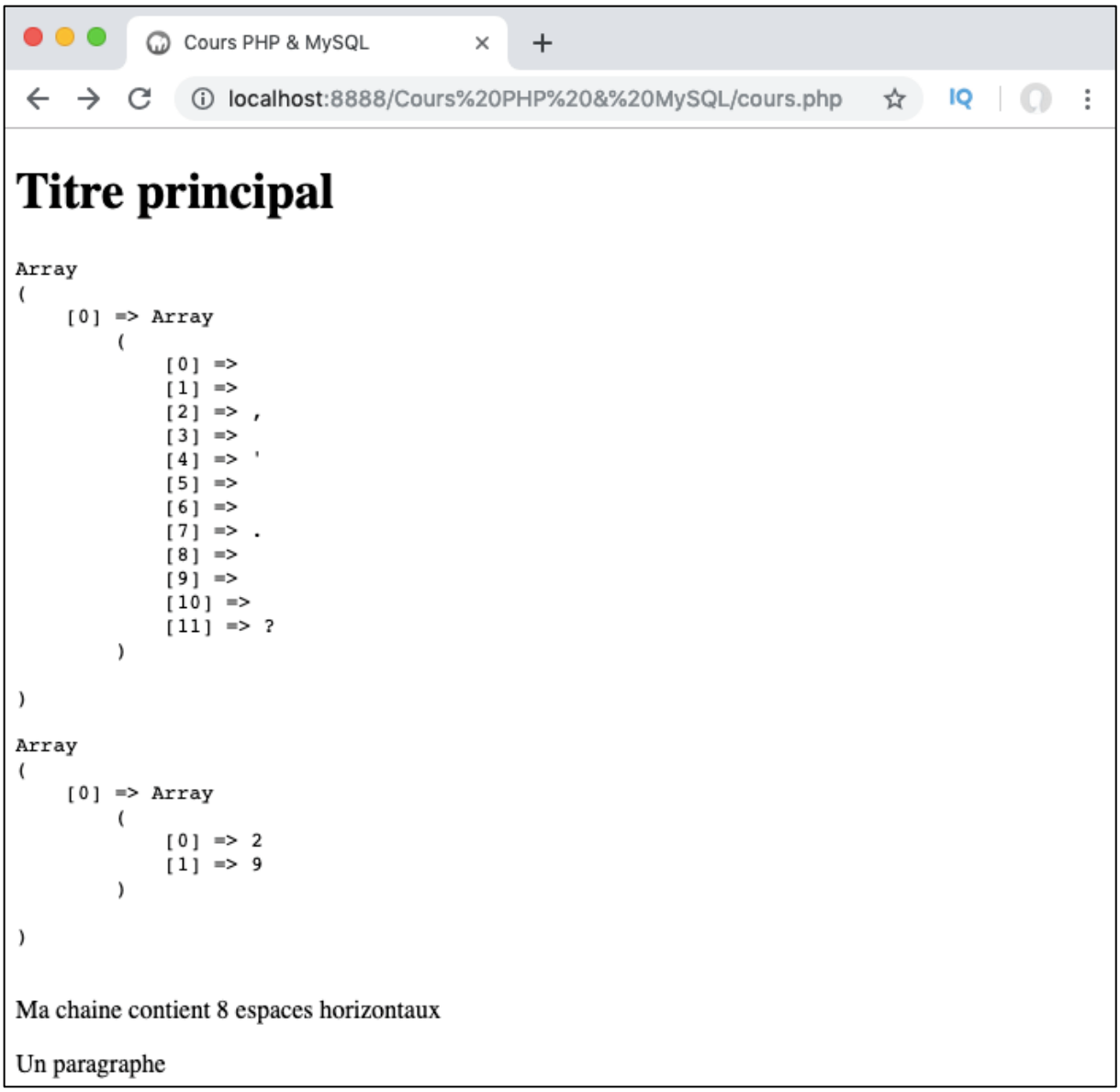
L'antislash va encore pouvoir être utilisé au sein de classes de caractères avec certains caractères « normaux » pour au contraire leur donner une signification spéciale.

On va ainsi pouvoir utiliser ce qu'on appelle des classes abrégées ou prédéfinies pour indiquer qu'on recherche un type de valeurs plutôt qu'une valeur ou qu'une plage de valeurs en particuliers. Les classes abrégées disponibles sont les suivantes (faites bien attention aux emplois de majuscules et de minuscules ici !) :

Classe abrégée	Description
\w	Représente tout caractère de « mot ». Équivalent à [a-zA-Z0-9_]
\W	Représente tout caractère qui n'est pas un caractère de « mot ». Equivalent à [^a-zA- Z0-9_]
\d	Représente un chiffre. Équivalent à [0-9]
\D	Représente tout caractère qui n'est pas un chiffre. Équivalent à [^0-9]

Classe abrégée	Description
\s	Représente un caractère blanc (espace, retour chariot ou retour à la ligne)
\S	Représente tout caractère qui n'est pas un caractère blanc
\h	Représente un espace horizontal
\H	Représente tout caractère qui n'est pas un espace horizontal
\v	Représente un espace vertical
\V	Représente tout caractère qui n'est pas un espace vertical

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque1 = '/[\W]/';
            $masque2 = '/[\d]/';
            $masque3 = '/[\h]/';
            $chaine = 'Je suis Pierre, j\'ai 29 ans. Et vous ?';

            preg_match_all($masque1, $chaine, $tb1);
            preg_match_all($masque2, $chaine, $tb2);
            $espaces = preg_match_all($masque3, $chaine);
            echo '<pre>';
            print_r($tb1);
            echo '<br>';
            print_r($tb2);
            echo '</pre><br>';
            echo 'Ma chaine contient ' .$espaces. ' espaces horizontaux';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, notre premier masque nous permet de trouver tous les caractères qui n'appartiennent pas à la classe `[a-ZA-Z-0-9_]`, c'est-à-dire tout caractère qui n'est ni une lettre de l'alphabet de base ni un chiffre ni un underscore.

Notre deuxième masque nous permet de trouver tous les caractères de type chiffres dans une chaine de caractères.

Notre troisième masque nous permet de trouver tous les espaces. Ici, on choisit d'utiliser la valeur renvoyée par défaut par `preg_match_all()` qui correspond au nombre de fois où le masque a été trouvé dans a chaine pour renvoyer le nombre d'espaces dans notre chaine.

Les métacaractères des regex PHP

Dans la leçon précédente, nous avons appris à créer des classes de caractères et avons découvert qu'on pouvait insérer dans nos classes de caractères des caractères qui possèdent une signification spéciale : les métacaractères.

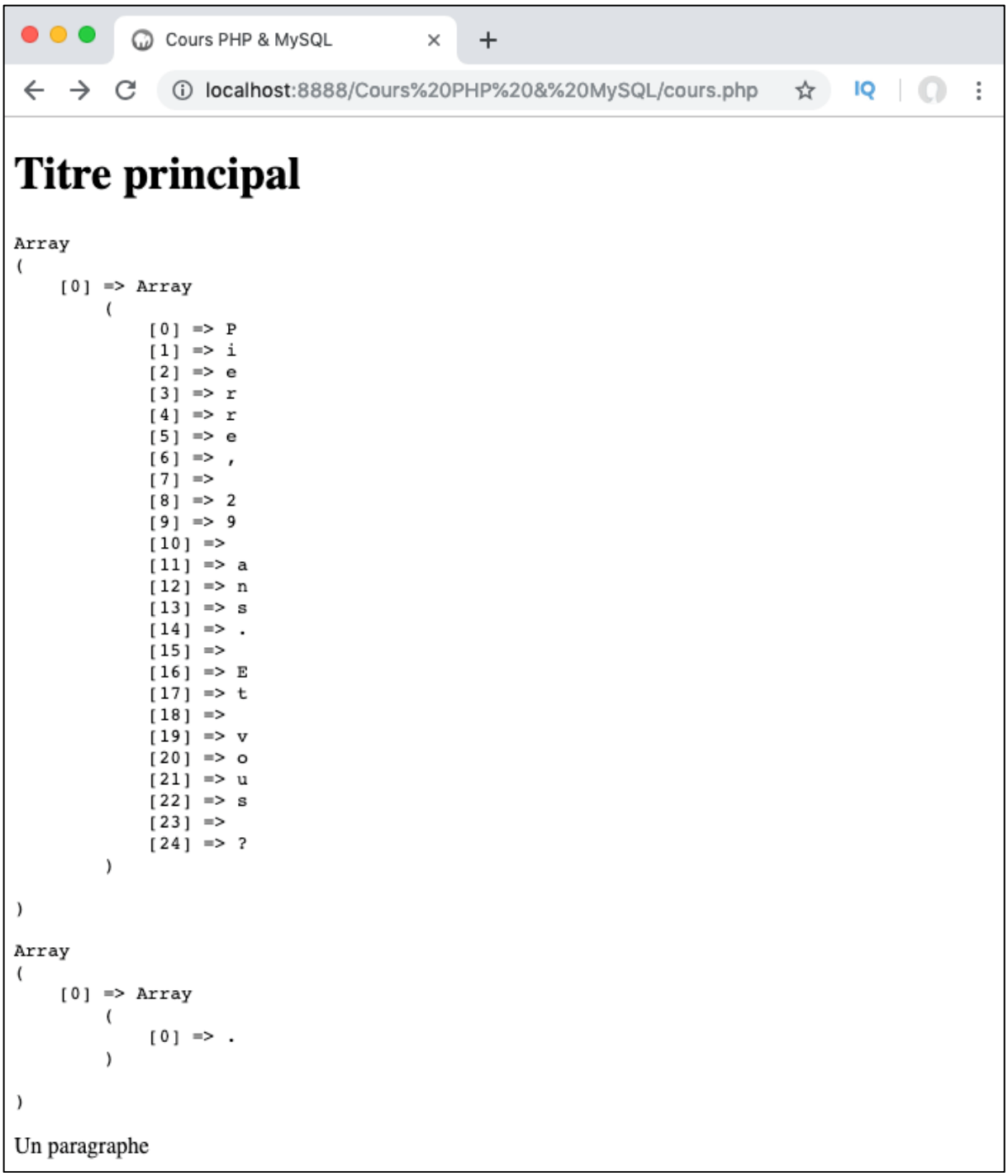
Nous n'avons accès qu'à trois métacaractères au sein des classes de caractères : les métacaractères `^`, `-` et `\`. A l'extérieur des classes de caractères, cependant, nous allons pouvoir en utiliser de nombreux autres.

Nous allons présenter ces différents métacaractères dans cette leçon.

Le point

Le métacaractère `.` (point) va nous permettre de rechercher n'importe quel caractère à l'exception du caractère représentant une nouvelle ligne qui est en PHP le `\n`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque1 = '/./';
            $masque2 = '/[.]/';
            $chaine = 'Pierre, 29 ans. Et vous ?';

            preg_match_all($masque1, $chaine, $tb1);
            preg_match_all($masque2, $chaine, $tb2);

            echo '<pre>';
            print_r($tb1);
            echo '<br>';
            print_r($tb2);
            echo '</pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



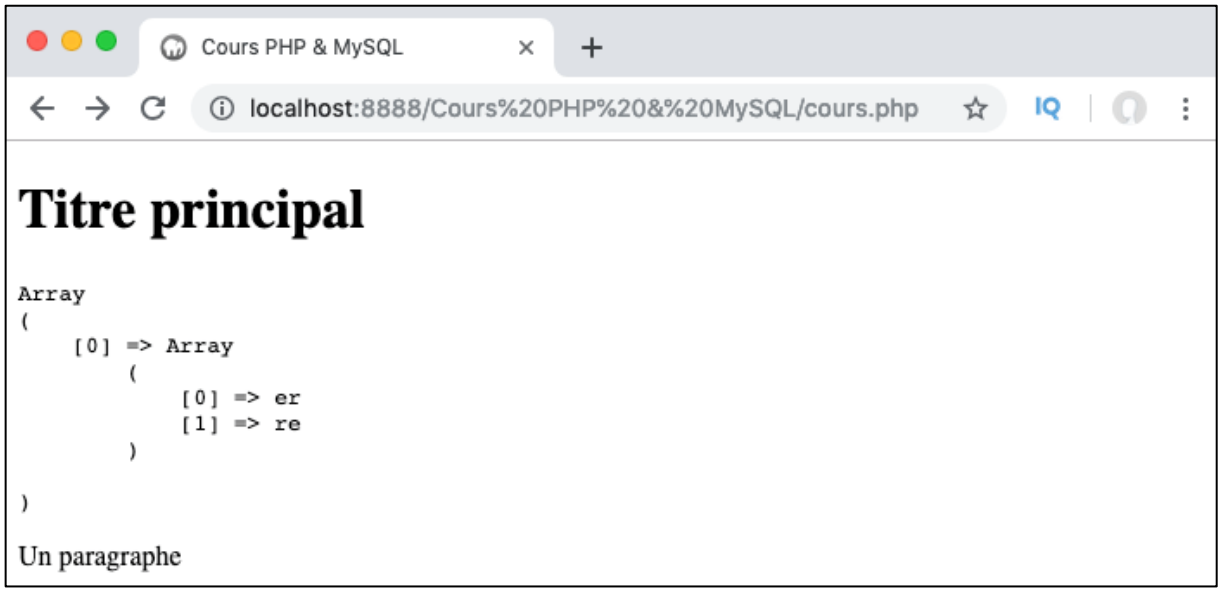
Comme vous pouvez le voir, le point a un sens bien différent selon qu'il soit spécifié dans une classe ou en dehors d'une classe de caractères : en dehors d'une classe de caractères, le point est un métacaractère qui permet de chercher n'importe quel caractère sauf une nouvelle ligne tandis que dans une classe de caractère le point sert simplement à rechercher le caractère point dans notre chaine de caractères.

Encore une fois, il n'existe que trois métacaractères, c'est-à-dire trois caractères qui vont posséder un sens spécial à l'intérieur des classes de caractères. Les métacaractères que nous étudions dans cette leçon ne vont avoir un sens spécial qu'en dehors des classes de caractères.

Les alternatives

Le métacaractère | (barre verticale) sert à séparer des alternatives. Concrètement, ce métacaractère va nous permettre de créer des masques qui vont pouvoir chercher une séquence de caractères ou une autre.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque1 = '/er|re/';
            $chaine = 'Pierre, 29 ans. Et vous ?';

            preg_match_all($masque1, $chaine, $tb1);

            echo '<pre>';
            print_r($tb1);
            echo '</pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on utilise le métacaractère | pour créer une alternative dans notre masque. Ce masque va nous permettre de chercher soit la séquence de caractères « er » soit la séquence « re » dans la chaine de caractères à analyser.

Les ancres

Les deux métacaractères ^ et $ vont nous permettre « d'ancrer » des masques.

Le métacaractère ^, lorsqu'il est utilisé en dehors d'une classe, va posséder une signification différente de lors de l'utilisation dans une classe. Attention donc à ne pas confondre les deux sens !

Utiliser le métacaractère ^ en dehors d'une classe nous permet de rechercher la présence du caractère suivant le ^ du masque en début de la chaine de caractères à analyser.
Il va falloir le placer en début de du masque ou tout au moins en début d'alternative pour qu'il exprime ce sens.

Au contraire, le métacaractère $ va nous permettre de rechercher la présence du caractère précédant le métacaractère en fin de chaine.

Il va falloir placer le métacaractère $ en fin de du masque ou tout au moins en fin d'alternative pour qu'il exprime ce sens.

Prenons immédiatement quelques exemples concrets :

```php
<body>
    <h1>Titre principal</h1>
    <?php
        $masque1 = '/^p/'; //Cherche "p" en début de chaine
        $masque2 = '/^p|^P/'; //Cherche "p" ou "P" en début de chaine
        $masque3 = '/^[A-Z]/'; //Cherche une majuscule en début de chaine
        $masque4 = '/\?$/'; //Cherche "?" en fin de chaine
        /*Cherche une chaine de deux caractères qui commence par "P" et finit
         *par "?"*/
        $masque5 = '/^p\?$|^P\?$/';
        $chaine = 'Pierre, 29 ans. Et vous ?';

        if(preg_match($masque1, $chaine)){
            echo '"p" trouvé en début de chaine <br>';
        }else{
            echo '"p" non trouvé en début de chaine <br>';
        }
        if(preg_match($masque2, $chaine)){
            echo '"p" ou "P" trouvé en début de chaine <br>';
        }else{
            echo '"p" ou "P" non trouvé en début de chaine <br>';
        }
        if(preg_match($masque3, $chaine)){
            echo 'La chaine commence par une lettre majuscule de A à Z <br>';
        }else{
            echo 'La chaine ne commence pas par une majuscule de A à Z <br>';
        }
        if(preg_match($masque4, $chaine)){
            echo '"?" trouvé en fin de chaine <br>';
        }else{
            echo '"?" non trouvé en fin de chaine <br>';
        }
        if(preg_match($masque5, $chaine)){
            echo 'La chaine est "p?" ou "P?"<br>';
        }else{
            echo 'La chaine n\'est pas "p?" ou "P?"<br>';
        }
    ?>
    <p>Un paragraphe</p>
</body>
```

Nos masques commencent ici à être relativement complexes et il faut bien faire attention à leur écriture. Avant tout, j'ai dû protéger le caractère `?` à chaque utilisation dans mes masques car c'est également un métacaractère que nous allons étudier juste après et car ici je souhaitais véritablement chercher la présence du caractère « ? » et non pas donner un sens différent à mon masque.

Mon premier masque nous permet de chercher la présence d'un « p » minuscule en début de chaine grâce au métacaractère `^` placé en début de masque.

Mon deuxième masque nous permet de chercher la présence d'un « p » minuscule ou d'un « P » majuscule en début de chaine. Ici, vous pouvez remarquer qu'on utiliser deux fois le métacaractère `^`. En effet, ce métacaractère doit être placé soit en début de masque soit en début d'alternative pour qu'on puisse utiliser son sens spécial.

Le troisième masque nous permet cette fois-ci de chercher la présence d'une lettre majuscule de l'alphabet commun (lettre non accentuée et sans cédille) en début de chaine. Notez bien ici que j'utilise mon métacaractère `^` en dehors de ma classe de caractères.

Notre quatrième masque permet de chercher la présence d'un point d'interrogation en fin de chaine. Ici, il faut protéger le point d'interrogation pour le chercher comme caractère en tant que tel.

Notre cinquième et dernier masque est un peu plus complexe à comprendre. On utilise cette fois-ci à la fois le métacaractère `^` et le métacaractère `$` ce qui va créer une vraie restriction sur ce qui va être recherché.

En effet, vous devez bien comprendre qu'ici on cherche une séquence de deux caractères avec le premier caractère qui doit être un « p » ou un « P » en début de chaine et le deuxième caractère qui doit être un « ? » et se situer en fin de chaine. On cherche donc exactement les chaines « p? » ou « P? ».

Nous allons voir ci-dessous comment spécifier qu'on cherche une chaine de taille quelconque qui commence par un certain caractère et se termine par un autre.

Les quantificateurs

Les quantificateurs sont des métacaractères qui vont nous permettre de rechercher une certaine quantité d'un caractère ou d'une séquence de caractères.

Les quantificateurs disponibles sont les suivants :

Quantificateur	Description
a{X}	On veut une séquence de X « a »
a{X,Y}	On veut une séquence de X à Y fois « a »
a{X,}	On veut une séquence d'au moins X fois « a » sans limite supérieure
a?	On veut 0 ou 1 « a ». Équivalent à a{0,1}
a+	On veut au moins un « a ». Équivalent à a{1,}
a*	On veut 0, 1 ou plusieurs « a ». Équivalent à a{0,}

Bien évidemment, les lettres « a », « X » et « Y » ne sont données ici qu'à titre d'exemple et on les remplacera par des valeurs effectives en pratique.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque1 = '/er?/';
            $masque2 = '/er+/';
            $masque3 = '/^[A-Z].{10,}\?$/';
            $masque4 = '/^\d{10,10}$/';

            $chaine = 'Pierre, 29 ans. Et vous ?';
            $chaine2 = '0665656565';

            preg_match_all($masque1, $chaine, $tb1);
            preg_match_all($masque2, $chaine, $tb2);
            preg_match_all($masque3, $chaine, $tb3);
            preg_match_all($masque4, $chaine2, $tb4);

            print_r($tb1);
            echo '<br>';
            print_r($tb2);
            echo '<br>';
            print_r($tb3);
            echo '<br>';
            print_r($tb4);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Notre premier masque va ici nous permettre de chercher un « e » suivi de 0 ou 1 « r ». La chose à bien comprendre ici est que si notre chaine contient un « e » suivi de plus d'un « r » alors la séquence « er » sera bien trouvée puisqu'elle est bien présente dans la chaine. Le fait qu'il y ait d'autres « r » derrière n'influe pas sur le résultat.

Notez également que les quantificateurs sont dits « gourmands » par défaut : cela signifie qu'ils vont d'abord essayer de chercher le maximum de répétition autorisé. C'est la raison pour laquelle ici « er » est renvoyé la première fois (séquence présente dans « Pierre ») et non pas simplement « e ». Ensuite, ils vont chercher le nombre de répétitions inférieur et etc. (le deuxième « e » de « Pierre » est également trouvé.

Notre deuxième masque va chercher un « e » suivi d'au moins un « r ». On trouve cette séquence dans « Pierre ». Comme les quantificateurs sont gourmands, c'est la séquence la plus grande autorisée qui va être trouvée, à savoir « err ».

Notre troisième masque est plus complexe et également très intéressant. Il nous permet de chercher une chaine qui commence par une lettre de l'alphabet commun en majuscule suivie d'au moins 10 caractères qui peuvent être n'importe quel caractère à part un retour à la ligne (puisqu'on utilise ici le métacaractère point) et qui se termine avec un « ? ».

Finalement, notre quatrième masque va nous permettre de vérifier qu'une chaine contient exactement et uniquement 10 chiffres. Ce type de masque va être très intéressant pour vérifier qu'un utilisateur a inscrit son numéro de téléphone correctement lors de son inscription sur notre site par exemple.

Les sous masques

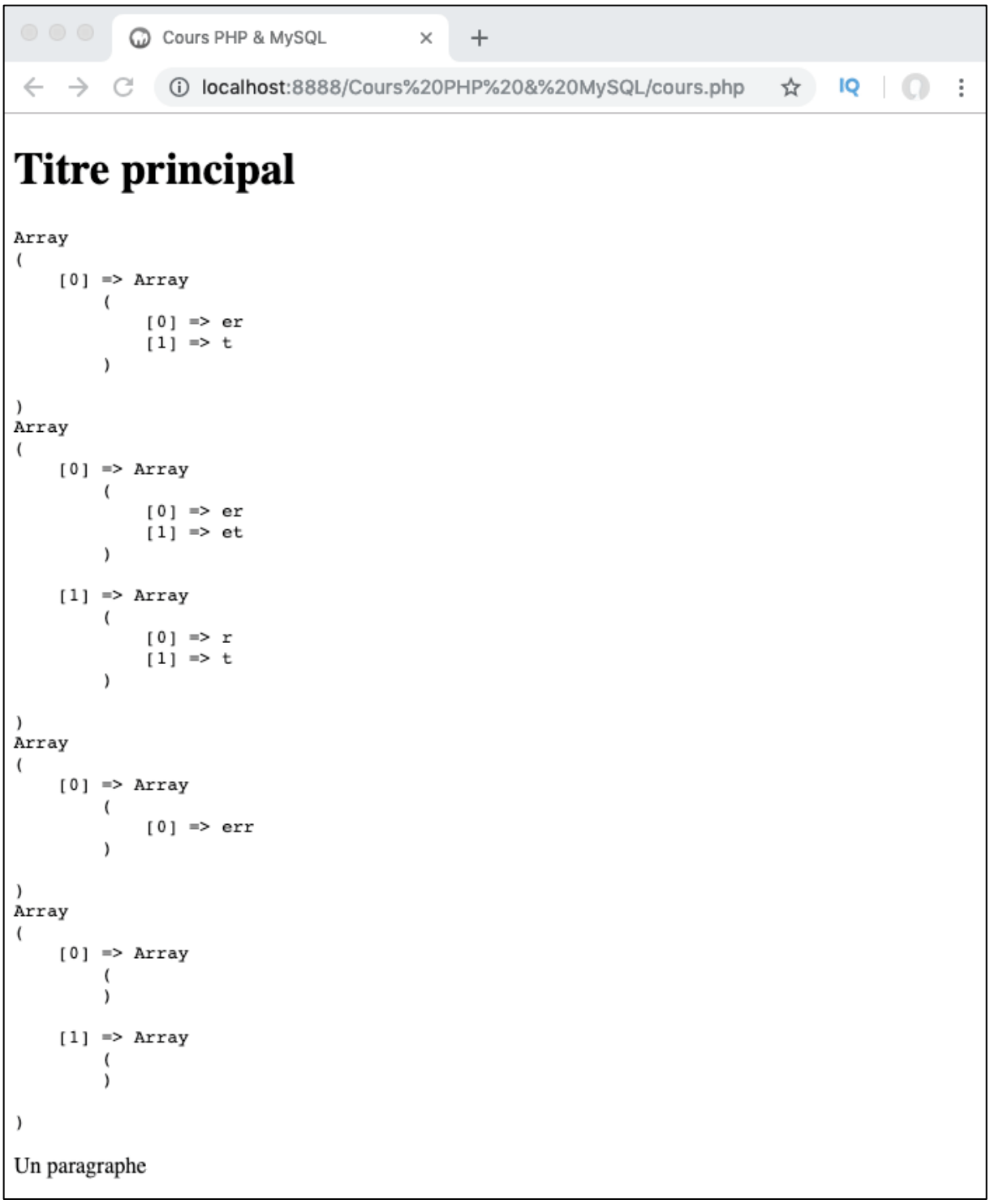
Les métacaractères (et) vont être utilisés pour délimiter des sous masques.

Un sous masque est une partie d'un masque délimités par un couple de parenthèses. Ces parenthèses vont nous permettre d'isoler des alternatives ou de définir sur quelle partie du masque un quantificateur doit s'appliquer.

De manière très schématique, et même si ce n'est pas strictement vrai, vous pouvez considérer qu'on va en faire le même usage que lors d'opérations mathématiques, c'est-à-dire qu'on va s'ne servir pour prioriser les calculs.

Par défaut, les sous masques vont être capturants. Cela signifie tout simplement que lorsqu'un sous masque est trouvé dans la chaine de caractères, la partie de cette chaine sera capturée.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque1 = '/er|t/'; //Chercher "er" ou "t"
            $masque2 = '/e(r|t)/'; //Cherche "er", "et", "r" et "t"
            $masque3 = '/er{2}/'; //Cherche "e" suivi de "r" suivi de "r"
            $masque4 = '/(er){2}/'; //Cherche "er" suivi de "er"

            $chaine = 'Je suis Pierre et j\'ai 29 ans.';

            preg_match_all($masque1, $chaine, $tb1);
            preg_match_all($masque2, $chaine, $tb2);
            preg_match_all($masque3, $chaine, $tb3);
            preg_match_all($masque4, $chaine, $tb4);

            echo '<pre>';
            print_r($tb1);
            print_r($tb2);
            print_r($tb3);
            print_r($tb4);
            echo '</pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Notre premier masque n'utilise pas les métacaractères de sous masque ( ). Il nous permet de chercher « er » ou « t ».

Notre deuxième masque contient un sous masque. Ce masque va nous permettre de chercher « er », « et », « r » et « t » car je vous rappelle que les sous masques sont capturants : si un motif du sous masque correspond, la partie de la chaine de caractères dans laquelle on recherche sera capturée. Ici, notre sous masque chercher « r » ou « t ». Dès que ces caractères vont être trouvés dans la chaine, ils vont être capturés et renvoyés.

Notre troisième masque nous permet de chercher un « e » suivi du caractère « r » répété deux fois.

Notre quatrième masque, en revanche, va nous permettre de chercher la séquence « er » répétée deux fois.

Les assertions

On appelle « assertion » un test qui va se dérouler sur le ou les caractères suivants ou précédent celui qui est à l'étude actuellement. Par exemple, le métacaractère $ est une assertion puisque l'idée ici est de vérifier qu'il n'y a plus aucun caractère après le caractère ou la séquence écrite avant $.

Ce premier exemple correspond à une assertion dite simple. Il est également possible d'utiliser des assertions complexes qui vont prendre la forme de sous masques.

Il existe à nouveau deux grands types d'assertions complexes : celles qui vont porter sur les caractères suivants celui à l'étude qu'on appellera également « assertion avant » et celles qui vont porter sur les caractères précédents celui à l'étude qu'on appellera également « assertion arrière ».

Les assertions avant et arrière vont encore pouvoir être « positives » ou « négatives ». Une assertion « positive » est une assertion qui va chercher la présence d'un caractère après ou avant le caractère à l'étude tandis qu'une assertion « négative » va au contraire vérifier qu'un caractère n'est pas présent après ou avant le caractère à l'étude.

Notez que les assertions, à la différence des sous masques, ne sont pas capturantes par défaut et ne peuvent pas être répétées.

Voici les assertions complexes qu'on va pouvoir utiliser ainsi que leur description rapide :

Assertion	Description
a(?=b)	Cherche « a » suivi de « b » (assertion avant positive)
a(?!b)	Cherche « a » non suivi de « b » (assertion avant négative)
(?<=b)a	Cherche « a » précédé par « b » (assertion arrière positive)
(?<!b)a	Cherche « a » non précédé par « b » (assertion arrière négative)

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $masque1 = '/e(?=r)/'; //Chercher "e" suivi de "r"
            $masque2 = '/e(?!r)/'; //Chercher "e" non suivi de "r"
            $masque3 = '/(?<=i)s/'; //Cherche "s" précédé de "i"
            $masque4 = '/(?<!i)s/'; //Cherche "s" non précédé de "i"

            $chaine = 'Je suis Pierre et j\'ai 29 ans.';

            $res1 = preg_match_all($masque1, $chaine);
            $res2 = preg_match_all($masque2, $chaine);
            $res3 = preg_match_all($masque3, $chaine);
            $res4 = preg_match_all($masque4, $chaine);

            echo 'La chaine complète est : "' .$chaine.'".<br>
                  Elle contient ' .$res1. ' "e" suivi d\'un "r". <br>
                  Elle contient ' .$res2. ' "e" non suivi d\'un "r". <br>
                  Elle contient ' .$res3. ' "s" précédé d\'un "i". <br>
                  Elle contient ' .$res4. ' "s" non précédé d\'un "i". <br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Résumé : liste complète des métacaractères des expressions régulières

Voici la liste de tous les métacaractères qu'on va pouvoir utiliser en dehors des classes de caractères :

Métacaractères	Description
\	Caractère dit d'échappement ou de protection qui va servir notamment à neutraliser le sens d'un métacaractère ou à créer une classe abrégée
[	Définit le début d'une classe de caractères
]	Définit la fin d'une classe de caractères
.	Permet de chercher n'importe quel caractère à l'exception du caractère de nouvelle ligne
|	Caractère d'alternative qui permet de trouver un caractère ou un autre
^	Permet de chercher la présence d'un caractère en début de chaine
$	Permet de chercher la présence d'un caractère en fin de chaine
?	Quantificateur de 0 ou 1. Peut également être utilisé avec « (» pour en modifier le sens
+	Quantificateur de 1 ou plus
*	Quantificateur de 0 ou plus
{	Définit le début d'un quantificateur
}	Définit la fin d'un quantificateur
(	Début de sous masque ou d'assertion
)	Fin de sous masque ou d'assertion

Les options des expressions régulières disponibles en PHP

En plus des métacaractères, nous allons également pouvoir ajouter des caractères qu'on appelle des options à nos masques pour construire nos expressions régulières.

Dans cette leçon, nous allons découvrir les différents caractères d'option disponibles et apprendre à les utiliser intelligemment.

Présentation des options des regex

Les options, encore appelées modificateurs, sont des caractères qui vont nous permettre d'ajouter des options à nos expressions régulières.

Les options ne vont pas à proprement parler nous permet de chercher tel ou tel caractère mais vont agir à un niveau plus élevé en modifiant le comportement par défaut des expressions régulières. Elles vont par exemple nous permettre de rendre une recherche insensible à la casse.

On va pouvoir facilement différencier une option d'un caractère normal ou d'un métacaractère dans une expression régulière puisque les options sont les seuls caractères qui peuvent et doivent obligatoirement être placés en dehors des délimiteurs du masque, après le délimiteur final.

Liste des options disponibles et exemples d'utilisation

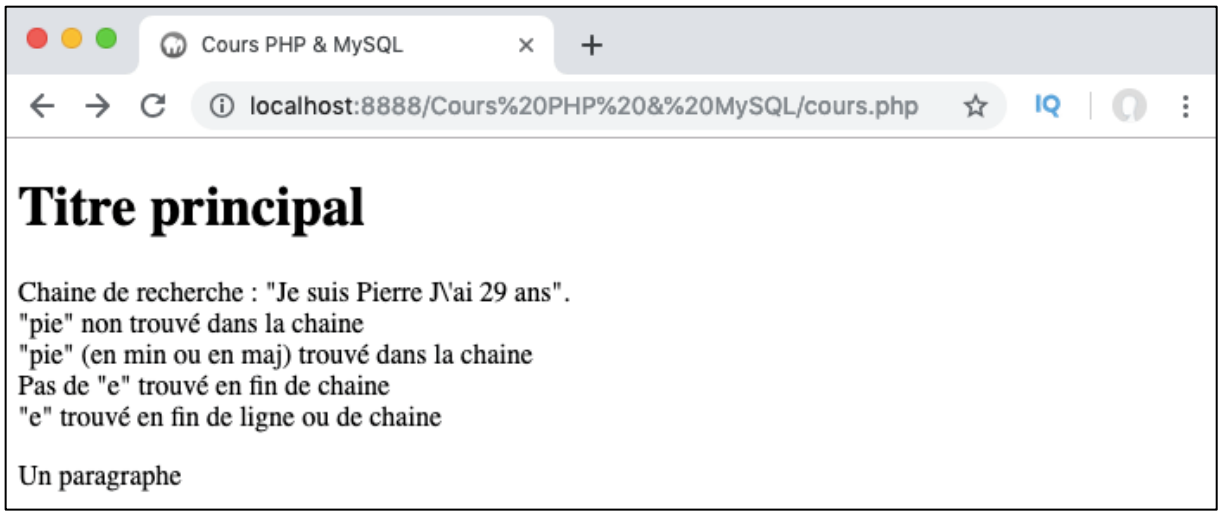
Certaines options sont complexes dans leur fonctionnement, peu utilisées ou ne sont pas toujours compatibles. Le tableau suivant ne présente que les options toujours disponibles et les plus utiles selon moi.

Option	Description
i	Rend la recherche insensible à la casse
m	Par défaut, les expressions régulières considèrent la chaine dans laquelle on fait une recherche comme étant sur une seule ligne et font qu'on ne peut donc utiliser les métacaractères ^ et $ qu'une seule fois. L'option m permet de tenir compte des caractères de retour à la ligne et de retour chariot et fait que ^ et $ vont pouvoir être utilisés pour chercher un début et une fin de ligne
s	Cette option permet au métacaractère . de remplacer n'importe quel caractère y compris un caractère de nouvelle ligne
x	Permet d'utiliser des caractères d'espacement dans nos masques sans que ceux-ci soient analysés afin de clarifier nos masques. Attention cependant à

Option	Description
	ne pas ajouter d'espace dans es séquences spéciales d'un masque, comme entre un « (» et un « ? » par exemple
u	Cette option permet de désactiver les fonctionnalités additionnelles de PCRE qui ne sont pas compatibles avec le langage Perl. Cela peut être très utile dans le cas où on souhaite exporter nos regex

Voyons immédiatement comment utiliser ces options en pratique. Notez qu'on va tout à fait pouvoir ajouter plusieurs options à un masque.

```
<body>
    <h1>Titre principal</h1>
    <?php
        $masque1 = '/pie/';
        $masque2 = '/pie/i';
        $masque3 = '/e$/';
        $masque4 = '/e$/m';

        /*On utilise des guillemets ici afin que le PHP interprète bien
         *le retour à la ligne \n*/
        $chaine = "Je suis Pierre\nJ\'ai 29 ans";

        echo 'Chaine de recherche : "' .$chaine.'".<br>';
        if(preg_match($masque1, $chaine)){
            echo '"pie" trouvé dans la chaine<br>';
        }else{
            echo '"pie" non trouvé dans la chaine<br>';
        }

        if(preg_match($masque2, $chaine)){
            echo '"pie" (en min ou en maj) trouvé dans la chaine<br>';
        }else{
            echo '"pie" (en min ou en maj) non trouvé dans la chaine<br>';
        }

        if(preg_match($masque3, $chaine)){
            echo '"e" trouvé en fin de chaine<br>';
        }else{
            echo 'Pas de "e" trouvé en fin de chaine<br>';
        }

        if(preg_match($masque4, $chaine)){
            echo '"e" trouvé en fin de ligne ou de chaine<br>';
        }else{
            echo 'Pas de "e" trouvé en fin de ligne ou de chaine<br>';
        }
    ?>
    <p>Un paragraphe</p>
</body>
```



Pour bien comprendre ce code, il faut déjà noter qu'on utilise ici un caractère de nouvelle ligne en PHP dans notre chaine (`\n`). Pour que ce caractère soit correctement interprété par le PHP, on entoure ici notre chaine de guillemets droits plutôt que d'apostrophes.

Je vous rappelle en effet que la grande différence entre les apostrophes et les guillemets lors de leur utilisation avec les chaines de caractères en PHP réside dans ce qui va être interprété ou pas par le PHP.

En utilisant des apostrophes, le PHP considèrera le contenu de la chaine comme du texte (à l'exception des caractères `\\` et `\'` qui vont nous permettre d'échapper un antislash et une apostrophe) tandis qu'en utilisant des guillemets le PHP interprètera les noms de variables et les séquences d'échappement comme `\n` par exemple.

Notre premier masque nous permet ici de chercher la séquence « pie » en minuscules dans notre chaine. Celle-ci n'est pas trouvée.

Notre deuxième masque utilise cette fois-ci l'option `i` qui rend la recherche insensible à la casse. Ce masque va donc nous permettre de trouver n'importe quelle séquence « pie » en minuscules ou en majuscules.

Notre troisième masque cherche le caractère « e » en fin de chaine. En effet, comme l'option `m` n'est pas présente, PCRE considèrera que notre chaine est sur une seule ligne.

Notre quatrième masque utilise l'option `m` qui va changer le comportement par défaut de PCRE qui va alors tenir compte des retours à la ligne (`\n`) et des retours chariots (`\r`) dans notre chaine. Ce masque nous permet de cherche le caractère « e » en fin de ligne ou de chaine.

Conclusion sur les expressions régulières en PHP

Nous avons couvert la majorité des concepts relatifs à l'utilisation des expressions régulières en PHP et sommes désormais capables de créer des masques de recherche puissants qui vont nous permettre d'analyser le contenu d'une chaine.

Une nouvelle fois, les expressions régulières vont s'avérer particulièrement utiles lorsqu'on voudra vérifier la forme des données envoyées par les utilisateurs. Elles vont par exemple nous permettre de nous assurer qu'un utilisateur a bien exactement envoyé une séquence de 10 chiffres lorsqu'on lui a demandé son numéro de téléphone, ou que le mot de passe choisi par l'utilisateur lors de son inscription contient au moins 8 caractères dont un caractère spécial, une majuscule et un chiffre par exemple.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <form action='cours.php' method='post'>
            <label for='pass'>Choisissez un mot de passe.</label>
            <input type='password' name='pass' id='pass'>
            <br>
            <p>Note : Le mot de passe doit possèder au moins 8 caractères dont
            au moins une majuscule, un chiffre et un caractère spécial</p>
            <br>
            <input type='submit' value='Envoyer'>
        </form>
        <?php
            $m = '/^\S*(?=\S{8,})(?=\S*[A-Z])(?=\S*[\d])(?=\S*[\W])\S*$/';

            if(isset($_POST['pass'])){
                if(preg_match($m, $_POST['pass'])){
                    echo 'Le mot de passe choisi convient';
                }else{
                    echo 'Le mot de passe choisi ne répond pas aux critères';
                }
            }

        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on commence par créer un formulaire qui demande un mot de passe aux utilisateurs en utilisant la méthode `post` et en envoyant les données reçues vers la page courante pour traitement avec `action='cours.php'`.

Les données envoyées vont être automatiquement stockées dans la superglobale `$_POST` et on va pouvoir y accéder côté PHP en indiquant `$_POST['pass']`.

Côté traitement PHP, on s'assure déjà qu'une valeur a bien été envoyée grâce à la ligne `if(isset($_POST['pass']))`. Si une valeur a bien été envoyée, le test de notre condition est validé et on rentre dedans. Dans le cas contraire, le test échoue et rien n'est affiché.

On utilise ensuite un masque plus complexe que ce qu'on a pu voir jusqu'à présent et qui nous permet de tester qu'une chaine ne contient pas d'espace et contient bien au moins 8 caractères avec au moins une majuscule, un chiffre et un caractère spécial.

Pour cela, on commence par utiliser `^\S*` et `\S*$` qui indique qu'on attend n'importe quel caractère à l'exception d'un caractère blanc 0 fois ou plus en début et en fin de chaine.

Ensuite, on utilise des assertions qui, je vous le rappelle, ne sont pas capturantes par défaut :

- L'assertion `(?=\S{8,})` permet de s'assurer que la chaine reçue fait au moins 8 caractères ;
- L'assertion `(?=\S*[A-Z])` permet de s'assurer que la chaine reçue possède au moins une lettre appartenant à l'intervalle de classe `[A-Z]`, c'est-à-dire au moins une lettre majuscule ;
- L'assertion `(?=\S*[\d])` permet de s'assurer que la chaine reçue possède au moins un chiffre ;
- L'assertion `(?=\S*[\W])` permet de s'assurer que la chaine reçue possède au moins un caractère spécial.

Encore une fois, ce masque est beaucoup plus complexe que tout ce qu'on a pu voir jusqu'à présent et nous commençons à utiliser différentes fonctionnalités du PHP ensemble dans cet exemple.

C’est donc tout à fait normal si cela vous semble « impossible à réaliser seul » de premier abord. Essayez simplement pour le moment de prendre un maximum de temps pour bien comprendre les différentes parties du masque ici et le reste viendra avec la pratique.

PARTIE X

Programmation orientée objet : concepts de base

Introduction à la programmation orientée objet en PHP

Dans cette nouvelle partie, nous allons redécouvrir le PHP sous un nouvel angle avec la programmation orientée objet. La programmation orientée objet est une façon différente de coder qui va suivre des règles différentes et va amener une syntaxe différente, ce qui fait qu'elle peut être perçue comme difficile à comprendre pour des débutants.

Je vais essayer de vous présenter le PHP orienté objet étape par étape et vous conseille fortement de ne pas faire l'impasse sur cette partie car cette nouvelle façon d'écrire son code va posséder des avantages indéniables qu'on va illustrer ensemble et est l'une des grandes forces du PHP.

Par ailleurs, il convient de noter que de nombreux langages serveurs possèdent une écriture orientée objet car encore une fois cette façon de coder va s'avérer très puissante.

Qu'est-ce que la programmation orientée objet (POO) ?

La programmation orientée objet (ou POO en abrégé) correspond à une autre manière d'imaginer, de construire et d'organiser son code.

Jusqu'à présent, nous avons codé de manière procédurale, c'est-à-dire en écrivant une suite de procédures et de fonctions dont le rôle était d'effectuer différentes opérations sur des données généralement contenues dans des variables et ceci dans leur ordre d'écriture dans le script.

La programmation orientée objet est une façon différente d'écrire et d'arranger son code autour de classes et d'objets qu'on va créer à partir de ces classes. Une classe est une entité qui va pouvoir contenir un ensemble de fonctions et de variables.

Les intérêts principaux de la programmation orientée objet vont être une structure générale du code plus claire, plus modulable et plus facile à maintenir et à déboguer.

La programmation orientée objet va introduire des syntaxes différentes de ce qu'on a pu voir jusqu'à présent et c'est l'une des raisons principales pour lesquelles le POO en PHP est vu comme une chose obscure et compliquée par les débutants.

Au final, si vous arrivez à comprendre cette nouvelle syntaxe et si vous faites l'effort de comprendre les nouvelles notions qui vont être amenées, vous allez vous rendre compte que la POO n'est pas si complexe : ce n'est qu'une façon différente de coder qui va amener de nombreux avantages.

Pour vous donner un aperçu des avantages concrets de la POO, rappelez-vous du moment où on a découvert les fonctions prêtes à l'emploi en PHP. Aujourd'hui, on les utilise constamment car celles-ci sont très pratiques : elles vont effectuer une tâche précise sans qu'on ait à imaginer ni à écrire tout le code qui les fait fonctionner.

Maintenant, imaginez qu'on dispose de la même chose avec les objets : ce ne sont plus des fonctions mais des ensembles de fonctions et de variables enfermées dans des objets

et qui vont effectuer une tâche complexe qu'on va pouvoir utiliser directement pour commencer à créer des scripts complexes et complets !

Classes, objets et instance : première approche

La programmation orientée objet se base sur un concept fondamental qui est que tout élément dans un script est un objet ou va pouvoir être considéré comme un objet. Pour comprendre ce qu'est précisément un objet, il faut avant tout comprendre ce qu'est une classe.

Une classe est un ensemble cohérent de code qui contient généralement à la fois des variables et des fonctions et qui va nous servir de plan pour créer des objets. Le but d'une classe va donc être de créer des objets similaires que nous allons ensuite pouvoir manipuler.

Il est généralement coutume d'illustrer ce que sont les objets et les classes en faisant des parallèles avec la vie de tous les jours. De manière personnelle, j'ai tendance à penser que ces parallèles embrouillent plus qu'ils n'aident à comprendre et je préfère donc vous fournir ici un exemple plus concret qui reste dans le cadre de la programmation.

Imaginons qu'on possède un site sur lequel les visiteurs peuvent s'enregistrer pour avoir accès à un espace personnel par exemple. Quand un visiteur s'enregistre pour la première fois, il devient un utilisateur du site.

Ici, on va essayer de comprendre comment faire pour créer le code qui permet cela. Pour information, ce genre de fonctionnalité est en pratique quasiment exclusivement réalisé en programmation orienté objet.

Qu'essaie-t-on de réaliser ici ? On veut « créer » un nouvel utilisateur à chaque fois qu'un visiteur s'enregistre à partir des informations qu'il nous a fournies. Un utilisateur va être défini par des attributs comme son nom d'utilisateur ou son mot de passe. En programmation, ces attributs vont être traduits par des variables.

Ensuite, un utilisateur va pouvoir réaliser certaines actions spécifiques comme se connecter, se déconnecter, modifier son profil, etc. En programmation, ces actions sont représentées par des fonctions.

A chaque fois qu'un visiteur s'inscrit et devient utilisateur, on va donc devoir créer des variables « nom d'utilisateur », « mot de passe », etc. et définir leurs valeurs et donner les permissions à l'utilisateur d'utiliser les fonctions connexion, déconnexion, etc.

Pour cela, on va créer un formulaire d'inscription sur notre site qui va demander un nom d'utilisateur et un mot de passe, etc. On va également définir les actions (fonctions) propres à nos utilisateurs : connexion, déconnexion, possibilité de commenter, etc.

Sur notre site, on s'attend à avoir régulièrement de nouveaux visiteurs qui s'inscrivent et donc de nouveaux utilisateurs. Il est donc hors de question de définir toutes ces choses manuellement à chaque fois.

A la place, on va plutôt créer un bloc de code qui va initialiser nos variables nom d'utilisateur et mot de passe par exemple et qui va définir les différentes actions que va pouvoir faire un utilisateur.

Ce bloc de code est le plan de base qui va nous servir à créer un nouvel utilisateur. On va également dire que c'est une classe. Dès qu'un visiteur s'inscrit, on va pouvoir créer un nouvel objet « utilisateur » à partir de cette classe et qui va disposer des variables et fonctions définies dans la classe. Lorsqu'on crée un nouvel objet, on dit également qu'on « instancie » ou qu'on crée une instance de notre classe.

Une classe est donc un bloc de code qui va contenir différentes variables, fonctions et éventuellement constantes et qui va servir de plan de création pour des objets similaires. Chaque objet créé à partir d'une même classe dispose des mêmes variables, fonctions et constantes définies dans la classe mais va pouvoir les implémenter différemment.

Classes et objets : exemple de création

Reprenons notre exemple précédent et créons une première classe qu'on va appeler `Utilisateur`. Bien évidemment, nous n'allons pas créer tout un script de connexion utilisateur ici, mais simplement définir une première classe très simple.

En PHP, on crée une nouvelle classe avec le mot clef `class`. On peut donner n'importe quel nom à une nouvelle classe du moment qu'on n'utilise pas un mot réservé du PHP et que le premier caractère du nom de notre classe soit une lettre ou un underscore.

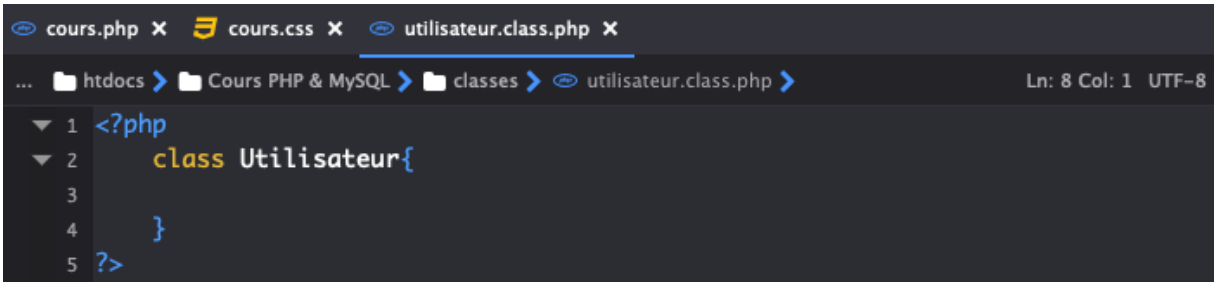
Par convention, on placera généralement chaque nouvelle classe créée dans un fichier à part et on placera également tous nos fichiers de classe dans un dossier qu'on pourra appeler `classes` par exemple pour plus de simplicité.

Comme je vous l'ai dit plus haut, l'un des grands avantages de la POO se situe dans la clarté du code produit et cette clarté est notamment le résultat d'une bonne séparation du code.

On n'aura ensuite qu'à inclure les fichiers de classes nécessaires à l'exécution de notre script principal dans celui-ci grâce à une instruction `require` par exemple.

On va donc créer un nouveau fichier en plus de notre fichier principal `cours.php` qu'on va appeler `utilisateur.class.php`. Notez qu'on appellera généralement nos fichiers de classe « maClasse.class.php » afin de bien les différencier des autres et par convention une nouvelle fois.

Dans ce fichier de classe, nous allons donc pour le moment tout simplement créer une classe `Utilisateur` avec le mot clef `class`.



Pour le moment, notre classe est vide. Vous pouvez remarquer que la syntaxe générale de déclaration d'une classe ressemble à ce qu'on a déjà vu avec les fonctions.

Nous allons également directement en profiter pour inclure notre classe dans notre fichier principal `cours.php` avec une instruction `require`. Ici, mon fichier de classe est dans un sous-dossier « classes » par rapport à mon fichier principal.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ensuite, nous allons rajouter la ligne suivante dans notre script principal :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';

            new Utilisateur();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ci-dessus, nous avons créé ce qu'on appelle une nouvelle instance de notre classe `Utilisateur`.

La syntaxe peut vous sembler particulière et c'est normal : rappelez-vous que je vous ai dit que le PHP orienté objet utilisait une syntaxe différente du PHP conventionnel.

Ici, le mot clef `new` est utilisé pour instancier une classe c'est-à-dire créer une nouvelle instance d'une classe.

Une instance correspond à la « copie » d'une classe. Le grand intérêt ici est qu'on va pouvoir effectuer des opérations sur chaque instance d'une classe sans affecter les autres instances.

Par exemple, imaginons que vous créiez deux nouveaux fichiers avec le logiciel Word. Chaque fichier va posséder les mêmes options : vous allez pouvoir changer la police, la taille d'écriture, etc. Cependant, le fait de personnaliser un fichier ne va pas affecter la mise en page du deuxième fichier.

A chaque fois qu'on instancie une classe, un objet est également automatiquement créé. Les termes « instance de classe » et « objet » ne désignent pas fondamentalement la même chose mais dans le cadre d'une utilisation pratique on pourra très souvent les confondre et c'est ce que nous allons faire dans ce cours.

Pour information, la grande différence est que chaque instance de classe est unique et peut donc être identifiée de manière unique ce qui n'est pas le cas pour les objets d'une même classe.

Lorsqu'on instancie une classe, un objet est donc créé. Nous allons devoir capturer cet objet pour l'utiliser. Pour cela, nous allons généralement utiliser une variable qui deviendra alors une « variable objet » ou plus simplement un « objet ».

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';

            $pierre = new Utilisateur();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Pour être tout à fait précis, notre variable en question ne va exactement contenir l'objet en soi mais plutôt une référence à l'objet. Nous reparlerons de ce point relativement complexe en fin de partie et allons pour le moment considérer que notre variable contient notre objet.

Classes, instances et objets : l'essentiel à retenir

Une classe est un « plan d'architecte » qui va nous permettre de créer des objets qui vont partager ses variables et ses fonctions. Chaque objet créé à partir d'une classe va disposer des mêmes variables et fonctions définies dans la classe mais va pouvoir implémenter ses propres valeurs.

Pour l'instant, nous n'avons créé qu'une classe vide et donc il est possible que vous ne compreniez pas encore l'intérêt des classes. Dès le chapitre suivant, nous allons inclure des variables et des fonctions dans notre classe et plus le cours va avancer, plus vous allez comprendre les différents avantages de programmer en orienté objet et des classes. Je ne peux juste pas tout vous révéler d'un coup, il va falloir y aller brique après brique.

Pour créer un objet, il faut instancier la classe en utilisant le mot clef `new`. Une instance est une « copie » de classe. On va stocker notre instance ou notre objet dans une variable pour pouvoir l'utiliser.

Propriétés et méthodes en PHP orienté objet

Dans cette nouvelle leçon, nous allons définir ce que sont les propriétés et les méthodes et allons ajouter des propriétés et des méthodes à notre classe `Utilisateur` créée dans la leçon précédente afin de tendre vers une classe fonctionnelle.

Les propriétés : définition et usage

Dans le chapitre précédent, nous avons créé une classe `Utilisateur` qui ne contenait pas de code.

Nous avons également dit qu'une classe servait de plan pour créer des objets. Vous pouvez donc en déduire qu'une classe vide ne sert pas à grand-chose.

Nous allons donc maintenant rendre notre classe plus fonctionnelle en lui ajoutant des éléments.

On va déjà pouvoir créer des variables à l'intérieur de nos classes. Les variables créées dans les classes sont appelées des propriétés, afin de bien les différencier des variables « classiques » créées en dehors des classes.

Une propriété, c'est donc tout simplement une variable définie dans une classe (ou éventuellement ajoutée à un objet après sa création).

Reprenons immédiatement notre classe `Utilisateur` et ajoutons lui deux propriétés qu'on va appeler `$user_name` et `$user_pass` par exemple (pour « nom d'utilisateur » et « mot de passe utilisateur »).

```php
<?php
    class Utilisateur{
        public $user_name;
        public $user_pass;
    }
?>
```

Comme vous pouvez le voir, on déclare une propriété exactement de la même façon qu'une variable classique, en utilisant le signe `$`.

Le mot clef `public` signifie ici qu'on va pouvoir accès à nos propriétés depuis l'intérieur et l'extérieur de notre classe. Nous reparlerons de cela un peu plus tard, n'y prêtez pas attention pour le moment.

Ici, nous nous contentons de déclarer nos propriétés sans leur attribuer de valeur. Les valeurs des propriétés seront ici passées lors de la création d'un nouvel objet (lorsqu'un visiteur s'inscrira sur notre site).

Notez qu'il est tout-à-fait permis d'initialiser une propriété dans la classe, c'est-à-dire lui attribuer une valeur de référence à la condition que ce soit une valeur constante (elle doit pouvoir être évaluée pendant la compilation du code et ne doit pas dépendre d'autres facteurs déterminés lors de l'exécution du code).

En situation réelle, ici, ce seront les visiteurs qui, lors de leur inscription, vont déclencher la création de nouveaux objets (qui vont être créés via notre classe `Utilisateur`.

Bien évidemment, réaliser le script complet qui va permettre cela est hors de question à ce niveau du cours. Nous allons donc nous contenter dans la suite de cette partie de créer de nouveaux objets manuellement pour pouvoir illustrer les différents concepts et leur fonctionnement.

Créons donc deux objets `$pierre` et `$mathilde` à partir de notre classe `Utilisateur` puis définissons ensuite des valeurs spécifiques pour les propriétés `$user_name` et `$user_pass` pour chaque objet.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';

            $pierre = new Utilisateur();
            $mathilde = new Utilisateur();

            $pierre->user_name = 'Pierre';
            $pierre->user_pass = 'abcdef';

            $mathilde->user_name = 'Math';
            $mathilde->user_pass = 123456;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, vous pouvez à nouveau observer une syntaxe que nous ne connaissons pas encore qui utilise le symbole `->`. Expliquons ce qu'il se passe ici.

Avant tout, souvenez-vous que les objets créés à partir d'une classe partagent ses propriétés et ses méthodes puisque chaque objet contient une « copie » de la classe.

Pour accéder aux propriétés définies originellement dans la classe depuis nos objets, on utilise l'opérateur `->` qui est appelé opérateur objet.

Cet opérateur sert à indiquer au PHP qu'on souhaite accéder à un élément défini dans notre classe via un objet créé à partir de cette classe. Notez qu'on ne précise pas de signe `$` avant le nom de la propriété à laquelle on souhaite accéder dans ce cas.

Dans le cas présent, on va pouvoir accéder à notre propriété depuis notre objet (c'est-à-dire depuis l'extérieur de notre classe) car nous l'avons définie avec le mot clef `public`.

Notez cependant qu'il est généralement considéré comme une mauvaise pratique de laisser des propriétés de classes accessibles directement depuis l'extérieur de la classe et de mettre à jour leur valeur comme cela car ça peut poser des problèmes de sécurité dans le script.

Pour manipuler des propriétés depuis l'extérieur de la classe, nous allons plutôt créer des fonctions de classe dédiées afin que personne ne puisse directement manipuler nos propriétés et pour protéger notre script.

Les méthodes : définition et usage

Nous allons également pouvoir déclarer des fonctions à l'intérieur de nos classes.

Les fonctions définies à l'intérieur d'une classe sont appelées des méthodes. Là encore, nous utilisons un nom différent pour bien les différencier des fonctions créées en dehors des classes.

Une méthode est donc tout simplement une fonction déclarée dans une classe. On va pouvoir créer des méthodes dans nos classes dont le rôle va être d'obtenir ou de mettre à jour les valeurs de nos propriétés.

Dans notre classe `Utilisateur`, nous allons par exemple pouvoir créer trois méthodes qu'on va appeler `getNom()`, `setNom()` et `setPass()`.

Le rôle de `getNom()` va être de récupérer la valeur contenue dans la propriété `$user_name`.

Les rôles de `setNom()` et de `setPass()` vont être respectivement de définir ou de modifier la valeur contenue dans les propriétés `$user_name` et `$user_pass` relativement à l'objet courant (la valeur de la propriété ne sera modifiée que pour l'objet couramment utilisé).

Profitez-en pour noter que les méthodes qui servent à définir / modifier / mettre à jour une valeur sont appelées des `setters`. Généralement, on fera commencer leur nom par `set` afin de bien les identifier comme on l'a fait pour nos méthodes `setNom()` et `setPass()`.

De même, les méthodes qui servent à récupérer des valeurs sont appelées des `getters` et on fera commencer leur nom par `get`. Ces notations sont des conventions qui ont pour but de clarifier les scripts et de simplifier la vie des développeurs.

Rajoutons nos méthodes à l'intérieur de notre classe :

```php
<?php
    class Utilisateur{
        private $user_name;
        private $user_pass;

        public function getNom(){
            return $this->user_name;
        }

        public function setNom($new_user_name){
            $this->user_name = $new_user_name;
        }

        public function setPasse($new_user_pass){
            $this->user_pass = $new_user_pass;
        }
    }
?>
```

Il y a beaucoup de nouvelles choses dans ce code que nous allons décortiquer ligne par ligne.

Tout d'abord, vous pouvez commencer par noter qu'on a modifié le niveau d'accessibilité (c'est-à-dire la portée) de nos propriétés `$user_name` et `$user_pass` dans la définition de notre classe en modifiant le mot clef devant la déclaration de nos propriétés.

En effet, nous avons utilisé le mot clef `private` à la place de `public` qui signifie que nos propriétés ne sont désormais plus accessibles que depuis l'intérieur de la classe.

En revanche, nous avons utilisé le mot clef `$public` devant nos deux méthodes afin de pouvoir les utiliser depuis l'extérieur de la classe.

Ensuite, vous devriez également avoir remarqué qu'on utilise un nouveau mot clef dans ces deux méthodes : le mot clef `$this`. Ce mot clef est appelé pseudo-variable et sert à faire référence à l'objet couramment utilisé.

Cela signifie que lorsqu'on va appeler une méthode de classe depuis un objet, la pseudo-variable `$this` va être remplacée (substituée) par l'objet qui utilise la méthode actuellement.

Prenons un exemple concret afin que vous compreniez bien ce point important. Pour cela, retournons dans notre fichier de script principal et modifions quelques lignes comme cela :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';

            $pierre = new Utilisateur();
            $mathilde = new Utilisateur();

            $pierre->setNom('Pierre');
            $pierre->setPasse('abcdef');
            echo $pierre->getNom();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on inclut notre fichier de classe puis on instancie ensuite notre classe et on stocke cette instance dans un objet qu'on appelle `$pierre`.

On va pouvoir accéder à nos méthodes `setNom()`, `getNom()` et `setPass()` définies dans notre classe depuis notre objet puisqu'on a utilisé le mot clef `public` lorsqu'on les a déclarées.

En revanche, on ne va pas pouvoir accéder directement aux propriétés définies dans la classe puisqu'elles sont privées. On ne va donc pouvoir les manipuler que depuis l'intérieur de la classe via nos méthodes publiques.

On commence donc par utiliser nos méthodes `setNom()` et `setPass()` pour définir une valeur à stocker dans nos propriétés `$user_name` et `$user_pass`.

Pour cela, on utilise à nouveau l'opérateur objet `->` pour exécuter notre méthode depuis notre objet.

Les méthodes `setNom()` et `setPass()` vont chacune avoir besoin qu'on leur passe un argument pour fonctionner normalement. Ces arguments vont correspondre au nom d'utilisateur et au mot de passe choisis par l'utilisateur qu'on va stocker dans les propriétés `$user_name` et `$user_pass` relatives à notre instance.

Note : encore une fois, en pratique, nous recevrons ces données de l'utilisateur lui-même. Il faudra donc bien faire attention à les vérifier et à les traiter (comme n'importe quelle autre donnée envoyée par l'utilisateur) avant d'effectuer toute opération afin d'être sûr que les données envoyées sont conformes au format attendu et ne sont pas dangereuses. Pour le moment, nous nous passons de cette étape.

Comme je vous l'ai dit plus haut, la pseudo-variable `$this` dans le code de notre méthode sert à faire référence à l'objet couramment utilisé.

Dans le cas présent, la ligne `$this->user_name = $new_user_name` signifie littéralement que l'on souhaite stocker dans la propriété `$user_name` définie dans notre classe le contenu de `$new_user_name` (c'est-à-dire l'argument qui va être passé) POUR un objet en particulier et en l'occurrence ici pour `$pierre`. La pseudo-variable `$this` fait référence dans ce cas à `$pierre`.

De la même façon, on retourne le contenu de `$user_name` relatif à notre objet `$pierre` grâce à `getNom()`.

Créons immédiatement un deuxième objet afin de bien illustrer une nouvelle fois le fait que `$this` sert de référence à l'objet couramment utilisé.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';

            $pierre = new Utilisateur();
            $mathilde = new Utilisateur();

            $pierre->setNom('Pierre');
            $pierre->setPasse('abcdef');

            $mathilde->setNom('Math');
            $mathilde->setPasse(123456);

            echo $pierre->getNom(). '<br>';
            echo $mathilde->getNom(). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Comme vous pouvez le voir, on se sert ici de la même classe au départ qu'on va instancier plusieurs fois pour créer autant d'objets. Ces objets vont partager les propriétés et méthodes définies dans la classe qu'on va pouvoir utiliser de manière indépendante avec chaque objet.

Constructeur et destructeur d'objets

Dans cette nouvelle leçon, nous allons nous intéresser à deux méthodes spéciales qui vont pouvoir être définies dans nos classes qui sont les méthodes constructeur et destructeur.

La méthode constructeur : définition et usage

La méthode constructeur ou plus simplement le constructeur d'une classe est une méthode qui va être appelée (exécutée) automatiquement à chaque fois qu'on va instancier une classe.

Le constructeur va ainsi nous permettre d'initialiser des propriétés dès la création d'un objet, ce qui va pouvoir être très intéressant dans de nombreuses situations.

Pour illustrer l'intérêt du constructeur, reprenons notre class `Utilisateur` créée précédemment.

```php
<?php
    class Utilisateur{
        private $user_name;
        private $user_pass;

        public function getNom(){
            return $this->user_name;
        }

        public function setNom($new_user_name){
            $this->user_name = $new_user_name;
        }

        public function setPasse($new_user_pass){
            $this->user_pass = $new_user_pass;
        }
    }
?>
```

Notre classe possède deux propriétés `$user_name` et `$user_pass` et trois propriétés `getNom()`, `setNom()` et `setPasse()` dont les rôles respectifs sont de retourner le nom d'utilisateur de l'objet courant, de définir le nom d'utilisateur de l'objet courant et de définir le mot de passe de l'objet courant.

Lorsqu'on créé un nouvel objet à partir de cette classe, il faut ici ensuite appeler les méthodes `setNom()` et `setPasse()` pour définir les valeurs de nos propriétés `$user_name` et `$user_pass`, ce qui est en pratique loin d'être optimal.

Ici, on aimerait idéalement pouvoir définir immédiatement la valeur de nos deux propriétés lors de la création de l'objet (en récupérant en pratique les valeurs passées par l'utilisateur). Pour cela, on va pouvoir utiliser un constructeur.

```php
<?php
    class Utilisateur{
        private $user_name;
        private $user_pass;

        public function __construct($n, $p){
            $this->user_name = $n;
            $this->user_pass = $p;
        }

        public function getNom(){
            return $this->user_name;
        }
    }
?>
```

On déclare un constructeur de classe en utilisant la syntaxe `function __construct()`. Il faut bien comprendre ici que le PHP va rechercher cette méthode lors de la création d'un nouvel objet et va automatiquement l'exécuter si elle est trouvée.

Nous allons utiliser notre constructeur pour initialiser certaines propriétés de nos objets dont nous pouvons avoir besoin immédiatement ou pour lesquelles il fait du sens de les initialiser immédiatement.

Dans notre cas, on veut stocker le nom d'utilisateur et le mot de passe choisi dans nos variables `$user_name` et `$user_pass` dès la création d'un nouvel objet.

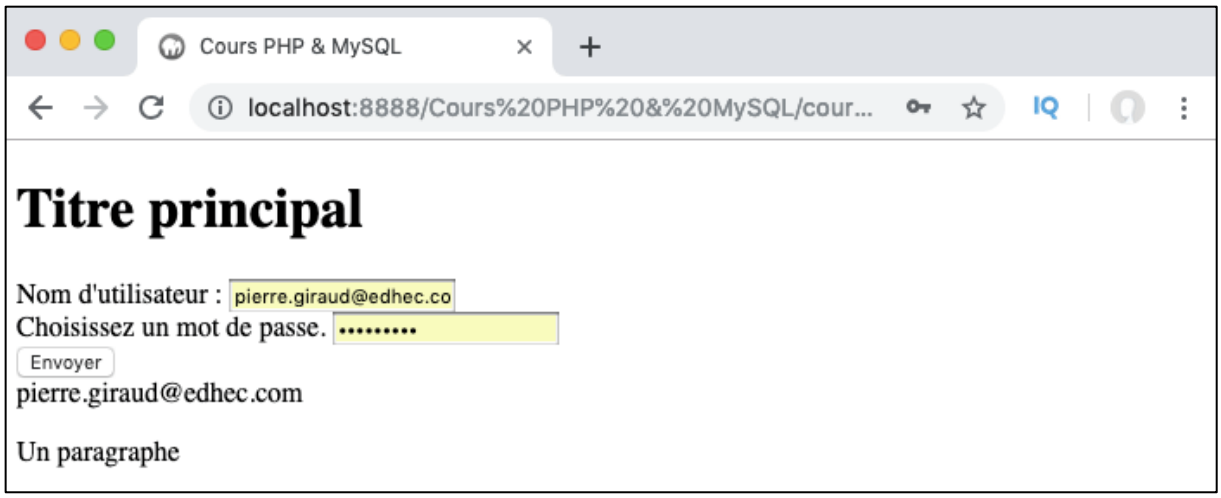
Pour cela, on va définir deux paramètres dans notre constructeur qu'on appelle ici `$n` et `$p`. Nous allons pouvoir passer les arguments à notre constructeur lors de l'instanciation de notre classe. On va ici passer un nom d'utilisateur et un mot de passe. Voici comment on va procéder en pratique :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';

            $pierre = new Utilisateur('Pierre', 'abcdef');
            $mathilde = new Utilisateur('Math', 123456);

            echo $pierre->getNom(). '<br>';
            echo $mathilde->getNom(). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Lors de l'instanciation de notre classe `Utilisateur`, le PHP va automatiquement rechercher une méthode `__construct()` dans la classe à instancier et exécuter cette méthode si elle est trouvée. Les arguments passés lors de l'instanciation vont être utilisés dans notre constructeur et vont ici être stockés dans `$user_name` et `$user_pass`.

Ici, on peut déjà avoir un premier aperçu d'un script « utile » en pratique si vous le désirez afin de bien comprendre à quoi vont servir les notions étudiées jusqu'à maintenant « en vrai » :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <form action='cours.php' method='post'>
            <label for='nom'>Nom d'utilisateur : </label>
            <input type='text' name='nom' id='nom'><br>
            <label for='pass'>Choisissez un mot de passe.</label>
            <input type='password' name='pass' id='pass'><br>
            <input type='submit' value='Envoyer'>
        </form>
        <?php
            require 'classes/utilisateur.class.php';
            //+ Vérification des données reçues (regex + filtres)
            //+ Stockage des données (base de données)
            $pierre = new Utilisateur($_POST['nom'], $_POST['pass']);
            echo $pierre->getNom(). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



On crée ici un formulaire en HTML qui va demander un nom d'utilisateur et un mot de passe à nos visiteurs. Vous pouvez imaginer que ce formulaire est un formulaire d'inscription. Ensuite, en PHP, on récupère les informations envoyées et on les utilise pour créer un nouvel objet. L'intérêt ici est que notre objet va avoir accès aux méthodes définies dans notre classe.

Bien évidemment, ici, notre script n'est pas complet puisqu'en pratique il faudrait analyser la cohérence des données envoyées et vérifier qu'elles ne sont pas dangereuses et car nous stockerions également les informations liées à un nouvel utilisateur en base de données pour pouvoir par la suite les réutiliser lorsque l'utilisateur revient sur le site et souhaite s'identifier.

La méthode destructeur

De la même façon, on va également pouvoir définir une méthode destructeur ou plus simplement un destructeur de classe.

La méthode destructeur va permettre de nettoyer les ressources avant que PHP ne libère l'objet de la mémoire.

Ici, vous devez bien comprendre que les variables-objets, comme n'importe quelle autre variable « classique », ne sont actives que durant le temps d'exécution du script puis sont ensuite détruites.

Cependant, dans certains cas, on voudra pouvoir effectuer certaines actions juste avant que nos objets ne soient détruits comme par exemple sauvegarder des valeurs de propriétés mises à jour ou fermer des connexions à une base de données ouvertes avec l'objet.

Dans ces cas-là, on va pouvoir effectuer ces opérations dans le destructeur puisque la méthode destructeur va être appelée automatiquement par le PHP juste avant qu'un objet ne soit détruit.

Il est difficile d'expliquer concrètement l'intérêt d'un destructeur ici à des personnes qui n'ont pas une connaissance poussée du PHP. Pas d'inquiétude donc si vous ne

comprenez pas immédiatement l'intérêt d'une telle méthode, on pourra illustrer cela de manière plus concrète lorsqu'on parlera des bases de données.

On va utiliser la syntaxe `function __destruct()` pour créer un destructeur. Notez qu'à la différence du constructeur, il est interdit de définir des paramètres dans un destructeur.

```php
<?php
    class Utilisateur{
        private $user_name;
        private $user_pass;

        public function __construct($n, $p){
            $this->user_name = $n;
            $this->user_pass = $p;
        }

        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            return $this->user_name;
        }
    }
?>
```

Encapsulation et visibilité des propriétés et méthodes

Dans cette nouvelle leçon, nous allons présenter le principe d'encapsulation et comprendre ses enjeux et intérêt et allons voir comment implémenter ce principe en pratique via les niveaux de visibilité des propriétés, méthodes et des constantes de classe.

Nous étudierons plus en détail les constantes dans une prochaine leçon, je n'en parlerai donc pas véritablement ici.

Le principe d'encapsulation

L'encapsulation désigne le principe de regroupement des données et du code qui les utilise au sein d'une même unité. On va très souvent utiliser le principe d'encapsulation afin de protéger certaines données des interférences extérieures en forçant l'utilisateur à utiliser les méthodes définies pour manipuler les données.

Dans le contexte de la programmation orientée objet en PHP, l'encapsulation correspond au groupement des données (propriétés, etc.) et des données permettant de les manipuler au sein d'une classe.

L'encapsulation va ici être très intéressante pour empêcher que certaines propriétés ne soient manipulées depuis l'extérieur de la classe. Pour définir qui va pouvoir accéder aux différentes propriétés, méthodes et constantes de nos classes, nous allons utiliser des limiteurs d'accès ou des niveaux de visibilité qui vont être représentés par les mots clefs `public`, `private` et `protected`.

Une bonne implémentation du principe d'encapsulation va nous permettre de créer des codes comportant de nombreux avantages. Parmi ceux-ci, le plus important est que l'encapsulation va nous permettre de garantir l'intégrité de la structure d'une classe en forçant l'utilisateur à passer par un chemin prédéfini pour modifier une donnée.

Le principe d'encapsulation est l'un des piliers de la programmation orientée objet et l'un des concepts fondamentaux, avec l'héritage, le l'orienté objet en PHP.

Le principe d'encapsulation et la définition des niveaux de visibilité devra être au centre des préoccupations notamment lors de la création d'une interface modulable comme par exemple la création d'un site auquel d'autres développeurs vont pouvoir ajouter des fonctionnalités comme WordPress ou PrestaShop (avec les modules) ou lors de la création d'un module pour une interface modulable.

L'immense majorité de ces structures sont construits en orienté objet car c'est la façon de coder qui présente la plus grande modularité et qui permet la maintenance la plus facile car on va éclater notre code selon différentes classes. Il faudra néanmoins bien réfléchir à qui peut avoir accès à tel ou tel élément de telle classe afin de garantir l'intégrité de la structure et éviter des conflits entre des éléments de classes (propriétés, méthodes, etc.).

Les niveaux de visibilité des propriétés, méthodes et constantes en POO PHP

On va pouvoir définir trois niveaux de visibilité ou d'accessibilité différents pour nos propriétés, méthodes et constantes (depuis PHP 7.1.0) grâce aux mots clefs `public`, `private` et `protected`.

Les propriétés, méthodes ou constantes définies avec le mot clef `public` vont être accessibles partout, c'est-à-dire depuis l'intérieur ou l'extérieur de la classe.

Les propriétés, méthodes ou constantes définies avec le mot clef `private` ne vont être accessibles que depuis l'intérieur de la classe qui les a définies.

Les propriétés, méthodes ou constantes définies avec le mot clef `protected` ne vont être accessibles que depuis l'intérieur de la classe qui les a définies ainsi que depuis les classes qui en héritent ou la classe parente. Nous reparlerons du concept d'héritage dans la prochaine leçon.

Lors de la définition de propriétés dans une classe, il faudra obligatoirement définir un niveau de visibilité pour chaque propriété. Dans le cas contraire, une erreur sera renvoyée.

Pour les méthodes et constantes, en revanche, nous ne sommes pas obligés de définir un niveau de visibilité même si je vous recommande fortement de la faire à chaque fois. Les méthodes et constantes pour lesquelles nous n'avons défini aucun niveau de visibilité de manière explicite seront définies automatiquement comme publiques.

Ici, lorsqu'on parle « d'accès », on se réfère à l'endroit où les propriétés et méthodes sont utilisées. Reprenons l'exemple de notre classe utilisateur pour bien comprendre :

```php
<?php
    class Utilisateur{
        private $user_name;
        private $user_pass;

        public function __construct($n, $p){
            $this->user_name = $n;
            $this->user_pass = $p;
        }

        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            return $this->user_name;
        }
    }
?>
```

Notre classe possède deux propriétés définies comme `private` et trois méthodes définies comme `public`. « L'intérieur » de la classe correspond au code ci-dessus.

Ici, on s'aperçoit par exemple que notre constructeur manipule nos propriétés `$user_name` et `$user_pass`. Il en a le droit puisque le constructeur est également défini dans la classe, tout comme la méthode `getNom()`.

Nos méthodes sont ici définies comme publiques, ce qui signifie qu'on va pouvoir les exécuter depuis l'extérieur de la classe. Lorsqu'on crée un nouvel objet dans notre script principal à partir de notre classe, par exemple, on appelle (implicitement) le constructeur depuis l'extérieur de la classe.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';

            $pierre = new Utilisateur('Pierre', 'abcdef');
            $mathilde = new Utilisateur('Math', 123456);

            echo $pierre->getNom(). '<br>';
            echo $mathilde->getNom(). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

On a le droit de la faire puisque notre constructeur est défini comme public. Ensuite, notre constructeur va modifier la valeur de nos propriétés depuis l'intérieur de la classe et c'est cela qu'il faut bien comprendre : on peut ici modifier la valeur de nos propriétés indirectement car c'est bien notre constructeur défini dans la classe qui les modifie.

La même chose se passe avec la méthode `getNom()` qui affiche la valeur de la propriété `$user_name`. Cette méthode est définie comme publique, ce qui signifie qu'on peut l'appeler depuis l'extérieur de la classe. Ensuite, notre méthode va récupérer la valeur de `$user_name` depuis l'intérieur de la classe puisque c'est là qu'elle a été définie.

L'idée à retenir est qu'on ne peut pas accéder directement à nos propriétés définies comme privées depuis notre script principal c'est-à-dire depuis l'extérieur de la classe. Il faut qu'on passe par nos fonctions publiques qui vont pouvoir les manipuler depuis l'intérieur de la classe.

Si vous essayez par exemple d'afficher directement le contenu de la propriété `$user_name` en écrivant `echo $pierre->user_name` dans notre script principal, par exemple, l'accès va être refusé et une erreur va être renvoyée. En revanche, si on définit notre propriété comme publique, ce code fonctionnera normalement.

Comment bien choisir le niveau de visibilité des différents éléments d'une classe

Vous l'aurez compris, la vraie difficulté ici va être de déterminer le « bon » niveau de visibilité des différents éléments de notre classe.

Cette problématique est relativement complexe et d'autant plus pour un débutant car il n'y a pas de directive absolue. De manière générale, on essaiera toujours de protéger un maximum notre code de l'extérieur et donc de définir le niveau d'accessibilité minimum possible.

Ensuite, il va falloir s'interroger sur le niveau de sensibilité de chaque élément et sur les impacts que peuvent avoir chaque niveau d'accès à un élément sur le reste d'une classe tout en identifiant les différents autres éléments qui vont avoir besoin d'accéder à cet élément pour fonctionner.

Ici, il n'y a vraiment pas de recette magique : il faut avoir une bonne expérience du PHP et réfléchir un maximum avant d'écrire son code pour construire le code le plus cohérent et le plus sécurisé possible. Encore une fois, cela ne s'acquière qu'avec la pratique.

Pour le moment, vous pouvez retenir le principe suivant qui fonctionnera dans la majorité des cas (mais qui n'est pas un principe absolu, attention) : on définira généralement nos méthodes avec le mot clef `public` et nos propriétés avec les mots clefs `protected` ou `private`.

Classes étendues et héritage

Dans cette nouvelle leçon, nous allons voir comment étendre une classe et comprendre les intérêts qu'il va y avoir à faire cela. Nous expliquerons également comment fonctionne l'héritage dans le cadre de la programmation orientée objet en PHP.

Étendre une classe : principe et utilité

Vous devriez maintenant normalement commencer à comprendre la syntaxe générale utilisée en POO PHP.

Un des grands intérêts de la POO est qu'on va pouvoir rendre notre code très modulable, ce qui va être très utile pour gérer un gros projet ou si on souhaite le distribuer à d'autres développeurs.

Cette modularité va être permise par le principe de séparation des classes qui est à la base même du PHP et par la réutilisation de certaines classes ou par l'implémentation de nouvelles classes en plus de classes de base déjà existantes.

Sur ce dernier point, justement, il va être possible plutôt que de créer des classes complètement nouvelles d'étendre (les possibilités) de classes existantes, c'est-à-dire de créer de nouvelles classes qui vont hériter des méthodes et propriétés de la classe qu'elles étendent (sous réserve d'y avoir accès) tout en définissant de nouvelles propriétés et méthodes qui leur sont propres.

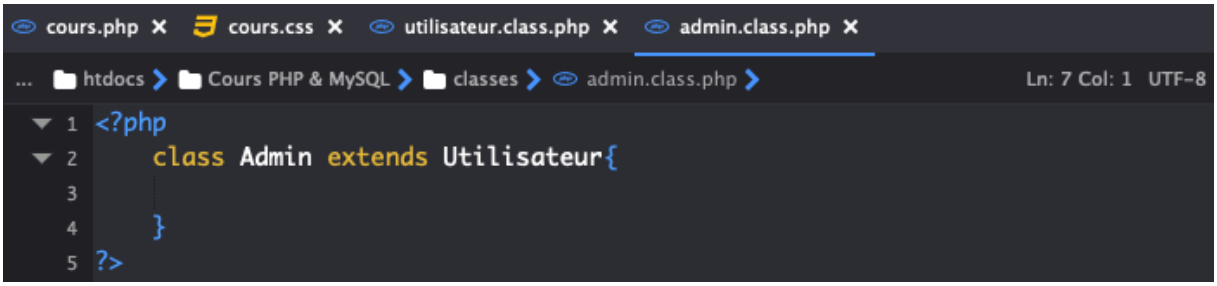
Certains développeurs vont ainsi pouvoir proposer de nouvelles fonctionnalités sans casser la structure originale de notre code et de nos scripts. C'est d'ailleurs tout le principe de la solution e-commerce PrestaShop (nous reparlerons de cela en fin de chapitre).

Comment étendre une classe en pratique

Nous allons pouvoir étendre une classe grâce au mot clef `extends`. En utilisant ce mot clef, on va créer une classe « fille » qui va hériter de toutes les propriétés et méthodes de son parent par défaut et qui va pouvoir les manipuler de la même façon (à condition de pouvoir y accéder).

Illustrons immédiatement cela en créant une nouvelle classe `Admin` qui va étendre notre classe `Utilisateur` définie dans les leçons précédentes par exemple.

Nous allons créer cette classe dans un nouveau fichier en utilisant le mot clef `extends` comme cela :



Notre classe `Admin` étend la classe `Utilisateur`. Elle hérite et va pouvoir accéder à toutes les méthodes et aux propriétés de notre classe `Utilisateur` qui n'ont pas été définies avec le mot clef `private`.

Nous allons désormais pouvoir créer un objet à partir de notre classe `Admin` et utiliser les méthodes publiques définies dans notre classe `Utilisateur` et dont hérite `Admin`.

Attention cependant : afin d'être utilisées, les classes doivent déjà être connues et la classe mère doit être définie avant l'écriture d'un héritage. Il faudra donc bien penser à inclure les classes mère et fille dans le fichier de script principal en commençant par la mère.

Les classes étendues et la visibilité

Dans le cas présent, notre classe mère `Utilisateur` possède deux propriétés avec un niveau de visibilité défini sur `private` et trois méthodes dont le niveau de visibilité est `public`.

Ce qu'il faut bien comprendre ici, c'est qu'on ne va pas pouvoir accéder aux propriétés de la classe `Utilisateur` depuis la classe étendue `Admin` : comme ces propriétés sont définies comme privées, elles n'existent que dans la classe `Utilisateur`.

Comme les méthodes de notre classe mère sont définies comme publiques, cependant, notre classe fille va en hériter et les objets créés à partir de la classe étendue vont donc pouvoir utiliser ces méthodes pour manipuler les propriétés de la classe mère.

Notez par ailleurs ici que si une classe fille ne définit pas de constructeur ni de destructeur, ce sont les constructeur et destructeur du parent qui vont être utilisés.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';

            $pierre = new Admin('Pierre', 'abcdef');
            $mathilde = new Utilisateur('Math', 123456);

            echo $pierre->getNom(). '<br>';
            echo $mathilde->getNom(). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on crée deux objets `$pierre` et `$mathilde`. Notre objet `$pierre` est créé en instanciant la classe étendue `Admin`.

Notre classe `Admin` est pour le moment vide. Lors de l'instanciation, on va donc utiliser le constructeur de la classe parent `Utilisateur` pour initialiser les propriétés de notre objet. Cela fonctionne bien ici puisqu'encore une fois le constructeur est défini à l'intérieur de la classe parent et peut donc accéder aux propriétés de cette classe, tout comme la fonction `getNom()`.

Cependant, si on essaie maintenant de manipuler les propriétés de notre classe parent depuis la classe `Admin`, cela ne fonctionnera pas car les propriétés sont définies comme privées dans la classe mère et ne vont donc exister que dans cette classe.

Si on définit une nouvelle méthode dans la classe `Admin` dont le rôle est de renvoyer la valeur de `$user_name` par exemple, le PHP va chercher une propriété `$user_name` dans la classe `Admin` et ne va pas la trouver.

Il va se passer la même chose si on réécrit une méthode de notre classe parent dans notre classe parent et qu'on tente de manipuler une propriété privée de la classe parent dedans, alors le PHP renverra une erreur.

Note : lorsqu'on redéfinit une méthode (non privée) ou une propriété (non privée) dans une classe fille, on dit qu'on « surcharge » celle de la classe mère. Cela signifie que les objets créés à partir de la classe fille utiliseront les définitions de la classe fille plutôt que celles de la classe mère.

Regardez plutôt l'exemple ci-dessous :

```php
<?php
    class Admin extends Utilisateur{
        ///On tente d'afficher $user_name qui n'existe pas dans Admin
        public function getNom2(){
            return $this->user_name;
        }

        /*On surcharge la méthode getNom() de Utilisateur. Ici, on conserve
         *le même code dans la méthode mais c'est cette méthode qui sera
         *utilisée par $pierre*/
        public function getNom(){
            return $this->user_name;
        }
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';

            $pierre = new Admin('Pierre', 'abcdef');
            $mathilde = new Utilisateur('Math', 123456);

            echo $pierre->getNom2(). '<br>';
            echo $pierre->getNom(). '<br>';
            echo $mathilde->getNom(). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, la valeur de `$user_name` de l'objet `$pierre` n'est jamais renvoyée puisqu'on essaie de la manipuler depuis la classe étendue `Admin`, ce qui est impossible.

Si on souhaite que des classes étendues puissent manipuler les propriétés d'une classe mère, alors il faudra définir le niveau de visibilité de ces propriétés comme `protected` dans la classe mère.

```php
<?php
    class Utilisateur{
        protected $user_name;
        protected $user_pass;

        public function __construct($n, $p){
            $this->user_name = $n;
            $this->user_pass = $p;
        }

        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            return $this->user_name;
        }
    }
?>
```

En définissant nos propriétés `$user_name` et `$user_pass` comme `protected` dans la classe `Utilisateur`, notre classe fille peut tout à fait les manipuler et les méthodes des classes étendues utilisant ces propriétés vont fonctionner normalement.

Définition de nouvelles propriétés et méthodes dans une classe étendue et surcharge

L'intérêt principal d'étendre des classes plutôt que d'en définir de nouvelles se trouve dans la notion d'héritage des propriétés et des méthodes : chaque classe étendue va hériter des propriétés et des méthodes (non privées) de la classe mère.

Cela permet donc une meilleure maintenance du code (puisqu'en cas de changement il suffit de modifier le code de la classe mère) et fait gagner beaucoup de temps dans l'écriture du code.

Cependant, créer des classes filles qui sont des « copies » d'une classe mère n'est pas très utile. Heureusement, bien évidemment, nous allons également pouvoir définir de

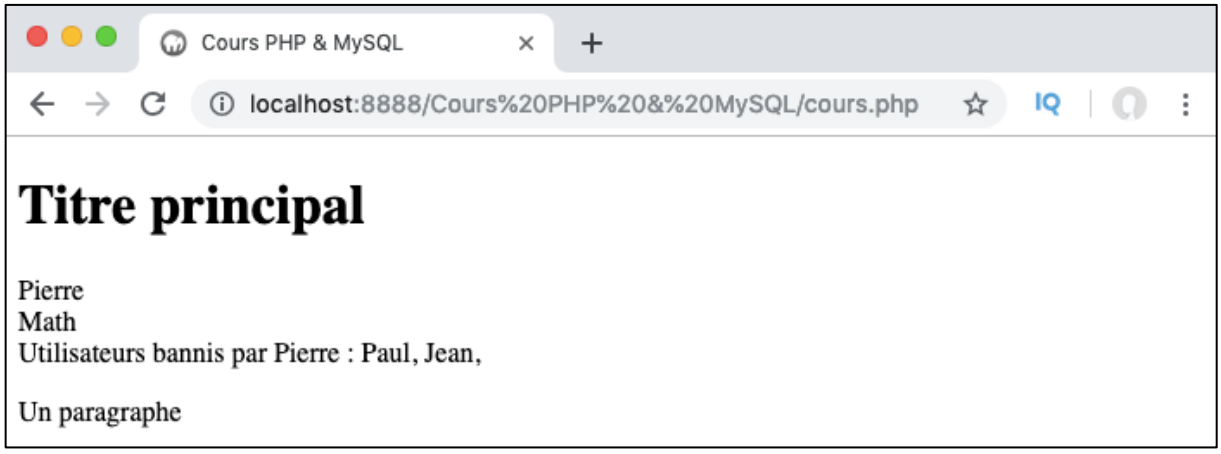
nouvelles propriétés et méthodes dans nos classes filles et ainsi pouvoir « étendre » les possibilités de notre classe de départ.

Ici, nous pouvons par exemple définir de nouvelles propriétés et méthodes spécifiques à notre classe `Admin`. On pourrait par exemple permettre aux objets de la classe `Admin` de bannir un utilisateur ou d'obtenir la liste des utilisateurs bannis.

Pour cela, on peut rajouter une propriété `$ban` qui va contenir la liste des utilisateurs bannis ainsi que deux méthodes `setBan()` et `getBan()`. Nous n'allons évidemment ici pas véritablement créer ce script mais simplement créer le code pour ajouter un nouveau prénom dans `$ban` et pour afficher le contenu de la propriété.

```php
<?php
    class Admin extends Utilisateur{
        protected $ban;

        public function setBan($b){
            $this->ban[] .= $b;
        }
        public function getBan(){
            echo 'Utilisateurs bannis par '.$this->user_name. ' : ';
            foreach($this->ban as $valeur){
                echo $valeur .', ';
            }
        }
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';

            $pierre = new Admin('Pierre', 'abcdef');
            $mathilde = new Utilisateur('Math', 123456);

            echo $pierre->getNom(). '<br>';
            echo $mathilde->getNom(). '<br>';

            $pierre->setBan('Paul');
            $pierre->setBan('Jean');
            echo $pierre->getBan();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



En plus de définir de nouvelles propriétés et méthodes dans nos classes étendues, nous allons également pouvoir surcharger, c'est-à-dire redéfinir certaines propriétés ou méthodes de notre classe mère dans nos classes filles. Pour cela, il va nous suffire de déclarer à nouveau la propriété ou la méthode en question en utilisant le même nom et en lui attribuant une valeur ou un code différent.

Dans ce cas-là, il va cependant falloir respecter quelques règles notamment au niveau de la définition de la visibilité qui ne devra jamais être plus réduite dans la définition

surchargée par rapport à la définition de base. Nous reparlerons de la surcharge dans la prochaine leçon et je vais donc laisser ce sujet de côté pour le moment.

Finalement, notez que rien ne nous empêche d'étendre à nouveau une classe étendue. Ici, par exemple, on pourrait tout à fait étendre notre classe `Admin` avec une autre classe `SuperAdmin`.

L'héritage va alors traverser les générations : les classes filles de `Admin` hériteront des méthodes et propriétés non privées de `Admin` mais également de celles de leur grand parent `Utilisateur`.

Comprendre la puissance et les risques liés aux classes étendues à travers l'exemple de la solution e-commerce PrestaShop

L'architecture du célèbre logiciel e-commerce PrestaShop a été créée en PHP orienté objet.

Cela rend PrestaShop modulable à l'infini et permet à des développeurs externes de développer de nouvelles fonctionnalités pour la solution.

En effet, le logiciel PrestaShop de base contient déjà de nombreuses classes et certaines vont pouvoir être étendues par des développeurs externes tandis que d'autres, plus sensibles ou essentielles au fonctionnement de la solution (ce qu'on appelle des classes « cœurs ») ne vont offrir qu'un accès limité.

Le fait d'avoir créé PrestaShop de cette manière est une formidable idée puisque ça permet aux développeurs de développer de nouveaux modules qui vont s'intégrer parfaitement à la solution en prenant appui sur des classes déjà existantes dans PrestaShop.

Cependant, c'est également le point principal de risque et l'ambiguïté majeure par rapport à la qualité de cette solution pour deux raisons.

Le premier problème qui peut survenir est que certains développeurs peuvent par mégarde ou tout simplement par manque d'application surcharger (c'est-à-dire réécrire ou encore substituer) certaines méthodes ou propriétés de classes lorsqu'ils créent leurs modules et le faire d'une façon qui va amener des bugs et des problèmes de sécurité sur les boutiques en cas d'installation du module en question.

A priori, l'équipe de validation des modules de PrestaShop est là pour éviter que ce genre de modules passent et se retrouvent sur la place de marché officielle.

Le deuxième problème est beaucoup plus insidieux et malheureusement quasiment impossible à éviter : imaginons que vous installiez plusieurs modules de développeurs différents sur votre solution de PrestaShop de base.

Si vous êtes malchanceux, il est possible que certains d'entre eux tentent d'étendre une même classe et donc de surcharger les mêmes méthodes ou propriétés, ou encore utilisent un même nom en créant une nouvelle classe ou en étendant une classe existante.

Dans ce cas-là, il y aura bien entendu un conflit dans le code et selon la gravité de celui-ci cela peut faire totalement planter votre boutique. Le problème étant ici que vous n'avez aucun moyen d'anticiper cela à priori lorsque vous êtes simple marchand et non un développeur aguerri.

Finalement, notez que si vous créez une architecture en POO PHP et que vous laissez la possibilité à des développeurs externes de modifier ou d'étendre cette architecture, vous devrez toujours faire bien attention à proposer une rétrocompatibilité de votre code à chaque mise à jour importante.

En effet, imaginons que vous modifiiez une classe de votre architecture : vous devrez toujours faire en sorte que les codes d'autres développeurs utilisant cette classe avant la mise à jour restent valides pour ne pas que tout leur code plante lorsqu'ils vont eux-mêmes mettre la solution à jour (ou tout au moins les prévenir avant de mettre la mise à jour en production pour qu'ils puissent adapter leur code).

Surcharge et opérateur de résolution de portée

Dans cette nouvelle leçon, nous allons voir précisément ce qu'est la surcharge d'éléments dans le cadre du PHP orienté objet ainsi que les règles liées à la surcharge.

Nous allons également en profiter pour introduire l'opérateur de résolution de portée, un opérateur qu'il convient de connaitre car il va nous servir à accéder à divers éléments dans nos classes et notamment aux éléments surchargés, aux constantes et aux éléments statiques qu'on étudiera dans les prochaines leçons.

La surcharge de propriétés et de méthodes en PHP orienté objet

En PHP, on dit qu'on « surcharge » une propriété ou une méthode d'une classe mère lorsqu'on la redéfinit dans une classe fille.

Pour surcharger une propriété ou une méthode, il va falloir la redéclarer en utilisant le même nom. Par ailleurs, si on souhaite surcharger une méthode, il faudra également que la nouvelle définition possède le même nombre de paramètres.

De plus, notez qu'on ne va pouvoir surcharger que des méthodes et propriétés définies avec des niveaux de visibilité `public` ou `protected` mais qu'il va être impossible de surcharger des éléments définis comme `private` puisque ces éléments n'existent / ne sont accessibles que depuis la classe qui les déclare.

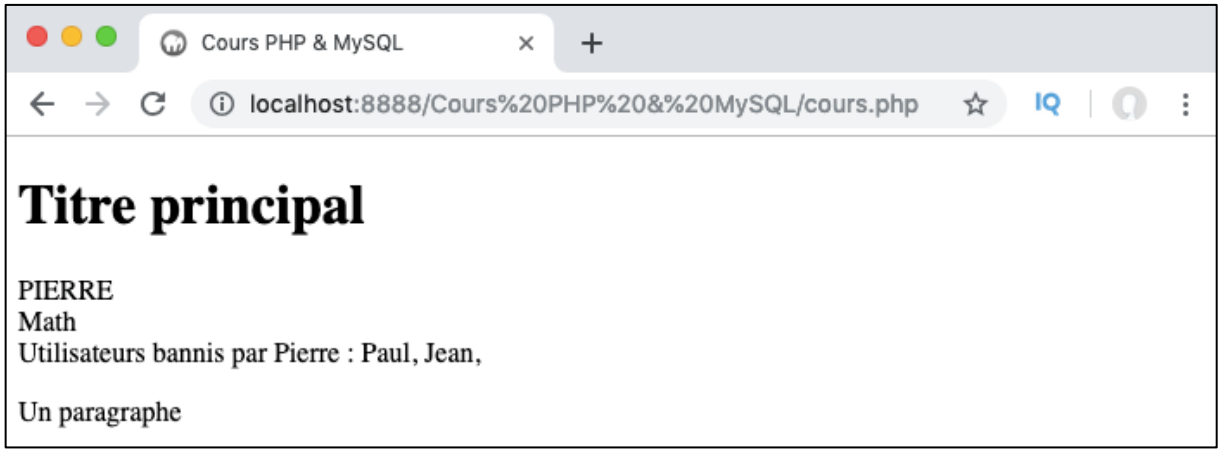
Finalement, notez que lorsqu'on surcharge une propriété ou une méthode, la nouvelle définition doit obligatoirement posséder un niveau de restriction de visibilité plus faible ou égal, mais ne doit en aucun cas avoir une visibilité plus restreinte que la définition de base. Par exemple, si on surcharge une propriété définie comme `protected`, la nouvelle définition de la propriété ne pourra être définie qu'avec `public` ou `protected` mais pas avec `private` qui correspond à un niveau de visibilité plus restreint.

Notez qu'il va être relativement rare d'avoir à surcharger des propriétés. Généralement, nous surchargerons plutôt les méthodes d'une classe mère depuis une classe fille. Prenons immédiatement un exemple concret en surchargeant la méthode `getNom()` de notre classe parent `Utilisateur` dans notre classe étendue `Admin`.

```php
<?php
    class Utilisateur{
        protected $user_name;
        protected $user_pass;

        public function __construct($n, $p){
            $this->user_name = $n;
            $this->user_pass = $p;
        }

        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            return $this->user_name;
        }
    }
?>
```

```php
<?php
    class Admin extends Utilisateur{
        protected $ban;

        public function getNom(){
            return strtoupper($this->user_name);
        }

        public function setBan($b){
            $this->ban[] .= $b;
        }
        public function getBan(){
            echo 'Utilisateurs bannis par '.$this->user_name. ' : ';
            foreach($this->ban as $valeur){
                echo $valeur .', ';
            }
        }
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';

            $pierre = new Admin('Pierre', 'abcdef');
            $mathilde = new Utilisateur('Math', 123456);

            echo $pierre->getNom(). '<br>';
            echo $mathilde->getNom(). '<br>';

            $pierre->setBan('Paul');
            $pierre->setBan('Jean');
            echo $pierre->getBan();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



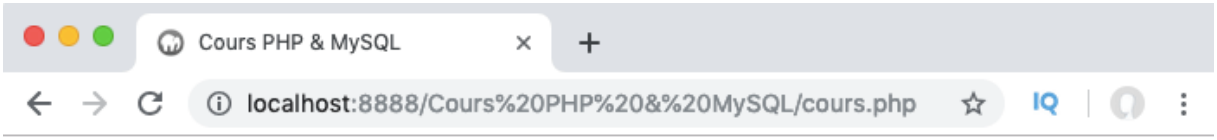
Ici, on modifie le code de notre méthode `getNom()` en transformant le résultat renvoyé en majuscules grâce à la fonction `strtoupper()` (« string to upper » ou « chaine en majuscules »).

Lorsqu'on appelle notre méthode depuis notre objet `$pierre` qui est un objet de la classe `Admin`, on voit bien que c'est la nouvelle définition de la méthode qui est exécutée.

On va de la même façon pouvoir surcharger le constructeur de la classe parent en définissant un nouveau constructeur dans la classe étendue. Dans le cas présent, on

pourrait définir le nom directement en majuscules pour les objets de la classe `Admin` afin qu'il s'affiche toujours en majuscules.

```php
<?php
    class Admin extends Utilisateur{
        protected $ban;

        public function __construct($n, $p){
            $this->user_name = strtoupper($n);
            $this->user_pass = $p;
        }

        public function setBan($b){
            $this->ban[] .= $b;
        }
        public function getBan(){
            echo 'Utilisateurs bannis par '.$this->user_name. ' : ';
            foreach($this->ban as $valeur){
                echo $valeur .', ';
            }
        }
    }
?>
```



Titre principal

PIERRE
Math
Utilisateurs bannis par PIERRE : Paul, Jean,

Un paragraphe

Notez que le PHP a une définition particulière de la surcharge par rapport à de nombreux autres langages. Pour de nombreux langages, « surcharger » une méthode signifie écrire différentes versions d'une même méthode avec un nombre différents de paramètres.

Accéder à une méthode ou une propriété surchargée grâce à l'opérateur de résolution de portée

Parfois, il va pouvoir être intéressant d'accéder à la définition de base d'une propriété ou d'une méthode surchargée. Pour faire cela, on va pouvoir utiliser l'opérateur de résolution de portée qui est symbolisé par le signe `::` (double deux points).

Nous allons également devoir utiliser cet opérateur pour accéder aux constantes et aux méthodes et propriétés définies comme statiques dans une classe (nous allons étudier tous ces concepts dans les prochains chapitres).

Pour le moment, concentrons-nous sur l'opérateur de résolution de portée et illustrons son fonctionnement dans le cas d'une méthode ou d'une propriété surchargée.

Nous allons pouvoir utiliser trois mots clefs pour accéder à différents éléments d'une classe avec l'opérateur de résolution de portée : les mots clefs `parent`, `self` et `static`.

Dans le cas où on souhaite accéder à une propriété ou à une méthode surchargée, le seul mot clef qui va nous intéresser est le mot clef `parent` qui va nous servir à indiquer qu'on souhaite accéder à la définition de la propriété ou de la méthode faite dans la classe mère.

Pour illustrer cela, nous allons modifier la méthode `getNom()` de notre classe mère `Utilisateur` afin qu'elle `echo` la nom de l'objet l'appelant plutôt qu'utiliser une instruction `return` (qui empêcherait l'exécution de tout code après l'instruction).

```php
<?php
    class Utilisateur{
        protected $user_name;
        protected $user_pass;

        public function __construct($n, $p){
            $this->user_name = $n;
            $this->user_pass = $p;
        }

        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            echo $this->user_name;
        }
    }
?>
```

Ensuite, on va surcharger notre méthode `getNom()` dans notre classe étendue `Admin`. La méthode de notre classe fille va reprendre le code de celle de la classe parent et ajouter de nouvelles instructions. Ici, plutôt que de réécrire le code de la méthode de base, on peut utiliser `parent::getNom()` pour appeler la méthode parent depuis notre méthode dérivée. Ensuite, on choisit d'`echo` un texte.

```php
<?php
    class Admin extends Utilisateur{
        protected $ban;

        public function __construct($n, $p){
            $this->user_name = strtoupper($n);
            $this->user_pass = $p;
        }

        public function getNom(){
            parent::getNom();
            echo ' (depuis la classe étendue)<br>';
        }

        public function setBan($b){
            $this->ban[] .= $b;
        }
        public function getBan(){
            echo 'Utilisateurs bannis par '.$this->user_name. ' : ';
            foreach($this->ban as $valeur){
                echo $valeur .', ';
            }
        }
    }
?>
```

Lorsqu'un objet de `Admin` appelle `getNom()`, la méthode `getNom()` de `Admin` va être utilisée. Cette méthode appelle elle-même la méthode de la classe parent qu'elle surcharge et ajoute un texte au résultat de la méthode surchargée.

Le mot clef `parent` fait ici référence à la classe parent de la classe dans laquelle la méthode appelante est définie. Dans notre cas, la méthode appelante se trouve dans `Admin`. Le code `parent::getNom()` va donc chercher une méthode nommée `getNom()` dans la classe parent de `Admin`, c'est-à-dire `Utilisateur`, et l'exécuter.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';

            $pierre = new Admin('Pierre', 'abcdef');
            $mathilde = new Utilisateur('Math', 123456);

            $pierre->getNom();
            $mathilde->getNom();

        echo '<br>';

            $pierre->setBan('Paul');
            $pierre->setBan('Jean');
            echo $pierre->getBan();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Constantes de classe

Nous n'en avons pas véritablement parlé jusqu'à présent mais nous allons également pouvoir définir des constantes dans une classe. Nous allons dans cette leçon apprendre à le faire et apprendre à accéder à la valeur d'une constante avec l'opérateur de résolution de portée.

Rappel sur les constantes et définition de constantes dans une classe

Une constante est un conteneur qui ne va pouvoir stocker qu'une seule et unique valeur durant la durée de l'exécution d'un script. On ne va donc pas pouvoir modifier la valeur stockée dans une constante.

Pour définir une constante de classe, on va utiliser le mot clef `const` suivi du nom de la constante en majuscules. On ne va pas utiliser ici de signe `$` comme avec les variables. Le nom d'une constante va respecter les règles communes de nommage PHP.

Depuis la version 7.1 de PHP, on peut définir une visibilité pour nos constantes (`public`, `protected` ou `private`).

Par défaut (si rien n'est précisé), une constante sera considérée comme publique et on pourra donc y accéder depuis l'intérieur et depuis l'extérieur de la classe dans laquelle elle a été définie.

Par ailleurs, notez également que les constantes sont allouées une fois par classe, et non pour chaque instance de classe. Cela signifie qu'une constante appartient intrinsèquement à la classe et non pas à un objet en particulier et que tous les objets d'une classe vont donc partager cette même constante de classe.

Ajoutons immédiatement une constante à notre classe mère `Utilisateur` :

```php
<?php
    class Utilisateur{
        protected $user_name;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __construct($n, $p){
            $this->user_name = $n;
            $this->user_pass = $p;
        }

        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            echo $this->user_name;
        }
    }
?>
```

Ici, on crée une constante `ABONNEMENT` qui stocke la valeur « 15 ». On peut imaginer que cette constante représente le prix d'un abonnement mensuel pour accéder au contenu de notre site.

Faites bien attention : si vous définissez un niveau de visibilité pour une constante, assurez-vous de travailler avec une version de PHP 7.1 ou ultérieure. Dans le cas où vous travaillez avec une version antérieure, la déclaration de la visibilité d'une constante sera illégale et le code ne fonctionnera pas.

Accéder à une constante avec l'opérateur de résolution de portée

Pour accéder à une constante, nous allons à nouveau devoir utiliser l'opérateur de résolution de portée. La façon d'accéder à une constante va légèrement varier selon qu'on essaie d'y accéder depuis l'intérieur de la classe qui la définit (ou d'une classe étendue) ou depuis l'extérieur de la classe.

Dans le cas où on tente d'accéder à la valeur d'une constante depuis l'intérieur d'une classe, il faudra utiliser l'un des deux mots clefs `self` ou `parent` qui vont permettre d'indiquer qu'on souhaite accéder à une constante définie dans la classe à partir de laquelle on souhaite y accéder (`self`) à qu'on souhaite accéder à une constante définie dans une classe mère (`parent`).

Essayons déjà de manipuler notre constante depuis la définition de la classe :

```php
<?php
    class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;

        /*Attention: si vous utilisez une version PHP < PHP 7.1, ce code ne
         *fonctionnera pas*/
        public const ABONNEMENT = 15;

        public function __construct($n, $p, $r){
            $this->user_name = $n;
            $this->user_pass = $p;
            $this->user_region = $r;
        }

        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            echo $this->user_name;
        }

        public function setPrixAbo(){
            /*On peut imaginer qu'on calcule un prix d'abonnement différent
             *selon les profils des utilisateurs*/
            if($this->user_region === 'Sud'){
                return $this->prix_abo = self::ABONNEMENT / 2;
            }else{
                return $this->prix_abo = self::ABONNEMENT;
            }
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }
    }
?>
```

L'idée ici va être de définir un tarif d'abonnement différent en fonction des différents profils des utilisateurs et en se basant sur notre constante de classe `ABONNEMENT`.

On va vouloir définir un tarif préférentiel pour les utilisateurs qui viennent du Sud (car c'est mon site et je fais ce que je veux !). On va ici rajouter deux propriétés dans notre classe : `$user_region` qui va contenir la région de l'utilisateur et `$prix_abo` qui va contenir le prix de l'abonnement après calcul.

On va commencer par modifier notre constructeur pour qu'on puisse initialiser la valeur de `$user_region` dès la création d'un objet.

Ensuite, on crée une méthode `setPrixAbo()` qui va définir le prix de l'abonnement en fonction de la région passée. Dans le cas où l'utilisateur indique venir du « Sud », le prix de l'abonnement sera égal à la moitié de la valeur de la constante `ABONNEMENT`.

Vous pouvez remarquer qu'on utilise `self::ABONNEMENT` pour accéder au contenu de notre constante ici. En effet, la constante a été définie dans la même classe que la méthode qui l'utilise.

On crée enfin une méthode `getPrixAbo()` qui renvoie la valeur contenue dans la propriété `$prix_abo` pour un objet.

On va maintenant se rendre dans notre classe étendue `Admin` :

```php
<?php
    class Admin extends Utilisateur{
        protected $ban;
        public const ABONNEMENT = 5;

        public function __construct($n, $p, $r){
            $this->user_name = strtoupper($n);
            $this->user_pass = $p;
            $this->user_region = $r;
        }

        public function setBan($b){
            $this->ban[] .= $b;
        }
        public function getBan(){
            echo 'Utilisateurs bannis par '.$this->user_name. ' : ';
            foreach($this->ban as $valeur){
                echo $valeur .', ';
            }
        }

        public function setPrixAbo(){
            if($this->user_region === 'Sud'){
                return $this->prix_abo = self::ABONNEMENT;
            }else{
                return $this->prix_abo = parent::ABONNEMENT / 2;
            }
        }
    }
?>
```

On surcharge ici la constante `ABONNEMENT` de la classe parente en lui attribuant une nouvelle valeur. On a tout à fait le droit puisqu'il s'agit ici de surcharge et non pas de changement dynamique de valeur d'une même constante (ce qui est interdit).

Pour les objets de la classe `Admin`, on définit le prix de l'abonnement de base à 5. Dans notre classe, on modifie le constructeur pour y intégrer un paramètre « région » et on supprime également la méthode `getNom()` créée précédemment.

Enfin, on surcharge la méthode `setPrixAbo()` : si l'administrateur indique venir du Sud, le prix de son abonnement va être égal à la valeur de la constante `ABONNEMENT` définie dans la classe courante (c'est-à-dire dans la classe `Admin`).

Dans les autres cas, le prix de l'abonnement va être égal à la valeur de la constante `ABONNEMENT` définie dans la classe parent (c'est-à-dire la classe `Utilisateur`) divisée par 2.

Ici, `self` et `parent` nous servent à indiquer à quelle définition de la constante `ABONNEMENT` on souhaite se référer.

Il ne nous reste plus qu'à exécuter nos méthodes et à afficher les différents prix de l'abonnement pour différents types d'utilisateurs. On va également vouloir renvoyer le contenu de la constante `ABONNEMENT` telle que définie dans notre classe `Utilisateur` et dans la classe étendue `Admin`.

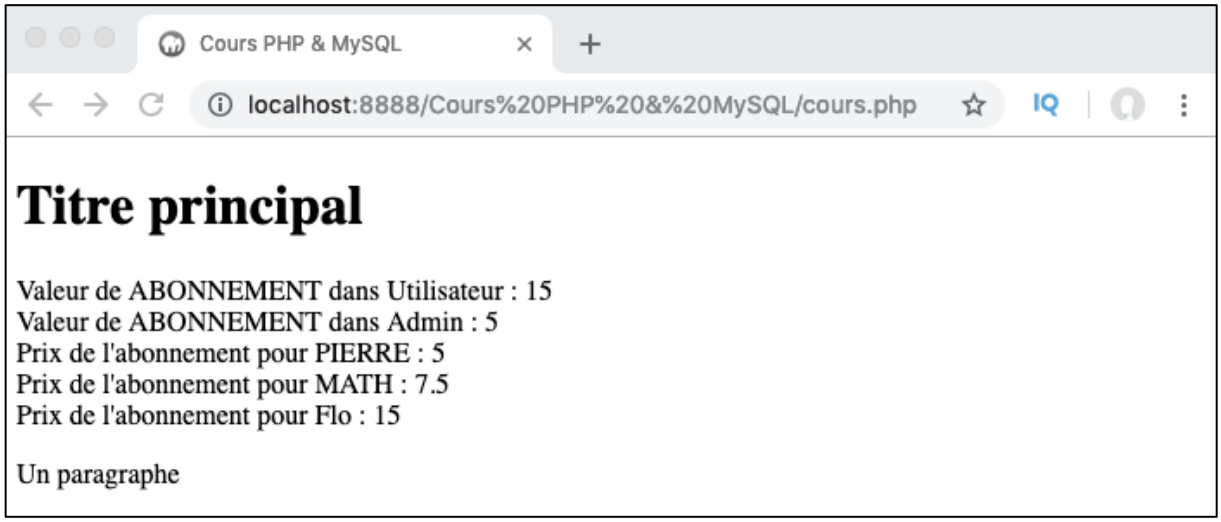
```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
      <h1>Titre principal</h1>
      <?php
        require 'classes/utilisateur.class.php';
        require 'classes/admin.class.php';

        $pierre = new Admin('Pierre', 'abcdef', 'Sud');
        $mathilde = new Admin('Math', 123456, 'Nord');
        $florian = new Utilisateur('Flo', 'flotri', 'Est');

        $pierre->setPrixAbo();
        $mathilde->setPrixAbo();
        $florian->setPrixAbo();

        $u = 'Utilisateur';
        echo 'Valeur de ABONNEMENT dans Utilisateur : ' .$u::ABONNEMENT. '<br>';
        echo 'Valeur de ABONNEMENT dans Admin : ' .Admin::ABONNEMENT. '<br>';

        echo 'Prix de l\'abonnement pour ';
        $pierre->getNom();
        echo ' : ';
        $pierre->getPrixAbo();
        echo '<br>Prix de l\'abonnement pour ';
        $mathilde->getNom();
        echo ' : ';
        $mathilde->getPrixAbo();
        echo '<br>Prix de l\'abonnement pour ';
        $florian->getNom();
        echo ' : ';
        $florian->getPrixAbo();
      ?>
      <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on tente d'accéder à notre constante depuis l'extérieur de la classe. On va utiliser une syntaxe différente qui va réutiliser le nom de la classe plutôt que les mots clefs `self` ou `parent` qui n'auraient aucun sens dans le cas présent.

On est obligés de préciser la classe ici car je vous rappelle qu'une constante n'appartient pas à une instance ou à un objet en particulier mais est définie pour la classe en soi.

Vous pouvez également remarquer que j'ai défini une variable `$u` dans le code ci-dessus. En effet, vous devez savoir qu'il est possible en PHP de référencer une classe en utilisant une variable, c'est-à-dire d'utiliser une variable pour faire référence à une classe. Pour faire cela, il suffit de stocker le nom exact de la classe dans la variable et on pourra ensuite l'utiliser comme référence à notre classe.

Dans mon code, l'écriture `$u::ABONNEMENT` est donc strictement équivalente à `UTILISATEUR::ABONNEMENT`.



Propriétés et méthodes statiques

Dans cette nouvelle leçon, nous allons découvrir ce que sont les propriétés et méthodes statiques, leur intérêt et comment créer et utiliser des propriétés et méthodes statiques.

Définition des propriétés et méthodes statiques

Une propriété ou une méthode statique est une propriété ou une méthode qui ne va pas appartenir à une instance de classe ou à un objet en particulier mais qui va plutôt appartenir à la classe dans laquelle elle a été définie.

Les méthodes et propriétés statiques vont donc avoir la même définition et la même valeur pour toutes les instances d'une classe et nous allons pouvoir accéder à ces éléments sans avoir besoin d'instancier la classe.

Pour être tout à fait précis, nous n'allons pas pouvoir accéder à une propriété statique depuis un objet. En revanche, cela va être possible dans le cas d'une méthode statique.

Attention à ne pas confondre propriétés statiques et constantes de classe : une propriété statique peut tout à fait changer de valeur au cours du temps à la différence d'une constante dont la valeur est fixée. Simplement, la valeur d'une propriété statique sera partagée par tous les objets issus de la classe dans laquelle elle a été définie.

De manière générale, nous n'utiliserons quasiment jamais de méthode statique car il n'y aura que très rarement d'intérêt à en utiliser. En revanche, les propriétés statiques vont s'avérer utiles dans de nombreux cas.

Définir et accéder à des propriétés et à des méthodes statiques

On va pouvoir définir une propriété ou une méthode statique à l'aide du mot clef `static`.

Prenons immédiatement un premier exemple afin que vous compreniez bien l'intérêt et le fonctionnement des propriétés et méthodes statiques.

Pour cela, retournons dans notre classe étendue `Admin`. Cette classe possède une propriété `$ban` qui contient la liste des utilisateurs bannis un l'objet courant de `Admin` ainsi qu'une méthode `getBan()` qui renvoie le contenu de `$ban`.

Imaginons maintenant que l'on souhaite stocker la liste complète des utilisateurs bannis par tous les objets de `Admin`. Nous allons ici devoir définir une propriété dont la valeur va pouvoir être modifiée et qui va être partagée par tous les objets de notre classe c'est-à-dire une propriété qui ne va pas appartenir à un objet de la classe en particulier mais à la classe en soi.

Pour faire cela, on va commencer par déclarer notre propriété `$ban` comme statique et modifier le code de nos méthodes `getBan()` et `setBan()`.

En effet, vous devez bien comprendre ici qu'on ne peut pas accéder à une propriété statique depuis un objet, et qu'on ne va donc pas pouvoir utiliser l'opérateur objet `->` pour accéder à notre propriété statique.

Pour accéder à une propriété statique, nous allons une fois de plus devoir utiliser l'opérateur de résolution de portée `::`.

Regardez plutôt le code ci-dessous :

```php
<?php
    class Admin extends Utilisateur{
        protected static $ban;
        public const ABONNEMENT = 5;

        public function __construct($n, $p, $r){
            $this->user_name = strtoupper($n);
            $this->user_pass = $p;
            $this->user_region = $r;
        }

        public function setBan(...$b){
            foreach($b as $banned){
                self::$ban[] .= $banned;
            }
        }
        public function getBan(){
            echo 'Utilisateurs bannis : ';
            foreach(self::$ban as $valeur){
                echo $valeur .', ';
            }
        }

        public function setPrixAbo(){
            if($this->user_region === 'Sud'){
                return $this->prix_abo = self::ABONNEMENT;
            }else{
                return $this->prix_abo = parent::ABONNEMENT / 2;
            }
        }
    }
?>
```

Ici, on commence par déclarer notre propriété `$ban` comme statique avec la syntaxe `protected static $ban`. La propriété va donc appartenir à la classe et sa valeur va être partagée par tous les objets de la classe.

Ensuite, on modifie notre fonction `setBan()` pour utiliser notre propriété statique. Ici, vous pouvez déjà noter que j'ai ajouté `...` devant la liste des paramètres de notre méthode.

Nous avons déjà vu cette écriture lors de la partie sur les fonctions : elle permet à une fonction d'accepter un nombre variable d'arguments. On utilise cette écriture ici pour permettre à nos objets de bannir une ou plusieurs personnes d'un coup.

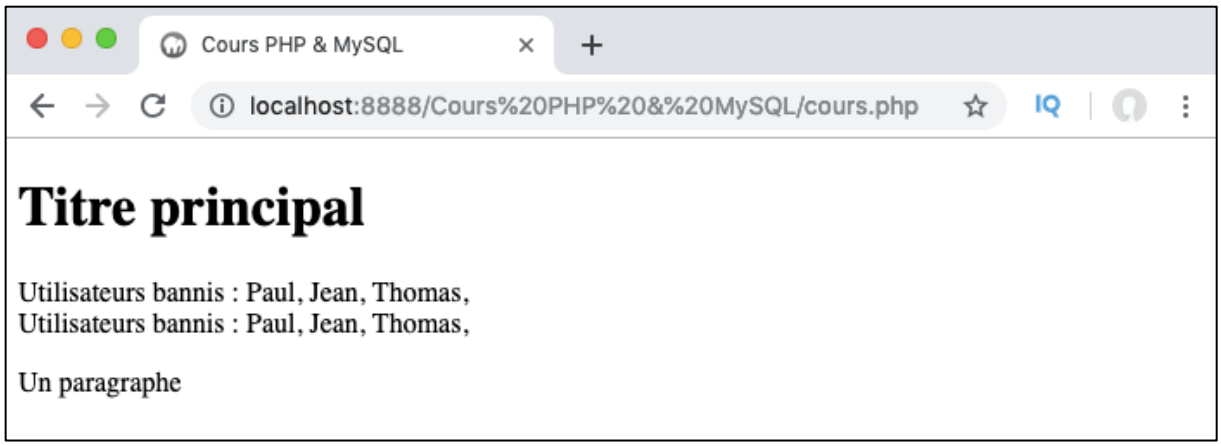
Dans notre méthode, on remplace `$this->ban` par `self::$ban` puisque notre propriété `$ban` est désormais statique et appartient à la classe et non pas à un objet en particulier. Il faut donc utiliser l'opérateur de résolution de portée pour y accéder.

La boucle `foreach` nous permet simplement d'ajouter les différentes valeurs passées en argument dans notre propriété statique `$ban` qu'on définit comme un tableau.

De même, on modifie le code de la méthode `getBan()` afin d'accéder à notre propriété statique et de pouvoir afficher son contenu en utilisant à nouveau la syntaxe `self::$ban`.

Chaque objet de la classe `Admin` va ainsi pouvoir bannir des utilisateurs en utilisant `setBan()` et chaque nouvel utilisateur banni va être stocké dans la propriété `$ban`. De même, chaque objet va pouvoir afficher la liste complète des personnes bannies en utilisant `getBan()`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
      <h1>Titre principal</h1>
      <?php
        require 'classes/utilisateur.class.php';
        require 'classes/admin.class.php';

        $pierre = new Admin('Pierre', 'abcdef', 'Sud');
        $mathilde = new Admin('Math', 123456, 'Nord');
        $florian = new Utilisateur('Flo', 'flotri', 'Est');

        $pierre->setBan('Paul', 'Jean');
        $mathilde->setBan('Thomas');

        $pierre->getBan();
        echo '<br>';
        $mathilde->getBan();
      ?>
      <p>Un paragraphe</p>
    </body>
</html>
```


Cours PHP & MySQL
localhost:8888/Cours%20PHP%20&%20MySQL/cours.php
Titre principal
Utilisateurs bannis : Paul, Jean, Thomas,
Utilisateurs bannis : Paul, Jean, Thomas,
Un paragraphe

Méthodes et classes abstraites

En PHP orienté objet, nous allons pouvoir définir des classes et des méthodes dites « abstraites ». Nous allons présenter dans cette leçon les intérêts des classes et méthodes abstraites et voir comment déclarer des classes et des méthodes comme abstraites en pratique.

Les classes et méthodes abstraites : définition et intérêt

Une classe abstraite est une classe qui ne va pas pouvoir être instanciée directement, c'est-à-dire qu'on ne va pas pouvoir manipuler directement.

Une méthode abstraite est une méthode dont seule la signature (c'est-à-dire le nom et les paramètres) va pouvoir être déclarée mais pour laquelle on ne va pas pouvoir déclarer d'implémentation (c'est-à-dire le code dans la fonction ou ce que fait la fonction).

Dès qu'une classe possède une méthode abstraite, il va falloir la déclarer comme classes abstraite.

Pour comprendre les intérêts des classes et des méthodes abstraites, il faut bien penser que lorsqu'on code en PHP, on code généralement pour quelqu'un ou alors on fait partie d'une équipe. D'autres développeurs vont ainsi généralement pouvoir / devoir travailler à partir de notre code, le modifier, ajouter des fonctionnalités, etc.

L'intérêt principal de définir une classe comme abstraite va être justement de fournir un cadre plus strict lorsqu'ils vont utiliser notre code en les forçant à définir certaines méthodes et etc.

En effet, une classe abstraite ne peut pas être instanciée directement et contient généralement des méthodes abstraites. L'idée ici va donc être de définir des classes mères abstraites et de pousser les développeurs à étendre ces classes.

Lors de l'héritage d'une classe abstraite, les méthodes déclarées comme abstraites dans la classe parent doivent obligatoirement être définies dans la classe enfant avec des signatures (nom et paramètres) correspondantes.

Cette façon de faire va être très utile pour fournir un rail, c'est-à-dire une ligne directrice dans le cas de développements futurs.

En effet, en créant un plan « protégé » (puisqu'une classe abstraite ne peut pas être instanciée directement) on force les développeurs à étendre cette classe et on les force également à définir les méthodes abstraites.

Cela nous permet de nous assurer que certains éléments figurent bien dans la classe étendue et permet d'éviter certains problèmes de compatibilité en nous assurant que les classes étendues possèdent une structure de base commune.

Définir des classes et des méthodes abstraites en pratique

Pour définir une classe ou une méthode comme abstraite, nous allons utiliser le mot clef `abstract`.

Vous pouvez déjà noter ici qu'une classe abstraite n'est pas structurellement différente d'une classe classique (à la différence de la présence potentielle de méthodes abstraites) et qu'on va donc tout à fait pouvoir ajouter des constantes, des propriétés et des méthodes classiques dans une classe abstraite.

Les seules différences entre les classes abstraites et classiques sont encore une fois qu'une classe abstraite peut contenir des méthodes abstraites et doit obligatoirement être étendue pour utiliser ses fonctionnalités.

Reprenons nos classes `Utilisateur` et `Admin` créées précédemment pour illustrer de manière pratique l'intérêt des classes et méthodes abstraites.

Précédemment, nous avons créé une méthode `setPrixAbo()` qui calculait le prix de l'abonnement pour un utilisateur classique dans notre classe `Utilisateur` et on avait surchargé le code de cette fonction dans `Admin` pour calculer un prix d'abonnement différent pour les admin.

Ici, cela rend notre code conceptuellement étrange car cela signifie que `Utilisateur` définit des choses pour un type d'utilisateur qui sont les utilisateurs « de base » tandis que `Admin` les définit pour un autre type d'utilisateur qui sont les « admin ». Le souci que j'ai avec ce code est que chacune de nos deux classes s'adresse à un type différent d'utilisateur mais que nos deux classes ne sont pas au même niveau puisque `Admin` est un enfant de `Utilisateur`.

Normalement, si notre code est bien construit, on devrait voir une hiérarchie claire entre ce que représentent nos classes mères et nos classes enfants. Dans le cas présent, j'aimerais que ma classe mère définisse des choses pour TOUS les types d'utilisateurs et que les classes étendues s'occupent chacune de définir des spécificités pour UN type d'utilisateur en particulier.

Encore une fois, ici, on touche à des notions qui sont plus de design de conception que des notions de code en soi mais lorsqu'on code la façon dont on crée et organise le code est au moins aussi importante que le code en soi. Il faut donc toujours essayer d'avoir la structure globale la plus claire et la plus pertinente possible.

Ici, nous allons donc partir du principe que nous avons deux grands types d'utilisateurs : les utilisateurs classiques et les administrateurs. On va donc transformer notre classe `Utilisateur` afin qu'elle ne définisse que les choses communes à tous les utilisateurs et allons définir les spécificités de chaque type utilisateur dans des classes étendues `Admin` et `Abonne`.

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __destruct(){
            //Du code à exécuter
        }

        abstract public function setPrixAbo();

        public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }
    }
?>
```

On commence déjà par modifier notre classe parent `Utilisateur` en la définissant comme abstraite avec le mot clef `abstract`. On supprime le constructeur qui va être défini dans les classes étendues et on déclare également la méthode `setPrixAbo()` comme abstraite.

Ici, définir la méthode `setPrixAbo()` comme abstraite fait beaucoup de sens puisque chaque type d'utilisateur va avoir un prix d'abonnement calculé différent (c'est-à-dire une implémentation de la méthode différente) et car on souhaite que le prix de l'abonnement soit calculé.

En définissant `setPrixAbo()` comme abstraite, on force ainsi les classes étendues à l'implémenter.

Les propriétés vont être partagées par tous les types d'utilisateurs et les méthodes comme `getNom()` vont avoir une implémentation identique pour chaque utilisateur. Cela fait donc du sens de les définir dans la classe abstraite.

Étendre des classes abstraites et implémenter des méthodes abstraites

Maintenant qu'on a défini notre classe `Utilisateur` comme abstraite, il va falloir l'étendre et également implémenter les méthodes abstraites.

On va commencer par aller dans notre classe étendue `Admin` et supprimer la constante `ABONNEMENT` puisque nous allons désormais utiliser celle de la classe abstraite. On va donc également modifier le code de notre méthode `setPrixAbo()`.

```php
<?php
    class Admin extends Utilisateur{
        protected static $ban;

        public function __construct($n, $p, $r){
            $this->user_name = strtoupper($n);
            $this->user_pass = $p;
            $this->user_region = $r;
        }

        public function setBan(...$b){
            foreach($b as $banned){
                self::$ban[] .= $banned;
            }
        }
        public function getBan(){
            echo 'Utilisateurs bannis : ';
            foreach(self::$ban as $valeur){
                echo $valeur .', ';
            }
        }

        public function setPrixAbo(){
            if($this->user_region === 'Sud'){
                return $this->prix_abo = parent::ABONNEMENT / 6;
            }else{
                return $this->prix_abo = parent::ABONNEMENT / 3;
            }
        }
    }
?>
```

Ici, lors de l'implémentation d'une méthode déclarée comme abstraite, on ne doit pas réécrire `abstract` puisque justement on implémente la méthode abstraite. Dans le code, on change `self::` par `parent::` ainsi que le calcul.

On va ensuite créer un nouveau fichier de classe qu'on va appeler `abonne.class.php`.

Dans ce fichier, nous allons définir un constructeur pour nos abonnés qui représentent nos utilisateurs de base et allons à nouveau implémenter la méthode `setPrixAbo()`.

```php
<?php
    class Abonne extends Utilisateur{
        public function __construct($n, $p, $r){
            $this->user_name = $n;
            $this->user_pass = $p;
            $this->user_region = $r;
        }

        public function setPrixAbo(){
            if($this->user_region === 'Sud'){
                return $this->prix_abo = parent::ABONNEMENT / 2;
            }else{
                return $this->prix_abo = parent::ABONNEMENT;
            }
        }
    }
?>
```

On peut maintenant retourner sur notre script principal et créer des objets à partir de nos classes étendues et voir comment ils se comportent :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';
            require 'classes/abonne.class.php';

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');

            $pierre->setPrixAbo();
            $mathilde->setPrixAbo();
            $florian->setPrixAbo();

            echo 'Prix de l\'abonnement pour ';
            $pierre->getNom();
            echo ' : ';
            $pierre->getPrixAbo();
            echo '<br>Prix de l\'abonnement pour ';
            $mathilde->getNom();
            echo ' : ';
            $mathilde->getPrixAbo();
            echo '<br>Prix de l\'abonnement pour ';
            $florian->getNom();
            echo ' : ';
            $florian->getPrixAbo();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```


Cours PHP & MySQL
localhost:8888/Cours%20PHP%20&%20MySQL/cours.php
Titre principal
Prix de l'abonnement pour PIERRE : 2.5
Prix de l'abonnement pour MATH : 5
Prix de l'abonnement pour Flo : 15
Un paragraphe

Interfaces

Dans cette nouvelle leçon, nous allons découvrir le concept d'interfaces en PHP orienté objet. Nous allons ici particulièrement insister la compréhension de ce qu'est une interface, la différence entre une interface et une classe abstraite ainsi que sur les cas où il va être intéressant de définir des interfaces.

Définition des interfaces en PHP objet et différence avec les classes abstraites

Les interfaces vont avoir un but similaire aux classes abstraites puisque l'un des intérêts principaux liés à la définition d'une interface va être de fournir un plan général pour les développeurs qui vont implémenter l'interface et de les forcer à suivre le plan donné par l'interface.

De la même manière que pour les classes abstraites, nous n'allons pas directement pouvoir instancier une interface mais devoir l'implémenter, c'est-à-dire créer des classes dérivées à partir de celle-ci pour pouvoir utiliser ses éléments.

Les deux différences majeures entre les interfaces et les classes abstraites sont les suivantes :

1. Une interface ne peut contenir que les signatures des méthodes ainsi qu'éventuellement des constantes mais pas de propriétés. Cela est dû au fait qu'aucune implémentation n'est faite dans une interface : une interface n'est véritablement qu'un plan ;
2. Une classe ne peut pas étendre plusieurs autres classes à cause des problèmes d'héritage. En revanche, une classe peut tout à fait implémenter plusieurs interfaces.

Je pense qu'il est ici intéressant de bien illustrer ces deux points et notamment d'expliquer pourquoi une classe n'a pas l'autorisation d'étendre plusieurs autres classes.

Pour cela, imaginons qu'on ait une première classe `A` qui définit la signature d'une méthode `diamond()` sans l'implémenter.

Nous créons ensuite deux classes `B` et `C` qui étendent la classe `A` et qui implémentent chacune d'une manière différente la méthode `diamond()`.

Finalement, on crée une classe `D` qui étend les classes `B` et `C` et qui ne redéfinit pas la méthode `diamond()`. Dans ce cas-là, on est face au problème suivant : la classe `D` doit-elle utiliser l'implémentation de `diamond()` faite par la classe `B` ou celle faite par la classe `C` ?

Ce problème est connu sous le nom du « problème du diamant » et est la raison principale pour laquelle la plupart des langages de programmation orientés objets (dont le PHP) ne permettent pas à une classe d'étendre deux autres classes.

En revanche, il ne va y avoir aucun problème par rapport à l'implémentation par une classe de plusieurs interfaces puisque les interfaces, par définition, ne peuvent que définir la signature d'une méthode et non pas son implémentation.

Profitez-en ici pour noter que les méthodes déclarées dans une classe doivent obligatoirement être publiques (puisqu'elles devront être implémentées en dehors de l'interface) et que les constantes d'interface ne pourront pas être écrasées par une classe (ou par une autre interface) qui vont en hériter.

Définir et implémenter une interface en pratique

On va pouvoir définir une interface de la même manière qu'une classe mais en utilisant cette fois-ci le mot clef `interface` à la place de `class`. Nous nommerons généralement nos fichiers d'interface en utilisant « interface » à la place de « classe ». Par exemple, si on crée une interface nommée `Utilisateur`, on enregistrera le fichier d'interface sous le nom `utiilsateur.interface.php` par convention.

```php
<?php
    interface Utilisateur{
        public const ABONNEMENT = 15;
        public function getNom();
        public function setPrixAbo();
        public function getPrixAbo();
    }
?>
```

On va ensuite pouvoir réutiliser les définitions de notre interface dans des classes. Pour cela, on va implémenter notre interface. On va pouvoir faire cela de la même manière que lors de la création de classes étendues mais on va cette fois-ci utiliser le mot clef `implements` à la place de `extends`.

```php
<?php
    class Abonne implements Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        private $user_pass;

        public function __construct($n, $p, $r){
            $this->user_name = $n;
            $this->user_pass = $p;
            $this->user_region = $r;
        }

        public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function setPrixAbo(){
            if($this->user_region === 'Sud'){
                return $this->prix_abo = Utilisateur::ABONNEMENT / 2;
            }else{
                return $this->prix_abo = Utilisateur::ABONNEMENT;
            }
        }
    }
?>
```

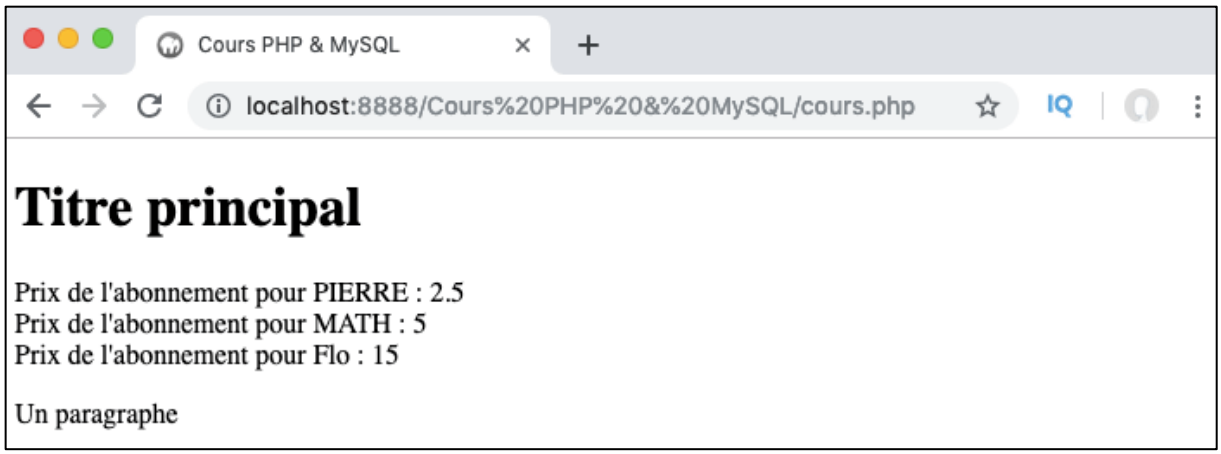
```php
<?php
    class Admin implements Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        private $user_pass;
        protected static $ban;

        public function __construct($n, $p, $r){
            $this->user_name = strtoupper($n);
            $this->user_pass = $p;
            $this->user_region = $r;
        }

        public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function setBan(...$b){
            foreach($b as $banned){
                self::$ban[] .= $banned;
            }
        }
        public function getBan(){
            echo 'Utilisateurs bannis : ';
            foreach(self::$ban as $valeur){
                echo $valeur .', ';
            }
        }

        public function setPrixAbo(){
            if($this->user_region === 'Sud'){
                return $this->prix_abo = Utilisateur::ABONNEMENT / 6;
            }else{
                return $this->prix_abo = Utilisateur::ABONNEMENT / 3;
            }
        }
    }
?>
```

Ici, on utilise une interface `Utilisateur` plutôt que notre classe abstraite `Utilisateur` qu'on laisse de côté pour le moment.

Notez bien ici que toutes les méthodes déclarées dans une interface doivent obligatoirement être implémentées dans une classe qui implémente une interface.

Vous pouvez également observer que pour accéder à une constante d'interface, il va falloir préciser le nom de l'interface devant l'opérateur de résolution de portée.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.interface.php';
            require 'classes/admin.class.php';
            require 'classes/abonne.class.php';

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');

            $pierre->setPrixAbo();
            $mathilde->setPrixAbo();
            $florian->setPrixAbo();

            echo 'Prix de l\'abonnement pour ';
            $pierre->getNom();
            echo ' : ';
            $pierre->getPrixAbo();
            echo '<br>Prix de l\'abonnement pour ';
            $mathilde->getNom();
            echo ' : ';
            $mathilde->getPrixAbo();
            echo '<br>Prix de l\'abonnement pour ';
            $florian->getNom();
            echo ' : ';
            $florian->getPrixAbo();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Par ailleurs, notez également qu'on va aussi pouvoir étendre une interface en utilisant le mot clef `extends`. Dans ce cas-là, on va créer des interfaces étendues qui devront être implémentées par des classes de la même manière que pour une interface classique.

Interface ou classe abstraite : comment choisir ?

Les interfaces et les classes abstraites semblent à priori servir un objectif similaire qui est de créer des plans ou le design de futures classes.

Cependant, avec un peu de pratique, on va vite s'apercevoir qu'il est parfois plus intéressant d'utiliser une classe abstraite ou au contraire une interface en fonction de la situation.

Les interfaces ne permettent aucune implémentation mais permet simplement de définir des signatures de méthodes et des constantes tandis qu'on va tout à fait pouvoir définir et implémenter des méthodes dans une classe abstraite.

Ainsi, lorsque plusieurs classes possèdent des similarités, on aura plutôt tendance à créer une classe mère abstraite dans laquelle on va pouvoir implémenter les méthodes communes puis d'étendre cette classe.

En revanche, si nous avons des classes qui vont pouvoir posséder les mêmes méthodes mais qui vont les implémenter de manière différente, alors il sera certainement plus adapté de créer une interface pour définir les signatures des méthodes communes et d'implémenter cette interface pour laisser le soin à chaque classe fille d'implémenter les méthodes comme elles le souhaitent.

Dans notre cas, par exemple, nos types d'utilisateurs partagent de nombreuses choses. Il est donc ici plus intéressant de créer une classe abstraite plutôt qu'une interface.

Encore une fois, on touche ici plus à des problèmes de structure et de logique du code qu'à des problèmes de code en soi et différents développeurs pourront parvenir au même résultat de différentes façons.

J'essaie ici simplement de vous présenter comment il est recommandé de travailler idéalement pour avoir la structure de code la plus propre et la plus facilement maintenable.

Une nouvelle fois, je ne saurais trop vous répéter l'importance de la réflexion face à un projet complexe avant l'étape de codage : il est selon moi essentiel de créer un premier plan sur papier décrivant ce à quoi on souhaite arriver et comment on va structurer notre code pour y arriver.

Les interfaces prédéfinies

De la même manière qu'il existe des classes prédéfinies, c'est-à-dire des classes natives ou encore prêtes à l'emploi en PHP, nous allons pouvoir utiliser des interfaces prédéfinies.

Parmi celles-ci, nous pouvons notamment noter :

- L'interface `Traversable` dont le but est d'être l'interface de base de toutes les classes permettant de parcourir des objets ;
- L'interface `Iterator` qui définit des signatures de méthodes pour les itérateurs ;
- L'interface `IteratorAggregate` qui est une interface qu'on va pouvoir utiliser pour créer un itérateur externe ;
- L'interface `Throwable` qui est l'interface de base pour la gestion des erreurs et des exceptions ;
- L'interface `ArrayAccess` qui permet d'accéder aux objets de la même façon que pour les tableaux ;
- L'interface `Serializable` qui permet de personnaliser la linéarisation d'objets.

Nous n'allons pas étudier ces interfaces en détail ici. Cependant, nous allons en utiliser certaines dans les parties à venir et notamment nous servir de `Throwable` et des classes dérivées prédéfinies `Error` et `Exception` dans la partie liée à la gestion des erreurs et des exceptions.

Méthodes magiques

Dans cette nouvelle leçon, nous allons discuter de méthodes spéciales en PHP objet : les méthodes magiques qui sont des méthodes qui vont être appelées automatiquement suive à un évènement déclencheur propre à chacune d'entre elles.

Nous allons établir la liste complète de ces méthodes magiques en PHP7 et illustrer le rôle de chacune d'entre elles.

Définition et liste des méthodes magiques PHP

Les méthodes magiques sont des méthodes qui vont être appelées automatiquement dans le cas d'un évènement particulier.

La méthode `__construct()`, par exemple, est une méthode magique. En effet, cette méthode s'exécute automatiquement dès que l'on instancie une classe dans laquelle on a défini un constructeur.

Les méthodes magiques reconnaissables en PHP au fait que leur nom commence par un double underscore `__`. En voici la liste :

- `__construct()`;
- `__destruct()` ;
- `__call()` ;
- `__callStatic()` ;
- `__get()` ;
- `__set()` ;
- `__isset()` ;
- `__unset()` ;
- `__toString()` ;
- `__clone()` ;
- `__sleep()` ;
- `__wakeup()` ;
- `__invoke()` ;
- `__set_state()` ;
- `__debugInfo()`.

Nous allons dans cette leçon comprendre le rôle de chacune de ces méthodes et voir comment les utiliser intelligemment.

Les méthodes magiques __construct() et __destruct()

Nous connaissons déjà les deux méthodes magiques `__construct()` et `__destruct()` encore appelées méthodes « constructeur » et « destructeur ».

La méthode `__construct()` va être appelée dès qu'on va instancier une classe possédant un constructeur.

Cette méthode est très utile pour initialiser les valeurs dont un objet a besoin dès sa création et avant toute utilisation de celui-ci.

La méthode magique `__destruct()` est la méthode « contraire » de la fonction constructeur et va être appelée dès qu'il n'y a plus de référence sur un objet donné ou dès qu'un objet n'existe plus dans le contexte d'une certaine application. Bien évidemment, pour que le destructeur soit appelé, vous devrez le définir dans la définition de la classe tout comme on a l'habitude de le faire avec le constructeur.

Le destructeur va nous servir à « nettoyer » le code en détruisant un objet une fois qu'on a fini de l'utiliser. Définir un destructeur va être particulièrement utile pour effectuer des opérations juste avant que l'objet soit détruit.

En effet, en utilisant un destructeur nous avons la maitrise sur le moment exact où l'objet va être détruit. Ainsi, on va pouvoir exécuter au sein de notre destructeur différentes commandes comme sauvegarder des dernières valeurs de l'objet dans une base de données, fermer la connexion à une base de données, etc. juste avant que celui-ci soit détruit.

Les méthodes magiques __call() et __callStatic()

La méthode magique `__call()` va être appelée dès qu'on essaie d'utiliser une méthode qui n'existe pas (ou qui est inaccessible) à partir d'un objet. On va passer deux arguments à cette méthode : le nom de la méthode qu'on essaie d'exécuter ainsi que les arguments relatifs à celle-ci (si elle en a). Ces arguments vont être convertis en un tableau.

La méthode magique `__callStatic()` s'exécute dès qu'on essaie d'utiliser une méthode qui n'existe pas (ou qui est inaccessible) dans un contexte statique cette fois-ci. Cette méthode accepte les mêmes arguments que `__call()`.

Ces deux méthodes vont donc s'avérer très utiles pour contrôler un script et pour éviter des erreurs fatales qui l'empêcheraient de s'exécuter convenablement ou même pour prévenir des failles de sécurité.

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __destruct(){
            //Du code à exécuter
        }

        abstract public function setPrixAbo();

        public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function __call($methode, $arg){
            echo 'Méthode ' .$methode. ' inaccessible depuis un contexte objet.
            <br>Arguments passés : ' .implode(', ', $arg). '<br>';
        }
        public static function __callStatic($methode, $arg){
            echo 'Méthode ' .$methode. ' inaccessible depuis un contexte statique.
            <br>Arguments passés : ' .implode(', ', $arg). '<br>';
        }
    }
?>
```

On définit ici deux méthodes magiques dans notre classe `Utilisateur` créée précédemment.

Si une méthode appelée depuis un contexte objet est inaccessible, `__call()` va être appelée automatiquement.

Si une méthode appelée depuis un contexte statique est inaccessible, `__callStatic()` va être appelée automatiquement.

Nos méthodes vont ici simplement renvoyer un texte avec le nom de la méthode inaccessible et les arguments passés.

Encore une fois, il faut bien comprendre que ce qui définit une méthode magique est simplement le fait que ces méthodes vont être appelées automatiquement dans un certain contexte.

Cependant, les méthodes magiques ne sont pas des méthodes prédéfinies et il va donc déjà falloir les définir quelque part pour qu'elles soient appelées et également leur fournir un code à exécuter.

Pour tester le fonctionnement de ces deux méthodes, il suffit d'essayer d'exécuter des méthodes qui n'existent pas dans notre script principal :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';
            require 'classes/abonne.class.php';

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');

            $pierre->test1('argument1');
            echo '<br>';
            Admin::test2('argument2', 'argument3');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Les méthodes magiques __get(), __set(), __isset() et __unset()

La méthode magique `__get()` va s'exécuter si on tente d'accéder à une propriété inaccessible (ou qui n'existe pas) dans une classe. Elle va prendre en argument le nom de la propriété à laquelle on souhaite accéder.

La méthode magique `__set()` s'exécute dès qu'on tente de créer ou de mettre à jour une propriété inaccessible (ou qui n'existe pas) dans une classe. Cette méthode va prendre comme arguments le nom et la valeur de la propriété qu'on tente de créer ou de mettre à jour.

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __destruct(){
            //Du code à exécuter
        }

        abstract public function setPrixAbo();

        public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function __get($prop){
            echo 'Propriété ' .$prop. ' inaccessible.<br>';
        }
        public function __set($prop, $valeur){
            echo 'Impossible de mettre à jour la valeur de ' .$prop. ' avec "'
            .$valeur. '" (propriété inaccessible)';
        }
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';
            require 'classes/abonne.class.php';

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');

            $pierre->prixAbo; //Inaccessible depuis un objet car protected
            echo '<br>';
            $pierre->prixAbo = 20;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

De manière similaire, la méthode magique `__isset()` va s'exécuter lorsque les fonctions `isset()` ou `empty()` sont appelées sur des propriétés inaccessibles.

La fonction `isset()` va servir à déterminer si une variable est définie et si elle est différente de `NULL` tandis que la fonction `empty()` permet de déterminer si une variable est vide.

Finalement, la méthode magique `__unset()` va s'exécuter lorsque la fonction `unset()` est appelée sur des propriétés inaccessibles.

La fonction `unset()` sert à détruire une variable.

Notez par ailleurs que les 4 méthodes magiques `__get()`, `__set()`, `__isset()` et `__empty()` ne vont pas pouvoir fonctionner dans un contexte statique mais uniquement dans un contexte objet.

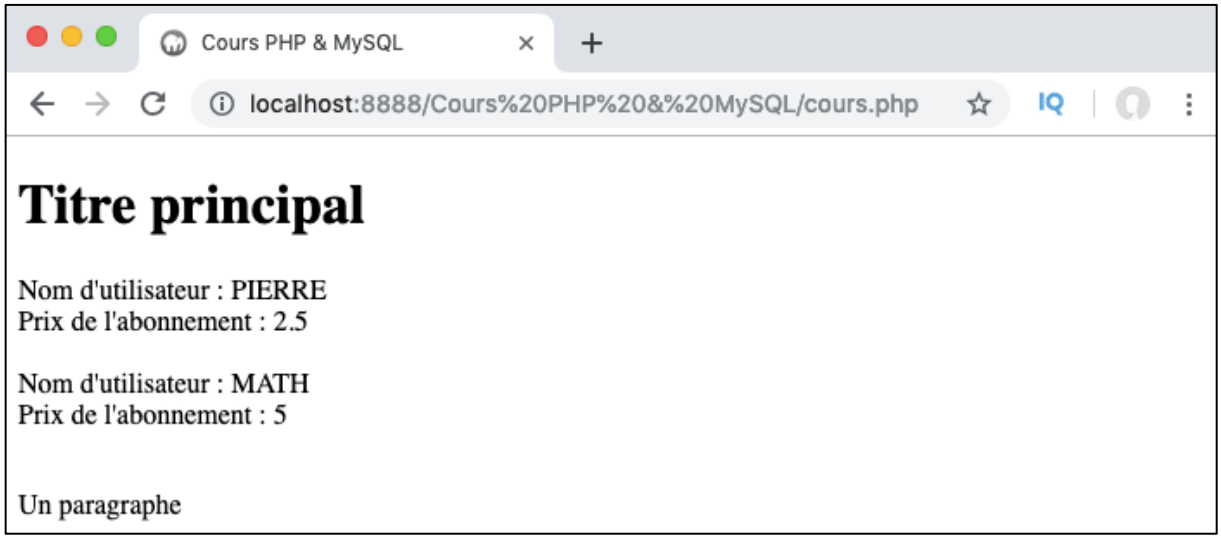
La méthode magique __toString()

La méthode magique `__toString()` va être appelée dès que l'on va traiter un objet comme une chaine de caractères (par exemple lorsqu'on tente d'`echo` un objet).

Attention, cette méthode doit obligatoirement retourner une chaine ou une erreur sera levée par le PHP.

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __destruct(){
            //Du code à exécuter
        }

        abstract public function setPrixAbo();

        public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function __toString(){
            return 'Nom d\'utilisateur : ' .$this->user_name. '<br>
            Prix de l\'abonnement : ' .$this->prix_abo. '<br><br>';
        }
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';
            require 'classes/abonne.class.php';

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');

            $pierre->setPrixAbo();
            $mathilde->setPrixAbo();
            echo $pierre;
            echo $mathilde;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



La méthode magique __invoke()

La méthode magique `__invoke()` va être appelée dès qu'on tente d'appeler un objet comme une fonction.

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __destruct(){
            //Du code à exécuter
        }

        abstract public function setPrixAbo();

        public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function __invoke($arg){
            echo 'Un objet a été utilisé comme une fonction.
            <br>Argument passé : ' .$arg. '<br><br>';
        }
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';
            require 'classes/abonne.class.php';

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');

            $pierre(1234);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

La méthode magique __clone()

La méthode magique `__clone()` s'exécute dès que l'on crée un clone d'un objet pour le nouvel objet créé.

Cette méthode va nous permettre notamment de modifier les propriétés qui doivent l'être après avoir créé un clone pour le clone en question.

Nous parlerons du clonage d'objet dans un chapitre ultérieur, je ne vais donc pas m'attarder sur cette méthode magique pour le moment.

Les méthodes magiques __sleep(), __wakeup(), __set_state() et debugInfo()

Nous n'étudierons pas ici les méthodes magiques `_sleep()`, `_wakeup()`, `_set_state()` et `debugInfo()` simplement car elles répondent à des besoins très précis et ne vont avoir d'intérêt que dans un contexte et dans un environnement spécifique.

PARTIE XI

Programmation orientée objet : notions avancées

Chainage de méthodes

Dans cette nouvelle leçon, nous allons découvrir ce qu'est le chainage de méthodes en PHP et apprendre à chainer des méthodes en pratique.

Principe et intérêt du chainage de méthodes en POO PHP

Chainer des méthodes nous permet d'exécuter plusieurs méthodes d'affilée de façon simple et plus rapide, en les écrivant à la suite les unes des autres, « en chaine ».

En pratique, il va suffire d'utiliser l'opérateur d'objet pour chainer différentes méthodes. On écrira quelque chose de la forme `$objet->methode1()->methode2()`.

Cependant, pour pouvoir utiliser le chainage de méthodes, il va falloir que nos méthodes chainées retournent notre objet afin de pouvoir exécuter la méthode suivante. Dans le cas contraire, une erreur sera renvoyée.

Le chaînage de méthodes en pratique

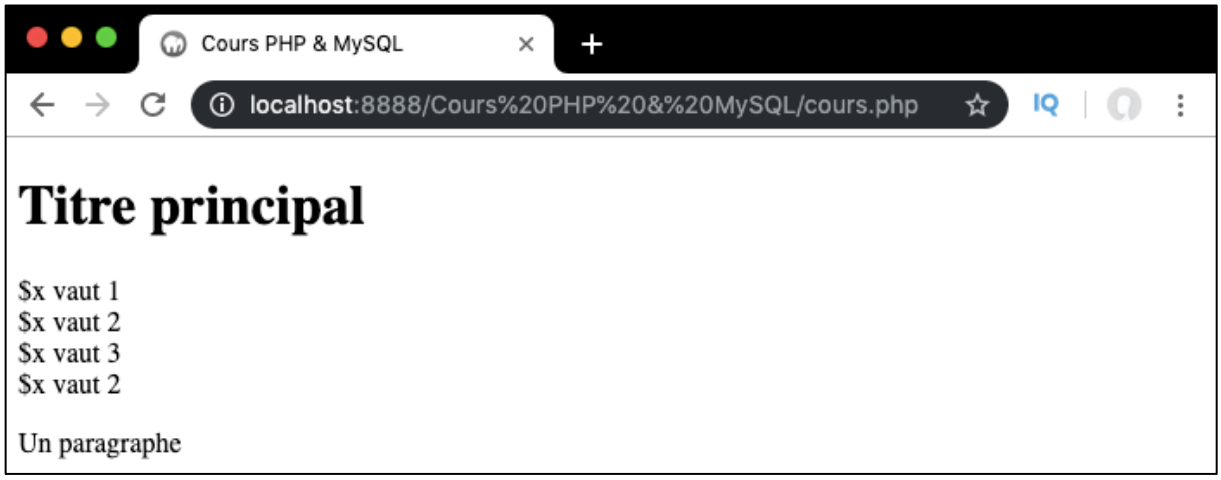
Prenons immédiatement un exemple afin de bien comprendre comment fonctionne le chainage de méthodes.

Pour cela, nous allons nous appuyer sur notre classe `Utilisateur` créée dans la partie précédente et à laquelle nous allons ajouter deux méthodes.

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        protected $x = 0;
        public const ABONNEMENT = 15;

        public function __destruct(){
            //Du code à exécuter
        }

        abstract public function setPrixAbo();

        public function getNom(){
            echo $this->user_name;
        }
        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function plusUn(){
            $this->x++;
            echo '$x vaut ' .$this->x. '<br>';
            return $this;
        }
        public function moinsUn(){
            $this->x--;
            echo '$x vaut ' .$this->x. '<br>';
            return $this;
        }
    }
?>
```

Ici, on commence par initialiser une propriété `$x` à zéro puis on crée deux nouvelles méthodes de classe `plusUn()` et `moinsUn()` dont le rôle est d'ajouter ou d'enlever un à la valeur de `$x` puis d'afficher un message avec la nouvelle position de notre objet.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            require 'classes/utilisateur.class.php';
            require 'classes/admin.class.php';
            require 'classes/abonne.class.php';

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');

            $pierre->plusUn()->plusUn()->plusUn()->moinsUn();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ensuite, on utilise l'opérateur objet pour exécuter toutes les méthodes à la suite, d'un seul coup. C'est précisément ce qu'on appelle le chainage de méthodes.

Notez bien l'instruction `return $this` à la fin du code de chacune de nos deux méthodes. Cette instruction est ici obligatoire. En effet, comme je l'ai précisé plus haut, vous devez impérativement retourner l'objet en soi pour pouvoir utiliser le chainage de méthodes. Si vous omettez le `return $this` vous allez avoir une erreur.

La grande limitation des méthodes chainées est donc qu'on doit retourner l'objet afin que la méthode suivante s'exécute. On ne peut donc pas utiliser nos méthodes pour retourner une quelconque autre valeur puisqu'on ne peut retourner qu'une chose en PHP.

Closures et classes anonymes

Dans cette nouvelle leçon, nous allons aborder la notion de classes anonymes. Pour bien comprendre comment vont fonctionner les classes anonymes, nous allons en profiter pour présenter les fonctions anonymes ou « closures ».

Découverte des fonctions anonymes et de la classe Closure

Les fonctions anonymes, qu'on appelle également des closures, sont des fonctions qui ne possèdent pas de nom.

On va créer une fonction anonyme de la même façon que l'on crée une fonction normale à la différence qu'on ne va ici pas préciser de nom.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            function(){
                echo 'Fonction anonyme bien exécutée';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

A ce stade, vous devriez déjà vous poser deux questions : quel est l'intérêt de créer de telles fonctions et comment peut-on les exécuter ou les appeler si elles ne possèdent pas de nom ?

Les closures en PHP ont été de manière historique principalement utilisées en tant que fonction de rappel car les fonctions de rappel. Une fonction de rappel est une fonction qui va être appelée par une autre fonction.

Notez que les fonctions anonymes sont implémentées en utilisant la classe prédéfinie `Closure`.

Appeler une fonction anonyme

Depuis PHP 7, il existe trois grands moyens simples d'appeler une fonction anonyme :

- En les auto-invoquant de manière similaire au langage JavaScript ;
- En les utilisant comme fonctions de rappel ;
- En les utilisant comme valeurs de variables.

Auto-invoquer une fonction anonyme

Depuis PHP 7, on peut auto-invoquer nos fonctions anonymes, c'est-à-dire faire en sorte qu'elles s'appellent elles-mêmes de manière automatique à la manière du JavaScript.

Pour cela, il va suffire d'entourer notre fonction anonyme d'un premier couple de parenthèses et d'ajouter un autre couple de parenthèses à la suite du premier couple comme cela :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            (function(){
                echo 'Fonction anonyme bien exécutée';
            })();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Créer des fonctions anonymes auto-invoquées va être très intéressant lorsqu'on ne va vouloir effectuer une tâche qu'une seule fois. Dans ce cas-là, en effet, il n'y a aucun intérêt de créer une fonction classique.

Par ailleurs, dans certains codes, nous n'allons pas pouvoir appeler une fonction manuellement mais allons vouloir que la fonction s'exécute automatiquement. Dans ces cas-là, utiliser la syntaxe ci-dessus va être très pertinent.

Utiliser une fonction anonyme comme fonction de rappel

Une fonction de rappel est une fonction qui va être appelée par une autre fonction. Pour cela, nous allons passer notre fonction de rappel en argument de la fonction appelante.

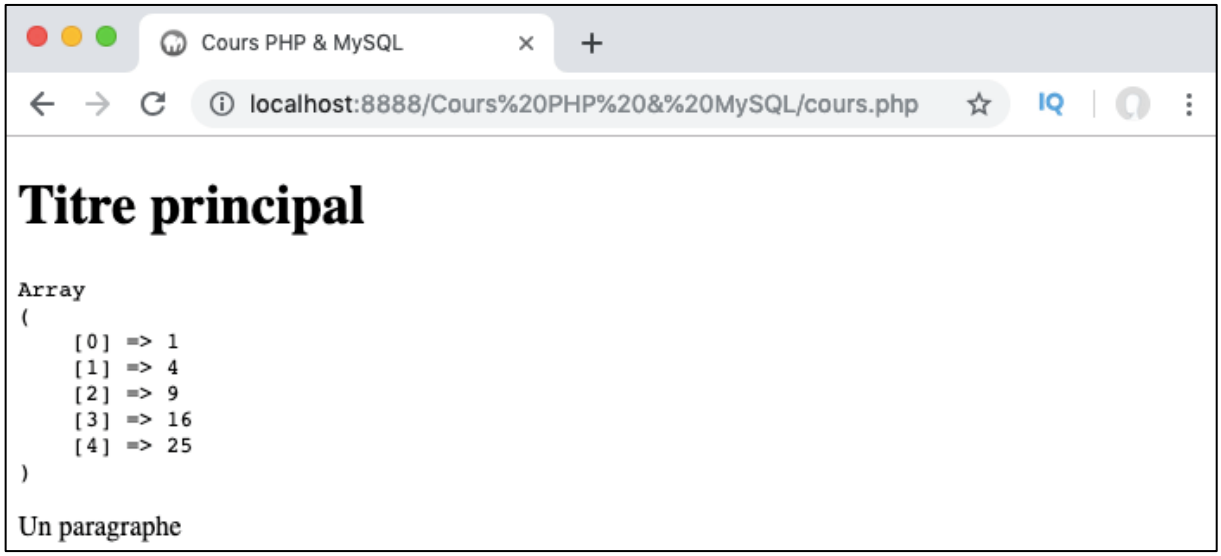
Les fonctions de rappel peuvent être de simples fonctions nommées, des fonctions anonymes ou encore des méthodes d'objets ou des méthodes statiques.

Dans le cas d'une fonction de rappel nommée, nous passerons le nom de la fonction de rappel en argument de la fonction appelante. Dans le cas d'une fonction de rappel anonyme, nous enfermerons la fonction dans une variable qu'on passera en argument de la fonction qui va l'appeler.

Bien évidemment, toutes les fonctions n'acceptent pas des fonctions de rappel en arguments mais seulement certaines comme la fonction `usort()` par exemple qui va servir à trier un tableau en utilisant une fonction de comparaison ou encore la fonction `array_map()` qui est la fonction généralement utilisée pour illustrer l'intérêt des closures.

La fonction `array_map()` va appliquer une fonction sur des éléments d'un tableau et retourner un nouveau tableau. On va donc devoir passer deux arguments à celle-ci : une fonction qui va dans notre cas être une closure et un tableau. Prenons immédiatement un exemple.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            /*La closure ci-dessous accepte un nombre en argument et
             *retourne son carré*/
            $squ = function(float $x){
                return $x**2;
            };

            //Définition d'un tableau
            $tb = [1, 2, 3, 4, 5];

            //array_map() exécute notre closure sur chaque élément du tableau
            $tb_squ = array_map($squ, $tb);

            echo '<pre>';
            print_r($tb_squ);
            echo '</pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Dans cet exemple, on commence par créer une closure dont le rôle est de calculer et de renvoyer le carré d'un nombre. On affecte notre closure à la variable `$squ` puis on crée ensuite une variable tableau stockant cinq chiffres.

Finalement, on utilise notre fonction `array_map()` en lui passant notre variable contenant notre fonction de rappel ainsi que notre tableau afin qu'elle renvoie un nouveau tableau et appliquant notre fonction de rappel à chaque élément du tableau passé.

On stocke le résultat renvoyé par `array_map()` dans une nouvelle variable `$tb_squ` qui est également une variable tableau et on affiche son contenu avec `print_r()`.

Appeler des fonctions anonymes en utilisant des variables

Lorsqu'on assigne une fonction anonyme en valeur de variable, notre variable va automatiquement devenir un objet de la classe prédéfinie `Closure`.

La classe `Closure` possède des méthodes qui vont nous permettre de contrôler une closure après sa création. Cette classe possède également une méthode magique `__invoke()` qui va ici s'avérer très utile puisqu'on va donc pouvoir exécuter nos closures simplement.

Je vous rappelle ici que la méthode magique `__invoke()` va s'exécuter dès qu'on se sert d'un objet comme d'une fonction.

Cela va nous permettre d'utiliser la syntaxe suivante pour appeler nos fonctions anonymes :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $txt = function(){
                echo 'Fonction anonyme bien exécutée';
            };

            $squ = function(float $x){
                return 'Le carré de ' .$x. ' est ' .$x**2;
            };

            $txt();
            echo '<br>';
            echo $squ(3);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Comme vous pouvez le constater, nos variables objets `$txt` et `$squ` sont utilisées comme fonctions pour exécuter les closures contenues à l'intérieur et les résultats sont bien renvoyés.

Définition et intérêt des classes anonymes

Les classes anonymes ont été implémentées récemment en PHP, puisque leur support n'a été ajouté qu'avec le PHP 7.

Les classes anonymes, tout comme les fonctions anonymes, sont des classes qui ne possèdent pas de nom.

Les classes anonymes vont être utiles dans le cas ou des objets simples et uniques ont besoin d'être créés à la volée.

Créer des classes anonymes va donc principalement nous faire gagner du temps et améliorer in-fine la clarté de notre code.

On va pouvoir passer des arguments aux classes anonymes via la méthode constructeur et celles-ci vont pouvoir étendre d'autres classes ou encore implémenter des interfaces et utiliser des traits comme le ferait une classe ordinaire.

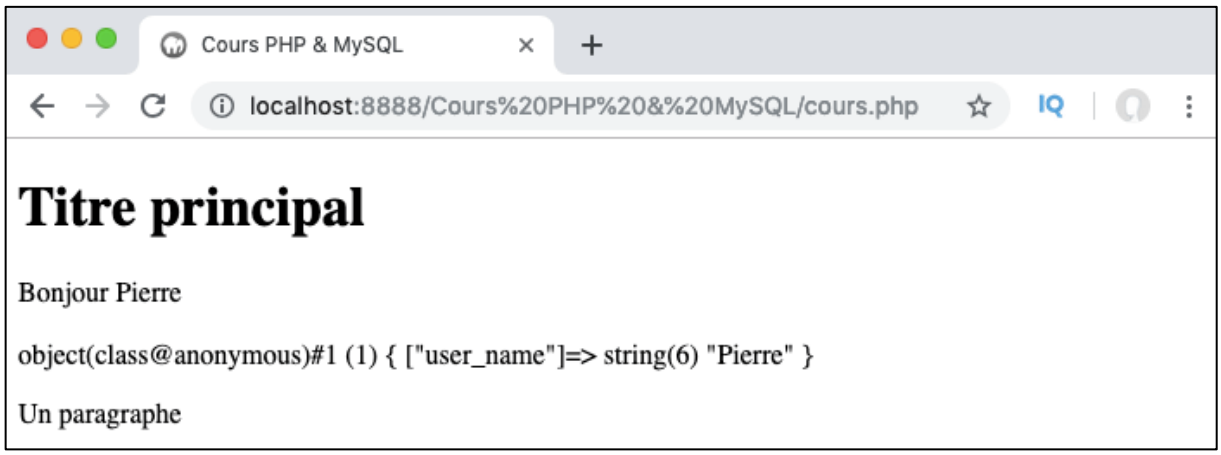
Notez qu'on va également pouvoir imbriquer une classe anonyme à l'intérieur d'une autre classe. Toutefois, on n'aura dans ce cas pas accès aux méthodes ou propriétés privées ou protégées de la classe contenante.

Pour utiliser des méthodes ou propriétés protégées de la classe contenante, la classe anonyme doit étendre celle-ci. Pour utiliser les propriétés privées de la classe contenant dans la classe anonyme, il faudra les passer via le constructeur.

Créer et utiliser des classes anonymes

Voyons immédiatement comment créer et manipuler des classes anonymes à travers différents exemples.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //On crée une classe anonyme qu'on stocke dans une variable (objet)
            $anonyme = new class{
                public $user_name;
                public const BONJOUR = 'Bonjour ';

                public function setNom($n){
                    $this->user_name = $n;
                }
                public function getNom(){
                    return $this->user_name;
                }
            };

            $anonyme->setNom('Pierre');
            echo $anonyme::BONJOUR;
            echo $anonyme->getNom();
            echo '<br><br>';

            //Affiche les infos de $anonyme
            var_dump($anonyme);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on commence avec un exemple simple en se contentant de déclarer une classe anonyme qu'on assigne à une variable qui devient de fait un objet.

Notre classe anonyme contient des propriétés, méthodes et constantes tout comme une classe classique.

Ensuite, on effectue différentes opérations simples : récupération et affichage des valeurs des propriétés, exécution des méthodes de notre classe anonyme, etc. afin que vous puissiez observer les différentes opérations que l'on peut réaliser.

Notez bien une nouvelle fois que le support des classes anonymes par le PHP est relativement récent : il est donc tout à fait possible que votre éditeur ne reconnaisse pas cette écriture (ce qui n'est pas grave en soi) ou que votre WAMP, MAMP, etc. n'arrive pas à l'exécuter si vous ne possédez une version PHP postérieure à la version 7.

On peut encore assigner une classe anonyme à une variable en passant par une fonction.

Dans ce cas-là, on pourra écrire quelque chose comme cela :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //On crée une classe anonyme qu'on stocke dans une variable (objet)
            function anonyme(){
                return new class{
                    public $user_name;
                    public const BONJOUR = 'Bonjour ';

                    public function setNom($n){
                        $this->user_name = $n;
                    }
                    public function getNom(){
                        return $this->user_name;
                    }
                };
            }
            $anonyme = anonyme();

            $anonyme->setNom('Pierre');
            echo $anonyme::BONJOUR;
            echo $anonyme->getNom();
            echo '<br><br>';

            //Affiche les infos de $anonyme
            var_dump($anonyme);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ce code est tout à fait comparable au précédent à la différence qu'on crée une fonction qui va retourner la définition de notre classe anonyme tout simplement.

Finalement, on peut également utiliser un constructeur pour passer des arguments à une classe anonyme lors de sa création.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //On crée une fonction qui retourne une classe anonyme
            function anonyme($n){
                return new class($n){
                    public $user_name;
                    public const BONJOUR = 'Bonjour ';

                    public function __construct($n){
                        $this->user_name = $n;
                    }
                    public function getNom(){
                        return $this->user_name;
                    }
                };
            }

            //On stocke le résultat de la fonction dans une variable objet
            $anonyme = anonyme('Pierre');
            echo $anonyme::BONJOUR;
            echo $anonyme->getNom();
            echo '<br><br>';

            //Affiche les infos de $anonyme
            var_dump($anonyme);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on définit le même paramètre `$n` lors de la définition de notre fonction et de notre classe anonyme. On passe ensuite l'argument `Pierre` lors de l'affectation du résultat de notre fonction dans la variable `$anonyme`. Cet argument va être stocké dans la propriété `$user_name` de notre classe.

Finalement, retenez que dans le cas où une classe anonyme est imbriquée dans une autre classe, la classe anonyme doit l'étendre afin de pouvoir utiliser ses propriétés et méthodes protégées. Pour utiliser ses méthodes et propriétés privées, alors il faudra également les passer via le constructeur.

Regardez plutôt l'exemple suivant :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            class Externe{
                private $age = 29;
                protected $nom = 'Pierre';

                public function anonyme(){
                    return new class($this->age) extends Externe{
                        private $a;
                        private $n;

                        public function __construct($age){
                            $this->a = $age;
                        }
                        public function getNomAge(){
                            return 'Nom : ' .$this->nom. ', âge : ' .$this->a;
                        }
                    };
                }
            }

            $obj = new Externe;
            echo $obj->anonyme()->getNomAge();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on déclare une première classe nommée `Externe` qui contient une propriété privée `$age` et une propriété protégée `$nom`.

Notre classe `Externe` contient également une méthode qui retourne une classe anonyme. On va vouloir utiliser les propriétés de code>Externe dans la classe anonyme. Pour cela, on passe notre variable protégée dans la définition de la classe et dans le constructeur.

Cette leçon et ce dernier exemple en particulier doivent vous sembler plus difficile à appréhender que le reste jusqu'ici. Pas d'inquiétude, c'est tout à fait normal car on commence à toucher à des notions vraiment avancées et il n'est pas simple d'en montrer l'intérêt à travers des exemples simples.

En pratique, je vous rassure, vous n'aurez que très rarement à faire ce genre de choses ou même à utiliser les classes anonymes.

Cependant, utiliser des classes anonymes peut s'avérer très pratique dans certaines situations et je dois donc vous présenter ce qu'il est possible de faire avec elles aujourd'hui en POO PHP.

L’auto chargement des classes

Il est considéré comme une bonne pratique en PHP orienté objet de créer un fichier par classe. Ceci a principalement pour but de conserver une meilleure clarté dans l’architecture générale d’un site et de simplifier la maintenabilité du code en séparant bien les différents éléments.

L’un des inconvénients de cette façon de procéder, cependant, est qu’on va possiblement avoir à écrire de longues séries d’inclusion de classes (une inclusion par classe) dans nos scripts lorsque ceux-ci ont besoin de plusieurs classes.

Pour éviter de rallonger le code inutilement et de nous faire perdre du temps, nous avons un moyen en PHP de charger (c’est-à-dire d’inclure) automatiquement nos classes d’un seul coup dans un fichier.

Pour cela, nous allons pouvoir utiliser la fonction `spl_autoload_register()`. Cette fonction nous permet d’enregistrer une ou plusieurs fonctions qui vont être mises dans une file d’attente et que le PHP va appeler automatiquement dès qu’on va essayer d’instancier une classe.

L’idée ici va donc être de passer une fonction qui permet de n’inclure que les classes dont on a besoin dans un script et de la passer à `spl_autoload_register()` afin qu’elle soit appelée dès que cela est nécessaire.

On va pouvoir ici soit utiliser une fonction nommée, soit idéalement créer une fonction anonyme :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');

            $pierre->getNom();
            echo '<br>';
            $mathilde->getNom();
            echo '<br>';
            $florian->getNom();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Dans ce script, on tente d'instancier nos classes `Admin` et `Abonne` créées précédemment. Pour rappel, ces deux classes étendent la classe parent `Utilisateur`.

On utilise ici la fonction `spl_autoload_register()` en lui passant une fonction anonyme en argument donc le rôle est d'inclure des fichiers de classe.

En résultat, la fonction `spl_autoload_register()` sera appelée dès qu'on va instancier une classe et va tenter d'inclure la classe demandée en exécutant la fonction anonyme. Notez

que cette fonction va également tenter de charger les éventuelles classes parents en commençant par les parents.

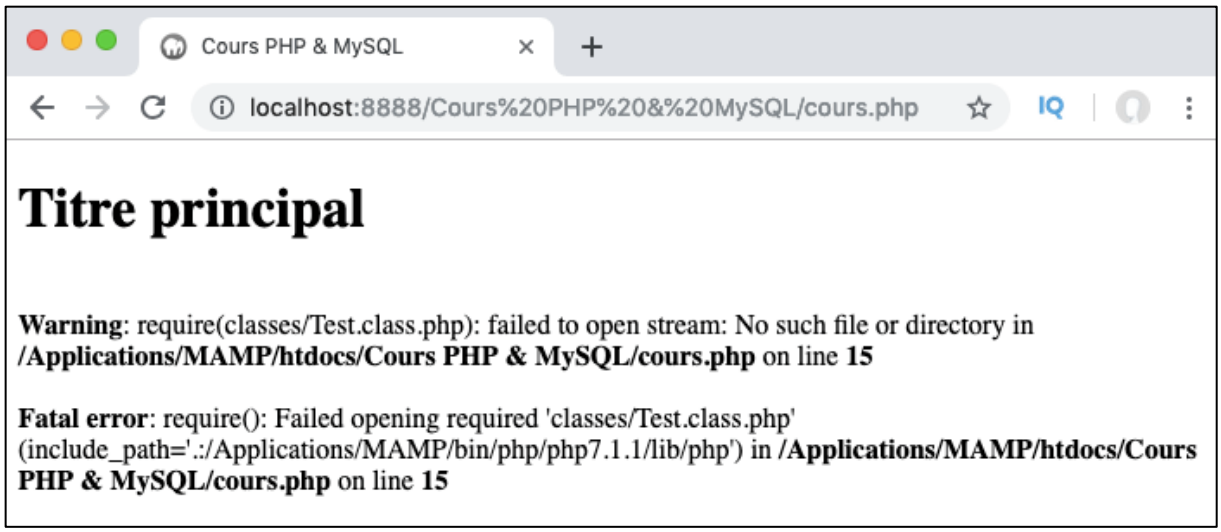
Ici, la fonction `spl_autoload_register()` va donc tenter d'inclure les fichiers `utilisateur.class.php`, `admin.class.php` et `abonne.class.php` situés dans un dossier « classes ».

Vous comprenez ici j'espère tout l'intérêt de placer tous nos fichiers de classes dans un même dossier et de respecter une norme d'écriture lorsqu'on nomme nos fichiers de classe puisque cela nous permet de pouvoir écrire des instructions formatées comme le `require` de notre fonction `spl_autoload_register()`.

Tenter d'auto-charger une classe inaccessible

Notez que si vous tentez d'inclure une classe qui est introuvable ou inaccessible le PHP renverra une erreur fatale.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');
            $test = new Test(); //La classe Test n'existe pas

            $pierre->getNom();
            echo '<br>';
            $mathilde->getNom();
            echo '<br>';
            $florian->getNom();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



On va pouvoir prendre en charge les erreurs et les exceptions en particulier en utilisant la classe `Exception`. La prise en charge des erreurs et des exceptions est cependant un sujet relativement complexe qui justifie une partie de cours en soi. Nous verrons comment cela fonctionne en détail dans la prochaine partie.

Le mot clef final en PHP objet

Depuis la version 5 de PHP, on peut empêcher les classes filles de surcharger une méthode en précisant le mot clef `final` avant la définition de celle-ci.

Si la classe elle-même est définie avec le mot clef `final` alors celle-ci ne pourra tout simplement pas être étendue.

Cela peut être utile si vous souhaitez empêcher explicitement certains développeurs de surcharger certaines méthodes ou d'étendre certaines classes dans le cas d'un projet Open Source par exemple.

Définir une méthode finale

Illustrons cela avec quelques exemples, en commençant avec la définition d'une méthode finale.

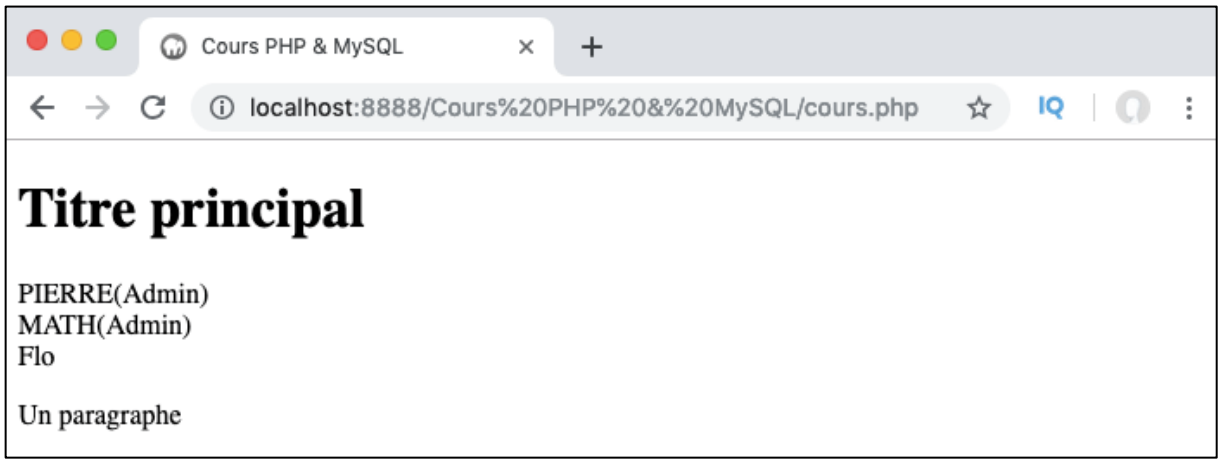
Pour cela, on peut reprendre nos classes `Utilisateur`, `Abonne` et `Admin` et par exemple déjà surcharger la méthode `getNom()` définie dans la classe parent `Utilisateur` depuis notre classe étendue `Admin` :

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        protected $x = 0;
        public const ABONNEMENT = 15;

        public function __destruct(){
            //Du code à exécuter
        }

        abstract public function setPrixAbo();

        public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function plusUn(){
            $this->x++;
            echo '$x vaut ' .$this->x. '<br>';
            return $this;
        }
        public function moinsUn(){
            $this->x--;
            echo '$x vaut ' .$this->x. '<br>';
            return $this;
        }
    }
?>
```

```php
<?php
    class Admin extends Utilisateur{
        protected static $ban;

        public function __construct($n, $p, $r){
            $this->user_name = strtoupper($n);
            $this->user_pass = $p;
            $this->user_region = $r;
        }

        public function getNom(){
            echo $this->user_name. '(Admin)';
        }

        public function setBan(...$b){
            foreach($b as $banned){
                self::$ban[] .= $banned;
            }
        }
        public function getBan(){
            echo 'Utilisateurs bannis : ';
            foreach(self::$ban as $valeur){
                echo $valeur .', ';
            }
        }

        public function setPrixAbo(){
            if($this->user_region === 'Sud'){
                return $this->prix_abo = parent::ABONNEMENT / 6;
            }else{
                return $this->prix_abo = parent::ABONNEMENT / 3;
            }
        }
    }
?>
```

Ici, lorsqu'on tente d'appeler notre méthode `getNom()` depuis un objet de la classe `Admin`, la définition de la méthode mère est bien surchargée et c'est la définition de la classe fille qui est utilisée.

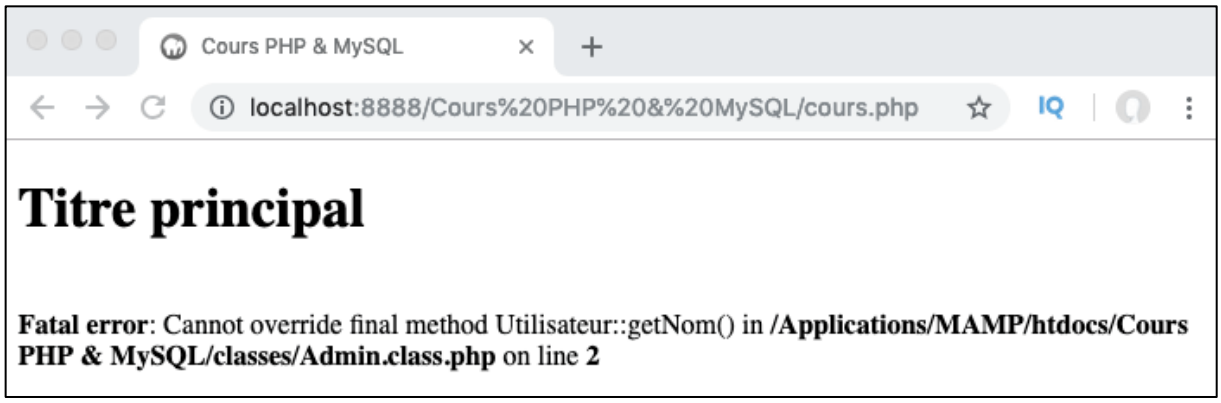
```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            $pierre = new Admin('Pierre', 'abcdef', 'Sud');
            $mathilde = new Admin('Math', 123456, 'Nord');
            $florian = new Abonne('Flo', 'flotri', 'Est');

            $pierre->getNom();
            echo '<br>';
            $mathilde->getNom();
            echo '<br>';
            $florian->getNom();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Essayons maintenant de définir notre méthode `getNom()` comme finale dans la classe `Utilisateur`.

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        protected $x = 0;
        public const ABONNEMENT = 15;

        public function __destruct(){
            //Du code à exécuter
        }

        abstract public function setPrixAbo();

        final public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function plusUn(){
            $this->x++;
            echo '$x vaut ' .$this->x. '<br>';
            return $this;
        }
        public function moinsUn(){
            $this->x--;
            echo '$x vaut ' .$this->x. '<br>';
            return $this;
        }
    }
?>
```

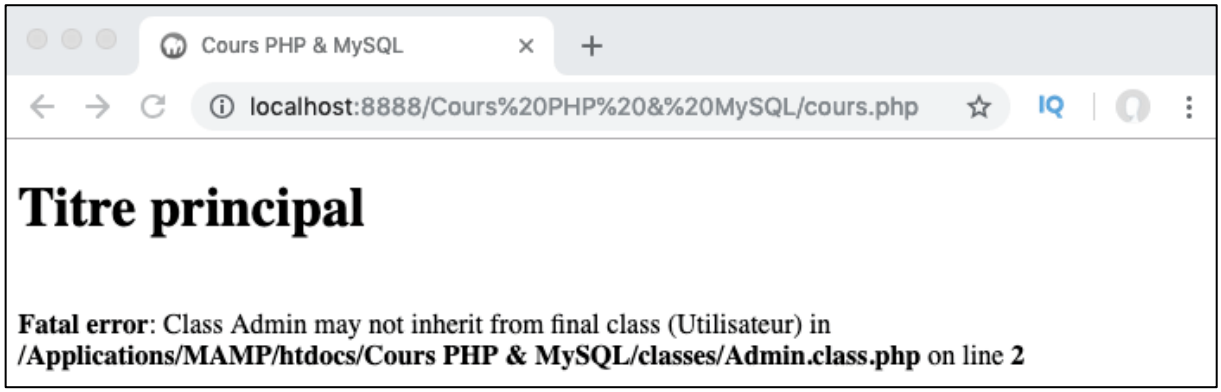
Comme notre méthode est définie avec le mot clef final, on n'a plus le droit de la surcharger dans une classe étendue. Si on tente de faire cela, une erreur fatale sera levée par le PHP :



Définir une classe finale

Si on définit une classe avec le mot clef final, on indique que la classe ne peut pas être étendue. Là encore, si on tente tout de même d'étendre la classe, le PHP renverra une erreur fatale.

```php
<?php
    final class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        protected $x = 0;
        public const ABONNEMENT = 15;

        public function __destruct(){
            //Du code à exécuter
        }

        public function setPrixAbo(){}

        final public function getNom(){
            echo $this->user_name;
        }

        public function getPrixAbo(){
            echo $this->prix_abo;
        }

        public function plusUn(){
            $this->x++;
            echo '$x vaut ' .$this->x. '<br>';
            return $this;
        }
        public function moinsUn(){
            $this->x--;
            echo '$x vaut ' .$this->x. '<br>';
            return $this;
        }
    }
?>
```



Notez ici que déclarer une classe comme abstraite et finale n'a aucun sens puisqu'une classe abstraite est par définition une classe qui va laisser à ses classes étendues le soin d'implémenter certains de ses éléments alors qu'une classe finale ne peut justement pas être étendue.

Par définition, une classe finale est une classe dont l'implémentation est complète puisqu'en la déclarant comme finale on indique qu'on ne souhaite pas qu'elle puisse être étendue. Ainsi, aucune méthode abstraite n'est autorisée dans une classe finale.

Résolution statique à la volée - late static bindings

Dans cette nouvelle leçon, nous allons découvrir une fonctionnalité très intéressante du PHP appelée la résolution statique à la volée ou « late static binding » en anglais et comprendre les problèmes qu'elle résout.

Définition et intérêt de la résolution statique à la volée

La résolution statique à la volée va nous permettre de faire référence à la classe réellement appelée dans un contexte d'héritage statique.

En effet, lorsqu'on utilise le `self::` pour faire référence à la classe courante dans un contexte statique, la classe utilisée sera toujours celle dans laquelle sont définies les méthodes utilisant `self::`.

Cela peut parfois produire des comportements inattendus. Regardez plutôt l'exemple ci-dessous pour vous en convaincre.

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        protected $x = 0;
        public const ABONNEMENT = 15;

        public function __construct($n, $p, $r){
            $this->user_name = $n;
            $this->user_pass = $p;
            $this->user_region = $r;
        }
        public function __destruct(){
            //Du code à exécuter
        }

        public static function getStatut(){
            self::statut();
        }
        public static function statut(){
            echo 'Utilisateur';
        }

        public function getNom(){
            echo $this->user_name;
        }
        public function getPrixAbo(){
            echo $this->prix_abo;
        }
        abstract public function setPrixAbo();
    }
?>
```

```php
<?php
    class Admin extends Utilisateur{
        protected static $ban;

        public function __construct($n, $p, $r){
            $this->user_name = strtoupper($n);
            $this->user_pass = $p;
            $this->user_region = $r;
        }
        public static function statut(){
            echo 'Admin';
        }
        public function getNom(){
            echo $this->user_name. '(Admin)';
        }
        public function setBan(...$b){
            foreach($b as $banned){
                self::$ban[] .= $banned;
            }
        }
        public function getBan(){
            echo 'Utilisateurs bannis : ';
            foreach(self::$ban as $valeur){
                echo $valeur .', ';
            }
        }
        public function setPrixAbo(){
            if($this->user_region === 'Sud'){
                return $this->prix_abo = parent::ABONNEMENT / 6;
            }else{
                return $this->prix_abo = parent::ABONNEMENT / 3;
            }
        }
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            /*On n'a pas beosin d'instancier nos classes ici puisqu'on se contente
             *d'utiliser une méthode statique (= qui appartient à la classe)*/
            Admin::getStatut();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on réutilise nos classes `Utilisateur` (classe mère) et `Admin` (classe étendue). Dans notre classe `Utilisateur`, on définit deux méthodes statiques `getStatut()` et `statut()`.

La méthode `statut()` de la classe `Utilisateur` renvoie le mot « Utilisateur ». La méthode `getStatut()` sert elle a exécuter la méthode `statut()` de la classe courante.

On surcharge ensuite notre méthode `statut()` dans notre classe étendue `Admin` afin qu'elle renvoie le texte « Admin ».

Finalement, dans notre script principal, on appelle notre méthode `getStatut()` depuis notre classe `Admin`.

Comme vous pouvez le voir, le résultat renvoyé est « Utilisateur » et non pas « Admin » comme on aurait pu le penser instinctivement. Cela est dû au fait que le code `self::` dans notre méthode `getStatut()` va toujours faire référence à la classe dans laquelle la méthode a été définie, c'est-à-dire la classe `Utilisateur`.

Ainsi, `self::statut()` sera toujours l'équivalent de `Utilisateur::statut()` et renverra toujours la valeur de la méthode `statut()` définie dans la classe `Utilisateur`.

La résolution statique à la volée a été introduite justement pour dépasser ce problème précis et pour pouvoir faire référence à la classe réellement utilisée.

Utilisation de la résolution statique à la volée et de static::

La résolution statique à la volée va donc nous permettre de faire référence à la classe réellement utilisée dans un contexte statique.

Pour utiliser la résolution statique à la volée, nous allons simplement devoir utiliser le mot clef `static` à la place de `self`. Ce mot clef va nous permettre de faire référence à la classe utilisée durant l'exécution de notre méthode.

Reprenons l'exemple précédent et changeons `self::` par `static::` dans le code de notre méthode `getStatut()`.

```php
<?php
    abstract class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        protected $x = 0;
        public const ABONNEMENT = 15;

        public function __construct($n, $p, $r){
            $this->user_name = $n;
            $this->user_pass = $p;
            $this->user_region = $r;
        }
        public function __destruct(){
            //Du code à exécuter
        }

        public static function getStatut(){
            static::statut();
        }
        public static function statut(){
            echo 'Utilisateur';
        }

        public function getNom(){
            echo $this->user_name;
        }
        public function getPrixAbo(){
            echo $this->prix_abo;
        }
        abstract public function setPrixAbo();
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            /*On n'a pas beosin d'instancier nos classes ici puisqu'on se contente
             *d'utiliser une méthode statique (= qui appartient à la classe)*/
            echo 'Statut de Admin : ' ;
            Admin::getStatut();
            echo '<br>Statut de Utilisateur : ' ;
            Utilisateur::getStatut();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Cette fois-ci, la méthode `getStatut()` va exécuter la méthode `statut()` de la classe utilisée, c'est-à-dire de la classe `Admin` et c'est donc la valeur « Admin » qui va être renvoyée.

Les traits

Dans cette nouvelle leçon, nous allons découvrir une fonctionnalité PHP qui va nous permettre de réutiliser du code dans des classes indépendantes et qu'on appelle « traits ».

Définition et intérêt des traits

Les traits sont apparus avec la version 5.4 du PHP. Très simplement, les traits correspondent à un mécanisme nous permettant de réutiliser des méthodes dans des classes indépendantes, repoussant ainsi les limites de l'héritage traditionnel.

En effet, rappelez-vous qu'en PHP une classe ne peut hériter que d'une seule classe mère.

Or, imaginons que nous devions définir la même opération au sein de plusieurs classes indépendantes, c'est-à-dire des classes qui ne partagent pas de fonctionnalité commune et pour lesquelles il n'est donc pas pertinent de créer une classe mère et de les faire étendre cette classe.

Dans ce cas-là, nous allons être obligé de réécrire le code correspondant à la méthode que ces classes ont en commun dans chacune des classes à moins justement d'utiliser les traits qui permettent à plusieurs classes d'utiliser une même méthode.

Par exemple, on peut imaginer qu'un site marchand possède deux classes `Utilisateur` et `Produit` qui vont être indépendantes mais qui vont posséder certaines méthodes en commun comme une méthode de comptage `plusUn()` par exemple.

Comme ces classes sont indépendantes et qu'on ne veut donc pas les faire hériter d'une même classe mère, on va être obligé de réécrire le code de notre méthode dans les deux classes si on n'utilise pas les traits :

cours.php | cours.css | utilisateur.class.php | produit.class.php | abonne.class.php

htdocs > Cours PHP & MySQL > classes > utilisateur.class.php — Ln: 34 Col: 1 UTF-8

```php
<?php
    class Utilisateur{
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        protected $nombre;
        public const ABONNEMENT = 15;

        public function __construct($n, $p, $r, $nb){
            $this->user_name = $n;
            $this->user_pass = $p;
            $this->user_region = $r;
            $this->nombre = $nb;
        }
        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            echo $this->user_name;
        }

        public function plusUn(){
            $this->nombre++;
            echo $this->nombre. '<br>';
            return $this;
        }

        //D'autres méthodes...
    }
?>
```

cours.php × cours.css × utilisateur.class.php × produit.class.php × abonne.class.php × a

... htdocs > Cours PHP & MySQL > classes > produit.class.php > Ln: 25 Col: 1 UTF-8

```php
<?php
    class Produit{
        protected $nom;
        protected $nombre;

        public function __construct($n, $nb){
            $this->nom = $n;
            $this->nombre = $nb;
        }

        public function getNom(){
            echo $this->nom;
        }

        public function plusUn(){
            $this->nombre++;
            echo $this->nombre. '<br>';
            return $this;
        }

        //D'autres méthodes...
    }
?>
```

Si on tente ensuite d'utiliser notre méthode, cela va bien évidemment fonctionner mais il ne sera pas optimisé puisqu'on a dû réécrire notre méthode deux fois.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            $pierre = new Utilisateur('Pierre', 'abcdef', 'Sud', 5);
            $yeti = new Produit('Yeti', 10);

            $pierre->plusUn();
            $yeti->plusUn()->plusUn();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Dans le cas présent, ce n'est pas trop grave mais imaginez maintenant que nous ayons des dizaines de classes utilisant certaines mêmes méthodes… … Cela va faire beaucoup de code écrit pour rien et en cas de modification d'une méthode il faudra modifier chaque classe, ce qui est loin d'être optimal !

Pour optimiser notre code, il va être intéressant dans ce cas d'utiliser les traits. Un trait est semblable dans l'idée à une classe mère en ce sens qu'il sert à grouper des fonctionnalités qui vont être partagées par plusieurs classes mais, à la différence des classes, on ne va pas pouvoir instancier un trait.

De plus, vous devez bien comprendre que le mécanisme des traits est un ajout à l'héritage « traditionnel » en PHP et que les méthodes contenues dans les traits ne vont pas être «

héritées » dans le même sens que ce qu'on a pu voir jusqu'à présent par les différentes classes.

Utiliser les traits en pratique

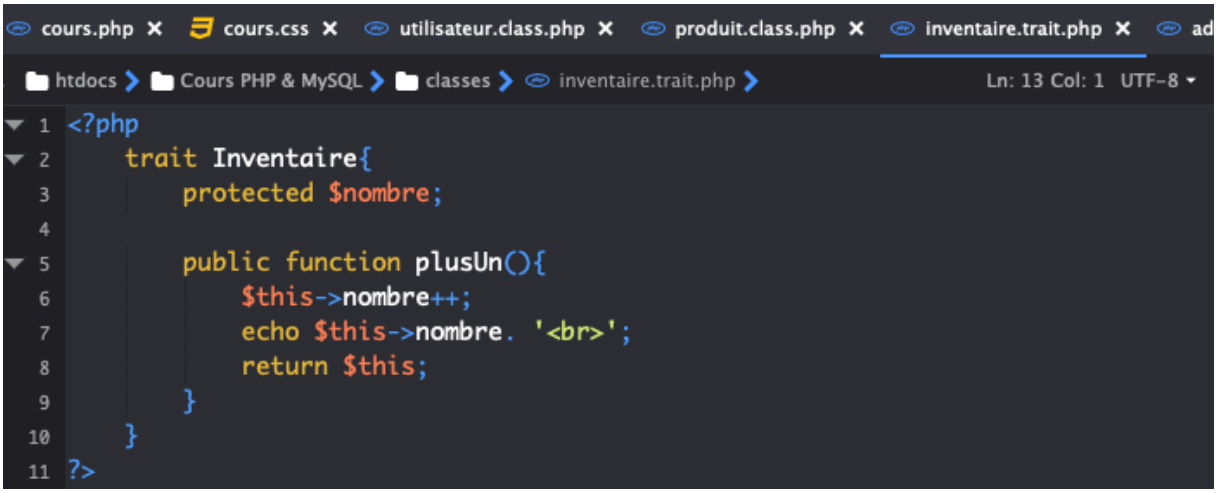
On va définir un trait de façon similaire à une classe, à la différence que nous allons utiliser le mot clef trait suivi du nom de notre trait.

Une bonne pratique consiste à utiliser un nouveau fichier pour chaque nouveau trait (on inclura ensuite les traits dans les classes qui en ont besoin). Ici, on peut déjà créer un trait qu'on va appeler `Inventaire`.

Dans ce trait, nous allons définir une propriété `$nombre` et une méthode `plusUn()`.

Notez qu'on peut tout à fait inclure des propriétés dans nos traits. Il faut cependant faire bien attention à la visibilité de celles-ci et ne pas abuser de cela au risque d'avoir un code au final moins clair et plus faillible.

Notez également que si on définit une propriété dans un trait, alors on ne peut pas à nouveau définir une propriété de même nom dans une classe utilisant notre trait à moins que la propriété possède la même visibilité et la même valeur initiale.



Une fois notre trait défini, nous devons préciser une instruction `use` pour pouvoir l'utiliser dans les différentes classes qui vont en avoir besoin. On peut également en profiter pour supprimer la propriété `$nombre` et la méthode `plusUn()` de ces classes.

```php
<?php
    class Utilisateur{
        use Inventaire;
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __construct($n, $p, $r, $nb){
            $this->user_name = $n;
            $this->user_pass = $p;
            $this->user_region = $r;
            $this->nombre = $nb;
        }
        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            echo $this->user_name;
        }

        //D'autres méthodes...
    }
?>
```

```php
<?php
    class Produit{
        use Inventaire;
        protected $nom;

        public function __construct($n, $nb){
            $this->nom = $n;
            $this->nombre = $nb;
        }

        public function getNom(){
            echo $this->nom;
        }

        //D'autres méthodes...
    }
?>
```

Dans notre script principal, nous allons également devoir inclure notre trait pour l'utiliser. On va faire cela de manière « classique », c'est-à-dire en dehors de la fonction `spl_autoload_register()` ici car il faudrait la modifier pour qu'elle accepte un fichier en `.trait.php`.

Les classes vont maintenant pouvoir utiliser les propriétés et les méthodes définies dans le trait et notre code va à nouveau fonctionner.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
        require 'classes/inventaire.trait.php';
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            $pierre = new Utilisateur('Pierre', 'abcdef', 'Sud', 5);
            $yeti = new Produit('Yeti', 10);

            $pierre->plusUn();
            $yeti->plusUn()->plusUn();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Utiliser un trait ici nous a permis de pouvoir réutiliser notre méthode `plusUn()` et notre propriété `$nombre` dans des classes indépendantes.

Ordre de précédence (ordre de priorité)

Dans le cas où une classe hérite d'une méthode d'une classe mère, celle-ci va être écrasée par une méthode (du même nom, bien évidemment) provenant d'un trait.

En revanche, dans le cas où une classe définit elle-même une méthode, celle-ci sera prédominante par rapport à celle du trait.

Ainsi, une méthode issue de la classe elle-même sera prioritaire sur celle venant d'un trait qui sera elle-même prédominante par rapport à une méthode héritée d'une classe mère. Pour illustrer cela, nous allons commencer par définir une méthode `precedence()` dans notre trait qui va renvoyer un texte.

```php
<?php
    trait Inventaire{
        protected $nombre;

        public function plusUn(){
            $this->nombre++;
            echo $this->nombre. '<br>';
            return $this;
        }

        public function precedence(){
            echo 'Méthode issue du trait<br>';
        }
    }
?>
```

Ensuite, nous allons récupérer notre classe `Utilisateur` et notre classes étendue `Admin` créée précédemment et allons utiliser notre trait dans chacune d'entre elles.

Nous allons également redéfinir notre méthode `precedence()` dans la classe `Utilisateur` mais pas dans `Admin`.

```php
<?php
    class Utilisateur{
        use Inventaire;
        protected $user_name;
        protected $user_region;
        protected $prix_abo;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __construct($n, $p, $r, $nb){
            $this->user_name = $n;
            $this->user_pass = $p;
            $this->user_region = $r;
            $this->nombre = $nb;
        }
        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            echo $this->user_name;
        }

        public function precedence(){
            echo 'Méthode issue de Utilisateur<br>';
        }
        //D'autres méthodes...
    }
?>
```

```php
<?php
    class Admin extends Utilisateur{
        use Inventaire;
        public function getNom(){
            echo $this->user_name. '(Admin)';
        }

        //Plus de méthodes...
    }
?>
```

Finalement, on instancie nos deux classes et on appelle notre méthode via nos deux objets créés pour observer le résultat.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
        require 'classes/inventaire.trait.php';
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            $pierre = new Utilisateur('Pierre', 'abcdef', 'Sud', 5);
            $mathilde = new Admin('Math', 123456, 'Nord', 2);

            $pierre->precedence();
            $mathilde->precedence();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Comme on peut le constater, la méthode définie dans `Utilisateur` est celle utilisée par-dessus celle définie dans le trait pour l'objet issu de cette classe.

En revanche, pour l'objet issu de `Admin`, c'est la méthode du trait qui va être utilisée par-dessus celle de la classe mère.

Inclusion de plusieurs traits et gestion des conflits

L'un des intérêts principaux des traits est qu'on va pouvoir utiliser plusieurs traits différents dans une même classe.

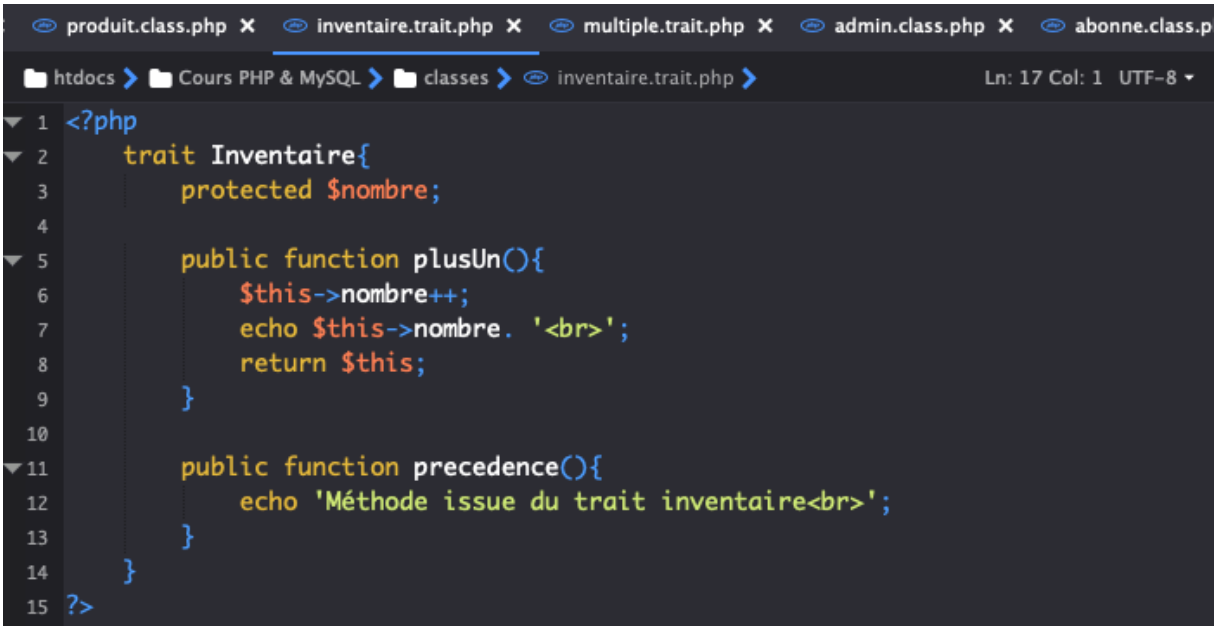
Cependant, cela nous laisse à priori avec le même problème d'héritage multiple vu précédemment et qui interdisait à une classe d'hériter de plusieurs classes parents.

En effet, imaginez le cas où une classe utilise plusieurs traits qui définissent une méthode de même nom mais de façon différente. Quelle définition doit alors être choisie par la classe ?

Dans ce genre de cas, il va falloir utiliser l'opérateur `insteadof` (« plutôt que » ou « à la place de » en français) pour choisir explicitement quelle définition de la méthode doit être choisie.

Utiliser l'opérateur `insteadof` nous permet en fait d'exclure les définitions d'une méthode qui ne devront pas être utilisées.

Pour illustrer cela, on peut par exemple créer un nouveau trait qu'on appellera `multiple` et qui va redéfinir la méthode `precedence()`.



On va ensuite inclure nos deux traits dans notre classe `Produit` et utiliser l'opérateur `insteadof` pour définir laquelle des deux définitions de notre méthode doit être utilisée avec la syntaxe suivante :

```php
<?php
    class Produit{
        use Inventaire, Multiple{
            Multiple::precedence insteadof Inventaire;
        }
        protected $nom;

        public function __construct($n, $nb){
            $this->nom = $n;
            $this->nombre = $nb;
        }

        public function getNom(){
            echo $this->nom;
        }

        //D'autres méthodes...
    }
?>
```

Ici, on déclare qu'on souhaite utiliser la définition de notre méthode `precedence()` du trait `Multiple` plutôt que celle du trait `Inventaire`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //On pense bien à inclure nos traits
            require 'classes/inventaire.trait.php';
            require 'classes/multiple.trait.php';
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            $yeti = new Produit('Yeti', 10);
            $yeti->precedence();

        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Notez que dans le cas (rare) où on souhaite utiliser les différentes définitions de méthodes portant le même nom définies dans différents traits, on peut également utiliser l'opérateur as qui permet d'utiliser une autre version d'une méthode de même nom en lui choisissant un nouveau nom temporaire.

Retenez ici bien que l'opérateur as ne renomme pas une méthode en soi mais permet simplement d'utiliser un autre nom pour une méthode juste pour le temps d'une inclusion et d'une exécution : le nom d'origine de la méthode n'est pas modifié.

```php
<?php
    class Produit{
        use Inventaire, Multiple{
            Inventaire::precedence insteadof Multiple;
            Multiple::precedence as prece;
        }
        protected $nom;

        public function __construct($n, $nb){
            $this->nom = $n;
            $this->nombre = $nb;
        }

        public function getNom(){
            echo $this->nom;
        }

        //D'autres méthodes...
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //On pense bien à inclure nos traits
            require 'classes/inventaire.trait.php';
            require 'classes/multiple.trait.php';
            spl_autoload_register(function($classe){
                require 'classes/' .$classe. '.class.php';
            });

            $yeti = new Produit('Yeti', 10);
            $yeti->precedence();
            $yeti->prece();

        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on commence par définir quelle définition de `precedence()` doit être utilisée grâce à l'opérateur `insteadof`. Ensuite, on demande à également utiliser la méthode `precedence()` du trait `Multiple` en l'utilisant sous le nom `prece()` pour éviter les conflits.

Une nouvelle fois, il est très rare d'avoir à effectuer ce genre d'opérations mais il reste tout de même bon de les connaitre, ne serait-ce que pour les reconnaitre dans les codes d'autres développeurs.

Par ailleurs, vous devez savoir qu'on peut également définir une nouvelle visibilité pour une méthode lorsqu'on utilise l'opérateur `as`. Pour cela, il suffit de préciser la nouvelle visibilité juste avant le nom d'emprunt de la méthode.

Les traits composés (héritage de traits)

Vous devez finalement savoir que des traits peuvent eux-mêmes utiliser d'autres traits et hériter de tout ou d'une partie de ceux-ci.

Pour cela, on va à nouveau utiliser le mot clef `use` pour utiliser un trait dans un autre.

On pourrait ainsi par exemple utiliser notre trait `Multiple` dans notre trait `Inventaire` (même si cela ne fait pas beaucoup de sens dans le cas présent) en utilisant la syntaxe suivante :

```php
<?php
    trait Inventaire{
        use Multiple;
        protected $nombre;

        public function plusUn(){
            $this->nombre++;
            echo $this->nombre. '<br>';
            return $this;
        }

        public function precedence(){
            echo 'Méthode issue du trait inventaire<br>';
        }
    }
?>
```

Notez que selon le système utilisé et l'emplacement de vos différents traits, vous devrez peut être spécifier le chemin complet du trait ou utiliser une instruction `require` pour inclure le trait directement dans l'autre avant d'utiliser `use`.

L’interface Iterator et le parcours d’objets

Dans cette nouvelle leçon, nous allons apprendre à parcourir rapidement les propriétés visibles d’un objet en utilisant une boucle `foreach`. Nous allons également découvrir l’interface `Iterator` et implémenter certaines de ses méthodes.

Parcourir un objet en utilisant une boucle foreach

Le PHP nous permet simplement de parcourir un objet afin d’afficher la liste de ses propriétés et leurs valeurs en utilisant une boucle `foreach`.

Attention cependant : par défaut, seules les propriétés visibles (publiques) seront lues. Prenons immédiatement un exemple pour illustrer cela.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //Définition d'une classe
            class Test{
                public $publique1 = 'Variable publique 1';
                public $publique2 = 'Variable publique 2';
                public $publique3 = 'Variable publique 3';

                protected $protegee = 'Variable protégée';
                private $privee   = 'Variable privée';
            }
            //Instanciation de la classe
            $test = new Test();

            //Parcours et affichage des propriétés visibles
            foreach ($test as $clef => $valeur){
                echo $clef. ' => ' .$valeur. '<br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on crée une classe `Test` qui contient cinq propriétés dont une protégée et une privée. Notez qu'ici je place tout le code sur une seule page pour plus de simplicité. Ce n'est cependant pas recommandé en pratique (rappelez-vous : une classe = un fichier).

Ensuite, on instancie la classe et on utilise une boucle `foreach` sur l'objet créé afin d'afficher la liste des propriétés visibles qu'il contient et leurs valeurs associées.

Comme vous pouvez le constater, seules les propriétés publiques sont renvoyées par défaut.

On va pouvoir gérer la façon dont un objet doit être traversé ou parcouru en implémentant une interface `Iterator` qui est une interface prédéfinie en PHP.

L'interface `Iterator` définit des méthodes qu'on va pouvoir implémenter pour itérer des objets en interne. On va ainsi pouvoir passer en revue certaines valeurs de nos objets à des moments choisis.

L'interface `Iterator` possède notamment cinq méthodes qu'il va être intéressant d'implémenter :

- La méthode `current()` ;
- La méthode `next()` ;
- La méthode `rewind()` ;
- La méthode `key()` ;
- La méthode `valid()`.

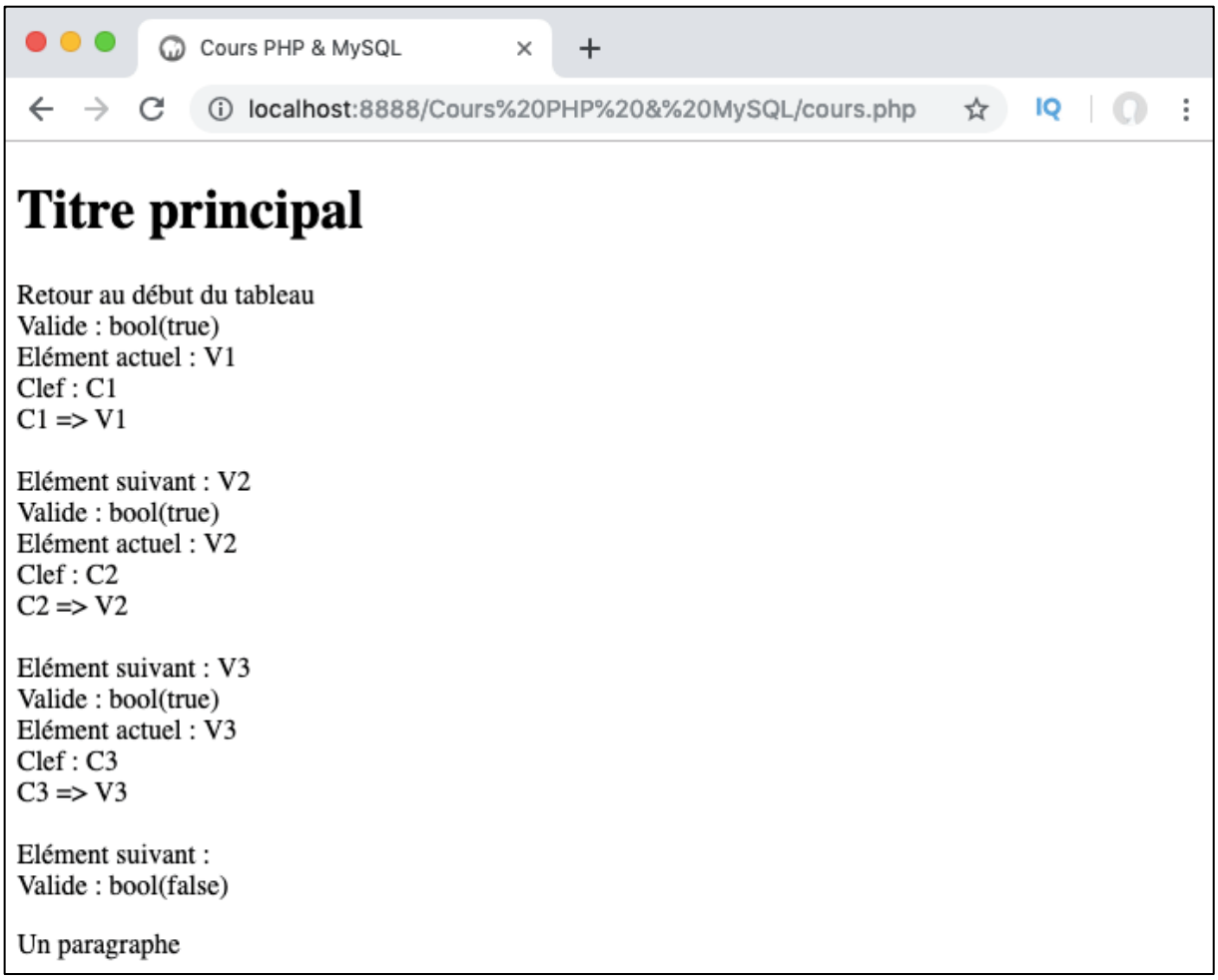
Une nouvelle fois, rappelez-vous bien ici qu'une interface n'est qu'un plan de base qui spécifie une liste de méthodes que les classes qui implémentent l'interface devront implémenter.

Les interfaces prédéfinies comme `Iterator` ne servent donc qu'à produire un code plus compréhensible et plus structuré (notamment car les autres développeurs vont « reconnaitre » l'utilisation d'une interface prédéfinie et donc immédiatement comprendre ce qu'on cherche à faire).

Nous allons donc devoir ici implémenter les méthodes définies dans `Iterator` dans la classe qui implémente cette interface. Bien évidemment, une nouvelle fois, on peut définir n'importe quelle implémentation pour nos méthodes mais dans ce cas utiliser une interface perd tout son sens.

Généralement, on va se baser sur le nom des méthodes pour définir une implémentation cohérente et utile. En effet, vous devez savoir que les fonctions `current()`, `next()`, `rewind()` et `key()` existent toutes déjà en tant que fonctions prédéfinies en PHP. Nous allons donc les utiliser pour définir l'implémentation de nos méthodes.

```php
<body>
    <h1>Titre principal</h1>
    <?php
        //Définition d'une classe
        class Test implements Iterator{
            private $tableau = [];

            public function __construct(array $tb){
                $this->tableau = $tb;
            }

            public function rewind(){
                echo 'Retour au début du tableau<br>';
                reset($this->tableau);
            }
            public function current(){
                $tableau = current($this->tableau);
                echo 'Elément actuel : ' .$tableau. '<br>';
                return $tableau;
            }
            public function key(){
                $tableau = key($this->tableau);
                echo 'Clef : ' .$tableau. '<br>';
                return $tableau;
            }
            public function next(){
                $tableau = next($this->tableau);
                echo 'Elément suivant : ' .$tableau. '<br>';
                return $tableau;
            }
            public function valid(){
                $clef = key($this->tableau);
                $tableau = ($clef !== NULL && $clef !== FALSE);
                echo 'Valide : ';
                var_dump($tableau);
                echo '<br>';
                return $tableau;
            }
        }
        $tbtest = ['C1' => 'V1', 'C2' => 'V2', 'C3' => 'V3'];
        $objet = new Test($tbtest);
        foreach ($objet as $c => $v){
            echo $c. ' => ' .$v. '<br><br>';
        }
    ?>
    <p>Un paragraphe</p>
</body>
```



Ce code contient de nombreuses choses intéressantes à expliquer et qui devraient vous permettre de mieux comprendre l'orienté objet en PHP en soi.

Tout d'abord, vous devez bien comprendre qu'une méthode de classe est un élément différent d'une fonction en PHP. En effet, la fonction `current()` par exemple est une fonction prédéfinie (ou prête à l'emploi) en PHP et on ne peut donc pas la redéfinir.

En revanche, la méthode `current()` n'est pas prédéfinie et on va donc pouvoir définir son implémentation. Ici, en l'occurrence, on utilise la fonction prédéfinie `current()` pour implémenter la méthode `current()`.

Notre classe `Test` implémente l'interface `Iterator`. Elle possède ici une propriété privée `$tableau` qui est un tableau vide au départ et définit un constructeur qui accepte un tableau en argument et le place dans la propriété privée `$tableau`. Ensuite, la classe se contente d'implémenter les méthodes de l'interface `Iterator`.

Ensuite, je pense qu'il convient d'expliquer ce que font les fonctions prédéfinies `current()`, `next()`, `rewind()` et `key()` pour comprendre le code ci-dessus.

La fonction `reset()` replace le pointeur interne au début du tableau et retourne la valeur du premier élément du tableau (avec une instruction de type `return`).

La fonction current() retourne la valeur de l'élément courant du tableau, c'est-à-dire de l'élément actuellement parcouru (l'élément au niveau duquel est situé le pointeur interne du tableau).

La fonction key() retourne la clef liée à la valeur de l'élément courant du tableau.

La fonction next() avance le pointeur interne d'un tableau d'un élément et retourne la valeur de l'élément au niveau duquel se situe le pointeur.

Une fois notre classe définie, on l'instancie en lui passant un tableau associatif qui va être utilisé comme argument pour notre constructeur.

Finalement, on utilise une boucle foreach pour parcourir l'objet créé à partir de notre classe.

Ce qu'il faut alors bien comprendre ici est que le PHP a un comportement bien défini lorsqu'on utilise un objet qui implémente l'interface Iterator et notamment lorsqu'on essaie de le parcourir avec une boucle foreach.

Notez par ailleurs que la plupart des langages web ont des comportements prédéfinis lorsqu'on utilise des éléments prédéfinis du langage et c'est généralement tout l'intérêt d'utiliser ces éléments.

Expliquons précisément ce qu'il se passe dans le cas présent. Tout d'abord, on sait qu'une interface impose aux classes qui l'implémentent d'implémenter toutes les méthodes de l'interface. Lorsqu'on crée un objet qui implémente l'interface Iterator, le PHP sait donc que l'objet va posséder des méthodes rewind(), current(), key(), next() et valid() et il va donc pouvoir les exécuter selon un ordre prédéfini.

Lorsqu'on utilise une boucle foreach avec un objet qui implémente l'interface Iterator, le PHP va automatiquement commencer par appeler Iterator::rewind() avant le premier passage dans la boucle ce qui va dans notre cas echo le texte « Retour au début du tableau » et va placer le pointeur interne du tableau au début de celui-ci.

Ensuite, avant chaque nouveau passage dans la boucle, Iterator::valid() est appelée et si false est retourné, on sort de la boucle. Dans le cas contraire, Iterator::current() et Iterator::key() sont appelées.

Finalement, après chaque passage dans la boucle, Iterator::next() est appelée et on recommence l'appel aux mêmes méthodes dans le même ordre (excepté pour rewind() qui n'est appelée qu'une fois en tout début de boucle).

Passage d'objets : identifiants et références

Dans cette nouvelle leçon, nous allons tenter d'illustrer des notions relativement complexes et abstraites concernant la façon dont les objets sont passés. Comprendre ces choses vous permettra de bien comprendre ce qu'il se passe en PHP orienté objet dans la plupart des cas d'assignation.

Le passage de variables par valeur ou par « référence » (alias)

On a vu plus tôt dans ce cours qu'il existait deux façons de passer une variable (à une fonction par exemple) en PHP : on pouvait soit la passer par valeur (ce qui est le comportement par défaut), soit par référence en utilisant le symbole `&` devant le nom de la variable.

Lorsqu'on parle de « passage par référence » en PHP, on devrait en fait plutôt parler d'alias au sens strict du terme et pour être cohérent par rapport à la plupart des autres langages de programmation.

Une « référence » en PHP ou plus précisément un alias est un moyen d'accéder au contenu d'une même variable en utilisant un autre nom. Pour le dire simplement, créer un alias signifie déclarer un autre nom de variable qui va partager la même valeur que la variable de départ.

Notez qu'en PHP le nom d'une variable et son contenu ou sa valeur sont identifiés comme deux choses distinctes par le langage. Cela permet donc de donner plusieurs noms à un même contenu (c'est-à-dire d'utiliser plusieurs noms pour accéder à un même contenu).

Ainsi, lorsqu'on modifie la valeur de l'alias, on modifie également la valeur de la variable de base puisque ces deux éléments partagent la même valeur.

Au contraire, lorsqu'on passe une variable par valeur (ce qui est le comportement par défaut en PHP), on travaille avec une « copie » de la variable de départ. Les deux copies sont alors indépendantes et lorsqu'on modifie le contenu de la copie, le contenu de la variable d'origine n'est pas modifié.

Regardez plutôt l'exemple ci-dessous pour bien vous en assurer :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 1;
            $y = $x;
            $z = &$y;
            $y = 2;

            echo 'Valeur de $x : ' .$x. '<br>
                  Valeur de $y : ' .$y. '<br>
                  Valeur de $z : ' .$z. '<br>';

            $a = 1;
            $b = 2;

            function parValeur($valeur){
                $valeur = 5;
                echo 'Valeur dans la fonction : ' .$valeur. '<br>';
            }
            parValeur($a);
            echo 'Valeur de $a : ' .$a. '<br>';

            function parReference(&$reference){
                $reference = 10;
                echo 'Valeur dans la fonction : ' .$reference. '<br>';
            }
            parReference($b);
             echo 'Valeur de $b : ' .$b. '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on commence par déclarer une variable `$x` à laquelle on assigne la valeur `1`.

Ensuite on définit une variable `$y` en lui assignant le contenu de `$x`. Par défaut, le passage se fait par valeur ce qui signifie qu'une copie de `$x` est créée. Si on manipule ensuite la copie (c'est-à-dire `$y`), le contenu de la variable de base `$x` n'est pas modifié puisqu'on a bien deux éléments indépendants ici stockant chacun une valeur bien différenciée.

Finalement, on définit une troisième variable `$z` en lui assignant le contenu de `$y` mais cette fois-ci on passe le contenu par référence avec le signe `&`. Notre variable `$z` est donc ici un alias de `$y`, ce qui signifie que `$z` et `$y` sont deux noms utilisés pour faire référence (= pour accéder ou pour manipuler) à la même valeur.

Nos deux variables `$y` et `$z` partagent donc ici la même valeur et lorsqu'on change la valeur assignée à l'une cela modifie forcément le contenu assigné à la seconde puisqu'encore une fois ces deux variables partagent la même valeur.

Ici, il faut bien comprendre une nouvelle fois qu'en PHP le nom d'une variable et son contenu sont deux éléments clairement identifiés. En fait, lorsqu'on assigne une valeur à un nom, on indique simplement au PHP qu'à partir de maintenant on va utiliser ce nom pour accéder à la valeur assignée.

Il se passe exactement la même chose lors du passage des arguments d'une fonction. Comme vous pouvez le voir, on passe l'argument dans notre fonction `parValeur()` par valeur. Lorsqu'on modifie la valeur de l'argument à l'intérieur de la fonction, la valeur de la variable externe n'est pas impactée puisque ces deux éléments sont bien différents.

En revanche, on passe l'argument de la fonction `parReference()` par référence. Ainsi, on crée un alias qui va servir de référence vers la même valeur que la variable qu'on va passer en argument à la fonction. Lorsqu'on modifie la valeur à l'intérieur de la fonction, on modifie donc également ce que stocke la variable à l'extérieur de la fonction.

Le passage des objets en PHP

Lorsqu'on crée une nouvelle instance de classe en PHP et qu'on assigne le résultat dans une variable, vous devez savoir qu'on n'assigne pas véritablement l'objet en soi à notre variable objet mais simplement un identifiant d'objet qu'on appelle également parfois un « pointeur ».

Cet identifiant va être utilisé pour accéder à l'objet en soi. Pour l'expliquer en d'autres termes, vous pouvez considérer que cet identifiant d'objet est à l'objet ce que le nom d'une variable est à la valeur qui lui est assignée.

Notre variable objet créée stocke donc un identifiant d'objet qui permet lui-même d'accéder aux propriétés de l'objet. Pour accéder à l'objet via son identifiant, on va utiliser l'opérateur `->` qu'on connait bien.

Ainsi, lorsqu'on passe une variable objet en argument d'une fonction, ou lorsqu'on demande à une fonction de retourner une variable objet, ou encore lorsqu'on assigne une variable objet à une autre variable objet, ce sont des copies de l'identifiant pointant vers le même objet qui sont passées.

Comme les copies de l'identifiant pointent toujours vers le même objet, on a tendance à dire que « les objets sont passés par référence ». Ce n'est cependant pas strictement vrai : encore une fois, ce sont des identifiants d'objets pointant vers le même objet qui vont être passés par valeur.

Regardez plutôt l'exemple suivant :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            class Utilisateur{
                protected $user_name;

                public function __construct($n){
                    $this->user_name = $n;
                }
                public function getNom(){
                    echo $this->user_name;
                }
                public function setNom($nom){
                    return $this->user_name = $nom;
                }
            };

            $pierre = new Utilisateur('Pierre');
            $victor = $pierre;
            $victor->setNom('Victor');
            $pierre->getNom();
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on crée une classe `Utilisateur` qu'on instancie une première fois. On assigne un identifiant d'objet à la variable objet `$pierre`.

On définit ensuite une deuxième variable $victor en lui assignant le contenu de $pierre. Notre variable va donc devenir de fait une variable objet et va stocker une copie de l'identifiant pointant vers le même objet que $pierre.

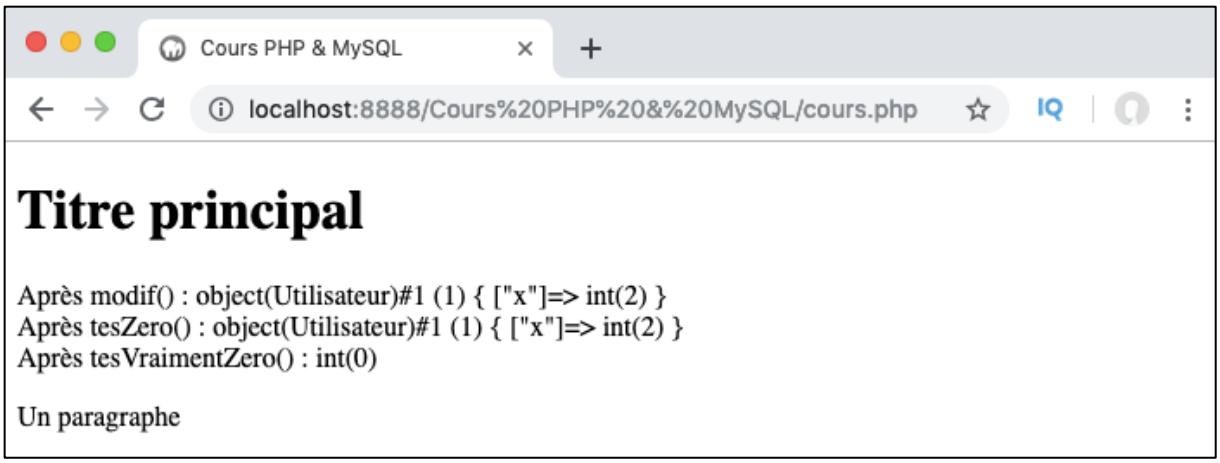
C'est la raison pour laquelle lorsqu'on accède à l'objet via $victor-> pour modifier la valeur de la propriété $user_name de l'objet, la valeur de $user_name de $pierre est également modifiée.

En effet, $pierre et $victor contiennent deux copies d'identifiant permettant d'accéder au même objet. C'est la raison pour laquelle le résultat ici peut faire pense que nos objets ont été passés par référence.

Ce n'est toutefois pas le cas, ce sont des copies d'identifiant pointant vers le même objet qui sont passées par valeur. Pour passer un identifiant d'objet par référence, nous allons une nouvelle fois devoir utiliser le signe &.

Regardez le nouvel exemple ci-dessous pour bien comprendre la différence entre un passage par référence et un passage par valeur via un identifiant.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            class Utilisateur{
                public $x = 1;

                public function modif(){
                    $this->x = 2;
                }
            }
            function tesZero($obj){
                $obj = 0;
            }
            function tesVraimentZero(&$obj){
                $obj = 0;
            }

            $pierre = new Utilisateur();
            $pierre->modif();
            echo 'Après modif() : ' ;
            var_dump($pierre);
            tesZero($pierre);
            echo '<br>Après tesZero() : ' ;
            var_dump($pierre);
            tesVraimentZero($pierre);
            echo '<br>Après tesVraimentZero() : ' ;
            var_dump($pierre);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on définit une classe qui contient une propriété et une méthode publiques et on instancie notre classe puis on assigne l'identifiant d'objet à notre variable objet `$pierre`.

On définit également deux fonctions en dehors de notre classe.

On appelle ensuite notre méthode `modif()` donc le rôle est de modifier la valeur de la propriété `$x` de l'objet courant puis on affiche les informations relatives à notre objet grâce à `var_dump()`. On constate que notre propriété `$x` stocke bien la valeur 2.

Ensuite, on utilise notre fonction `tesZero()` en lui passant `$pierre` en argument. Le rôle de cette fonction est d'assigner la valeur 0 à la variable passée en argument. Pourtant, lorsqu'on `var_dump()` à nouveau `$pierre`, on s'aperçoit que le même objet que précédemment est renvoyé.

Cela est dû au fait qu'ici notre fonction `tesZero()` n'a modifié que l'identifiant d'objet et non pas l'objet en soi.

Notre fonction `tesVraimentZero()` utilise elle le passage par référence. Dans ce cas-là, c'est bien une référence à l'objet qui va être passée et on va donc bien pouvoir écraser l'objet cette fois-ci.

Je tiens ici à préciser que ces notions sont des notions abstraites et complexes et qu'il faut généralement beaucoup de pratique et une très bonne connaissance au préalable du langage pour bien les comprendre et surtout comprendre leurs implications. J'essaie ici de vous les présenter de la manière la plus simple possible, mais ne vous inquiétez pas si certaines choses vous échappent pour le moment : c'est tout à fait normal, car il faut du temps et du recul pour maitriser parfaitement un langage.

Le clonage d'objets

Dans la leçon précédente, on a vu que nos variables objets stockaient en fait des identifiants d'objets servant à accéder aux objets.

Lorsqu'on instancie une fois une classe, on crée un objet et on stocke généralement un identifiant d'objet dans une variable qu'on appelle une variable objet ou un objet par simplification.

Si on assigne le contenu de notre variable objet dans une nouvelle variable, on ne va créer qu'une copie de l'identifiant qui va continuer à pointer vers le même objet.

Cependant, dans certains cas, on voudra plutôt créer une copie d'un objet en soi. C'est exactement ce que va nous permettre de réaliser le clonage d'objet que nous allons étudier dans cette nouvelle leçon.

Les principes du clonage d'objets

Parfois, on voudra « copier » un objet afin de manipuler une copie indépendante plutôt que l'objet original.

Dans ces cas-là, on va « cloner » l'objet. Pour cela, on va utiliser le mot clef `clone` qui va faire appel à la méthode magique `__clone()` de l'objet si celle-ci a été définie. Notez qu'on ne peut pas directement appeler la méthode `__clone()`.

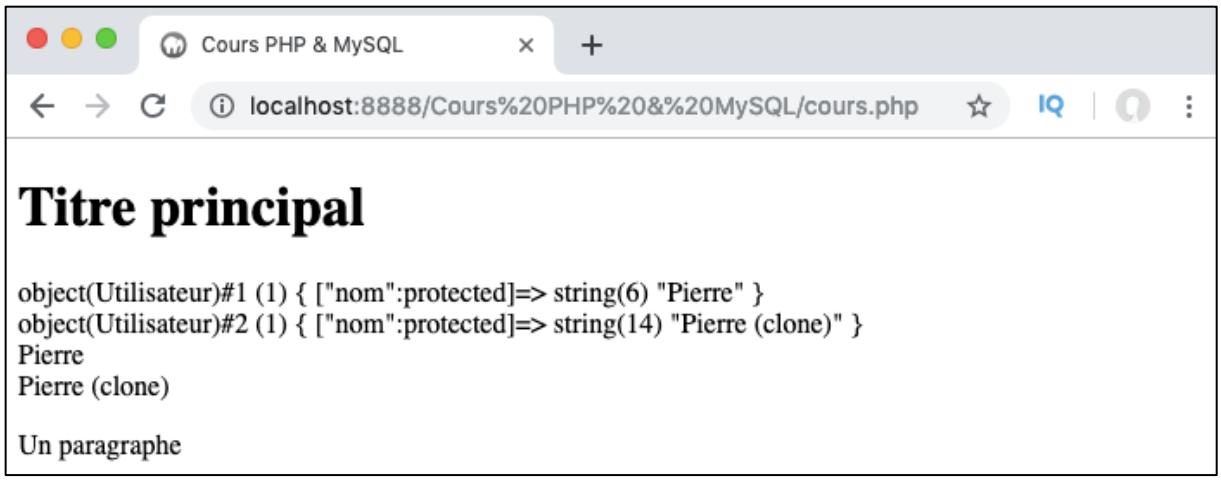
Lorsque l'on clone un objet, le PHP va en fait réaliser une copie « superficielle » de toutes les propriétés relatives à l'objet, ce qui signifie que les propriétés qui sont des références à d'autres variables (objets) demeureront des références.

Dès que le clonage d'objet a été effectué, la méthode `__clone()` du nouvel objet (le clone) va être automatiquement appelée. Cela va généralement nous permettre de mettre à jour les propriétés souhaitées.

Exemple de clonage d'objets

Pour cloner un objet en pratique nous allons simplement devoir utiliser le mot clef `clone` et éventuellement pouvoir définir une méthode `__clone()` qui va nous permettre de mettre à jour les éléments du clone par rapport à l'original.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            class Utilisateur{
                protected $nom;

                public function __construct($n){
                    $this->nom = $n;
                }

                public function __clone(){
                    $this->nom = $this->nom. ' (clone)';
                }
                public function getNom(){
                    echo $this->nom;
                }
            }

            $pierre = new Utilisateur('Pierre');
            $victor = clone $pierre;

            var_dump($pierre);
            echo '<br>';
            var_dump($victor);
            echo '<br>';
            $pierre->getNom();
            echo '<br>';
            $victor->getNom();

        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



On crée ici une classe qu'on appelle à `Utilisateur`. Cette classe possède une propriété `$nom`, un constructeur dont le rôle est d'assigner une valeur à `$nom` pour l'objet courant, une méthode `getNom()` qui sert à renvoyer la valeur de `$nom` de l'objet courant et finalement une méthode `__clone()`.

Le rôle de la méthode `__clone()` est ici de modifier la valeur stockée dans `$nom` pour le clone en lui ajoutant « (clone) ».

On instancie ensuite notre classe et on assigne le résultat à une variable objet `$pierre` qu'on va cloner grâce au mot clef `clone`.

Lors du clonage, notre clone `$victor` va recevoir une copie superficielle des propriétés et des méthodes de l'objet de départ. Dès que le clonage est terminé, la méthode `__clone()` est appelée et va ici mettre à jour la valeur de la propriété `$nom` pour notre clone.

On voit bien ici qu'on a créé une copie indépendante de l'objet de départ (et donc une nouvelle instance de la classe) et qu'on ne s'est pas contenté de créer une copie d'un identifiant pointant vers le même objet dès qu'on affiche le contenu de nos deux objets et qu'on tente d'accéder à la valeur de leur propriété `$nom`.

Comparer des objets

Plus tôt dans ce cours nous avons appris à comparer différentes variables qui stockaient des valeurs simples (une chaine de caractères, un chiffre, un booléen, etc.).

On va également pouvoir comparer des variables objets de manière similaire, ce qui va pouvoir être très utile pour s'assurer par exemple qu'un objet est unique.

Principe de la comparaison d'objets

Pour comparer deux variables objets entre elles, nous allons à nouveau devoir utiliser des opérateurs de comparaison. Cependant, étant donné que les valeurs comparées vont cette fois-ci être des valeurs complexes (car un objet est composé de diverses propriétés et méthodes), nous n'allons pas pouvoir utiliser ces opérateurs de comparaison aussi librement que lors de la comparaison de valeurs simples.

La première chose à savoir est qu'on ne va pouvoir tester que l'égalité (en valeur ou en identité) entre les objets. En effet, cela n'aurait aucun sens de demander au PHP si un objet est « inférieur » ou « supérieur » à un autre puisqu'un objet regroupe un ensemble de propriétés et de méthodes.

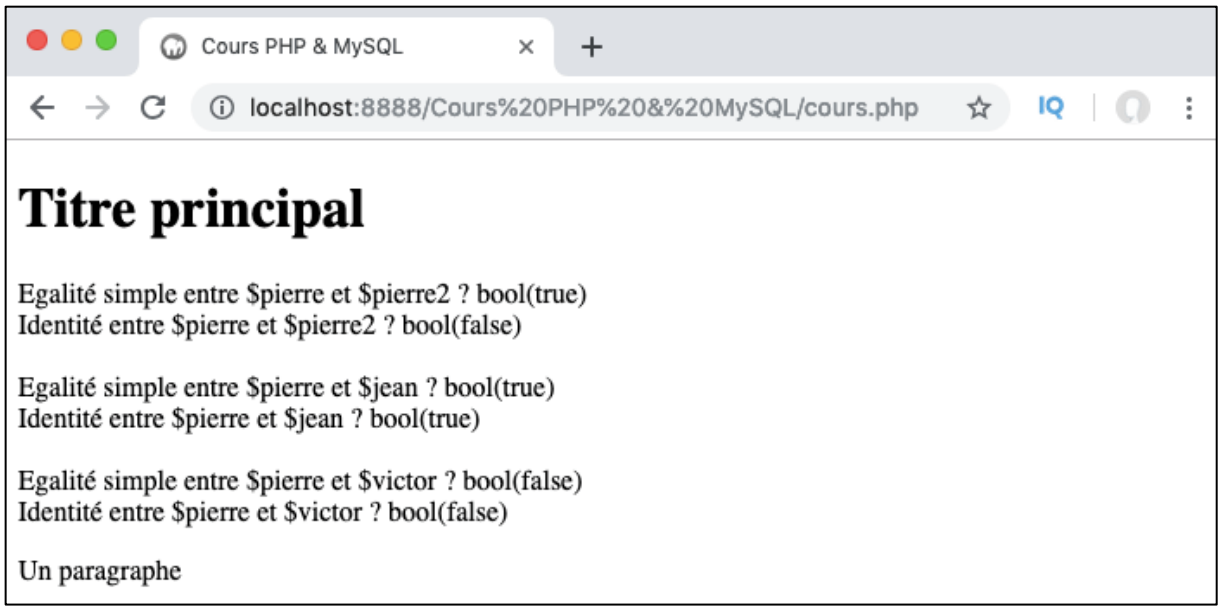
En utilisant l'opérateur de comparaison simple `==`, les objets vont être considérés comme égaux s'ils possèdent les mêmes attributs et valeurs (valeurs qui vont être comparées à nouveau avec `==` et si ce sont des instances de la même classe.

En utilisant l'opérateur d'identité `===`, en revanche, les objets ne vont être considérés comme égaux que s'ils font référence à la même instance de la même classe.

Comparer des objets en pratique

Pour illustrer la façon dont le PHP va comparer différents objets et pour commenter les résultats renvoyés, nous allons nous baser sur l'exemple suivant :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            class Utilisateur{
                protected $nom;
                public function __construct($n){
                    $this->nom = $n;
                }
                public function __clone(){
                    $this->nom = $this->nom. ' (clone)';
                }
                public function getNom(){
                    echo $this->nom;
                }
            }

            $pierre = new Utilisateur('Pierre');
            $pierre2 = new Utilisateur('Pierre');
            $jean = $pierre;
            $victor = clone $pierre;

            echo 'Egalité simple entre $pierre et $pierre2 ? ';
            var_dump($pierre == $pierre2);
            echo '<br>Identité entre $pierre et $pierre2 ? ';
            var_dump($pierre === $pierre2);
            echo '<br><br>Egalité simple entre $pierre et $jean ? ';
            var_dump($pierre == $jean);
            echo '<br>Identité entre $pierre et $jean ? ';
            var_dump($pierre === $jean);
            echo '<br><br>Egalité simple entre $pierre et $victor ? ';
            var_dump($pierre == $victor);
            echo '<br>Identité entre $pierre et $victor ? ';
            var_dump($pierre === $victor);

        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on réutilise notre classe `Utilisateur` créée précédemment. Cette classe définit une propriété `$nom` ainsi qu'un constructeur qui va initialiser notre propriété, une méthode `getNom()` dont le rôle est de renvoyer la valeur de `$nom` de l'objet courant et une méthode `__clone()` qui va mettre à jour la valeur de la propriété `$nom` d'un clone.

On instancie ensuite deux fois notre classe `Utilisateur` et on stocke le résultat dans deux variables objet `$pierre` et `$pierre2`.

En dessous, on assigne le contenu de `$pierre` dans une nouvelle variable objet `$jean`. Je vous rappelle ici que nos deux objets contiennent deux copies d'un identifiant pointant vers le même objet (la même instance de la classe).

Finalement, on crée un clone de `$pierre` grâce au mot clef `clone` qu'on appelle `$victor`. Ici, une véritable copie de l'objet est créée et donc nos deux variable objets vont bien représenter deux instances différentes et indépendantes de la classe. Dès que le clonage est terminé, notre méthode `__clone()` est appelée et la valeur de `$nom` de `$victor` est mise à jour.

L'idée va alors ici être de comparer nos différents objets. Pour cela, on utilise les opérateurs de comparaison `==` et `===` qui renvoient 1 si la comparaison réussit ou 0 si la comparaison échoue. On passe la valeur renvoyée à la fonction `var_dump()` afin qu'elle nous renvoie le type de résultat (la valeur 1 correspond au booléen `true` tandis que 0 correspond à `false`.

La première comparaison simple entre `$pierre` et `$pierre2` réussit. En effet, nos deux objets sont issus de la même classe et leur propriété `$nom` contient la même valeur puisqu'on a passé « Pierre » au constructeur dans les deux cas.

En revanche, la comparaison en termes d'identité échoue. La raison est que nos deux objets ont été créés en instanciant la classe à chaque fois et représentent donc deux instances différentes de la classe.

Cela n'est pas le cas pour notre objet `$jean` qui contient la copie d'un identifiant pointant vers le même objet (la même instance de la classe) que `$pierre` et qui va donc pouvoir être comparé à `$pierre` en termes d'identité avec succès.

Notre dernier objet, `$victor`, est un clone (ou une copie) de `$pierre` et stocke donc une instance différente de `$pierre`. De plus, juste après que le clonage ait été terminé, la valeur de `$nom` du clone a été mise à jour. La comparaison entre `$victor` et `$pierre` échoue donc à la fois en termes d'identité et en termes de comparaison simple.

PARTIE XII

Espaces de noms, filtres et gestion des erreurs

Les espaces de noms

Dans cette nouvelle leçon, nous allons définir ce que sont les espaces de noms et découvrir leurs usages en PHP.

Présentation des espaces de noms

Les espaces de nom en PHP sont des sortes de dossiers virtuels qui vont nous servir à encapsuler (c'est-à-dire à isoler) certains éléments de certains autres.

Les espaces de noms vont notamment permettre d'éliminer les conflits possibles entre deux éléments de même nom. Cette ambiguïté autour du nom de plusieurs éléments de même type peut survenir lorsqu'on a défini des fonctions, classes ou constantes personnalisées et qu'on fait appel à des extensions ou des bibliothèques externes PHP.

Ici, vous devez savoir qu'une extension ou une bibliothèque externe est un ensemble de code qui ne fait pas partie du langage nativement mais qui a pour but de rajouter des fonctionnalités au langage de base en proposant notamment par exemple des classes préconstruites pour créer une connexion à une base de données ou pour filtrer des données externes.

Il est possible que certaines classes, fonctions ou constantes d'une extension qu'on va utiliser dans un script possèdent des noms identiques à des classes, fonctions ou constantes personnalisées qu'on a défini dans un script.

Sans définition d'un espace de noms, cela va évidemment créer des conflits, puisque le reste du script utilisant la fonction, classe ou constante ne va pas savoir à laquelle se référer.

Pour prendre un exemple concret, vous pouvez considérer que les espaces de noms fonctionnent comme les dossiers sur notre ordinateur. Alors qu'il est impossible de stocker deux fichiers de même nom dans un même dossier, on va tout à fait pouvoir stocker deux fichiers de même nom dans deux dossiers différents. En effet, dans ce dernier cas, il n'y a plus d'ambiguïté puisque les deux fichiers ont un chemin d'accès et de fait une adresse différente sur notre ordinateur.

Définir un espace de noms simple

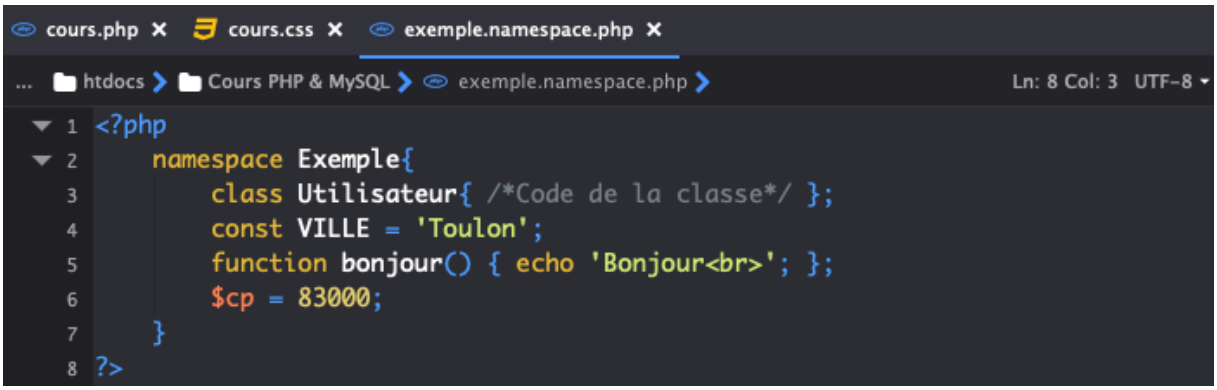
Pour définir un espace de noms, on va utiliser le mot clef `namespace` suivi de notre espace de noms.

Notez déjà que les noms d'espaces de noms ne sont pas sensibles à la casse et qu'on ne va pas pouvoir définir un espace de noms avec un nom commençant par « PHP ».

Notez également que seuls les classes, les traits, les interfaces, les fonctions et les constantes vont être affectés par un espace de noms c'est-à-dire vont pouvoir effectivement être différenciés du reste du code grâce à l'espace de noms.

On va pouvoir placer d'autres types d'éléments comme des variables dans un espace de noms mais ces éléments ne seront pas affectés par l'espace de noms.

Par ailleurs, vous devez également savoir qu'un espace de noms doit être déclaré avant tout autre code dans un fichier à l'exception de la commande `declare` qui peut être déclarée avant.




```php
<?php
    namespace Exemple{
        class Utilisateur{ /*Code de la classe*/ };
        const VILLE = 'Toulon';
        function bonjour() { echo 'Bonjour<br>'; };
        $cp = 83000;
    }
?>
```

Ici, on crée un premier espace de noms simple qu'on appelle `Exemple` dans un fichier séparé. J'enregistre mon fichier sous le nom `exemple.namespace.php`. Ensuite, on définit différents éléments dans notre espace de noms.

Notez qu'il existe deux syntaxes pour la déclaration d'un espace de noms : une syntaxe utilisant un point-virgule de type `namespace MonEspaceDeNoms ;` et une syntaxe utilisant plutôt un couple de parenthèses comme ci-dessus.

Personnellement, je vous conseille la syntaxe avec les parenthèses qui est plus claire et qui fonctionnent dans toutes les situations au contraire de celle avec le point-virgule.

Notez ici qu'aucun autre code PHP ne va pouvoir être défini en dehors d'un espace de noms. Si on souhaite créer du code « global » (du code en dehors d'un espace de noms défini), il faudra déclarer un espace de noms sans nom, en utilisant simplement le mot clef `namespace`.

```php
<?php
    namespace Exemple{
        class Utilisateur{ /*Code de la classe*/ };
        const VILLE = 'Toulon';
        function bonjour() { echo 'Bonjour<br>'; };
        $cp = 83000;
    }
    namespace{
        //Code global
    }
?>
```

Finalement, notez également qu'on va pouvoir définir et utiliser le même espace de noms dans plusieurs fichiers ce qui va être très utile pour scinder le contenu d'un espace de noms entre plusieurs fichiers.

On va également en théorie pouvoir définir plusieurs espaces de noms dans un même fichier mais cela est considéré comme une mauvaise pratique et donc on essaiera tant que possible de ne pas faire cela en pratique.

Définir un sous espace de noms

De la même façon qu'avec nos dossiers réels d'ordinateur, nous allons pouvoir définir des niveaux de hiérarchie d'espaces de noms et ainsi créer des sous espaces de noms.
Pour cela, on va préciser les différents noms liés aux niveaux et séparés par des antislashs.

Ici, j'enregistre mon sous espace de noms dans un nouveau fichier que j'appelle sousexemple.namespace.php. Mis à part la relation hiérarchique entre nos deux espaces, ceux-ci sont complètement différenciés et vont pouvoir contenir les éléments qu'ils souhaitent.

Accéder aux éléments d'un espace de noms

Pour utiliser des éléments d'un espace de noms en particulier, il va falloir que PHP sache à quel espace de noms on fait référence. Pour cela, des règles similaires à la recherche d'un fichier sur notre ordinateur sont utilisées en PHP.

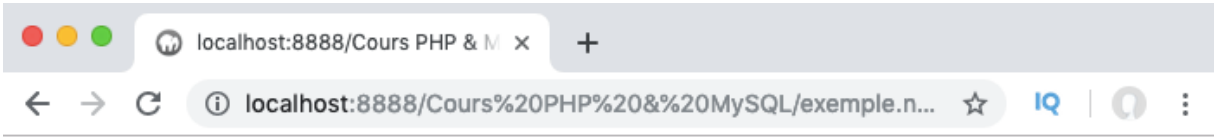
Pour accéder à un élément d'un espace de noms, on peut déjà préciser un nom non qualifié, c'est-à-dire ne préciser que le nom de l'élément en question.

Si on précise le nom d'un élément sans qualificatif (c'est-à-dire sans préfixe) dans un espace de noms nommé, alors c'est l'élément de même nom dans l'espace nommé qui sera utilisé.

Si on précise le nom d'un élément sans qualificatif depuis l'espace global, alors c'est l'élément de même nom dans l'espace global qui sera utilisé.

Attention ici : dans le cas où on souhaite utiliser une fonction ou une constante avec un nom sans qualificatif depuis un espace de noms nommé et que la fonction ou la constante en question n'est pas trouvée dans l'espace, alors une fonction / constante du même nom sera cherchée dans l'espace global. Ce ne sera pas le cas pour une classe.

```php
<?php
    namespace Exemple{
        class Utilisateur{ /*Code de la classe*/ };
        const VILLE = 'Toulon';
        function bonjour() { echo 'Bonjour<br>'; };
        $cp = 83000;

        /*Tente d'appeler bonsoir() depuis l'espace de noms courant, puis
         *depuis l'espace global en cas d'échec*/
        bonsoir();
    }
    namespace{
        function bonsoir(){
            echo 'Bonsoir !';
        }
    }
?>
```



Bonsoir !

On peut également utiliser un nom qualifié pour utiliser un élément d'un espace de noms. Le nom qualifié correspond au nom de l'élément préfixé par son chemin d'accès à partir de l'endroit où il est appelé (c'est-à-dire le nom de l'élément + le nom des sous espaces de noms séparés par des antislashs).

Si on utilise l'écriture `sous\bonjour()` depuis notre espace de noms `Exemple`, par exemple, on indique qu'on souhaite accéder à la fonction `bonjour()` située dans `exemple\sous\bonjour()`.

Finalement, on peut encore préciser un nom absolu pour accéder à un élément d'un espace de noms. Un nom absolu correspond au chemin complet de l'élément, c'est-à-dire au nom de l'élément préfixé de tous les espaces et sous espaces et commençant avec un antislash.

```php
<?php
    namespace Exemple{
        //On inclut notre espace de noms exemple\sous
        include 'sousexemple.namespace.php';
        class Utilisateur{ /*Code de la classe*/ };
        const VILLE = 'Toulon';
        function bonjour() { echo 'Bonjour<br>'; };
        $cp = 83000;

        bonjour();
        sous\bonjour();
        \exemple\sous\bonjour();
    }
    namespace{
        //Code global
    }
?>
```

cours.php × cours.css × exemple.namespace.php × sousexemple.namespace.php ×

... MAMP > htdocs > Cours PHP & MySQL > cours.php > Ln: 20 Col: 1 UTF-8

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            include 'exemple.namespace.php';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Cours PHP & MySQL

localhost:8888/Cours%20PHP%20&%20MySQL/cours.php

Titre principal

Bonjour
Salut
Salut

Un paragraphe

La commande namespace et la constante magique __NAMESPACE__

La constante magique `__NAMESPACE__` contient le nom de l'espace de noms courant sous forme de chaine de caractères. Dans le cas où on l'appelle depuis l'espace global, elle contient une chaine vide.

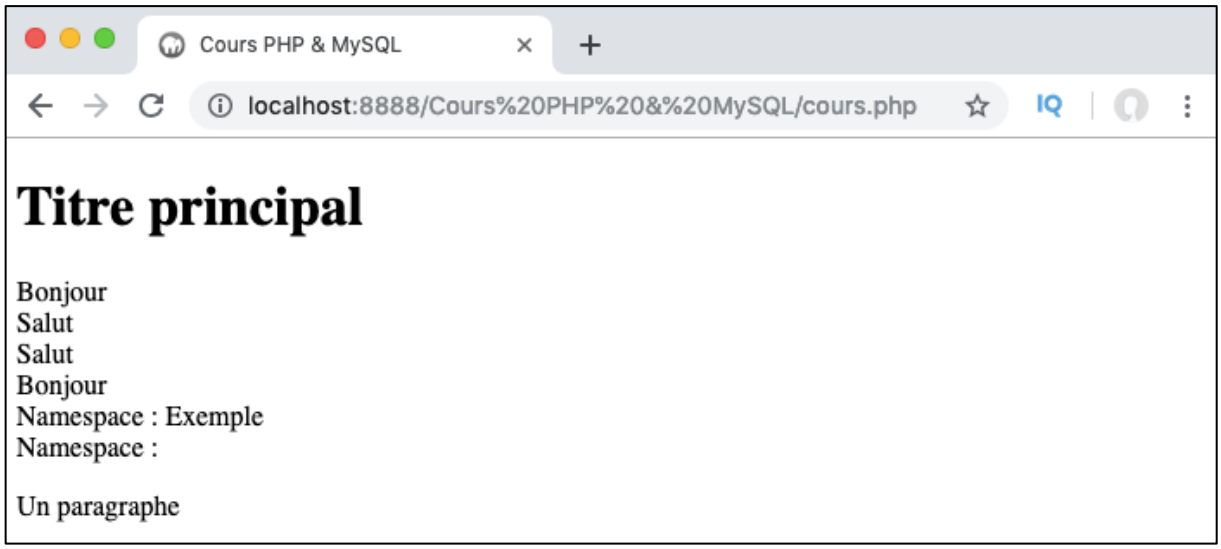
On va pouvoir utiliser cette constante pour récupérer le nom de l'espace de noms courant et accéder à ses éléments ou à d'autres éléments en partant de ce nom dans des situations où on n'a pas accès directement au nom.

La commande `namespace` va représenter l'espace de noms courant. C'est l'équivalent de l'opérateur `self` des classes vu précédemment mais pour les espaces de noms.

On va donc également pouvoir s'en servir pour accéder à des éléments d'un espace de noms en représentant cet espace à l'aide de la commande `namespace`.

```php
<?php
    namespace Exemple{
        //On inclut notre espace de noms exemple\sous
        include 'sousexemple.namespace.php';
        class Utilisateur{ /*Code de la classe*/ };
        const VILLE = 'Toulon';
        function bonjour() { echo 'Bonjour<br>'; };
        $cp = 83000;

        bonjour();
        sous\bonjour();
        \exemple\sous\bonjour();
        namespace\bonjour();

        echo 'Namespace : ' .__NAMESPACE__. '<br>';
    }
    namespace{
        echo 'Namespace : ' .__NAMESPACE__. '<br>';
    }
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            include 'exemple.namespace.php';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Accéder à un élément du contexte global depuis un espace de noms

Lorsqu'on ne définit pas d'espace de noms, les éléments (classes, fonctions, etc.) sont automatiquement placés dans l'espace de noms global.

Pour accéder à un élément de l'espace global depuis un espace de noms, on peut préfixer le nom de l'élément avec un simple antislash.

```php
<?php
    namespace Exemple{
        //On inclut notre espace de noms exemple\sous
        include 'sousexemple.namespace.php';
        class Utilisateur{ /*Code de la classe*/ };
        const VILLE = 'Toulon';
        function bonjour() { echo 'Bonjour<br>'; };
        $cp = 83000;

        bonjour();
        \bonjour();
    }
    namespace{
        function bonjour(){
            echo 'Bonjour depuis l\'espace global';
        }
    }
?>
```

Importation (des éléments) d'un espace de noms et création d'alias

Créer un « alias » en PHP correspond à définir un autre nom pour désigner un élément déjà existant dans notre script.

Pour importer un élément d'un espace de noms, nous allons utiliser l'instruction `use` qu'on a déjà eu l'occasion de rencontrer dans la leçon relative aux traits.

Cette instruction, en important un élément d'un espace de noms, va également automatiquement créer un alias en utilisant le nom en soi de l'élément. On va cependant également pouvoir définir un alias personnalisé avec le mot clef `as`.

Notez qu'avant la version 5.6 de PHP, nous ne pouvions importer et définir d'alias que pour un nom de classe, un nom d'interface et pour un espace de noms en soi.

Depuis PHP 5.6, on peut également importer et créer un alias personnalisé pour une fonction et importer une constante.

Notez également que depuis PHP 7.0, les classes, fonctions et constantes importées depuis le même espace de noms peuvent être regroupées dans une seule instruction use. Enfin, lorsqu'on précise un nom absolu lors d'une importation avec use, il faut savoir qu'il est recommandé d'omettre l'antislash initial dans le chemin du fichier.

Les exemples ci-dessous illustrent comment importer et créer des alias des différents éléments d'un espace de noms.

```php
<?php
    namespace Exemple\Sous{
        class Utilisateur{ /*Code de la classe*/ };
        const VILLE = 'Lyon';
        function bonjour() { echo 'Salut<br>'; };
        function bonsoir() { echo 'Bonsoir<br>'; };
        function bonne_nuit() { echo 'Bonne nuit<br>'; };
        $cp = 69000;
    }
?>
```

```php
<?php
    namespace Exemple{
        //On inclut notre espace de noms exemple\sous
        include 'sousexemple.namespace.php';
        class Utilisateur{ /*Code de la classe*/ };
        const VILLE = 'Toulon';
        function bonjour() { echo 'Bonjour<br>'; };
        $cp = 83000;

        use Exemple\Sous; //Revient à écrire use Exemple\Sous as Sous
        use Exemple\Sous\Utilisateur as Sousutil; //Alias d'une classe
        use function Exemple\Sous\bonjour as bjr; //Alias d'une fonction
        use const Exemple\Sous\VILLE; //Importation d'une constante

        //Import et alias de plusieurs fonctions avec une instruction use
        use function Exemple\Sous\{bonsoir as bns, bonne_nuit as nuit};

        var_dump($obj = new Sousutil);
        echo '<br>';
        bjr();
        echo VILLE;
        echo '<br>';
        bns();
        nuit();
    }
    namespace{
        function bonjour(){
            echo 'Bonjour depuis l\'espace global';
        }
    }
?>
```

Notez finalement que dans le cas d'une inclusion de fichiers, les fichiers inclus n'hériteront pas des règles d'importation du fichier parent puisque ces règles ne concernent que le fichier dans lequel elles ont été définies.

Présentation des filtres

Dans cette nouvelle leçon, nous allons découvrir l'extension filter et ce que sont les filtres PHP. Nous allons pour le moment particulièrement nous attarder sur les fonctions de cette extension.

Présentation des filtres et de l'extension PHP filter

Les filtres en PHP vont nous permettre de filtrer des données et notamment des données externes comme des données provenant d'utilisateurs et qui ont été transmises par des formulaires.

Les filtres en PHP sont disponibles via l'extension filter qui est une extension activée par défaut depuis PHPH 5.2. Il n'y a donc aucune manipulation à faire pour utiliser les filtres.

Il existe deux grands types de filtres en PHP : des filtres de validation et des filtres de nettoyage.

« Valider » des données correspond à déterminer si les données reçues possèdent la forme attendue. Par exemple, on va pouvoir vérifier si une adresse email possède bien un caractère « @ ».

« Nettoyer » ou « assainir » des données correspond à retirer les caractères indésirables de celles-ci. Nous allons par exemple pouvoir supprimer des espaces non nécessaires ou certains caractères spéciaux gênants.

Des options ou des « drapeaux » vont également pouvoir être utilisées avec certains filtres pour préciser leur comportement dans le cas où on ait un besoin spécifique. Nous aurons l'occasion d'illustrer cela plus tard.

A noter que l'extension filtre, en plus de fournir une liste de filtres puissants, possède également des fonctions et des constantes prédéfinies qu'on va pouvoir utiliser.

Les fonctions de l'extension filtre et la liste des filtres disponibles

L'extension filter de PHP nous fournit donc différentes fonctions et filtres qui vont nous permettre de vérifier la conformité des données envoyées par rapport à ce qu'on attend. Pour être exact, l'extension filter met les fonctions suivantes à notre disposition :

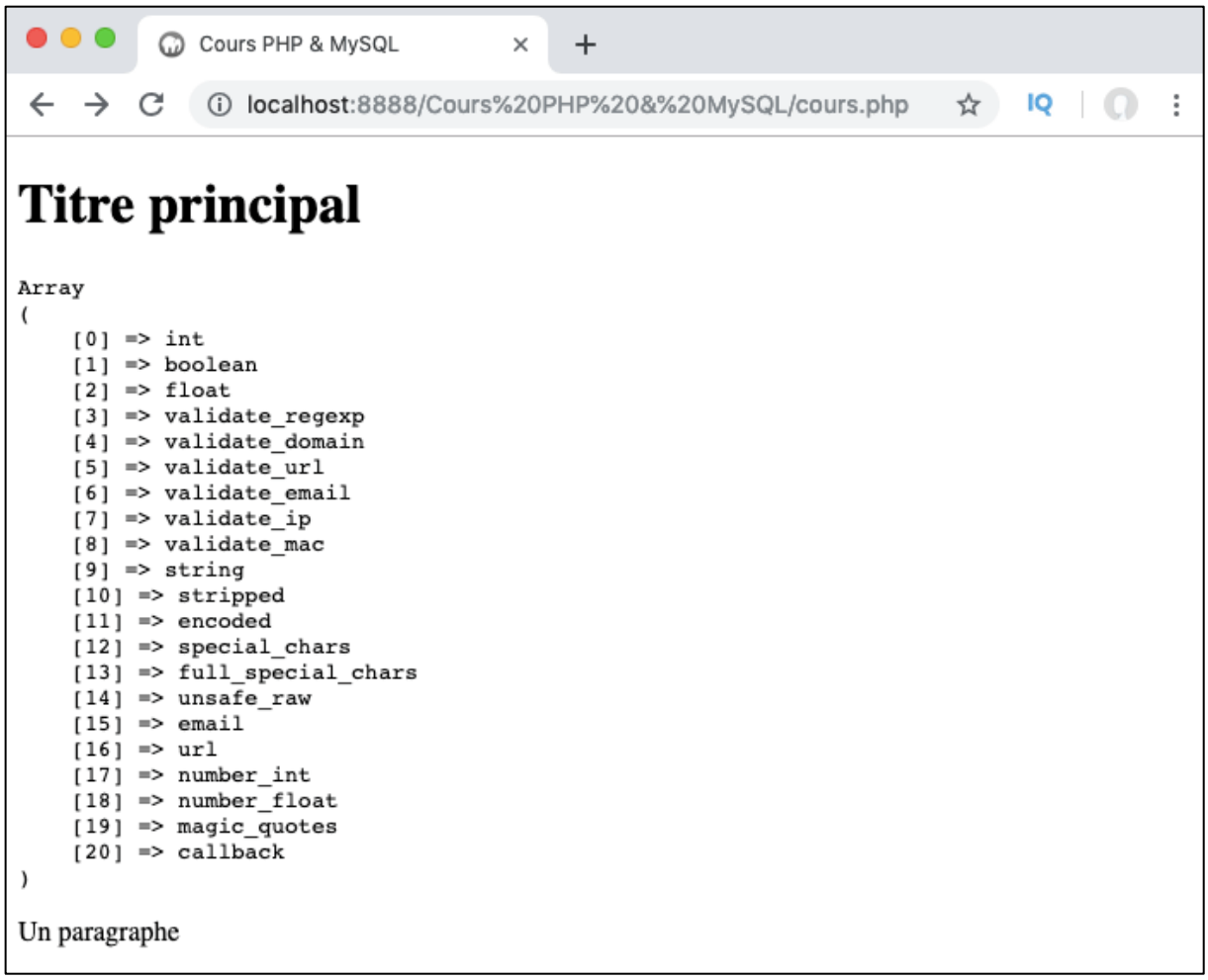
Fonction	Description
filter_list()	Retourne une liste de tous les filtres supportés
filter_id()	Retourne l'identifiant d'un filtre nommé

Fonction	Description
filter_input()	Récupère une variable externe et la filtre
filter_var()	Filtre une variable avec un filtre spécifique
filter_var_array()	Récupère plusieurs variables et les filtre
filter_input_array()	Récupère plusieurs variables externes et les filtre
filter_has_var()	Vérifie si une variable d'un type spécifique existe

Nous allons utiliser la plupart de ces fonctions dans la suite de cette leçon. Pour le moment, essayons déjà d'obtenir la liste des filtres disponibles en utilisant la fonction `filter_list()`.

La fonction `filter_list()` renvoie un tableau avec une liste de noms de tous les filtres qu'on va pouvoir utiliser.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo '<pre>';
            print_r(filter_list());
            echo '</pre>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



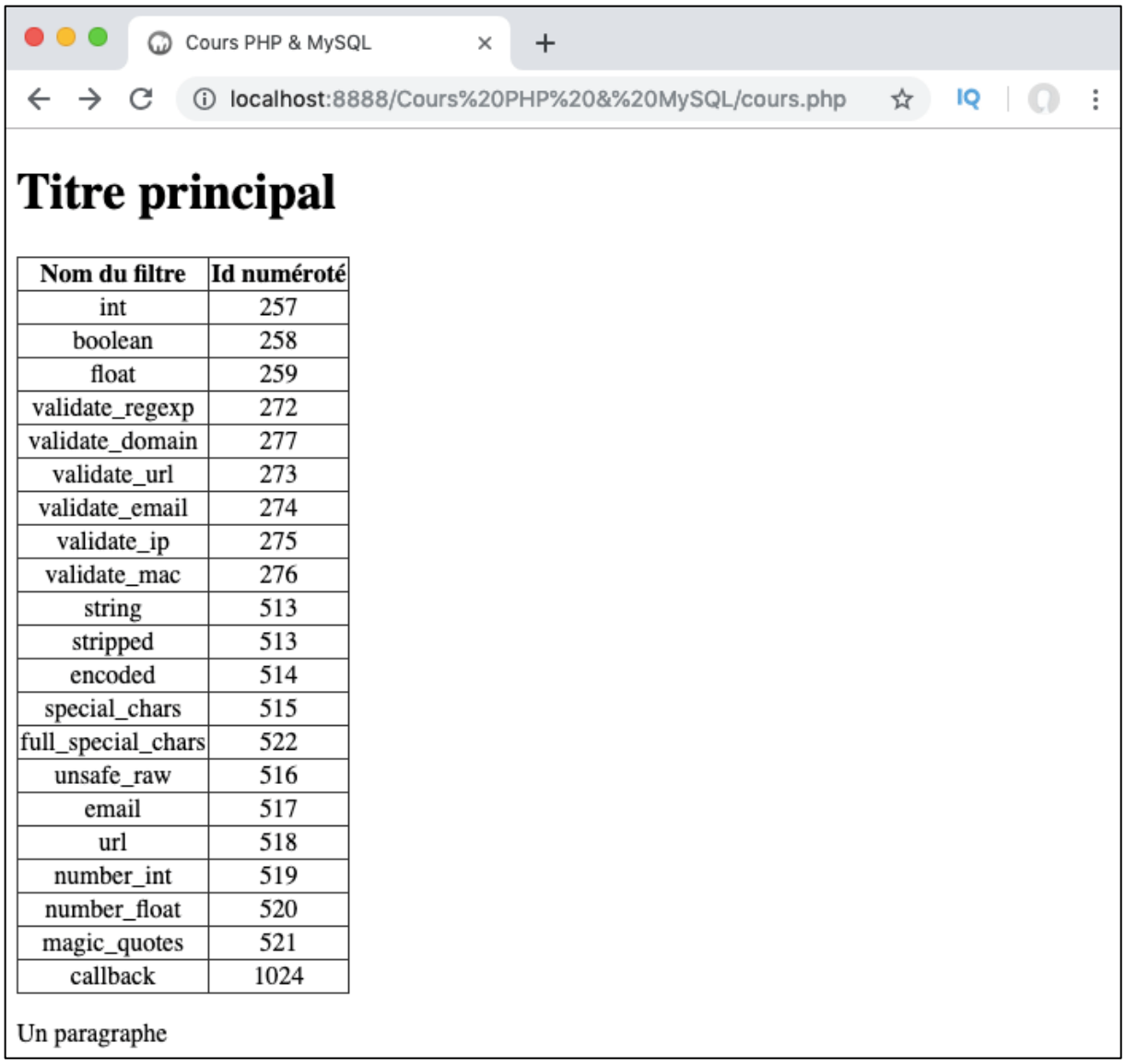
En plus du simple nom des filtres, vous devez savoir que l'extension `filter` utilise d'autres moyens d'identifier un filtre et notamment un système d'id nommés et numérotés. L'id d'un filtre sous forme de chaine de caractères est constitué du mot `FILTER` suivi du type de filtre et suivi du nom du filtre.

Par exemple, l'id nommé du filtre `string` est `FILTER_SANITIZE_STRING`. Notez que certaines fonctions vont accepter le nom des filtres tandis que d'autres vont accepter un id du filtre pour fonctionner.

Pour obtenir l'id numéroté d'un filtre, on va pouvoir utiliser la fonction `filter_id()`. Cette fonction va prendre un nom de filtre en argument et retourner l'id correspond au filtre si le nom passé correspond bien à un filtre ou `false` dans le cas contraire.

Pour obtenir directement les id de chaque filtre disponible, on va pouvoir créer une boucle `foreach` en bouclant sur les valeurs renvoyées par `filter_list()` et en passant les noms de filtre à `filter_id()`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            echo '
                <table>
                    <tr>
                        <th>Nom du filtre</th>
                        <th>Id numéroté</th>
                    </tr>';
            $filtres_tb = filter_list();
            foreach($filtres_tb as $clef => $nom){
                echo '
                    <tr>
                        <td>' .$nom. '</td>
                        <td>' .filter_id($nom). '</td>
                    </tr>';
            }
            echo '</table>'
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on choisit de retourner un tableau HTML pour une meilleure présentation. Notez que j'ai déclaré certains styles pour mon tableau dans mon fichier `cours.css` et notamment appliqué des bordures entre les cellules de mon tableau.

Nettoyer et valides des données en PHP : choisir la fonction et le filtre adaptés

Pour filtrer des données et avoir le résultat attendu, il va avant tout falloir utiliser la bonne fonction de l'extension `Filter`.

Si on ne souhaite filtrer qu'une donnée en particulier, alors on pourra utiliser les fonctions `filter_var()` ou `filter_input()` dans le cas où la variable à filtrer est une variable externe au script comme une donnée envoyée via un formulaire par exemple.

Pour filtrer plusieurs données en même temps, on va pouvoir utiliser les fonctions `filter_var_array()` ou `filter_input_array()` pour des variables externes.

Finalement, si on souhaite tester l'existence d'un type spécifique de variable, on pourra utiliser la fonction filter_has_var(). Ici, il faut comprendre « type » au sens de la provenance de la variable (récupérées via un formulaire et get ou <code<post, via un cookie, etc).

Dans ce cours, nous allons nous concentrer sur les fonctions filter_var(), filter_input() et filter_has_var().

Pour utiliser la fonction filter_var(), nous allons devoir lui passer le nom de la variable que l'on souhaite filtrer, l'id (sous forme de nombre ou de chaine de caractères) du filtre que l'on souhaite appliquer à la variable et facultativement un tableau associatif d'options ou des drapeaux qui vont servir à préciser notre filtre.

La fonction filter_var() va retourner les données filtrées en cas de succès ou false si le filtre échoue.

La fonction filter_input() va s'utiliser de manière similaire à filter_var() à la différence qu'on va également devoir lui passer en tout premier argument une constante qui va indiquer la façon dont les données ont été transmises pour lever toute ambiguïté sur la variable qu'on souhaite filtrer.

On va pouvoir choisir parmi les constantes suivantes :

- INPUT_GET : donnée récupérée via un formulaire et la méthode get ;
- INPUT_POST : donnée récupérée via un formulaire et la méthode post ;
- INPUT_COOKIE : donnée récupérée via un cookie ;
- INPUT_SERVER : donnée de serveur ;
- INPUT_ENV : données d'environnement ;

La fonction filter_input() va également retourner les données filtrées en cas de succès ou false si le filtre échoue.

Finalement, nous allons devoir passer à la fonction filter_has_var() une constante parmi la liste ci-dessus ainsi que le nom de la variable dont on souhaite vérifier l'existence.

Filtres de validation, de nettoyage et drapeaux

Maintenant que nous avons une première idée du fonctionnement et du résultat des fonctions de l'extension filtre, il est temps de les utiliser en pratique.

Pour utiliser ces fonctions intelligemment, il va falloir renseigner le filtre qui répond à nos besoins. Il existe deux grands types de filtres : les filtres de validation et de nettoyage.

Les filtres de validation vont permettre, comme leur nom l'indique, de « valider » des données c'est-à-dire de vérifier que les données filtrées possèdent bien une certaine forme.

Les filtres de nettoyage vont eux nous permettre de « nettoyer » des données, c'est-à-dire par exemple de supprimer certains caractères non voulus comme des caractères spéciaux.

Dans cette leçon, nous allons présenter les différents filtres disponibles dans chaque groupe et également illustrer le comportement de certains d'entre eux à travers différent exemples.

Nous allons également présenter et utiliser les options et les drapeaux des filtres qui vont nous servir à préciser ou à modifier le comportement par défaut d'un filtre.

Les filtres de validation

Les filtres de validation vont nous permettre de vérifier qu'une certaine donnée possède une forme conforme à ce qu'on attend. Ces filtres se basent sur des règles de validation précises et parfois complexes qui sont issues de certaines normes. Le filtre de validation de mail, par exemple, va notamment vérifier qu'une donnée possède bien un caractère « @ » entre autres.

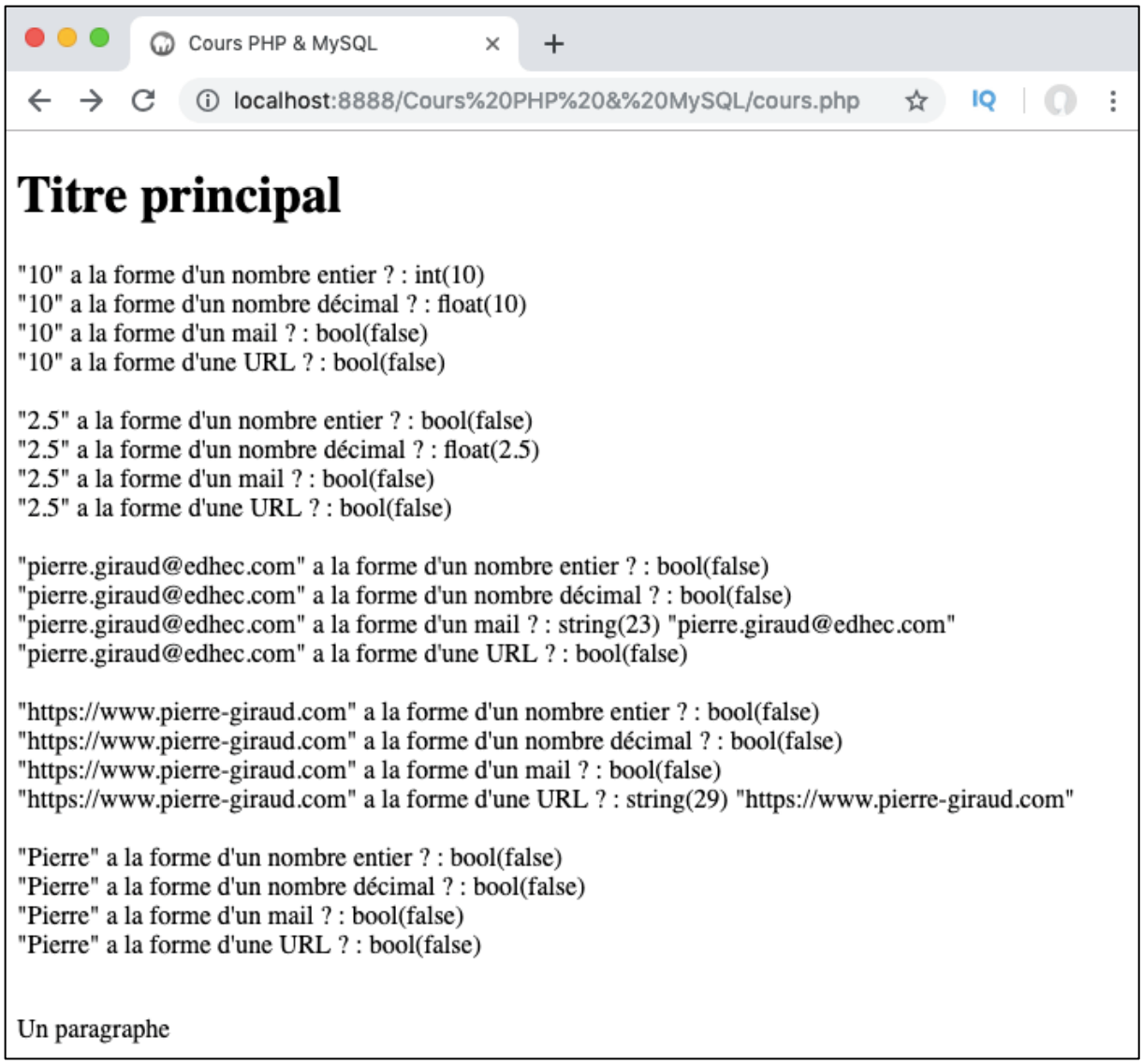
Nous allons pouvoir utiliser les filtres de validation suivants :

Nom du filtre	Id (texte) du filtre	Id (nb) du filtre	Description
boolean	FILTER__VALIDATE_BOOLEAN	258	Retourne `true` pour « 1 », « true », « on » et « yes » et `false` sinon. De fait, s'utilise souvent avec le drapeau `FILTER_NULL_ON_FAILURE` qui fait que le filtre ne retournera `false` que pour « 0 », « false », « off » et « no » et `NULL` pour toutes les autres valeurs non booléennes.
validate_domain	FILTER__VALIDATE_DOMAIN	277	Permet de vérifier qu'une donnée a bien la forme d'un nom de domaine

Nom du filtre	Id (texte) du filtre	Id (nb) du filtre	Description
validate_email	FILTER__VALIDATE_EMAIL	274	Permet de vérifier qu'une donnée a bien la forme d'une adresse mail
float	FILTER__VALIDATE_FLOAT	259	Permet de vérifier qu'une donnée a bien la forme d'un nombre décimal
int	FILTER__VALIDATE_INT	257	Permet de vérifier qu'une donnée a bien la forme d'un nombre entier
validate_ip	FILTER__VALIDATE_IP	275	Permet de vérifier qu'une donnée a bien la forme d'une adresse IP. S'utilise souvent avec les drapeaux `FILTER_FLAG_IPV4` ou `FILTER_FLAG_IPV6` pour valider une IPv4 ou une IPv6 spécifiquement
validate_mac	FILTER__VALIDATE_MAC	276	Permet de vérifier qu'une donnée a bien la forme d'une adresse MAC
validate_regexp	FILTER__VALIDATE_REGEXP	272	Permet de vérifier qu'une donnée a bien la forme d'une expression rationnelle regexp compatible Perl
validate_url	FILTER__VALIDATE_URL	273	Permet de vérifier qu'une donnée a bien la forme d'une URL

Illustrons immédiatement le fonctionnement de ces filtres grâce à quelques exemples concrets d'application.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $tb = [
                10,
                2.5,
                'pierre.giraud@edhec.com',
                'https://www.pierre-giraud.com',
                'Pierre'
            ];

            foreach($tb as $valeur){
                echo '"' .$valeur. '" a la forme d\'un nombre entier ? : ';
                var_dump(filter_var($valeur, FILTER_VALIDATE_INT));
                echo '<br>"' .$valeur. '" a la forme d\'un nombre décimal ? : ';
                var_dump(filter_var($valeur, FILTER_VALIDATE_FLOAT));
                echo '<br>"' .$valeur. '" a la forme d\'un mail ? : ';
                var_dump(filter_var($valeur, FILTER_VALIDATE_EMAIL));
                echo '<br>"' .$valeur. '" a la forme d\'une URL ? : ';
                var_dump(filter_var($valeur, FILTER_VALIDATE_URL));
                echo '<br><br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on commence par créer un tableau numéroté qui stocke 5 valeurs. Nous allons déjà filtrer les valeurs de notre tableau en utilisant `filter_var()` (puisque les données sont ici internes = définies dans notre script) et différents filtres de validation.

Pour cela, on crée une boucle `foreach` qui va parcourir notre tableau. Pour chaque valeur du tableau, on va utiliser successivement 4 filtres qui sont les filtres `int`, `float`, validate_email et `validate_url`.

On choisit ici de passer leur id sous forme de texte à `filter_var()` mais on aurait aussi bien pu passer leur id numéroté. Si la validation réussit, la fonction `filter_var()` renvoie la donnée filtrée et dans le cas contraire renvoie `false`.

On utilise ici `var_dump()` en lui passant le résultat de `filter_var()` pour afficher finalement si la validation a réussi ou si elle a échoué.

Les filtres de nettoyage

Les filtres de nettoyage permettent de « préparer » des données en les nettoyant. Ils vont par exemple nous servir à nous débarrasser de certains caractères non souhaités lors de la réception de données et donc d'obtenir au final des données assainies qu'on va pouvoir manipuler et toute sécurité.

Les filtres de nettoyage sont donc bien différents des filtres de validation puisqu'à la différence de ces derniers ils ne vont pas nous permettre de valider la forme d'une donnée mais plutôt de modifier la forme des données reçues en suivant certains schémas qui vont dépendre du filtre utilisé.

Notez bien que les filtres de validation et les filtres de nettoyage sont souvent utilisés conjointement. On va par exemple pouvoir commencer par nettoyer une donnée reçue puis ensuite valider la forme de la donnée après nettoyage (nous verrons cela plus en détail dans la prochaine leçon).

Nous allons pouvoir utiliser les filtres de nettoyage suivants :

Nom du filtre	**Id (texte) du filtre**	**Id (nb) du filtre**	**Description**
email	FILTER_SANITIZE_EMAIL	517	Supprime tous les caractères sauf les lettres, chiffres, et les caractères !#$%&'*+-=?^_`{|}~@.[]
encoded	FILTER_SANITIZE_ENCODED	514	Applique l'encodage URL, et supprime ou encode les caractères spéciaux selon le drapeau passé
magic_quotes	FILTER_SANITIZE_MAGIC_QUOTES	521	Applique la function addslashes() qui ajoute des antislashs pour échapper les caractères qui doivent l'être dans une chaine
number_float	FILTER_SANITIZE_NUMBER_FLOAT	520	Supprime tous les caractères sauf les chiffres, les signes + et – et les expressions de type exponentielle (« e ») avec un drapeau
number_int	FILTER_SANITIZE_NUMBER_INT	519	Supprime tous les caractères sauf les chiffres et les signes + et –
special_chars	FILTER_SANITIZE_SPECIAL_CHARS	515	Transforme en entité HTML les caractères ' »<>& et les caractères ASCII de valeur inférieur à 32, et supprime ou

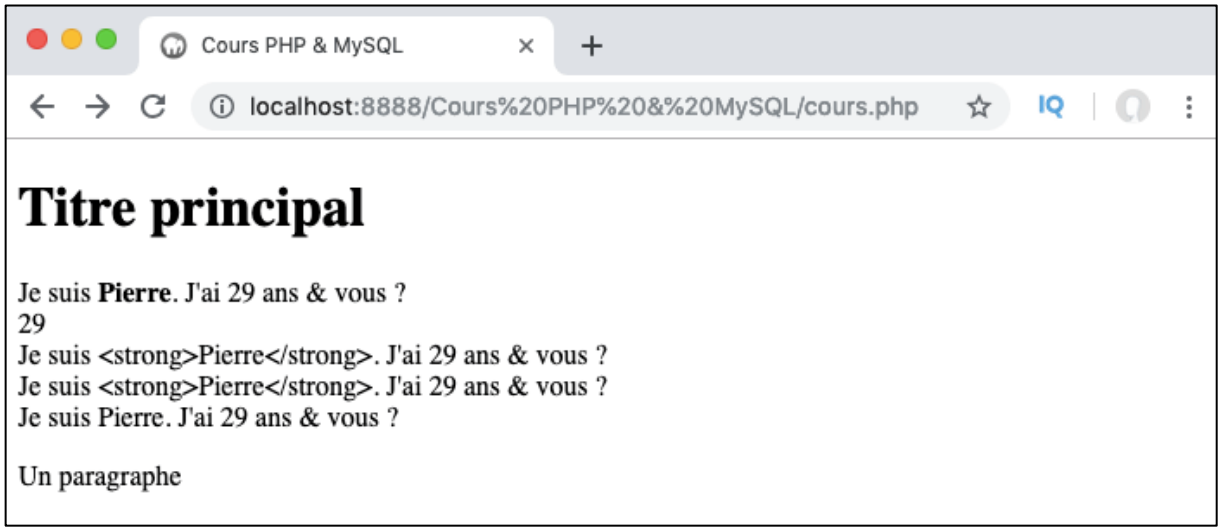
Nom du filtre	Id (texte) du filtre	Id (nb) du filtre	Description
			encode les autres caractères spéciaux selon le drapeau choisi
full_special_chars	FILTER_SANITIZE_FULL_SPECIAL_CHARS	522	Équivaut à appeler la fonction htmlspecialchars() qui convertit les caractères spéciaux en entités HTML avec ENT_QUOTES défini. ENT_QUOTES est un drapeau de htmlspecialchars() qui permet à la fonction de convertir les guillemets doubles et les guillemets simples.
string	FILTER_SANITIZE_STRING	513	Supprime les balises, et supprime ou encode les caractères spéciaux en fonction du drapeau choisi
stripped	FILTER_SANITIZE_STRIPPED	513	Ce filtre est un alias du filtre string
url	FILTER_SANITIZE_URL	518	Supprime tous les caractères sauf les lettres, chiffres et $-_.+!*'(),{}|\\^~[]`<>#% »;/?:@&=
unsafe_raw	FILTER_UNSAFE_RAW	516	Ne fait rien (par défaut) ou supprime ou encode les caractères spéciaux selon le drapeau choisi

Attention : certains noms se ressemblent entre les filtres de nettoyage et de validation tout simplement car certains filtres vont permettre de nettoyer ou de valider une donnée de même type comme un email ou un entier par exemple.

Vous pouvez noter ici que c'est la raison pour laquelle certains filtres de validation possèdent « validate » dans leur nom : tout simplement car le nom simple était déjà pris par un filtre de nettoyage. Faites donc bien attention à ne pas confondre les différents noms !

Utilisons immédiatement quelques filtres de nettoyage pour nettoyer une chaine de caractères par exemple :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $texte = 'Je suis <strong>Pierre</strong>. J\'ai 29 ans & vous ?';

            echo $texte. '<br>';
            echo filter_var($texte, FILTER_SANITIZE_NUMBER_INT). '<br>';
            echo filter_var($texte, FILTER_SANITIZE_SPECIAL_CHARS). '<br>';
            echo filter_var($texte, FILTER_SANITIZE_FULL_SPECIAL_CHARS). '<br>';
            echo filter_var($texte, FILTER_SANITIZE_STRING). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Notre chaine de caractères possède ici des balises HTML, ainsi que des caractères spéciaux et des caractères « chiffres ». On commence déjà par `echo` la chaine telle quelle pour voir le résultat « naturel ».

On utilise ensuite un filtre `FILTER_SANITIZE_NUMBER_INT` qui va tout supprimer dans la chaine sauf les caractères chiffres, le signe + et le signe – avec notre fonction `filter_var()`. Ici, pas la peine d'utiliser `var_dump()` car les filtres de nettoyage vont quasiment toujours renvoyer des valeurs et on va donc plutôt directement `echo` le résultat qui est le nombre 29 trouvé dans notre chaine.

Nos deuxième et troisième filtres `FILTER_SANITIZE_SPECIAL_CHARS` et `FILTER_SANITIZE_FULL_SPECIAL_CHARS` vont

par défaut convertir certains caractères spéciaux HTML et les transformer en entité ce qui va nous permettre d'échapper leur signification spéciale et de les afficher tels quels. Ici, les deux filtres renvoient le même résultat et comme vous pouvez le voir les balises « strong » ont bien été échappées.

Finalement, notre dernier filtre `FILTER_SANITIZE_STRING` va lui supprimer les balises HTML et encoder les autres caractères spéciaux.

Liste et utilisation des options et des drapeaux avec les filtres

Comme je vous le disais précédemment, on va également pouvoir utiliser des drapeaux ou des tableaux d'options avec la plupart de nos filtres.

Ces options et ces drapeaux vont nous permettre de modifier le comportement par défaut de nos filtres et / ou de les rendre plus ou moins strict.

Notez que les options vont devoir être passées via un tableau multidimensionnel qui devra être nommé `options`.

A titre d'information, voici la liste des drapeaux disponibles avec les filtres. Il n'est évidemment pas question de les apprendre par cœur, mais il est bon d'en connaitre quelques-uns et de savoir que des drapeaux existent.

Identifiant du drapeau	**Utilisé avec**	**Description**
FILTER_FLAG_STRIP_LOW	FILTER_SANITIZE_ENCODED, FILTER_SANITIZE_SPECIAL_CHARS, FILTER_SANITIZE_STRING, FILTER_UNSAFE_RAW	Supprime les caractères dont la valeur numérique est <32.
FILTER_FLAG_STRIP_HIGH	FILTER_SANITIZE_ENCODED, FILTER_SANITIZE_SPECIAL_CHARS, FILTER_SANITIZE_STRING, FILTER_UNSAFE_RAW	Supprime les caractères dont la valeur numérique est >127.
FILTER_FLAG_STRIP_BACKTICK	FILTER_SANITIZE_ENCODED, FILTER_SANITIZE_SPECIAL_CHARS, FILTER_SANITIZE_STRING, FILTER_UNSAFE_RAW	Supprime les caractères « accent grave ».

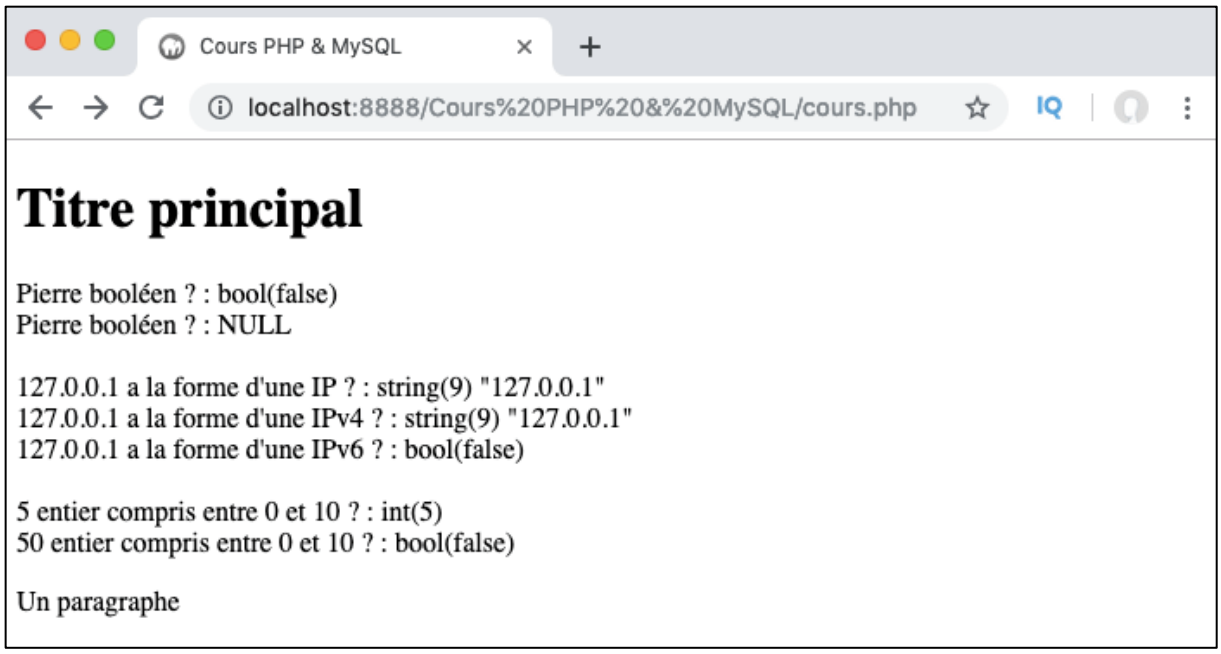
Identifiant du drapeau	Utilisé avec	Description
FILTER_FLAG_ALLOW_FRACTION	FILTER_SANITIZE_NUMBER_FLOAT	Autorise un point (.) comme séparateur fractionnaire pour les nombres.
FILTER_FLAG_ALLOW_THOUSAND	FILTER_SANITIZE_NUMBER_FLOAT, FILTER_VALIDATE_FLOAT	Autorise une virgule (,) comme séparateur fractionnaire pour les nombres.
FILTER_FLAG_ALLOW_SCIENTIFIC	FILTER_SANITIZE_NUMBER_FLOAT	Autorise un e ou un E pour la notation scientifique dans les nombres.
FILTER_FLAG_NO_ENCODE_QUOTES	FILTER_SANITIZE_STRING	Si ce drapeau est présent, les guillemets simples (‘) et les doubles («) ne seront pas encodés.
FILTER_FLAG_ENCODE_LOW	FILTER_SANITIZE_ENCODED, FILTER_SANITIZE_STRING, FILTER_SANITIZE_RAW	Encode tous les caractères dont la valeur numérique est <32.
FILTER_FLAG_ENCODE_HIGH	FILTER_SANITIZE_ENCODED, FILTER_SANITIZE_SPECIAL_CHARS, FILTER_SANITIZE_STRING, FILTER_SANITIZE_RAW	Encode tous les caractères dont la valeur numérique est >127.
FILTER_FLAG_ENCODE_AMP	FILTER_SANITIZE_STRING, FILTER_SANITIZE_RAW	Encode les &.
FILTER_NULL_ON_FAILURE	FILTER_VALIDATE_BOOLEAN	Retourne NULL pour les valeurs booléennes non reconnues.
FILTER_FLAG_ALLOW_OCTAL	FILTER_VALIDATE_INT	Prend en compte les nombres octaux précédés d’un zéro (0). Ceci ne fonctionne que pour les chiffres 0-7.
FILTER_FLAG_ALLOW_HEX	FILTER_VALIDATE_INT	Prend en compte les nombres hexadécimaux précédés de 0x ou 0X. Ceci ne fonctionne que pour a-fA-F0-9.

Identifiant du drapeau	Utilisé avec	Description
FILTER_FLAG_EMAIL_UNICODE	FILTER_VALIDATE_EMAIL	Permet à la partie locale de l'adresse électronique de contenir des caractères Unicode.
FILTER_FLAG_IPV4	FILTER_VALIDATE_IP	Autorise une adresse IP au format IPv4.
FILTER_FLAG_IPV6	FILTER_VALIDATE_IP	Autorise une adresse IP au format IPv6.
FILTER_FLAG_NO_PRIV_RANGE	FILTER_VALIDATE_IP	Échoue la validation pour les intervalles privés IPv4: 10.0.0.0/8, 172.16.0.0/12 et 192.168.0.0/16. Échoue la validation pour les adresses IPv6 commençant par FD ou FC.
FILTER_FLAG_NO_RES_RANGE	FILTER_VALIDATE_IP	Echoue la validation pour les intervalles IPv4 réservés : 0.0.0.0/8, 169.254.0.0/16, 127.0.0.0/8 et 240.0.0.0/4 et pour les intervalles IPv6 réservés ::1/128, ::/128, ::ffff:0:0/96 et fe80::/10.
FILTER_FLAG_SCHEME_REQUIRED	FILTER_VALIDATE_URL	Requière de l'URL qu'elle contienne une partie schéma.
FILTER_FLAG_HOST_REQUIRED	FILTER_VALIDATE_URL	Requière de l'URL qu'elle contienne une partie hôte.
FILTER_FLAG_PATH_REQUIRED	FILTER_VALIDATE_URL	Oblige URL à contenir un chemin.
FILTER_FLAG_QUERY_REQUIRED	FILTER_VALIDATE_URL	Oblige URL à contenir une chaine de requête.
FILTER_REQUIRE_SCALAR		Oblige la valeur à être un scalaire.
FILTER_REQUIRE_ARRAY		Oblige la valeur à être un tableau.
FILTER_FORCE_ARRAY		Si la valeur est un scalaire, elle sera traitée comme un tableau

Identifiant du drapeau	Utilisé avec	Description
		avec les valeurs scalaires comme seul élément.

Illustrons le fonctionnement des options et des drapeaux avec quelques filtres de validation. Notez bien ici qu'on va pouvoir de la même façon utiliser des options et des drapeaux avec des filtres de nettoyage.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = 'Pierre';
            $ipv4 = '127.0.0.1';
            $x = 5;
            $y = 50;
            $options = ['options' => ['min_range' => 0, 'max_range' => 10]];

            echo $prenom. ' booléen ? : ';
            var_dump(filter_var($prenom, 258)); // 258 = FILTER_VALIDATE_BOOLEAN
            echo '<br>' .$prenom. ' booléen ? : ';
            var_dump(filter_var($prenom, 258, FILTER_NULL_ON_FAILURE));

            echo '<br><br>' .$ipv4. ' a la forme d\'une IP ? : ';
            var_dump(filter_var($ipv4, 275)); //275 = FILTER_VALIDATE_IP
            echo '<br>' .$ipv4. ' a la forme d\'une IPv4 ? : ';
            var_dump(filter_var($ipv4, 275, FILTER_FLAG_IPV4));
            echo '<br>' .$ipv4. ' a la forme d\'une IPv6 ? : ';
            var_dump(filter_var($ipv4, 275, FILTER_FLAG_IPV6));

            echo '<br><br>' .$x. ' entier compris entre 0 et 10 ? : ';
            var_dump(filter_var($x, 257, $options)); //257 = FILTER_VALIDATE_INT
            echo '<br>' .$y. ' entier compris entre 0 et 10 ? : ';
            var_dump(filter_var($y, 257, $options));
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



On passe cette fois-ci des id de filtre numérotés à `filter_var()` pour raccourcir notre écriture (j'ai mis en commentaire dans le code les équivalents en termes d'id nommé pour chaque filtre).

On commence par filtrer la valeur « Pierre » avec le filtre `boolean`. Notre fonction `filter_var()` renvoie `false` dans le premier exemple car le filtre `boolean`, utilisé sans drapeau, renvoie `false` pour toute valeur différente de « 1 », « true », « on » et « yes ».

On utilise ensuite à nouveau le filtre `boolean` pour filtrer la valeur de `$prenom` mais avec cette fois-ci le drapeau `FILTER_NULL_ON_FAILURE`. Ce drapeau va faire que le filtre `boolean` renverra `NULL` pour toute valeur non booléenne.

On tente ensuite de valider une adresse IP Notre variable `$ipv4` stocke ici une valeur qui a la forme d'un IPv4 (IP version 4) mais pas la forme d'une IPv6.

On commence par utiliser le filtre `validate_ip` sans drapeau. Le filtre va donc filtrer n'importe quelle IP. Ensuite, on retente de filtrer en utilisant le drapeau `FILTER_FLAG_IPV4` qui permet de filtrer des IPv4 en particulier. Finalement, on utilise le filtre `FILTER_FLAG_IPV6` qui permet de ne filtrer que des IPv6 et la filtre échoue donc.

On tente ensuite de valider deux entiers stockés dans des variables `$x` et `$y` et dans le cas présent on va également valider le fait que les entiers stockés se situent dans un certain intervalle.

Pour faire cela, on va utiliser un tableau d'options. Le tableau d'options utilisé est un tableau multidimensionnel nommé `options` et qui va dans notre cas contenir un tableau associatif stockant les éléments `min_range => 0` et `max_range => 10`.

Les clefs de tableau `min_range` et `max_range` sont des mots clefs qui possèdent déjà un sens en PHP : ils vont servir à définir des bornes d'intervalle.

Ici, on va donc valider le fait que nos deux variables contiennent des entiers et que ces entiers se situent dans l'intervalle de valeurs [0-10].

Cas concret d'utilisation des filtres

Dans cette leçon, nous allons voir les cas les plus courants d'utilisation des filtres en PHP en situation « réelle ». Nous allons donc construire un formulaire HTML et allons utiliser les filtres pour filtrer les données suivantes :

- Utiliser les filtres PHP pour valider une adresse IP ;
- Utiliser les filtres PHP pour nettoyer et valider une adresse mail ;
- Utiliser les filtres PHP pour nettoyer et valider une URL.

Nous discuterons également des limites des filtres et de ce que les filtres ne peuvent ou ne doivent pas faire.

Valider une adresse IP

Commençons par l'exemple le plus facile qui consiste en une simple validation d'adresse IP. Pour valider une adresse IP, on va utiliser le filtre `validate_ip`.

Nous allons ici vouloir valider des données externes pour rendre l'exemple plus réel et allons donc créer un formulaire HTML très simple qui va contenir un champ de texte demandant une adresse IP.

On va ensuite utiliser la fonction `filter_input()` dont le rôle est de récupérer une variable externe pour la filtrer.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <form method='post' action='cours.php'>
            <label for='ip'>Entrez une IP ici :</label>
            <input type='text' id='ip' name='ip'>

            <input type='submit' value='Envoyer'>
        </form>
        <?php
            //On vérifie déjà qu'une donnée ait bien été envoyée
            if(isset($_POST['ip'])){
                if(filter_input(INPUT_POST, 'ip', FILTER_VALIDATE_IP)){
                    echo $_POST['ip']. ' ressemble à une IP valide <br>';
                }else{
                    echo $_POST['ip']. ' ne ressemble pas à une IP valide <br>';
                }
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, les données de notre formulaire sont renvoyées vers la page courante avec `action='cours.php'` qui est la page qui abrite le formulaire. On utilise la méthode `post` pour envoyer les données qui vont donc être stockées dans la superglobale `$_POST`.

Dans notre script PHP, on commence déjà par s'assurer que notre superglobale `$_POST` contient bien un élément `ip`. C'est le rôle de la première

instruction `if` englobante. Si ce n'est pas le cas, on ne rentre pas dans le `if` et rien ne sera affiché.

Si notre variable `$_POST['ip']` a bien été définie, on va alors tester que la valeur passée a bien la forme d'une adresse IP. Notez que les filtres de validation ne servent bien sûr qu'à vérifier la concordance de forme et non pas dans le cas présent si l'IP existe vraiment et a été attribuée à quelqu'un.

Lorsqu'on utilise la fonction `filter_input()`, il faut préciser le nom simple de la donnée qu'on souhaite filtrer plutôt que l'emplacement de la valeur dans la superglobale. Cela est dû au fait que le premier argument de la fonction permet déjà de préciser d'où vient la donnée.

Pour une validation d'IP plus stricte, on aurait pu également utiliser les drapeaux `FILTER_FLAG_IPV4` et `FILTER_FLAG_IPV6` qui permettent de valider respectivement une IPv4 et une IPv6.

Nettoyer et valider une adresse mail

Pour vérifier la validité d'une adresse email envoyée par un utilisateur en utilisant les filtres PHP, nous allons pouvoir procéder en deux étapes en commençant par nettoyer les données envoyées en supprimant tous les caractères « illégaux » envoyés (une espace, une virgule, etc.) puis en vérifiant que les données restantes ont bien la forme d'une adresse email.

Une nouvelle fois, nous n'allons pas pouvoir nous assurer que l'adresse email envoyée existe bien de cette manière, nous allons simplement pouvoir vérifier que les données envoyées ont la forme d'une adresse email (présence d'un symbole « @ » par exemple). Pour illustrer cela, nous allons cette fois-ci créer un champ d'`input` de type email dans notre formulaire.

Notez que la plupart des navigateurs effectuent déjà une vérification sur les données passées dans les `input type='email'` et vérifient notamment la présence du caractère « @ » suivi par au moins une lettre.

Les filtres vont nous permettre d'aller plus loin. Si vous souhaitez pouvoir inscrire n'importe quoi dans le champ mail du formulaire pour bien vérifier votre filtre, vous pouvez créer un champ de type `text` plutôt que `email`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <form method='post' action='cours.php'>
            <label for='mail'>Entrez une adresse mail ici :</label>
            <input type='email' id='mail' name='mail'><br>

            <input type='submit' value='Envoyer'>
        </form>
        <?php
            //On vérifie déjà qu'une donnée ait bien été envoyée
            if(isset($_POST['mail'])){
                $m = filter_input(INPUT_POST, 'mail', FILTER_SANITIZE_EMAIL);
                echo 'Valeur retenue : ' .$m.
                     '<br> Valeur originale : ' .$_POST['mail']. '<br>';
                if(filter_var($m, FILTER_VALIDATE_EMAIL)){
                    echo $m. ' ressemble à une adresse mail valide <br>';
                }else{
                    echo $m. ' ne ressemble pas à une adresse mail valide <br>';
                }
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

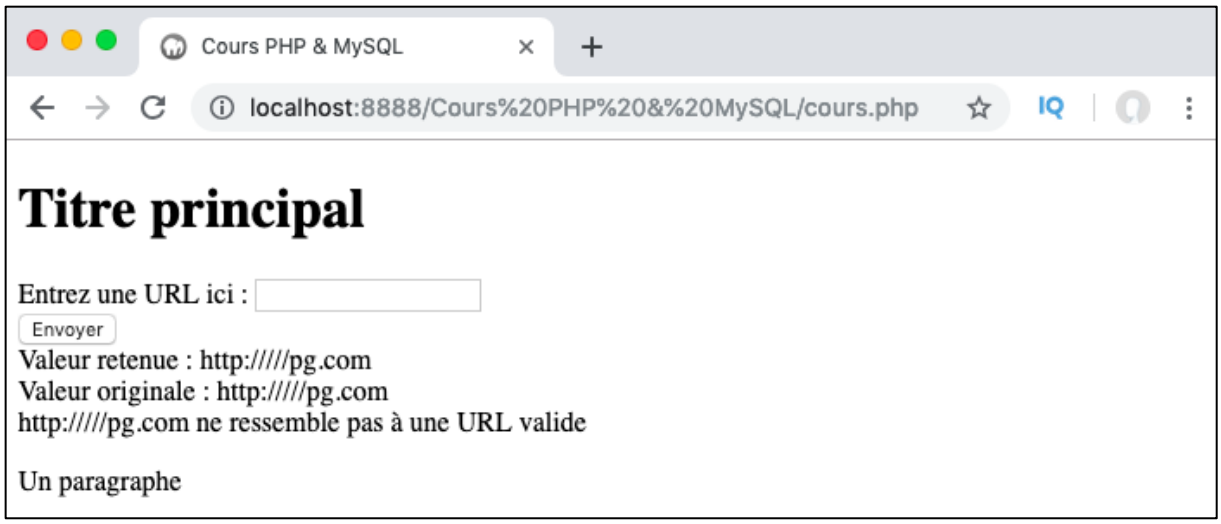
Ici, on commence par utiliser un premier filtre de nettoyage `FILTER_SANITIZE_EMAIL` puis on récupère le résultat (la valeur filtrée) dans une variable `$m`.

On utilise ensuite un filtre de validation pour nous assurer que l'adresse contenue dans `$m` a bien la forme d'une adresse mail valide. On utilise ici `filter_var()` plutôt que `filter_input()` car notre variable `$m` a été définie en interne.

Nettoyer et valider une URL

On va pouvoir de la même façon qu'avec nos adresses mail pouvoir filtrer des URL. Pour illustrer cela, on peut reprendre le code précédent et ajuster les filtres et les noms des variables :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <form method='post' action='cours.php'>
            <label for='url'>Entrez une URL ici :</label>
            <input type='url' id='url' name='url'><br>

            <input type='submit' value='Envoyer'>
        </form>
        <?php
            //On vérifie déjà qu'une donnée ait bien été envoyée
            if(isset($_POST['url'])){
                $url = filter_input(INPUT_POST, 'url', FILTER_SANITIZE_URL);
                echo 'Valeur retenue : ' .$url.
                     '<br> Valeur originale : ' .$_POST['url']. '<br>';
                if(filter_var($url, FILTER_VALIDATE_URL)){
                    echo $url. ' ressemble à une URL valide <br>';
                }else{
                    echo $url. ' ne ressemble pas à une URL valide <br>';
                }
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Les limites des filtres

Les filtres sont un bon outil pour s'assurer qu'une valeur possède une forme attendue ou pour échapper ou supprimer certains caractères problématiques d'une valeur.

Cependant, les filtres sont loin d'être parfaits et ne vont pas pouvoir être utilisés pour vérifier des données dans n'importe quelle condition.

Typiquement, les filtres vont être performants lorsqu'on souhaite afficher des données traitées dans une page web car la plupart des filtres se concentrent sur l'échappement des balises HTML et de certains caractères pouvant poser des problèmes lors d'un affichage.

En revanche, on ne pourra pas se reposer uniquement sur les filtres pour l'enregistrement de données en base de données par exemple, tout simplement car les filtres vont laisser passer certains caractères et schémas qui vont être potentiellement dangereux.

Définition et gestion des erreurs

Dans cette nouvelle leçon, nous allons définir ce qu'est une erreur en PHP et comprendre comment les erreurs sont générées par le PHP ainsi que comment créer un gestionnaire d'erreurs personnalisé.

Qu'est-ce qu'une erreur en PHP ?

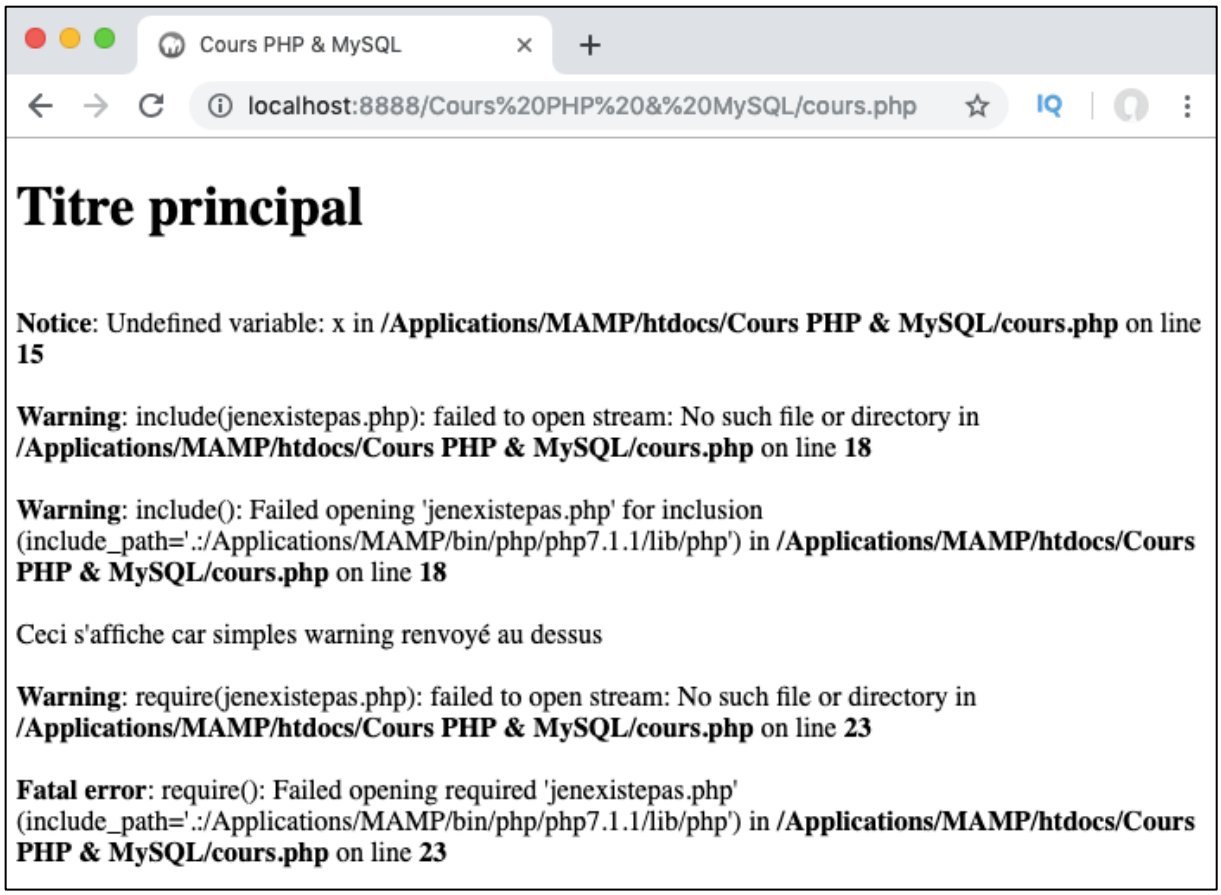
Tout code peut rencontrer une situation anormale qui va l'empêcher de s'exécuter correctement.

Ces « situations anormales » peuvent parfois provenir du code lui-même, c'est-à-dire d'erreurs faites par les développeurs qui l'ont écrit comme par exemple l'utilisation d'une variable non définie ou le passage d'arguments incorrects dans une fonction ou encore l'oubli d'une parenthèse ou d'un point-virgule quelque part dans le code.

Parfois, également, un code ne va pas pouvoir s'exécuter correctement à cause d'un problème externe comme par exemple dans le cas où on souhaite inclure un fichier dans un script mais que le fichier en question ne peut pas être inclus pour le moment pour diverses raisons.

Dans ces cas-là, le PHP va signaler les problèmes rencontrés lors de l'exécution du code en renvoyant ce qu'on appelle une erreur. A chaque fois que HP génère une erreur, il inclut également un type qui prend la forme d'une constante. Selon le problème rencontré, le niveau ou « type » d'erreur renvoyé sera différent et selon le type d'erreur le script pourra continuer à s'exécuter normalement ou au contraire sera stoppé brutalement.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            //La variable $x n'a pas été définie
            echo $x;

            //Le fichier jenexistepas.php n'existe pas
            include 'jenexistepas.php';

            echo '<br>Ceci s\'affiche car simples warning renvoyé au dessus<br>';

            //Le fichier jenexistepas.php n'existe pas
            require 'jenexistepas.php';

            echo 'Ceci ne s\'affichera pas car erreur fatale ci-dessus';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Dans l'exemple ci-dessus, je commence par essayer d'`echo` une variable `$x` non définie. Comme PHP n'arrive pas à trouver la variable, il renvoie une erreur de type `E_NOTICE`.

Ensuite, je tente d'inclure un fichier qui n'existe pas avec `include`. PHP renvoie deux erreurs mais de type `E_WARNING` indiquant que la ressource n'a pas été trouvée et qu'elle n'a pas pu être ouverte (dans certains cas, le fichier sera trouvé mais ne pourra pas être ouvert pour telle ou telle raison).

Finalement, je tente à nouveau d'inclure un fichier qui n'existe pas mais en utilisant cette fois-ci `require` qui est plus stricte que `include`. Cette fois-ci, le PHP va renvoyer une erreur de type `E_WARNING` pour indiquer que le fichier n'a pas été trouvé et une deuxième de type `E_ERROR` (erreur fatale) qui va stopper l'exécution du script.

La gestion des erreurs par le PHP et les types d'erreurs

Il existe 16 niveaux ou types d'erreurs différentes en PHP. Chaque type d'erreur possède une constante et une valeur associées et le script va se comporter différemment en fonction de l'erreur rencontrée.

Ces erreurs peuvent être rangées en différents groupes : les erreurs du code classiques (erreur du développeur), les erreurs liées au code source de PHP, les erreurs générées par le moteur Zend (le moteur ou « framework » sur lequel PHP se base) et les erreurs générées par l'utilisateur.

Vous pouvez trouver ci-dessous la liste des différentes constantes d'erreurs, des valeurs associées et une description de chaque type d'erreur :

Valeur	Constante	Description
1	E_ERROR	Erreur fatale qui va être affichée par défaut. Ce sont des erreurs qui ne peuvent pas être ignorées. L'exécution du script est interrompue.
2	E_WARNING	Alerte qui va être affichée par défaut. Elles indiquent un problème qui doit être intercepté par le script durant l'exécution du script. L'exécution du script n'est pas interrompue.
4	E_PARSE	Les erreurs d'analyse ne doivent être générées que par l'analyseur. Vous n'aurez donc pas à vous en préoccuper.
8	E_NOTICE	Les notices ou « remarques » ne sont pas affichées par défaut, et indiquent que le script a rencontré quelque chose qui peut être une erreur, mais peut aussi être un événement normal dans la vie du script (ex : essayer d'accéder à une valeur non déclarée)
16	E_CORE_ERROR	Comparable à `E_ERROR` mais générée par le code source de PHP. Les fonctions de PHP ne doivent pas générer ce genre d'erreur.
32	E_CORE_WARNING	Comparable à `E_WARNING` mais générée par le code source de PHP. Les fonctions de PHP ne doivent pas générer ce genre d'erreur.
64	E_COMPILE_ERROR	Comparable à `E_ERROR` mais générée par le moteur Zend. Les fonctions de PHP ne doivent pas générer ce genre d'erreur.
128	E_COMPILE_WARNING	Comparable à `E_WARNING` mais générée par le moteur Zend. Les fonctions de PHP ne doivent pas générer ce genre d'erreur.
256	E_USER_ERROR	Message d'erreur généré par l'utilisateur. Comparable à `E_ERROR`. Elle est générée par le programmeur en PHP par l'utilisation de la fonction `trigger_error()`. Les fonctions

Valeur	Constante	Description
		de PHP ne doivent pas générer ce genre d'erreur.
512	E_USER_WARNING	Message d'erreur généré par l'utilisateur. Comparable à E_WARNING. Elle est générée par le programmeur en PHP par l'utilisation de la fonction trigger_error(). Les fonctions de PHP ne doivent pas générer ce genre d'erreur.
1024	E_USER_NOTICE	Message d'erreur généré par l'utilisateur. Comparable à E_NOTICE. Elle est générée par le programmeur en PHP par l'utilisation de la fonction trigger_error(). Les fonctions de PHP ne doivent pas générer ce genre d'erreur.
2048	E_STRICT	Permet d'obtenir des suggestions de PHP pour modifier notre code, assurant ainsi une meilleure interopérabilité et compatibilité de celui-ci
4096	E_RECOVERABLE_ERROR	Erreur fatale qui peut être captée. Ceci indique qu'une erreur probablement dangereuse s'est produite, mais n'a pas laissé le moteur Zend dans un état instable. Si on ne capture pas l'erreur, le script est stoppé comme pour une erreur E_ERROR
8192	E_DEPRECATED	Alertes d'exécution. On va activer cette option pour recevoir des alertes sur les portions de code qui pourraient ne pas fonctionner avec les futures versions
16384	E_USER_DEPRECATED	Message d'alerte généré par l'utilisateur. Fonctionne de la même façon que E_DEPRECATED, mise à part que le message est généré par notre code PHP en utilisant la fonction trigger_error()
32767	E_ALL	Toutes les erreurs et alertes supportées

Si aucun gestionnaire d'erreur n'est défini, PHP gèrera les erreurs qui se produisent en fonction de sa configuration. La directive error_reporting du fichier php.ini (le fichier de configuration principal de PHP) va définir quelles erreurs vont être rapportées et quelles erreurs vont être ignorées.

Notez qu'on va tout à fait pouvoir définir quelles erreurs doivent être rapportées et quelles erreurs doivent être ignorées en modifiant manuellement cette directive. Dans la plupart des cas, il sera préférable de tout rapporter par défaut en utilisant la valeur `E_ALL`.

Ensuite, l'affichage ou le non affichage des erreurs rapportées va être défini par une autre directive du fichier `php.ini` qui est la directive `display_errors`. Si vous ne voyez pas les erreurs comme moi, il vous faudra probablement modifier la valeur de cette directive.

Créer des gestionnaires d'erreurs personnalisés

Parfois, la prise en charge d'une erreur par défaut par le PHP ne va nous convenir. Dans ce cas-là, on voudra créer un gestionnaire personnalisé d'erreurs.

Pour réaliser cela, on va pouvoir utiliser la fonction `set_error_handler()` à laquelle on va passer une fonction de rappel qui sera notre gestionnaire d'erreurs.

Notez qu'on va également pouvoir passer en deuxième argument de `set_error_handler()` une constante d'erreur qui va définir le type d'erreurs pour lequel notre fonction gestionnaire d'erreurs doit être appelée. Si on ne précise rien, le gestionnaire sera appelé pour toutes les erreurs.

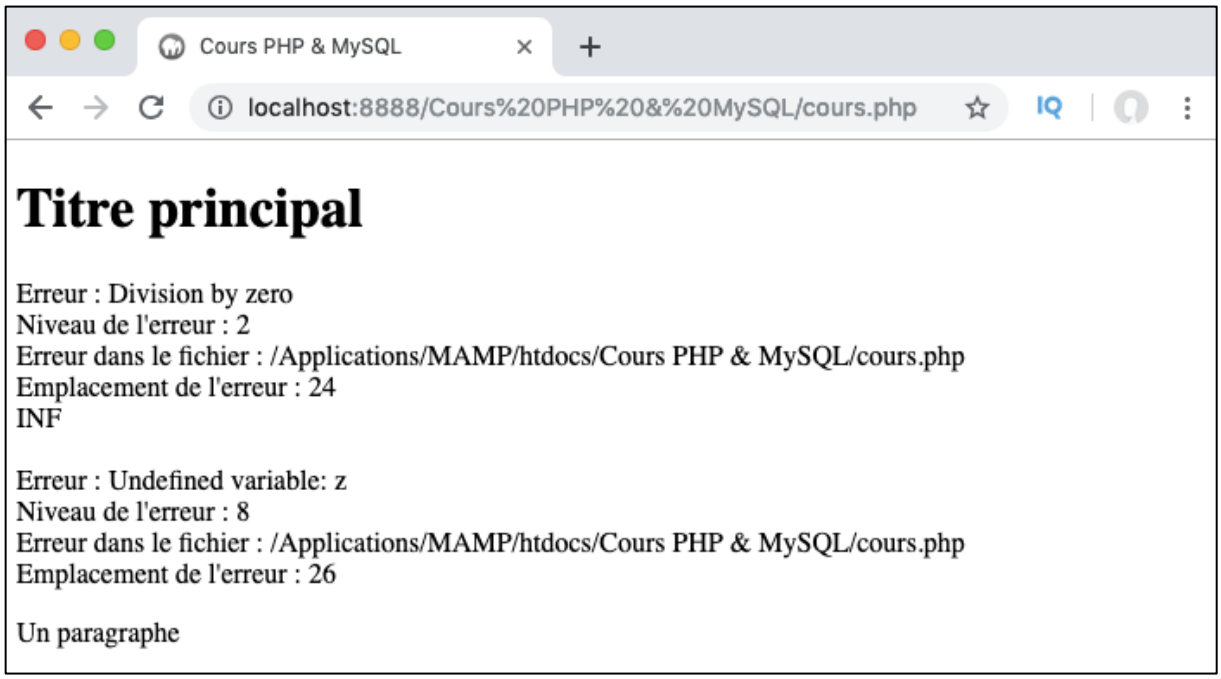
Notez également qu'on ne va pas pouvoir gérer tous les types d'erreurs de cette façon. En particulier, les types d'erreur `E_ERROR`, `E_PARSE`, `E_CORE_ERROR`, `E_CORE_WARNING`, `E_COMPILE_ERROR`, `E_COMPILE_WARNING` ainsi que la plupart des `E_STRICT` du fichier dans lequel `set_error_handler()` est appelée ne pourront pas être gérés de cette façon.

Le but de la fonction de rappel créée va être de récupérer les différentes informations renvoyées par le PHP en cas d'erreur et de les utiliser pour effectuer différentes opérations. Pour cela, on va pouvoir définir entre 2 et 4 paramètres pour notre fonction :

- Le premier paramètre va représenter le niveau d'erreur ;
- Le deuxième paramètre représente le message d'erreur renvoyé par le PHP ;
- Le troisième paramètre (optionnel) représente le nom du fichier dans lequel l'erreur a été identifiée ;
- Le quatrième paramètre (optionnel) représente le numéro de ligne à laquelle l'erreur a été identifiée.

Le PHP va lui-même transmettre les valeurs en arguments de notre fonction en cas d'erreur. Regardez plutôt l'exemple ci-dessous :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $x = 5;
            $y = 0;

            set_error_handler(function($niveau, $message, $fichier, $ligne){
                echo 'Erreur : ' .$message. '<br>';
                echo 'Niveau de l\'erreur : ' .$niveau. '<br>';
                echo 'Erreur dans le fichier : ' .$fichier. '<br>';
                echo 'Emplacement de l\'erreur : ' .$ligne. '<br>';
            });

            echo $x / $y;
            echo '<br><br>';
            echo $z;
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



On utilise ici une fonction anonyme comme fonction de rappel de `set_error_handler()` et donc comme gestionnaire d'erreurs. On définit 4 paramètres pour cette fonction.

Dans le cas présent, la fonction va se contenter d'`echo` les différentes informations liées à l'erreur de façon structurée et elle n'est donc pas très utile en l'état. Cependant, on va également pouvoir définir des fonctions bien plus complexes qui vont effectuer telle ou telle action en fonction des valeurs reçues (par exemple « si l'erreur est de telle niveau alors exécute ceci…).

Deux erreurs sont générées ici : mathématiquement parlant, la division par zéro est une aberration (le résultat va tendre vers l'infini d'où le « INF » de renvoyé) et une première erreur est donc renvoyée. Ensuite, on tente d'`echo` une variable qui n'a pas été définie et une deuxième erreur est donc renvoyée.

Déclenchement, capture et gestion d'exceptions

Dans cette nouvelle leçon, nous allons présenter une nouvelle façon de gérer les erreurs qui va s'avérer souvent meilleure que de simplement créer et passer un gestionnaire d'erreur à la fonction `set_error_handler()`.

On va ainsi apprendre à gérer les erreurs en manipulant des objets qu'on appelle des exceptions et qui vont être crées et « lancés » dès qu'une erreur définie va survenir.

Présentation des exceptions en PHP 5

La version 5 de PHP a introduit une nouvelle façon de gérer la plupart des erreurs en utilisant de qu'on appelle des exceptions. Cette nouvelle façon de procéder se base sur le PHP orienté objet et sur la classe `Exception`.

L'idée ici va être de créer ou de « lancer » un nouvel objet `Exception` lorsqu'une erreur spécifique est détectée. Dès qu'une exception est lancée, le script va suspendre son exécution et le PHP va chercher un endroit dans le script où l'exception va être « attrapée ».

Utiliser des exceptions va nous permettre de gérer les erreurs de manière plus fluide et de personnaliser la façon dont un script doit gérer certaines erreurs.

Notez que quasiment tous les langages serveurs utilisent le concept d'exceptions pour prendre en charge les erreurs car c'est la meilleure façon de procéder à ce jour.

Lancer et attraper une exception

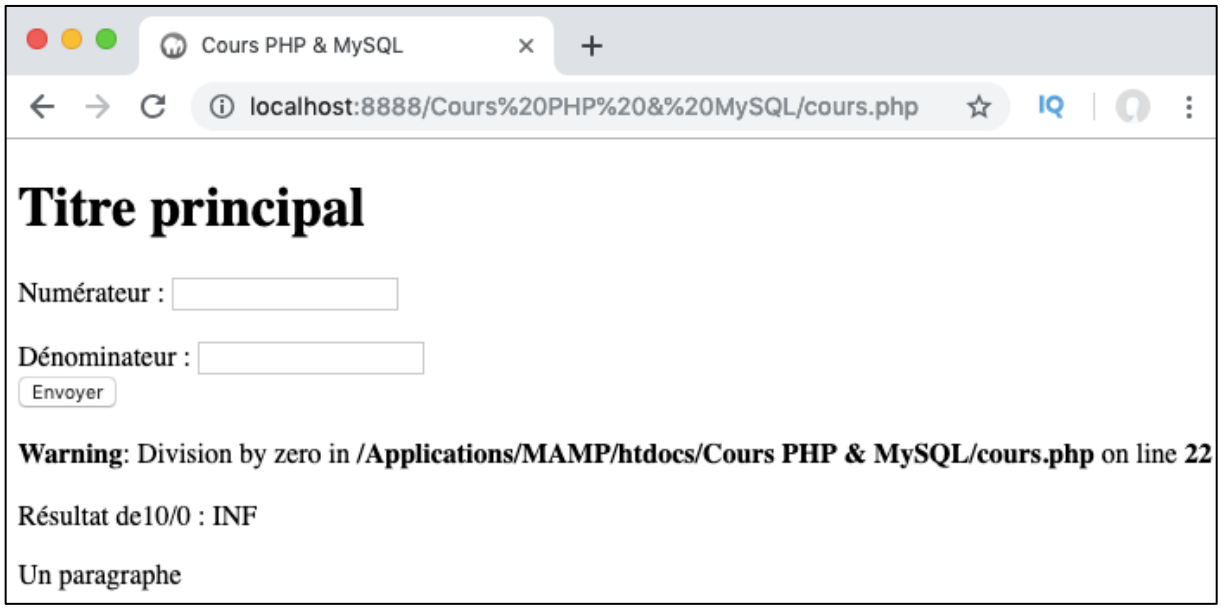
En pratique, la gestion d'une erreur via une exception va se faire en trois temps :

1. On va définir quand une exception doit être lancée avec une instruction `throw` ;
2. On va créer un bloc `catch` dont le but va être d'attraper l'exception si celle-ci a été lancée et de définir la façon dont doit être gérée l'erreur ;
3. On va utiliser un bloc `try` dans lequel le code qui peut potentiellement retourner une erreur va être exécuté.

Illustrons immédiatement tout cela avec un exemple concret. Pour cela, imaginons que l'on définisse une fonction qui divise deux nombres non connus à l'avance entre eux. Comme vous le savez, on ne peut pas diviser un nombre par zéro. Si on tente de le faire, une erreur sera renvoyée par le PHP. Nous allons justement lancer une exception pour gérer ce cas particulier.

Commençons déjà par créer notre script sans aucune prise en charge des erreurs :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <form action='cours.php' method='post'>
            <label for='n1'>Numérateur :</label>
            <input type='number' id='n1' name='n1'><br><br>
            <label for='n2'>Dénominateur :</label>
            <input type='number' id='n2' name='n2'><br>
            <input type='submit' value='Envoyer'>
        </form>
        <?php
            function division($x, $y){
                echo '<br>Résultat de' .$x. '/' .$y. ' : ' .($x / $y);
            }

            //En pratique, il faudrait vérifier les données envoyées
            if(isset($_POST['n1']) && isset($_POST['n2'])){
                division($_POST['n1'],$_POST['n2']);
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Ici, on définit une fonction `division()` dont le rôle est de diviser un nombre par un autre et de renvoyer le résultat. Les nombres seront passés par nos utilisateurs via un formulaire.

Évidemment, en pratique, il faudra vérifier les données envoyées avant de les manipuler mais ce n'est pas l'objet de la leçon.

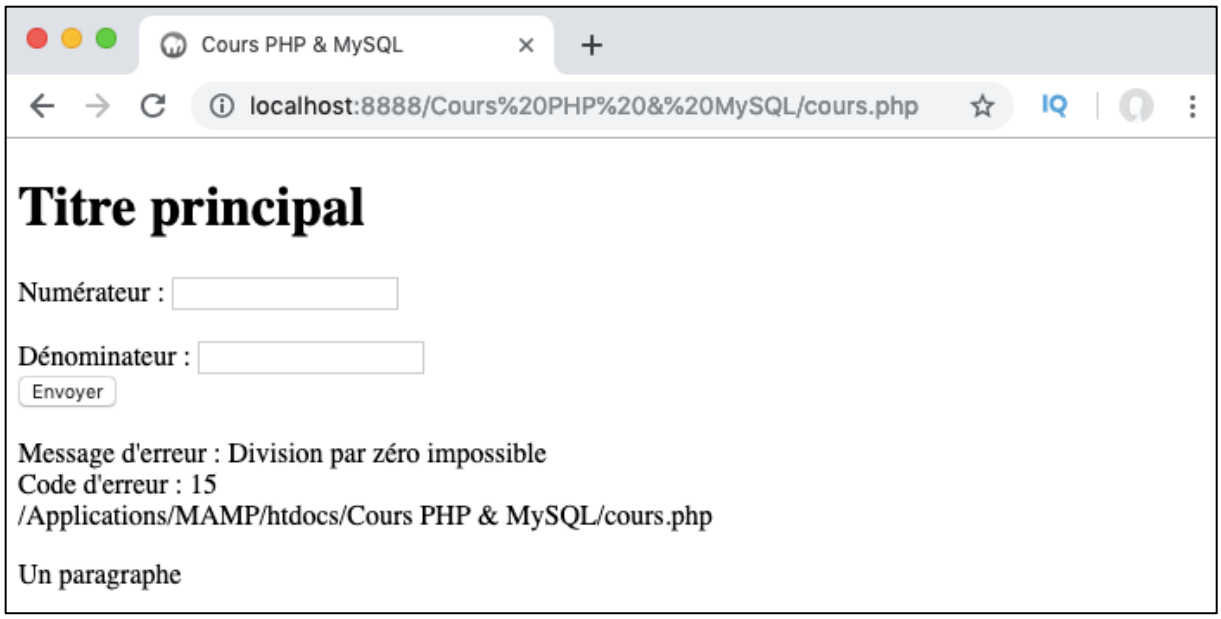
Ici, on se contente simplement de s'assurer que des données ont bien été envoyées et si c'est le cas on exécute `division()`. Évidemment, si on renseigne 0 comme dénominateur, une erreur est renvoyée puisqu'on n'a pas le droit de diviser par zéro.

Pour gérer cette erreur, on pourrait tout à fait utiliser un gestionnaire d'erreurs et la fonction `set_error_handler()` comme ceci :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <form action='cours.php' method='post'>
            <label for='n1'>Numérateur :</label>
            <input type='number' id='n1' name='n1'><br><br>
            <label for='n2'>Dénominateur :</label>
            <input type='number' id='n2' name='n2'><br>
            <input type='submit' value='Envoyer'><br><br>
        </form>
        <?php
            function division($x, $y){
                echo '<br>Résultat de' .$x. '/' .$y. ' : ' .($x / $y);
            }

            set_error_handler(function($niveau, $message, $fichier, $ligne){
                echo 'Erreur : ' .$message. '<br>';
                echo 'Niveau de l\'erreur : ' .$niveau. '<br>';
                echo 'Erreur dans le fichier : ' .$fichier. '<br>';
                echo 'Emplacement de l\'erreur : ' .$ligne. '<br>';
            });

            //En pratique, il faudrait vérifier les données envoyées
            if(isset($_POST['n1']) && isset($_POST['n2'])){
                division($_POST['n1'],$_POST['n2']);
            }

        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Cela fonctionne bien ici. Cependant, vous devez savoir que dans un script plus complexe, il va être difficile de s'assurer que la fonction `set_error_handler()` soit appelée avec les bonnes valeurs et au bon moment. Pour ces raisons, on préfèrera souvent utiliser les exceptions pour gérer les erreurs. Regardez à nouveau le code :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <form action='cours.php' method='post'>
            <label for='n1'>Numérateur :</label>
            <input type='number' id='n1' name='n1'><br><br>
            <label for='n2'>Dénominateur :</label>
            <input type='number' id='n2' name='n2'><br>
            <input type='submit' value='Envoyer'><br><br>
        </form>
        <?php
            function division($x, $y){
                if($y == 0){
                    throw new Exception('Division par zéro impossible', 15);
                }
                else{
                    echo '<br>Résultat de' .$x. '/' .$y. ' : ' .($x / $y);
                }
            }

            //En pratique, il faudrait vérifier les données envoyées
            if(isset($_POST['n1']) && isset($_POST['n2'])){
                try{
                    division($_POST['n1'],$_POST['n2']);
                }
                catch(Exception $e){
                    echo 'Message d\'erreur : ' .$e->getMessage();
                    echo '<br>';
                    echo 'Code d\'erreur : ' .$e->getCode();
                    echo '<br>';
                    echo $e->getFile();
                }
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



Dans l'exemple ci-dessus, on utilise cette fois-ci les exceptions pour prendre en charge les erreurs. L'idée derrière les exceptions va être d'anticiper les situations problématiques (situations qui vont pouvoir causer une erreur) et de lancer une exception si la situation est rencontrée.

C'est ce qu'on fait ici dans le code de notre fonction `division()` : on a identifié que le PHP renverrait une erreur dans le cas où un utilisateur tenterait une division par zéro. Dans ce cas là, on va lancer une exception. Pour cela, on utilise la syntaxe `throw new Exception` qui lance un objet de la classe `Exception`.

La classe `Exception` possède un constructeur qui va pouvoir accepter un message d'erreur et un code d'erreur personnalisé de notre choix. Ici, je passe le message « Division par zéro impossible » et je choisis le code « 15 ».

Ce code va être très utile dans le cas d'un « vrai » site si on souhaite effectuer un suivi et un rapport des erreurs survenues puisqu'on va ainsi pouvoir identifier très rapidement et précisément les erreurs.

Dès qu'une exception est lancée, il va falloir l'attraper. On va faire cela dans un deuxième bloc `catch`. Le but de ce bloc va déjà être de capturer une exception si une exception a été lancée. Ici, si c'est le cas, on place les informations liées à l'exception dans `$e` (le nom `$e` est choisi arbitrairement, vous pouvez choisir le nom de votre choix).

Au sein du bloc `catch`, nous allons préciser les actions à mener en cas d'exception. Ici, on se contente de renvoyer notre message d'erreur, notre code d'erreur et le fichier dans lequel l'exception a été lancée.

Pour comprendre ce code, vous devez savoir que `$e` est un objet de la classe `Exception` et que la classe `Exception` possède des méthodes dont notamment :

- `getMessage()` qui va renvoyer le message d'erreur défini lors du lancement de l'exception ;
- `getCode()` qui va renvoyer le code d'erreur défini lors du lancement de l'exception ;

- `getFile()` qui va renvoyer le chemin du fichier depuis lequel l'exception a été lancée ;
- `getLine()` qui va renvoyer la ligne du fichier depuis lequel l'exception a été lancée.

Finalement, on va exécuter le code potentiellement problématique dans un bloc `try` (bloc « d'essai »). Si aucune erreur n'est rencontrée, alors aucune exception ne sera lancée et le code s'exécutera normalement. Si une erreur est rencontrée, alors une exception sera lancée et attrapée dans le bloc `catch` dans lequel on va décider de la marche à suivre.

Les exceptions et les erreurs en PHP 7

L'une des grandes limites du PHP 5 est qu'il était quasiment impossible de gérer des erreurs fatales. En effet, une erreur fatale n'appelait pas le gestionnaire d'erreurs défini dans `set_error_handler()` et il était également impossible de lancer une exception en cas d'erreur fatale.

Ainsi, en PHP 5, dès qu'une erreur fatale était rencontrée, le script s'arrêtait brutalement. Certains développeurs disent qu'il s'agit là d'un comportement normal puisqu'une erreur fatale ne devrait pas pouvoir être gérée d'une autre façon / ignorée. Ce point de vue est parfois vrai mais pas toujours justifié en fonction de l'erreur fatale.

La version 7 du PHP modifié cela et nous offre désormais un moyen de gérer certaines erreurs fatales via le lancement d'exceptions. La difficulté ici est que pour des raisons de rétrocompatibilité du code (cohérence entre différentes versions de PHP qui coexistent), les exceptions lancées par les erreurs fatales ne vont pas être des instances de la classe `Exception` mais d'une nouvelle classe créée en PHP 7 pour cela : la classe `Error` qui est très similaire dans sa construction à la classe `Exception`.

Note : A ce point du cours, vous n'êtes plus tout à fait débutants et il y a donc une notion que vous devriez intégrer et qui va grandement faciliter la compréhension des différents langages de programmation : les langages de programmation ne sont JAMAIS parfaits.

Certains langages prennent des directions de développement puis reviennent sur leur pas ou intègrent de nouveaux composants au fil du temps. Cela fait que chaque langage dispose d'un héritage de composants désuets et que parfois il est difficile de maintenir un ensemble qui fasse du sens.

C'est le cas du PHP qui est un langage très performant mais cependant également très loin d'être parfait d'un point de vue sémantique et qui possède encore certains comportements qui ne font tout simplement aucun sens. Vous devez absolument garder cela en tête et essayer d'utiliser les langages au mieux tout en gardant à l'esprit que la perfection n'existe pas !

Retour à nos erreurs. Une nouvelle classe a donc été créée en PHP 7 pour utiliser les exceptions avec certains types d'erreurs. Pour apporter un peu de cohérence dans la gestion des erreurs et tenter une unification, une interface a également été créée : l'interface `Throwable`.

Cette interface est en PHP 7 l'interface de base pour tout objet qui peut être jeté ou lancé grâce à la déclaration `throw` et va être implémentée à la fois par la classe `Error` et par la classe `Exception`.

On va ainsi pouvoir utiliser l'interface `Throwable` pour attraper à la fois des exceptions issues de la classe `Exception` et des exceptions issues de la classe `Error`.

Cependant, comme `Throwable` est une interface, on ne va pas pouvoir l'instancier directement.

En PHP 7, lorsqu'une erreur fatale et récupérable (erreur fatale de type « recoverable ») survient, une exception de la classe `Error` est automatiquement lancée. On va ainsi pouvoir l'attraper dans un bloc `catch` comme nos exceptions « classiques » à part que cette fois-ci le bloc `catch` devra utiliser la syntaxe `catch(Error $e){}` ou éventuellement plus généralement `catch(Throwable $t){}` qui va permettre d'attraper les exceptions issues de la classe `Error` et celles de la classe `Exception`.

Notez qu'il est généralement considéré comme une bonne pratique d'utiliser les classes les plus précises pour attraper les exceptions et les gérer de la meilleure façon.

Regardez l'exemple ci-dessous pour bien comprendre :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            try{
                bonjour();
            }
            catch(Error $e){
               echo $e->getMessage();
            }

            echo '<br><br>';

            try{
                if(!function_exists('test')){
                    throw new Error('La fonction n\'est pas définie');
                }
            }
            catch(Error $e){
               echo $e->getMessage();
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Ici, on commence par essayer d'appeler une fonction `bonjour()` qui n'est pas définie. Vous devez savoir que lorsqu'on tente d'exécuter une fonction non définie, le PHP lève une erreur fatale. En PHP 7, une exception de la classe `Error` est également lancée. On capture cette erreur dans le bloc `catch` juste en dessous et on affiche les informations relatives à l'erreur avec la méthode `getMessage()` de la classe `Error`.

On va ensuite gérer une autre erreur du même type en lançant nous-même l'exception cette fois-ci pour personnaliser le message par exemple. Pour cela, on utilise la fonction `function_exists()` qui vérifie si une fonction a été définie et renvoie `true` si c'est le cas ou `false` dans le cas contraire.

Ici, on demande à `function_exists()` de vérifier si une fonction `test()` a été définie. Ce n'est pas le cas ; notre fonction va donc renvoyer `false` et on va lancer une exception issue de la classe `Error` en précisant un message personnalisé.

Ici, notez bien qu'on rentre dans le `if` lorsque `function_exists()` renvoie `false` puisqu'on a utilisé l'opérateur `!` pour inverser la valeur logique de notre test.

Ensuite, on utilise un bloc `catch` pour capturer l'exception et renvoyer le message personnalisé.

Ici, nous ne faisons qu'`echo` les informations relatives à l'erreur, ce qui n'est pas très utile en soi. Cependant, vous allez pouvoir définir toute sorte de code dans votre bloc `catch` pour dicter le comportement du script en cas d'exception.

Le bloc finally

Depuis PHP 5.5, on va pouvoir spécifier un bloc `finally` à la suite des blocs `catch`.

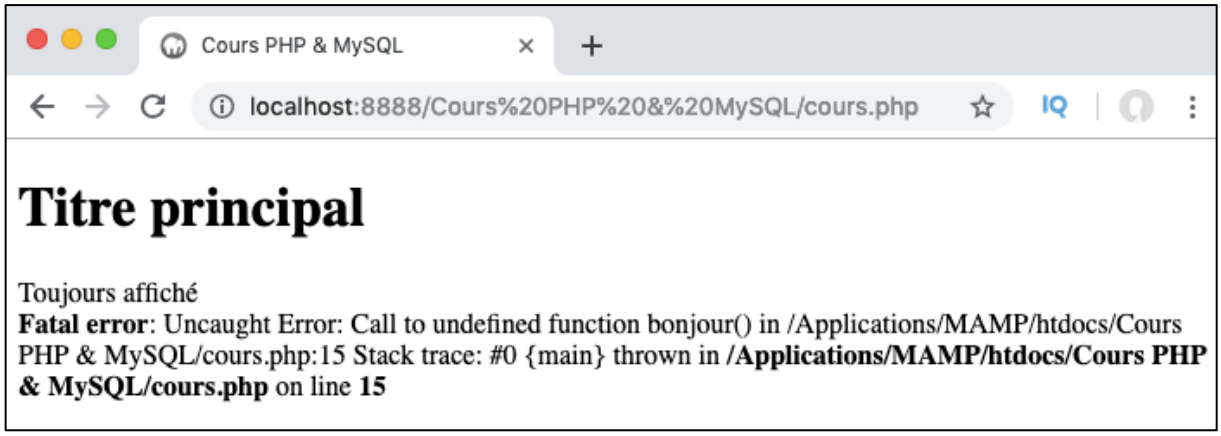
L'intérêt principal de ce bloc est que le code contenu dans `finally` sera toujours exécuté à la suite des blocs `try` et `catch` et avant la reprise de l'exécution normale du script, et ceci qu'une exception ait été lancée ou pas.

On va donc vouloir utiliser ce bloc lorsqu'on souhaite absolument qu'un code s'exécute quelle que soit la situation, comme par exemple dans le cas où on voudrait fermer une connexion à une base de données.

La grande différence entre un code dans un bloc finally et un code dans l'espace global du script (en dehors des blocs cités précédemment) est qu'un code suivant les blocs try et finally et en dehors de ceux-ci ne sera pas exécuté dans le cas où une exception a été lancée mais pas attrapée (car dans ce cas une erreur fatale sera levée).

Regardez plutôt l'exemple ci-dessous :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            try{
                bonjour();
            }
            finally{
                echo 'Toujours affiché';
            }
            echo 'Non affiché ici car exception lancée et non capturée';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```



PARTIE XIII

Introduction aux bases de données

Introduction aux bases de données, au SQL et à MySQL

Avec l'utilisation et la manipulation des bases de données, nous entrons dans la partie « complexe » de la programmation.

Il est possible que vous ne compreniez pas immédiatement comment fonctionnent les bases de données en soi ni comment s'articulent les différents langages entre eux.

Ce premier chapitre a pour vocation de définir justement les différents éléments qui vont entrer dans la manipulation de ces bases de données ainsi que l'utilité de celles-ci.

Je vous conseille donc de le lire attentivement et de ne pas vous précipiter dans l'apprentissage de vos premières requêtes SQL car vous aurez besoin de cette compréhension pour comprendre comment créer un « vrai » site.

Qu'est-ce qu'une base de données ?

Une base de données est un conteneur qui va servir à stocker toutes sortes de données : des dates, chiffres, mots, etc. de façon organisée et sans date d'expiration.
De manière pratique, une base de données va être constitué d'un ensemble de fichiers.

Pourquoi utiliser les bases de données ?

Pourquoi créer des bases de données et stocker des données dedans plutôt que simplement utiliser un fichier quelconque ?

Les bases de données possèdent deux grands avantages par rapport aux autres méthodes de stockage :

1. Nous allons pouvoir stocker de très grandes quantités de données ;
2. Nous allons pouvoir récupérer certaines données en particulier simplement et rapidement.

Si vous avez un nombre limité de données à stocker et peu d'opérations à faire, l'usage des bases de données peut ainsi être contestable. En revanche, dans le cas d'un blog ou d'un e-commerce par exemple, vous allez avoir besoin de stocker de nombreuses informations (informations de connexion et de profil de vos visiteurs ou clients, liste de vos produits et de leurs caractéristiques, liste des commandes, etc.) et vous allez avoir besoin de récupérer telle ou telle information très souvent (pour l'affichage d'une page produit, pour la connexion à votre site, etc.). Dans ces cas-là, l'usage d'une base de données est plus que recommandé, il est essentiel.

Qu'est-ce que le SQL ?

SQL est l'abréviation de Structured Query Language ou langage de requêtes structuré en français.

Le SQL est le langage principal utilisé pour accéder aux bases de données et les manipuler. Nous allons utiliser ce langage pour exécuter toutes sortes de requêtes dans une base de données : récupérer des données, les mettre à jour, en insérer de nouvelles ou même créer de nouvelles bases de données.

Le SQL est un langage à part entière : il possède sa propre syntaxe que nous allons découvrir dans les chapitres suivants.

Qu'est-ce que MySQL ?

MySQL est ce qu'on appelle un système de gestion de bases de données. De manière très schématique, c'est un programme qui va nous permettre de manipuler simplement nos bases de données.

En effet, les bases de données sont des systèmes très complexes. Nous utilisons un système de gestion de bases de données pour cacher cette complexité et effectuer simplement les opérations dont nous avons besoin sur nos bases de données.

MySQL est loin d'être le seul système de gestion de bases de données ; il en existe bien d'autres. Parmi les plus connus, on peut ici notamment citer SQL Server, MS Access ou encore Oracle. Chacun de ces systèmes de gestion de bases de données fonctionne de manière similaire (ils permettent d'envoyer des instructions SQL) et propose des fonctionnalités relativement équivalentes.

Dans ce cours, nous utiliserons MySQL pour des raisons simples : il est gratuit, simple d'utilisation, utilise du SQL standard et le PHP supporte son usage.

Nous allons donc pouvoir utiliser MySQL en PHP pour passer des ordres à nos bases de données : MySQL va nous servir à envoyer nos requêtes écrites en SQL standard à nos bases de données.

Dans ce cours, nos bases de données seront ainsi des bases de données MySQL.

Qu'est-ce que phpMyAdmin ?

Nous allons avoir deux moyens d'interagir avec nos bases de données MySQL : soit en envoyant nos requêtes à partir de nos fichiers de code PHP, soit directement via l'interface phpMyAdmin.

phpMyAdmin est un logiciel gratuit code en PHP qui sert à gérer directement nos bases de données MySQL. Dans phpMyAdmin, nous allons par exemple pouvoir directement créer une nouvelle base de données ou envoyer toutes sortes de requêtes SQL à nos bases de données.

phpMyAdmin est un logiciel qui nous permet donc d'accéder directement aux données de nos bases de données et à gérer nos bases de données.

Notez que phpMyAdmin est embarqué et proposé sur tous les serveurs utilisant les bases de données MySQL. Vous allez également pouvoir y accéder en local et ceci que vous utilisiez Wamp, Mamp ou Lamp. Référez-vous à la documentation Wamp, Mamp ou Lamp pour plus d'informations.

Pourquoi utiliser le PHP et MySQL si je peux directement utiliser phpMyAdmin ?

Ces deux façons d'accéder aux données de nos bases de données et de les gérer concernent des utilisations et des situations totalement différentes.

phpMyAdmin est un logiciel formidable pour créer, modifier ou effectuer des opérations directement sur nos bases de données. En d'autres mots, il permet une utilisation manuelle de nos bases de données.

En revanche, nous n'allons pas pouvoir utiliser phpMyAdmin pour récupérer ou mettre à jour dynamiquement nos bases de données. Par dynamiquement, j'entends ici à un moment non connu à l'avance.

Par exemple, lorsqu'un utilisateur s'inscrit sur notre site, nous allons vouloir stocker différentes informations le concernant en base de données, afin de pouvoir s'en resservir par la suite : nom d'utilisateur, mot de passe, etc. Pour cela, nous devrons créer un formulaire d'inscription et utiliser le PHP et MySQL pour traiter et stocker les données envoyées.

Pour faire très simple, vous pouvez retenir que dès qu'un utilisateur entre dans l'équation, nous allons utiliser le PHP et MySQL. Encore une fois, phpMyAdmin ne va nous servir qu'à modifier directement et manuellement nos bases de données.

En résumé : les notions à retenir

- On appelle base de données une collection de données stockées dans des fichiers particuliers ;
- Les bases de données vont nous permettre de stocker de grandes quantités de données sans date d'expiration. Nous allons ensuite pouvoir manipuler ces données ;
- Le langage des bases de données est le SQL. C'est un langage de requêtes qui va nous permettre d'accéder aux bases de données et de les manipuler ;
- Nous n'allons pas pouvoir communiquer directement en SQL avec nos bases de données. Pour se faire, nous devrons utiliser un système de gestion de bases de données comme MySQL ;
- MySQL va nous permettre d'envoyer des requêtes SQL. A la manière des expressions régulières, nous allons pouvoir l'utiliser avec le PHP ;
- Nous allons pouvoir envoyer nos requêtes SQL via MySQL de deux façons : soit dans nos fichiers de code PHP, soit en passant par l'interface phpMyAdmin qi est un logiciel également codé en PHP ;
- Nous allons utiliser phpMyAdmin lorsque nous voudrons effectuer des actions manuelles directes sur nos bases de données. Dès que les appels à la base de

données seront conditionnés par l'utilisateur, il faudra plutôt utiliser nos fichiers de code PHP.

Structure d’une base de données MySQL et découverte de phpMyAdmin

Nous comprenons désormais l’intérêt d’avoir recours aux bases de données et possédons une vue d’ensemble des différents outils et langages que nous allons devoir ou pouvoir mobiliser pour les manipuler.

Cette nouvelle leçon a pour objet de découvrir et de comprendre la structure d’une base de données et d’une table à proprement parler.

Pour bien comprendre la complexité de certaines bases de données, nous allons prendre l’exemple de bases de données d’un WordPress et d’un PrestaShop que nous visualiserons grâce à phpMyAdmin.

Structure d’une base de données

Une base de données est généralement constituée de tables. Une table est une collection cohérente de données. Par exemple, dans le cas d’un site e-commerce, vous aurez certainement une table « Clients », une autre table « Commandes », etc.

On représente habituellement une table sous forme de tableau. Une table va ainsi être constituée de lignes qu’on appelle également entrées et de colonnes.

L’intersection entre une ligne et une colonne est ce qu’on appelle un champ. Un champ est l’équivalent d’une cellule dans un tableau et va contenir une donnée particulière (un ID, le nom d’un utilisateur, un numéro de téléphone, etc.).

Dans une table, chacun des champs d’une même ligne ou entrée va généralement être relatif à un même sujet. Si votre base de données possède une table « Clients » par exemple, la première ligne va regrouper des informations relatives à un client en particulier.

En colonne, nous allons trouver des informations de même type. Une table « Clients » par exemple pourra contenir des colonnes comme « Id du client », « nom du client », « adresse mail », « numéro de téléphone », etc.

Voici par exemple comment pourrait se présenter une table « Clients » d’un site e-commerce :

IdClient	NomClient	Adresse	Ville	CodePostal	Pays	Mail
1	Pierre Giraud	30 avenue des Acacias	Toulon	83000	France	pierre.giraud@edhec.com

2	Victor Durand	50 boulevard Jean Jaurès	Lille	59000	France	victor.durand@gmail.com
3	Julia Palaz	113 avenue de Versailles	Paris	75016	France	ju.palaz@gmail.com
4	Chloé Joly	28 rue Sainte Catherine	Bordeaux	33000	France	cjoly@outlook.fr
5	Florian Buisson	88 allée des sportifs	Lyon	69002	France	florian.b@gmail.com

Dans ce cas-là, notre table « Clients » possède 7 colonnes et 5 entrées.

Chacun de vos tables va généralement posséder une colonne de type « ID » (identifiant) qui va nous permettre par la suite d'établir des correspondances entre tables en identifiant précisément une donnée en particulier.

Par exemple, dans un site e-commerce, nous aurons généralement deux tables « Clients » et « Commandes » dans notre base de données. La table « Clients » va contenir des informations relatives à chaque client tandis que la table « Commandes » va contenir les informations relatives à chaque commande (montant, date, etc.).

Si ces deux tables ont été créées de manière intelligente, elles vont chacune contenir une colonne « ID » (« IdClient » pour la table « Clients » et « IdCommande » pour la table « Commandes »). Ces Ids vont nous permettre d'accéder et de récupérer simplement les informations relatives à un client ou à une commande,

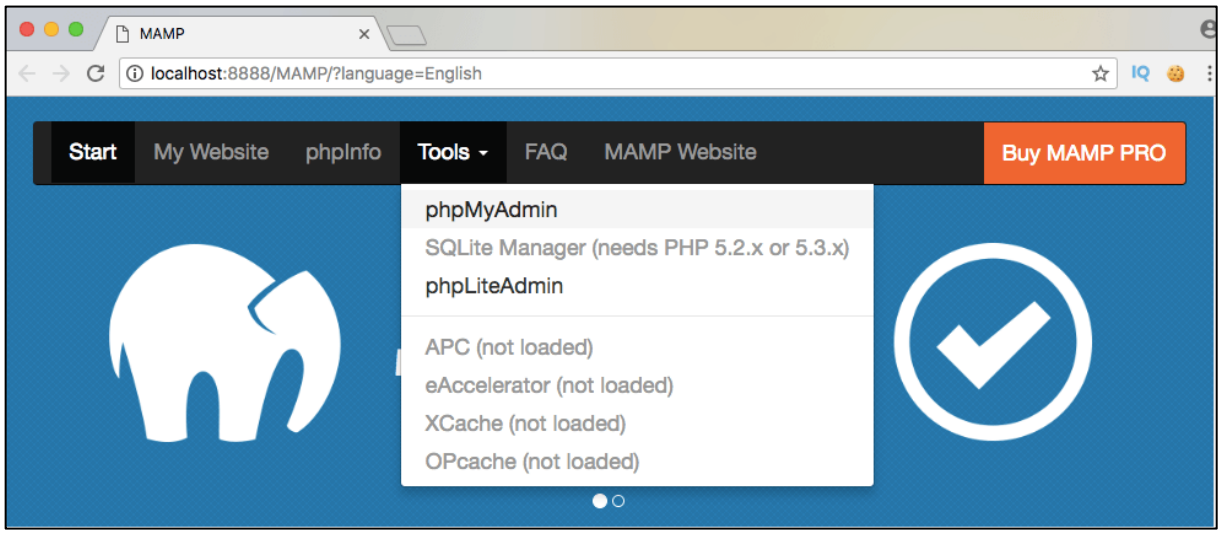
De plus, dans ce cas précis, il va être ici très intéressant d'avoir une même colonne « Id client » dans la table « Clients » et dans la table « Commandes » pour ensuite pouvoir récupérer l'historique des commandes d'un client ainsi que les informations relatives à ce client d'un coup.

Exemple d'une base de données WordPress vue depuis phpMyAdmin

Voyons immédiatement comment se présente une « vraie » base de données en la visualisant avec phpMyAdmin et profitons-en également pour découvrir ce logiciel.

Ici, je vais vous présenter deux bases de données : l'une issue d'une installation WordPress et l'autre issue d'un PrestaShop, tous les deux vierges (sans aucune modification).

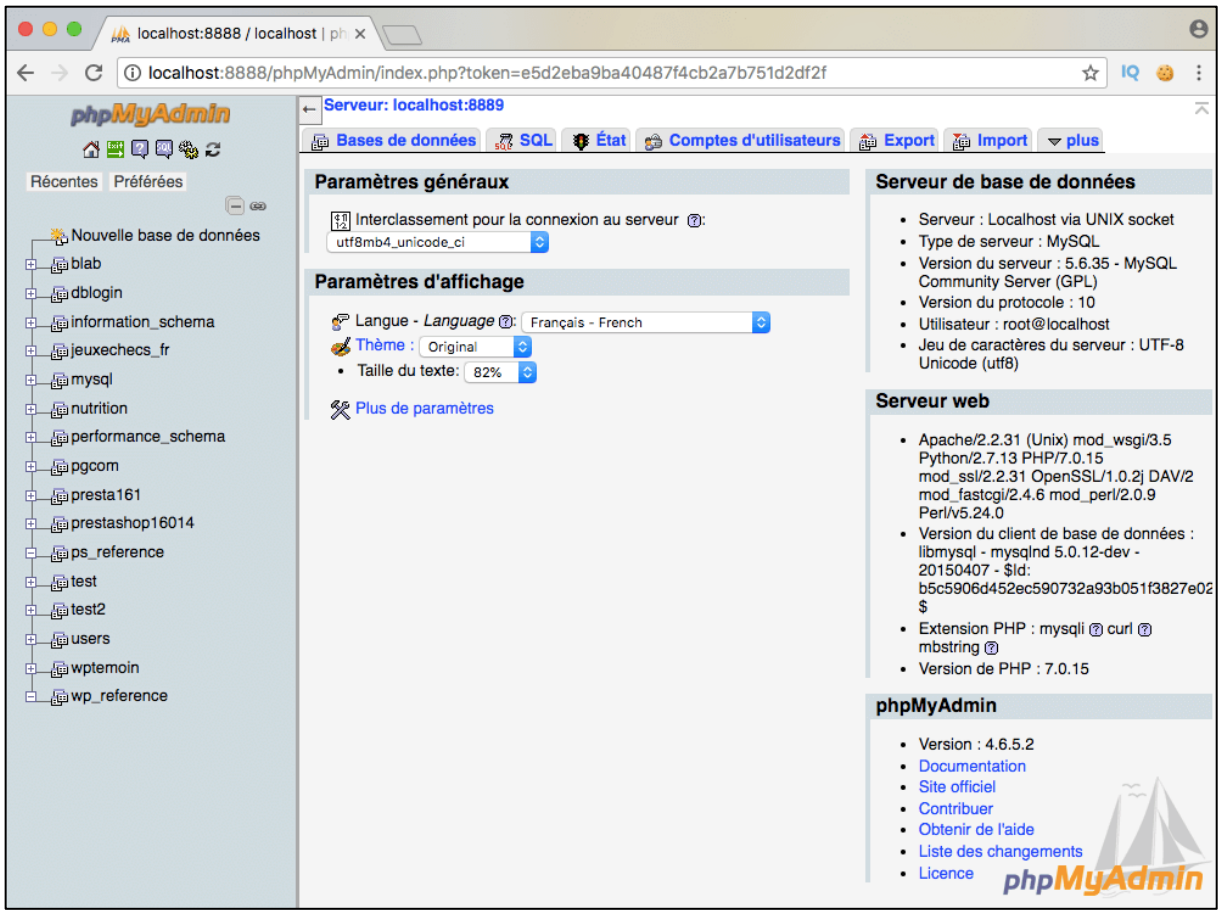
Normalement, que vous utilisiez MAMP ou WAMP, vous devriez pouvoir accéder à phpMyAdmin depuis la page d'accueil du logiciel comme ci-dessous :



Notes :

- Il est possible que vous ne possédiez pas les droits d'accès à phpMyAdmin. Si l'accès n'est pas autorisé, vous devrez modifier votre fichier de configuration.
- Si vous utilisez LAMP, vous devrez au préalable installer phpMyAdmin. Référez vous à la documentation officielle pour effectuer cette opération.

En cliquant sur phpMyAdmin, nous arrivons sur la page d'accueil du logiciel.



En haut de la page, vous avez différents onglets :

- Un onglet « Bases de données » à partir duquel nous allons pouvoir créer de nouvelles bases de données en un clic ;
- Un onglet « SQL » dans lequel nous allons pouvoir exécuter directement des requêtes SQL ;
- Un onglet « Etat » qui nous donne des informations sur l'état / le statut du serveur ainsi que des statistiques sur les requêtes exécutées, etc. ;
- Un onglet « Comptes d'utilisateurs » qui nous permet de gérer et d'ajouter des utilisateurs pour gérer les bases de données ;
- Un onglet « Exporter » qui nous permet d'exporter des bases de données ;
- Un onglet « Importer » qui nous permet d'importer des bases de données ;
- Un onglet « Paramètres » qui va nous permettre de choisir nos préférences d'affichage, de création de bases de données ou de requêtes SQL, les paramètres par défaut lors d'un import ou d'un export, etc. ;
- Un onglet « Réplication » qui va nous permettre de répliquer (dupliquer) une base de données en créant une base « slave » ;
- Un onglet « Variables » qui liste toutes les variables utilisées et utilisables ainsi que leurs valeurs actuelle et globale et qui nous permet de les modifier;
- Un onglet « Jeux de caractères » qui affiche la liste des jeux de caractères (charset) disponibles ;
- Un onglet « Moteurs » qui affiche la liste des moteurs (engines) disponibles sur ce serveur ;
- Un onglet « Greffons » qui correspond en fait à la liste des extensions (plugins) disponibles.

Nous n'allons bien évidemment pas voir ici en détail ce que signifie chaque onglet et comment utiliser telle ou telle fonctionnalité de phpMyAdmin car ce n'est pas le sujet de cette leçon.

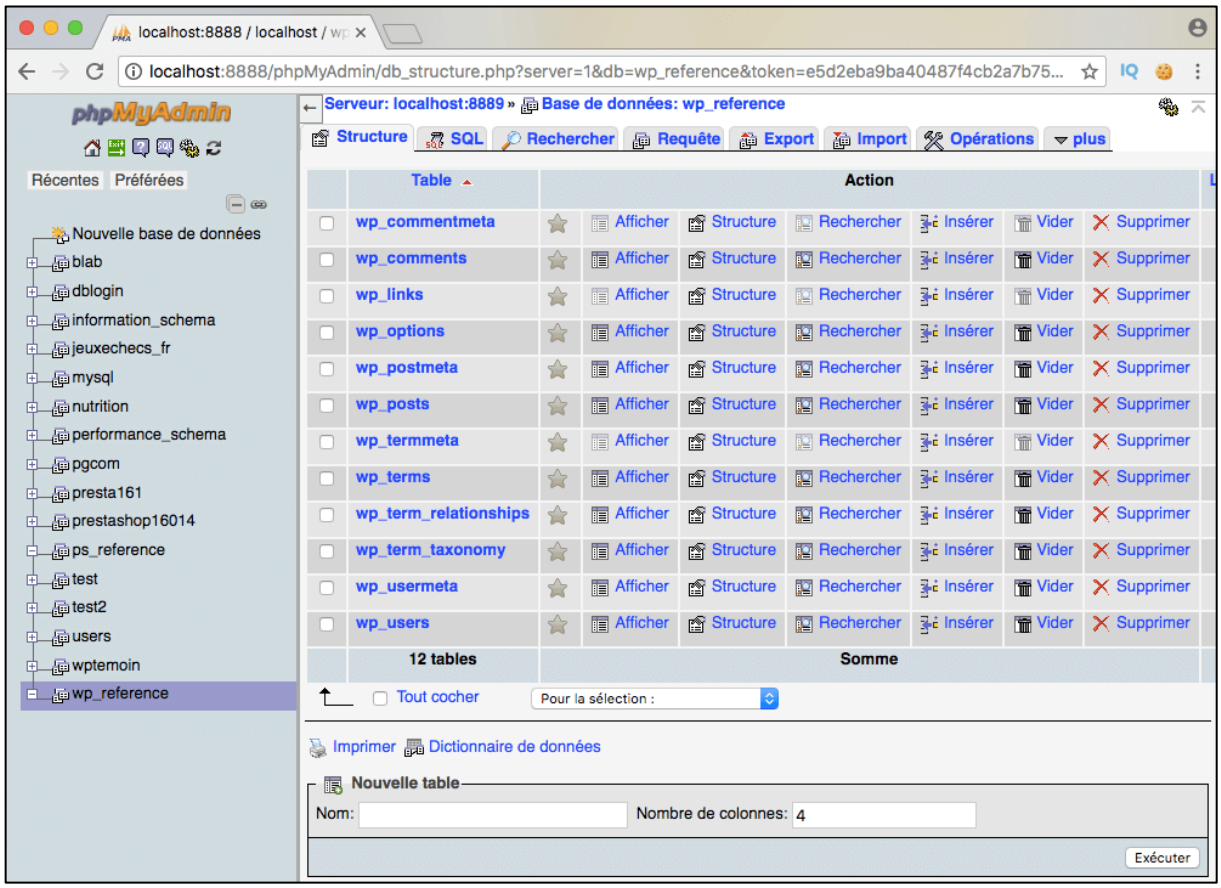
Retenez simplement pour le moment que phpMyAdmin est un logiciel complet qui propose toutes les fonctionnalités nécessaires pour nous permettre de gérer les paramètres et de manipuler nos bases de données.

Pour l'instant, ce qui nous intéresse est la liste que vous pouvez voir à gauche de la page. Ceci est la liste des bases de données que j'ai créées.

Lorsque vous cliquez sur une base de données, vous avez accès à la liste des tables de celle-ci. Ici, nous allons particulièrement nous intéresser à mes deux bases « ps_reference » et « wp_reference » qui contiennent les tables créées automatiquement lors de l'installation en local d'un PrestaShop et d'un WordPress.

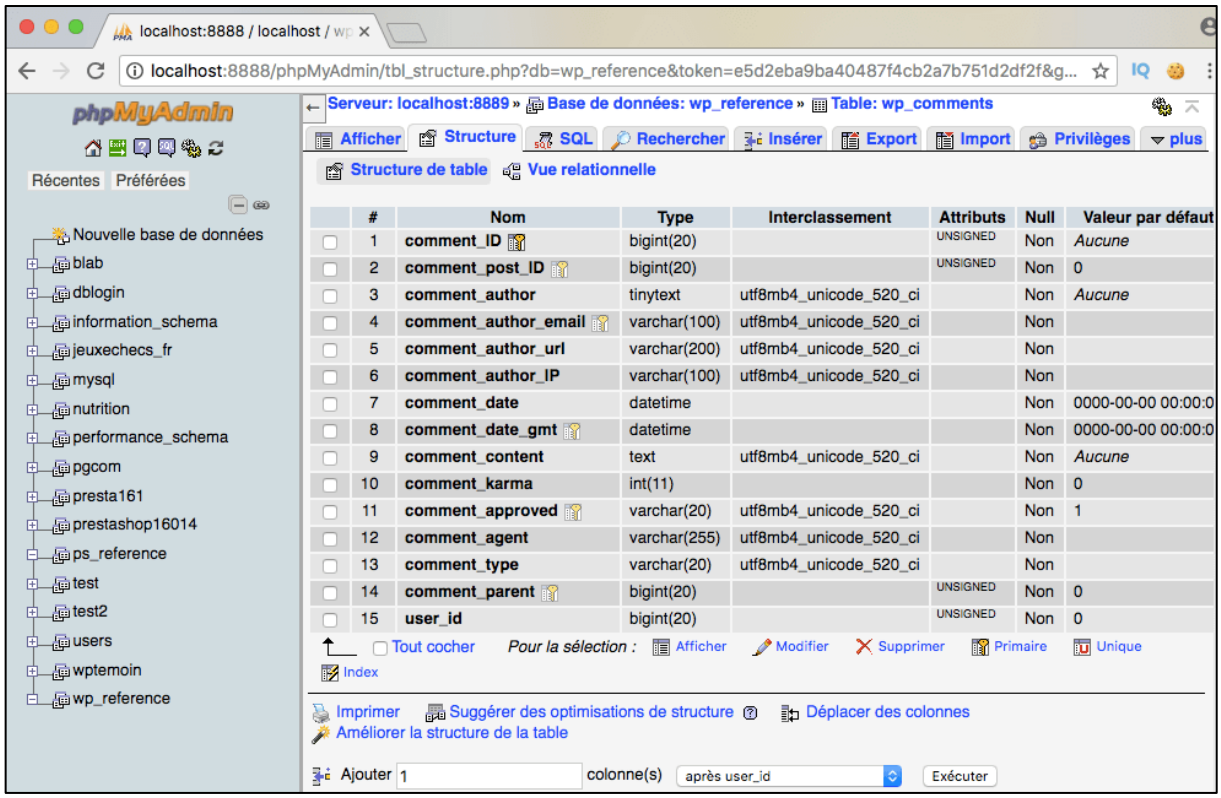
En cliquant sur ma base de données « wp_reference », on peut voir que cette base contient 12 tables, dont la table « wp_comments » qui va par exemple stocker les commentaires des utilisateurs ou la table « wp_users » qui va stocker les différents utilisateurs inscrits sur notre blog et les informations les concernant.

Note : Certaines images par la suite seront tronquées tout simplement car certaines tables sont très longues et je ne peux pas tout afficher dans une image ou cela serait illisible pour vous.

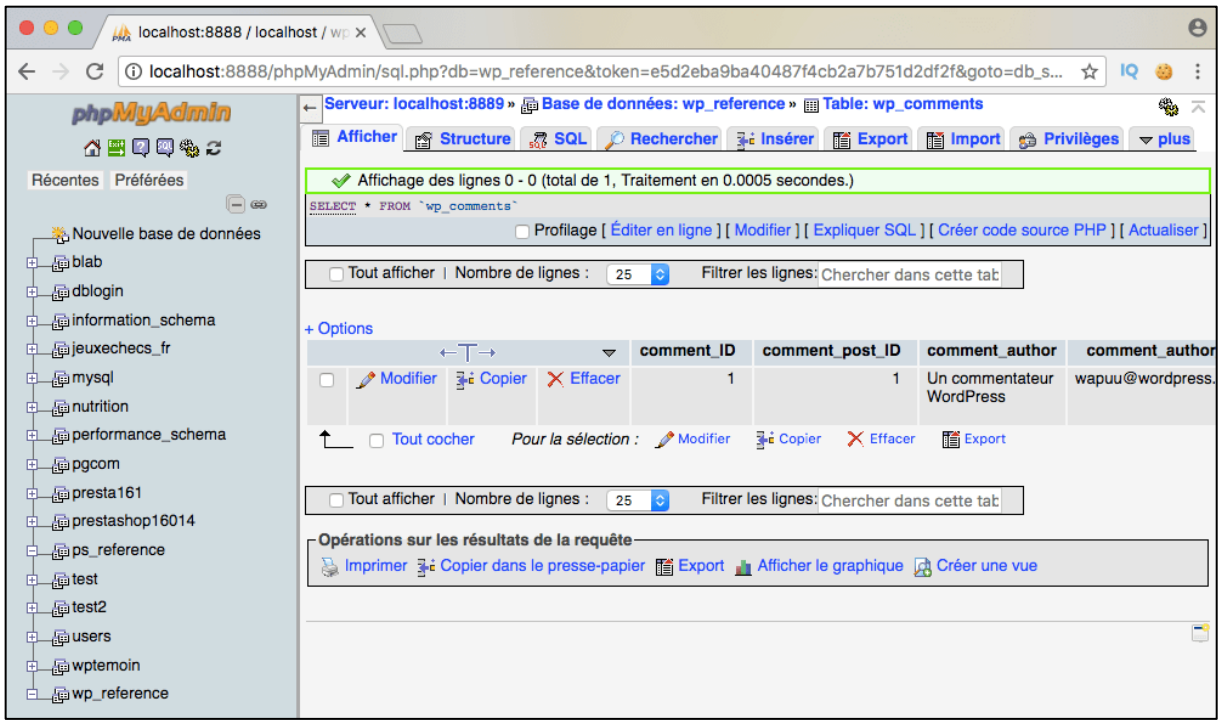


Vous pouvez déjà voir que l'interface de phpMyAdmin nous permet d'effectuer toutes sortes d'action sur nos tables : affichage, recherche, insertion, suppression…

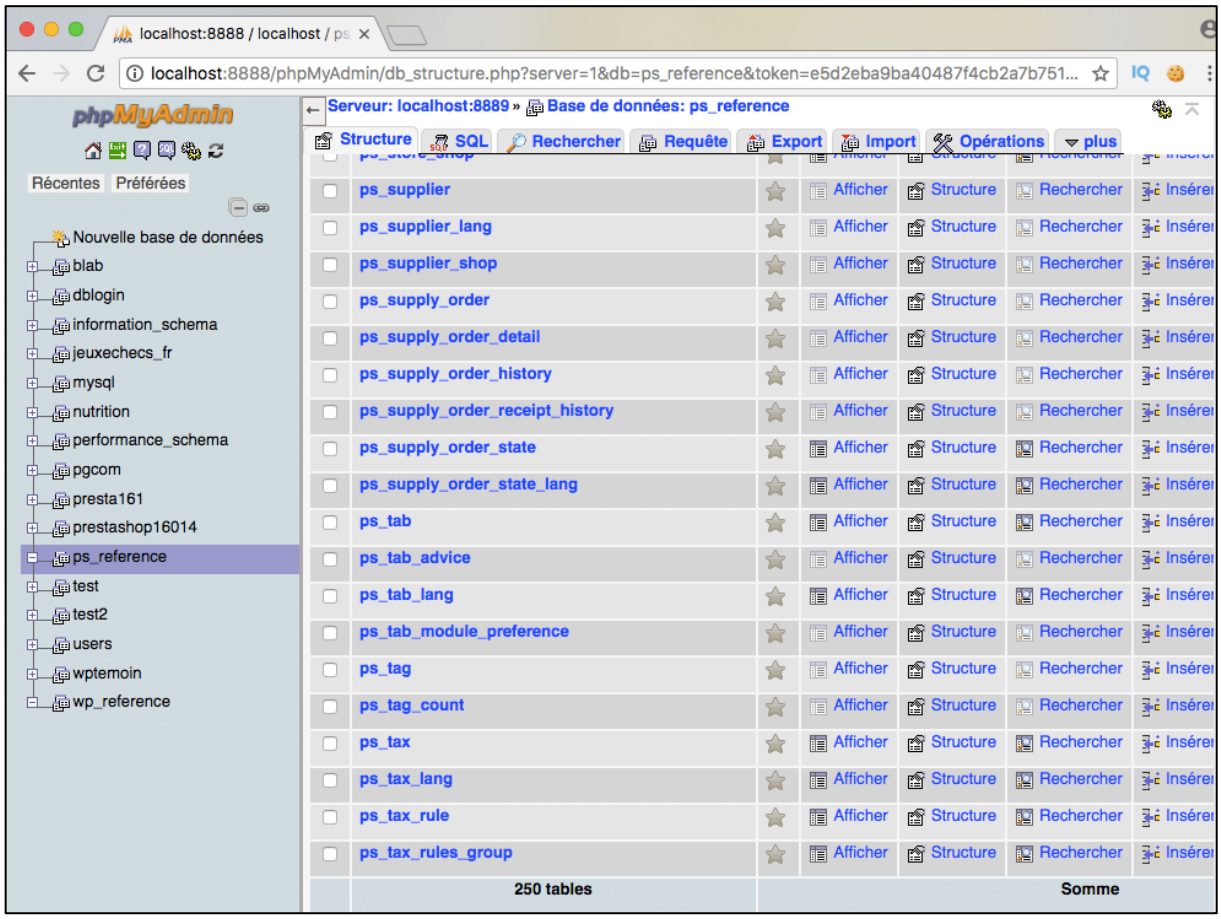
En cliquant sur « Structure » de la table « wp_comments », nous allons par exemple pouvoir voir la structure de notre table, c'est-à-dire les différentes colonnes de celles-ci et obtenir des informations par rapport à ces colonnes (type de données attendues, etc.).



En cliquant sur « Afficher », nous avons cette fois-ci accès à la liste des entrées c'est-à-dire, pour notre table « wp_comments », à la liste des commentaires postés sur notre blog. Comme je viens jute d'installer ce WordPress et comme je ne me suis même jamais connecté à son back office, cette table est vide car je n'ai pour le moment aucun commentaire sur mon blog (qui encore une fois n'est même pas à proprement dit créé).



Revenons maintenant en arrière et intéressons-nous cette fois ci à notre base de données « ps_reference ». En cliquant dessus, on s'aperçoit que PrestaShop a installé beaucoup plus de tables que WordPress. En effet, nous avons 250 tables dans notre base de données !



Je ne vais bien entendu pas expliquer à quoi correspond chaque table. L'idée à retenir ici est que selon les besoins d'une solution, la base de données ne va pas du tout être créée de la même manière que pour une autre solution.

Qui crée et comment créer les bases de données et les tables ?

Une nouvelle fois, en informatique, rien n'est magique et c'est bien là la difficulté de la gestion des bases de données.

Vous avez deux choix lors de la création d'un site Internet : soit utiliser une solution préconçue et prête à installer comme un PrestaShop ou un WordPress par exemple et le modifier ensuite selon vos besoins, soit créer une solution sur mesure.

Si vous choisissez d'installer une solution comme PrestaShop ou WordPress, vous n'aurez pas à vous soucier de la construction ni de la cohérence de vos bases de données. En effet, les solutions comme WordPress ou PrestaShop vont se charger de créer automatiquement les tables dont elles ont besoin.

Il vous suffira simplement de créer la base de données vide et d'indiquer son nom ainsi que les bons identifiants lors de l'installation de WordPress ou PrestaShop afin que la solution arrive à établir une connexion avec votre système de gestion de bases de données et puisse créer les tables. Notez que nombre d'hébergeurs vont même faire cela à votre place en vous proposant des « installations en 1 clic ».

En revanche, si vous décidez de créer un site vous-même de A à Z et d'utiliser les bases de données, alors il va bien falloir réfléchir à la structure et celles-ci.

Pour cela, vous devrez toujours vous demander quel type de données vous souhaitez stocker et pourquoi et également comment récupérer telle ou telle donnée par la suite.

Je ne vais pas vous mentir : bien créer une base de données requiert une grande expérience en tant que développeur et gestionnaire de projet et également une capacité d'anticipation et une vision d'ensemble du projet. Si vous devez un jour créer une structure relativement complexe, la simple création de la structure de votre base de données peut prendre des mois !

Dans la suite du cours, je vais vous donner les clefs pour manipuler les bases de données. Cependant, vous devrez faire ce travail de réflexion sur quelles données vous souhaitez stocker et pourquoi par vous-même.

Se connecter à une base de données MySQL en PHP

Dans ce nouveau chapitre, nous allons passer en revue les différents moyens que nous avons de nous connecter au serveur et donc à nos bases de données MySQL en PHP. Nous discuterons des avantages et des inconvénients de telle ou telle méthode et allons également apprendre à nous connecter à nos bases de données à proprement parler.

Se connecter à MySQL en PHP : les API proposées par le PHP

Pour pouvoir manipuler nos bases de données MySQL en PHP (sans passer par phpMyAdmin), nous allons déjà devoir nous connecter à MySQL.

Pour cela, le PHP met à notre disposition deux API (Application Programming Interface) :

- L'extension MySQLi ;
- L'extension PDO (PHP Data Objects).

Note : Auparavant, nous pouvions également utiliser l'extension MySQL. Cependant, cette extension est désormais dépréciée et a été remplacée par MySQLi (« i » signifie « improved », c'est-à-dire « amélioré » en français).

Quelle API préférer : MySQLi ou PDO ?

Le PHP nous fournit donc deux API pour nous connecter à MySQL et manipuler nos bases de données.

Chacune de ces deux API possède des forces différentes et comme vous vous en doutez elles ne sont pas forcément interchangeables.

Il existe notamment une différence notable entre ces deux API : l'extension MySQLi ne va fonctionner qu'avec les bases de données MySQL tandis que PDO va fonctionner avec 12 systèmes de bases de données différents.

Pour cette raison, nous préférerons généralement le PDO car si vous devez un jour utiliser un autre système de bases de données, le changement sera beaucoup plus simple que si vous avez tout codé en MySQLi auquel cas vous devrez réécrire le code dans son ensemble.

En termes de fonctionnement, MySQLi et PDO sont tous les deux orienté objet (bien que MySQLi propose également une API en procédural), et ils supportent également tous les deux les requêtes préparées qui servent à se prémunir des injections SQL (nous reparlerons de cela dans la suite du cours).

Dans ce cours, j'utiliserai donc PDO sauf pour ce chapitre où il me semble intéressant de vous montrer les différences d'écriture pour un script de connexion à une base de données MySQL.

Connexion au serveur avec MySQLi orienté objet

Pour se connecter au serveur et accéder à nos bases de données MySQL en MySQLi orienté objet, nous allons avoir besoin de trois choses : le nom du serveur ainsi qu'un nom d'utilisateur (avec des privilèges de type administrateur) et son mot de passe.

Dans le cas où votre site est hébergé sur un serveur, votre hébergeur vous donnera ces différents éléments. Ici, bien évidemment, nous travaillons en local. Le nom de notre serveur est donc `localhost`.

Concernant les identifiants au serveur local, ils peuvent changer selon vos paramétrages et selon le système que vous utilisez. Cependant, si vous disposez des réglages par défaut, le nom d'utilisateur devrait toujours être `root` et le mot de passe associé devrait être soit `root` soit une chaine de caractère vide.

Nous allons devoir procéder à deux opérations lors de la connexion au serveur : se connecter à proprement parler et vérifier que la connexion a bien été établie et si ce n'est pas le cas afficher le message d'erreur correspondant.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <h1>Bases de données MySQL</h1>

        <?php
            $servername = "localhost";
            $username = "root";
            $password = "root";

            //On crée la connexion
            $conn = new mysqli($servername, $username, $password);

            //On vérifie la connexion
            if ($conn->connect_error){
                die("Erreur : " . $conn->connect_error);
            }
            echo "Connexion réussie";
        ?>
    </body>
</html>
```

Pour se connecter, nous instancions la classe prédéfinie `mysqli` en passant au constructeur les informations suivantes : nom du serveur auquel on doit se connecter, nom d'utilisateur et mot de passe.

Nous stockons les informations de connexion dans un objet qu'on appelle ici `$conn`. Cet objet représente notre connexion en soi.

Ensuite, nous devons tester que la connexion a bien été établie car dans le cas où celle-ci échoue on voudra renvoyer un message d'erreur

Il est en essentiel de considérer les potentielles erreurs de connexion à nos bases de données pour éviter que des utilisateurs mal intentionnés tentent de récupérer les informations relatives à la tentative de connexion.

Pour cela, nous utilisons la propriété `connect_error` de la classe `mysqli` qui retourne un message d'erreur relatif à l'erreur rencontrée en cas d'erreur de connexion MySQL ainsi que la fonction `die()` pour stopper l'exécution du script en cas d'erreur.

Attention : La propriété `connect_error` de `mysqli` ne fonctionne correctement que depuis la version 5.3 de PHP. Utilisez la fonction `mysqli_connect_error()` pour les versions antérieures.

Notez ici qu'on aurait également pu utiliser les exceptions et des blocs `try` et `catch` pour gérer les erreurs potentielles. Je voulais juste vous présenter une autre manière de faire ici.

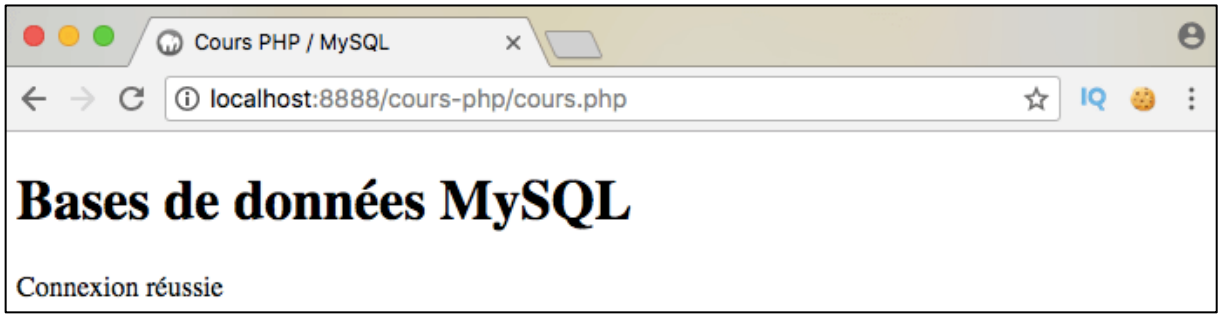
Dans le cas où la connexion réussit, on se contente d'afficher un message « connexion réussie ».

Si vous désirez la liste complète des propriétés et méthodes de la classe `mysqli`, je vous invite à consulter la documentation officielle.

Connexion au serveur avec MySQLi procédural

Nous allons également pouvoir utiliser un script en procédural avec MySQLi pour nous connecter au serveur et à la base de données MySQL.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <h1>Bases de données MySQL</h1>

        <?php
            $servername = "localhost";
            $username = "root";
            $password = "root";

            //On crée la connexion
            $conn = mysqli_connect($servername, $username, $password);

            //On vérifie la connexion
            if (!$conn){
                die("Erreur : " . mysqli_connect_error());
            }
            echo "Connexion réussie";
        ?>
    </body>
</html>
```



Ce script ressemble à priori au précédent et pourtant il est bien très différent : nous n'avons cette fois-ci plus recours à notre classe `mysqli` ni à l'orienté objet.

A la place, nous utilisons les fonctions `mysqli_connect()` pour nous connecter à la base de données et `mysqli_connect_error()` pour obtenir des informations sur l'erreur de connexion si il y en a une.

En dehors de ça, le principe reste le même : nous devons toujours fournir le nom du serveur ainsi que des identifiants de connexion (nom d'utilisateur et mot de passe) pour se connecter avec la fonction `mysqli_connect()` et nous prenons toujours en charge les cas d'erreur de connexion et stoppant l'exécution du script avec la fonction `die()`.

Connexion au serveur avec PDO

Pour se connecter en utilisant PDO, nous allons devoir instancier la classe PDO en passant au constructeur la source de la base de données (serveur + nom de la base de données) ainsi qu'un nom d'utilisateur et un mot de passe.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <h1>Bases de données MySQL</h1>

        <?php
            $servername = "localhost";
            $username = "root";
            $password = "root";

            $conn = new PDO("mysql:host=$servername;dbname=bddtest", $username, $password);
        ?>
    </body>
</html>
```

Vous pouvez déjà remarquer ici que pour se connecter à une base de données avec PDO, vous devez passer son nom dans le constructeur de la classe PDO. Cela implique donc qu'il faut que la base ait déjà été créée au préalable (avec phpMyAdmin par exemple) ou qu'on la crée dans le même script.

Notez également qu'avec PDO il est véritablement indispensable que votre script gère et capture les exceptions (erreurs) qui peuvent survenir durant la connexion à la base de données.

En effet, si votre script ne capture pas ces exceptions, l'action par défaut du moteur Zend va être de terminer le script et d'afficher une trace. Cette trace contient tous les détails de connexion à la base de données (nom d'utilisateur, mot de passe, etc.). Nous devons donc la capturer pour éviter que des utilisateurs malveillants tentent de la lire.

Pour faire cela, nous allons utiliser des blocs try et catch.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <h1>Bases de données MySQL</h1>

        <?php
            $servername = "localhost";
            $username = "root";
            $password = "root";

            //On teste la connexion
            try{
                $conn = new PDO("mysql:host=$servername;dbname=bddtest", $username, $password);
                //On définit le mode d'erreur de DPO sur Exception
                $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
                echo "Connexion réussie";
            }

            /*On capture les exceptions si une exception est lancée et on affiche
             *les informations relatives à celle ci*/
            catch(PDOException $e){
                echo "Erreur : " . $e->getMessage();
            }
        ?>
    </body>
</html>
```

Ici, nous utilisons également la méthode `setAttribute()` en lui passant deux arguments `PDO::ATTR_ERRMODE` et `PDO::ERRMODE_EXCEPTION`.

La méthode `setAttribute()` sert à configurer un attribut PDO. Dans ce cas précis, nous lui demandons de configurer l'attribut `PDO::ATTR_ERRMODE` qui sert à créer un rapport d'erreur et nous précisons que l'on souhaite qu'il émette une exception avec `PDO::ERRMODE_EXCEPTION`.

Plus précisément, en utilisant `PDO::ERRMODE_EXCEPTION` on demande au PHP de lancer une exception issue de la classe `PDOException` (classes étendue de `Exception`) et d'en définir les propriétés afin de représenter le code d'erreur et les informations complémentaires.

Ensuite, nous n'avons plus qu'à capturer cette exception `PDOException` et à afficher le message d'erreur correspondant. C'est le rôle de notre bloc `catch`.

Fermer la connexion à la base de données

Une fois la connexion à la base de données ouverte, celle-ci reste active jusqu'à la fin de l'exécution de votre script.

Pour fermer la connexion avant cela, nous allons devoir utiliser différentes méthodes selon la méthode d'ouverture choisie.

Si on utilise MySQLi orienté objet, alors il faudra utiliser la méthode `close()`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <h1>Bases de données MySQL</h1>

        <?php
            $servername = "localhost";
            $username = "root";
            $password = "root";

            $conn = new mysqli($servername,$username, $password);

            if($conn->connect_error){
                die("Erreur : " . $conn->connect_error);
            }
            echo "Connexion réussie";

            //On ferme la connexion
            $conn->close();
        ?>
    </body>
</html>
```

Si on utilise MySQLi procédural, on utilisera la fonction `mysqli_close()`.

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <h1>Bases de données MySQL</h1>

        <?php
            $servername = "localhost";
            $username = "root";
            $password = "root";

            $conn = mysqli_connect($servername,$username, $password);

            if(!$conn){
                die("Erreur : " . mysqli_connect_error());
            }
            echo "Connexion réussie";

            //On ferme la connexion
            mysqli_close($conn);
        ?>
    </body>
</html>
```

Si on utilise PDO, il faudra détruire l'objet représentant la connexion et effacer toutes ses références. Nous pouvons faire cela en assignant la valeur null à la variable gérant l'objet.

```php
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <h1>Bases de données MySQL</h1>

        <?php
            $servername = "localhost";
            $username = "root";
            $password = "root";

            //On tente d'établir la connexion
            try{
                $conn = new PDO("mysql:host=$servername;dbname=bddtest", $username, $password);
                //On définit le mode d'erreur de DPO sur Exception
                $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
                echo "Connexion réussie";
            }

            /*On capture les exceptions si une exception est lancée et on affiche
             *les informations relatives à celle ci*/
            catch(PDOException $e){
                echo "Erreur : " . $e->getMessage();
            }

            //On ferme la connexion
            $conn = null;
        ?>
    </body>
</html>
```

Créer une base de données et une table

Nous savons désormais nous connecter au serveur de différentes manières, il est donc maintenant temps d'apprendre à manipuler nos bases de données à proprement parler.

Pour cela, nous allons déjà devoir créer une base de données et des tables.

Dans cette leçon, je vous propose donc de voir comment créer une base de données et des tables avec PDO ou directement avec phpMyAdmin.

Création d'une base de données en utilisant PDO

Nous allons pouvoir créer une nouvelle base de données avec PDO en PHP en utilisant la requête SQL `CREATE DATABASE` suivie du nom que l'on souhaite donner à notre base de données.

Note : A partir de maintenant, nous allons commencer à découvrir et à utiliser le langage SQL. Comme nous l'avons expliqué précédemment, nous allons envoyer nos requêtes SQL via PDO en PHP.

Pour exécuter une requête SQL en PDO, nous allons devoir utiliser la méthode `exec()` qui va prendre en paramètre une requête SQL.

Voyons immédiatement le code de création d'une base de données qu'on appelle « pdodb ».

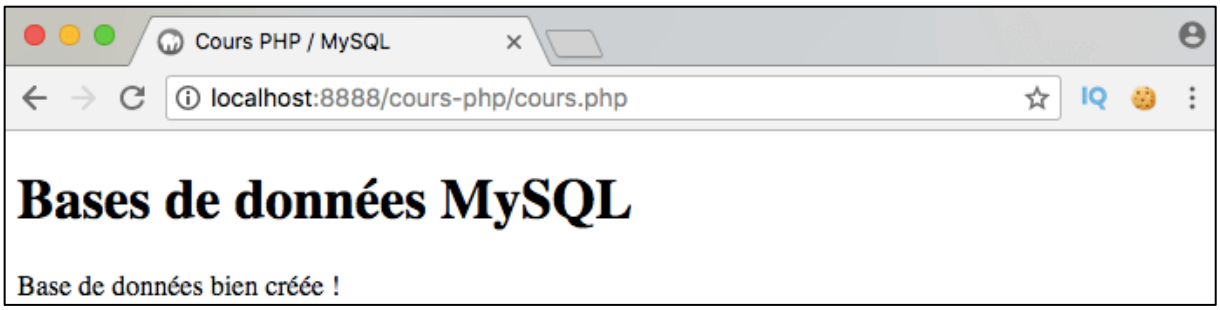
```php
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <h1>Bases de données MySQL</h1>

        <?php
            $servername = "localhost";
            $username = "root";
            $password = "root";

            try{
                $dbco = new PDO("mysql:host=$servername", $username, $password);
                $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

                $sql = "CREATE DATABASE pdodb";
                $dbco->exec($sql);

                echo "Base de données bien créée !";
            }

            catch(PDOException $e){
                echo "Erreur : " . $e->getMessage();
            }
        ?>
    </body>
</html>
```

On commence déjà par se connecter au serveur. Notez ici qu'on ne précise plus de nom de bases de données puisque nous allons la créer explicitement dans la suite du script. Bien évidemment, nous prenons toujours en charge les erreurs et exceptions.

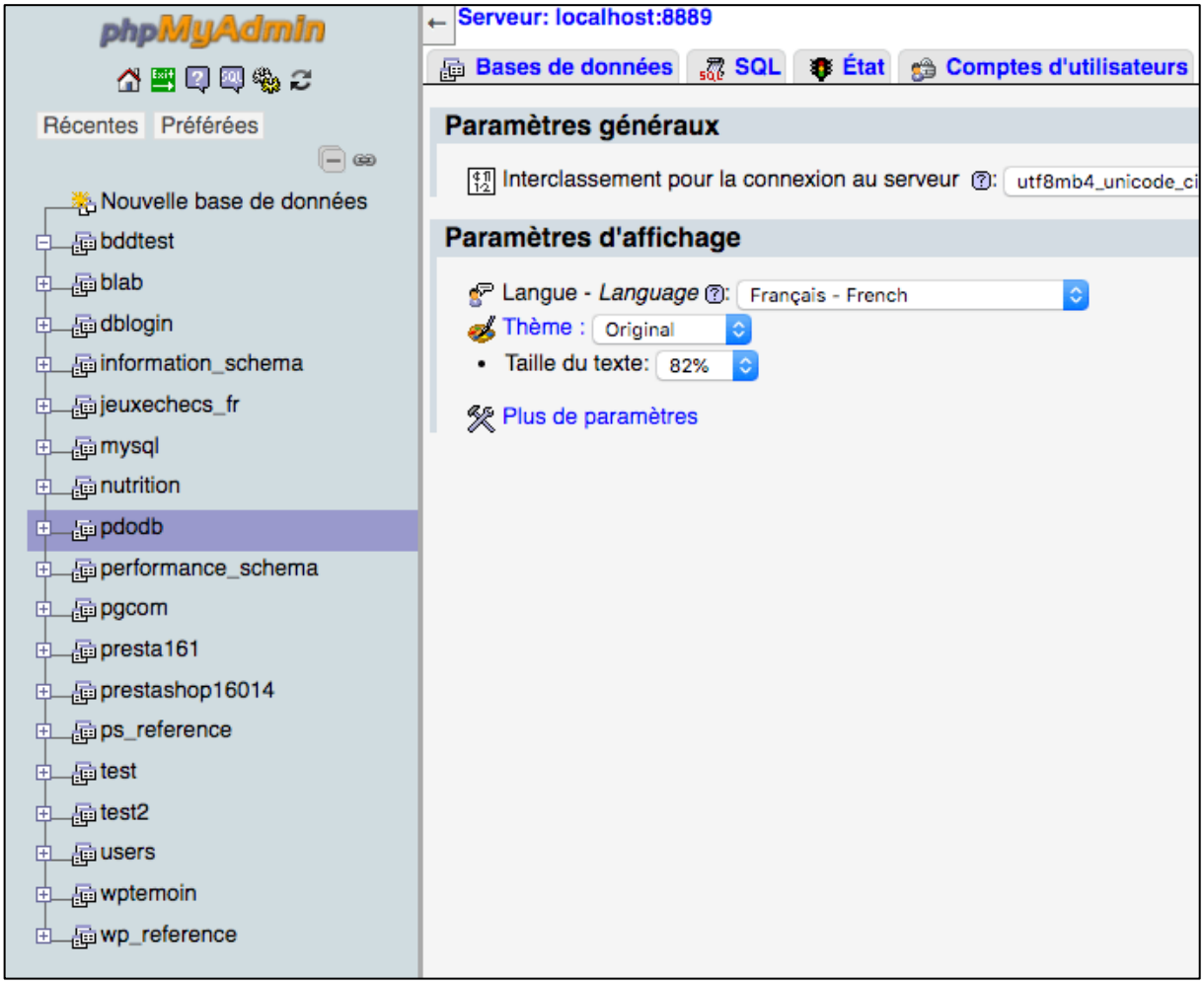
Ensuite, nous écrivons notre requête SQL que nous enfermons dans une variable pour une plus grande liberté d'utilisation par la suite. Notez qu'on aurait aussi bien pu placer la requête directement en argument de `exec()`. Par convention, nous écrirons toujours nos requêtes SQL en majuscule pour bien les séparer du reste du code.

On utilise donc enfin notre méthode `exec()` (méthode qui appartient bien évidemment à notre classe `PDO`) et on lui passe la variable contenant notre requête SQL en argument.

La méthode `exec()` va se charger d'exécuter notre requête et de créer la base de données « pdodb ».



Vous pouvez aller vérifier dans votre phpMyAdmin, la nouvelle base de données a bien été créée.



Création d'une table en utilisant MySQL et PDO

Une base de données est constituée de tables. Les tables sont les « casiers » dans lesquelles nous allons stocker nos données. Mais avant de pouvoir stocker des données, il va déjà falloir apprendre à créer des tables dans notre base de données !

Pour créer une nouvelle table dans une base de données, nous allons utiliser la requête SQL `CREATE TABLE` suivie du nom que l'on souhaite donner à notre table et nous allons

également pouvoir préciser entre parenthèse le nom des colonnes de notre table ainsi que le type de données qui doit être stocké dans chaque colonne.

MySQL nous offre beaucoup de choix de types de données différents nous permettant de créer des tables de manière vraiment précise. Pour le moment, vous pouvez retenir qu'il existe quatre grands types de données principaux en MySQL : les données de type texte, les données de type nombre, les données de type date et les données de type spatial.

Les sous types de valeurs les plus courants et les plus utilisés sont :

- `INT` : accepte un nombre entier de 4 octets. La fourchette pour les entiers relatifs est [-2 147 483 648, 2 147 483 647], celle pour les entiers positifs est [0, 4 294 967 295] ;
- `VARCHAR` : accepte une chaine de longueur variable (entre 0 et 65 535 caractères). La longueur effective réelle de la chaine dépend de la taille maximum d'une ligne ;
- `TEXT` : accepte une chaine de caractère d'une longueur maximum de 65 535 caractères ;
- `DATE` : accepte une date se situant entre le 1er janvier de l'an 1000 et le 31 décembre de l'an 9999.

En plus de cela, nous allons également pouvoir spécifier des attributs ou contraintes pour chacune des colonnes de notre table. Ces attributs ou contraintes vont venir apporter des contraintes supplémentaires sur les données attendues (non nulle, etc.) ou vont définir des comportements.

Voici les attributs qu'on va pouvoir ajouter à nos colonnes durant la création de notre table :

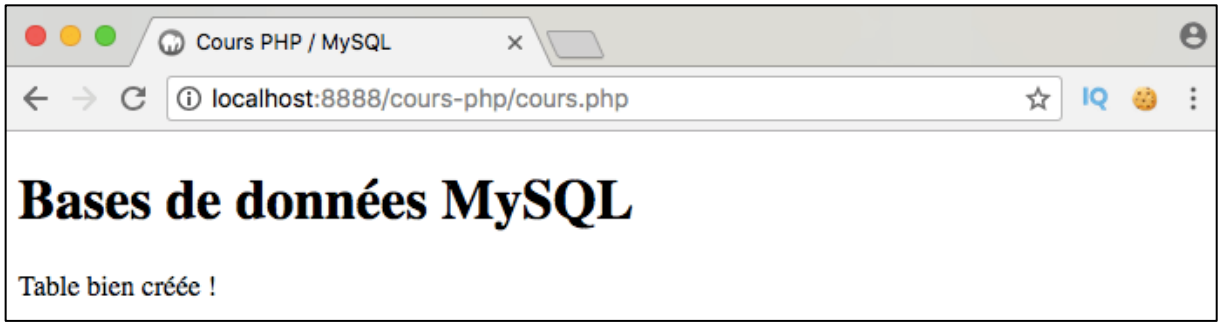
- `NOT NULL` – Signifie que chaque entrée doit contenir une valeur pour cette colonne. La valeur `null` n'est pas acceptée ;
- `UNIQUE` – Chacune des valeurs dans la colonne doit être unique (est utile par exemple lorsqu'on reçoit des adresses mail, cela évite qu'un utilisateur s'inscrive deux fois sur notre site entre autres) ;
- `PRIMARY KEY` – Est utilisé pour identifier de manière unique chaque nouvelle entrée dans une table. C'est une combinaison de `NOT NULL` et de `UNIQUE`. `PRIMARY KEY` ne doit s'appliquer qu'à une colonne dans une table mais chaque table doit obligatoirement posséder une colonne avec une `PRIMARY KEY`. La colonne avec `PRIMARY KEY` est souvent une colonne d'ID (nombres) qui s'auto-incrémentent ;
- `FOREIGN KEY` – Utilisée pour empêcher des actions qui pourraient détruire les liens entre des tables. La `FOREIGN KEY` sert à identifier une colonne qui est identique à une colonne portant une `PRIMARY KEY` dans une autre table ;
- `CHECK` – Sert à s'assurer que toutes les valeurs dans une colonne satisfont à une certaine condition ou se trouve dans un certain intervalle spécifié ;
- `DEFAULT` value – Sert à définir une valeur par défaut qui va être renseignée si aucune valeur n'est fournie ;
- `AUTO_INCREMENT` – MySQL va automatiquement incrémenter (c'est-à-dire ajouter 1) au champ pour chaque nouvelle entrée ;
- `UNSIGNED` – Utilisé pour les données de type nombre, cette contrainte permet de limiter les données reçues aux nombres positifs (0 inclus).

Voyons immédiatement en pratique comment on pourrait créer une table « Clients » dans notre base « pdodb ».

Notre table va contenir 9 colonnes :

- Id
- Nom
- Prenom
- Adresse
- Ville
- CodePostal
- Pays
- Mail
- DateInscription

```php
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <h1>Bases de données MySQL</h1>

        <?php
            $servname = "localhost";
            $dbname = "pdodb";
            $user = "root";
            $pass = "root";

            try{
                $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
                $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

                $sql = "CREATE TABLE Clients(
                Id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
                Nom VARCHAR(30) NOT NULL,
                Prenom VARCHAR(30) NOT NULL,
                Adresse VARCHAR(70) NOT NULL,
                Ville VARCHAR(30) NOT NULL,
                Codepostal INT UNSIGNED NOT NULL,
                Pays VARCHAR(30) NOT NULL,
                Mail VARCHAR(50) NOT NULL,
                DateInscription TIMESTAMP,
                UNIQUE (Mail))";

                $dbco->exec($sql);
                echo "Table bien créée !";
            }

            catch(PDOException $e){
                echo "Erreur : " . $e->getMessage();
            }
        ?>
    </body>
</html>
```



Ici, nous créons donc la table « Clients » en utilisant la requête SQL `CREATE TABLE Clients`.

Entre les parenthèses, nous précisons les colonnes que doit contenir la table en indiquant déjà le type de données attendues et les contraintes relatives à chaque colonne et en définissant l'une de nos colonnes comme `PRIMARY KEY`.

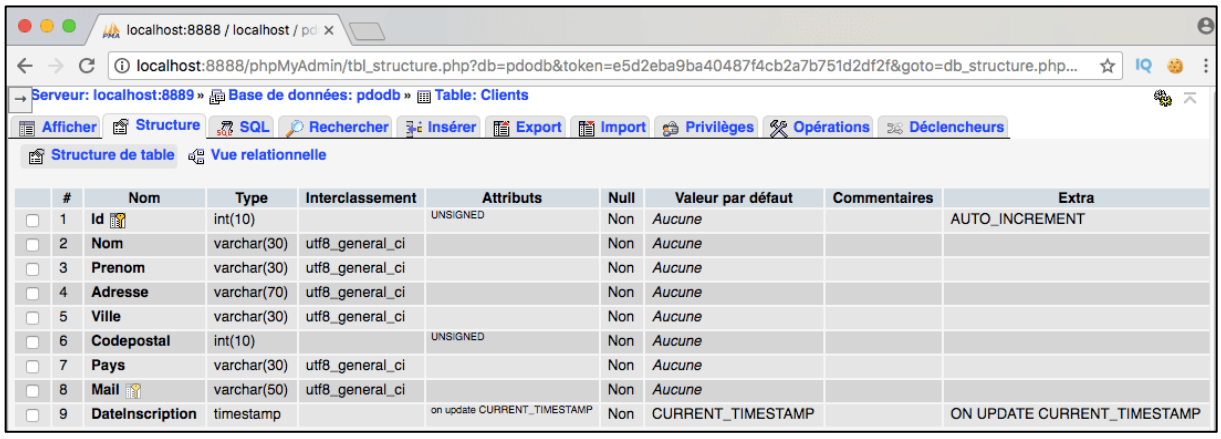
La syntaxe du SQL nous impose de séparer la déclaration de chaque colonne par une virgule.

Les chiffres entre parenthèses après les `VARCHAR` sont facultatifs : ils permettent juste d'indiquer le maximum de caractère que la colonne peut accepter pour une valeur. Indiquer cela permet d'optimiser très marginalement la table mais est surtout considéré comme une bonne pratique.

Le type de valeur `TIMESTAMP` signifie que la date courante sera stockée lors de chaque nouvelle entrée dans la table.

Finalement, vous pouvez remarquer qu'on ajoute une contrainte `UNIQUE` pour notre colonne Mail de manière un peu différente du reste. C'est l'écriture conseillée en SQL.

Vous pouvez vérifier dans phpMyAdmin que la table a bien été créée avec ses colonnes en cliquant sur le nom de la table dans notre base de données puis en cliquant sur « Structure » :

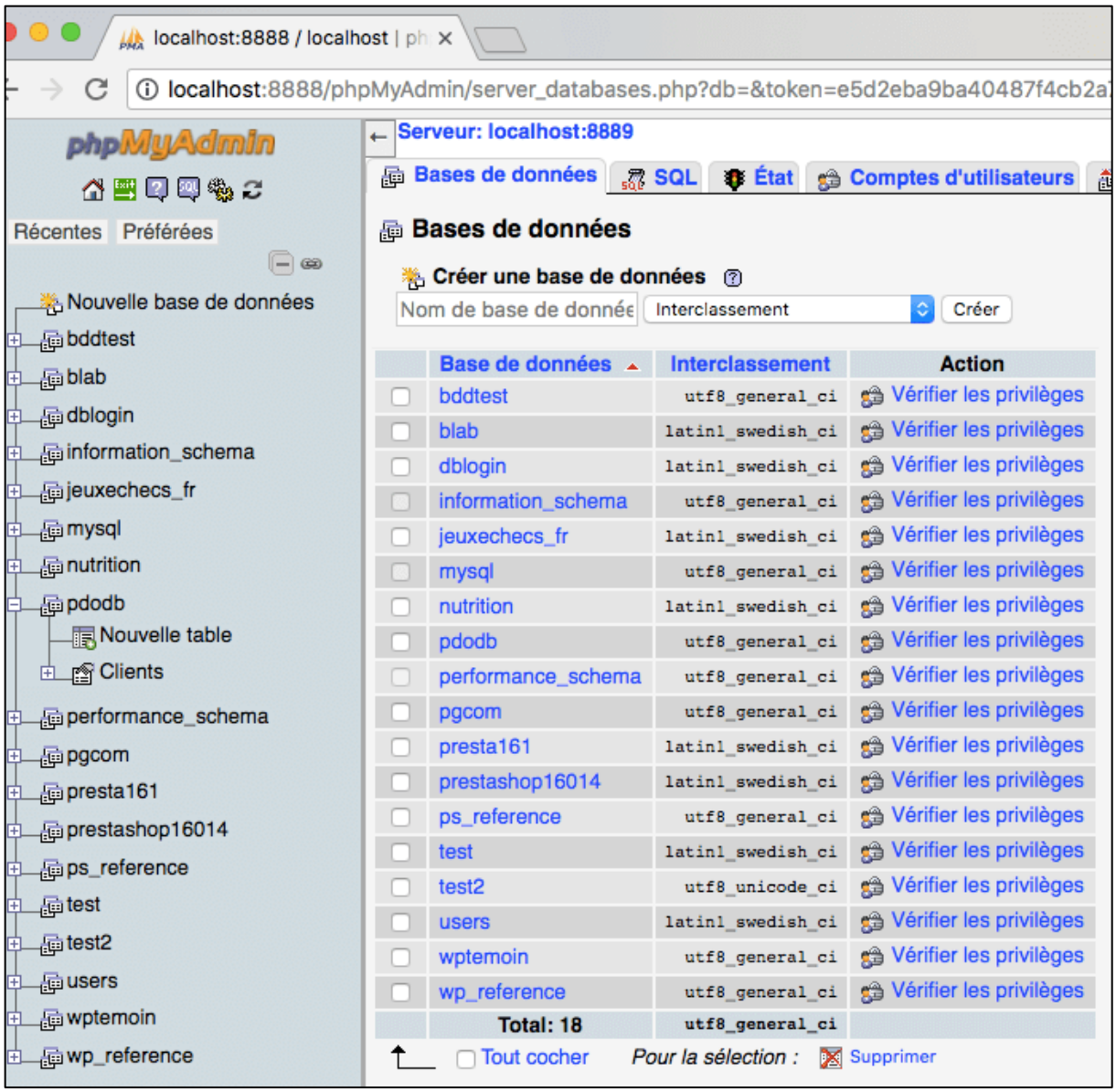


	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
☐	1	Id	int(10)		UNSIGNED	Non	*Aucune*		AUTO_INCREMENT
☐	2	Nom	varchar(30)	utf8_general_ci		Non	*Aucune*		
☐	3	Prenom	varchar(30)	utf8_general_ci		Non	*Aucune*		
☐	4	Adresse	varchar(70)	utf8_general_ci		Non	*Aucune*		
☐	5	Ville	varchar(30)	utf8_general_ci		Non	*Aucune*		
☐	6	Codepostal	int(10)		UNSIGNED	Non	*Aucune*		
☐	7	Pays	varchar(30)	utf8_general_ci		Non	*Aucune*		
☐	8	Mail	varchar(50)	utf8_general_ci		Non	*Aucune*		
☐	9	DateInscription	timestamp		on update CURRENT_TIMESTAMP	Non	CURRENT_TIMESTAMP		ON UPDATE CURRENT_TIMESTAMP

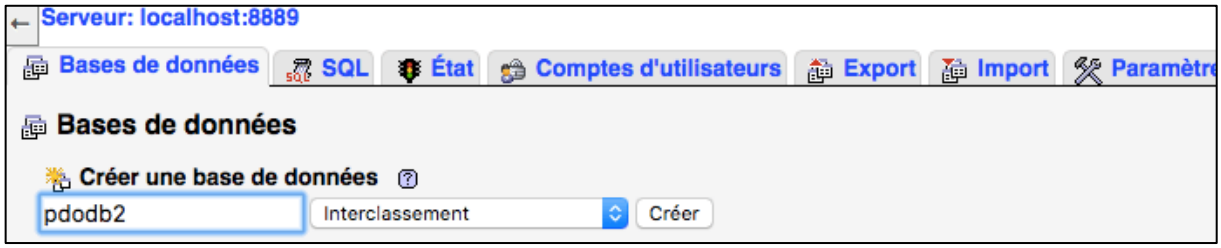
Création d'une BDD et d'une table avec phpMyAdmin

Il va être très facile de créer une base de données et une table avec l'outil phpMyAdmin. Cependant, encore une fois, la limite est que nous devons faire cela « manuellement » et non pas dynamiquement.

Pour cela, rendez-vous sur la page d'accueil de phpMyAdmin. A partir de là, vous pouvez soit cliquer sur « Nouvelle base de données » dans la colonne de gauche, soit sur l'onglet « Bases de données en haut ».

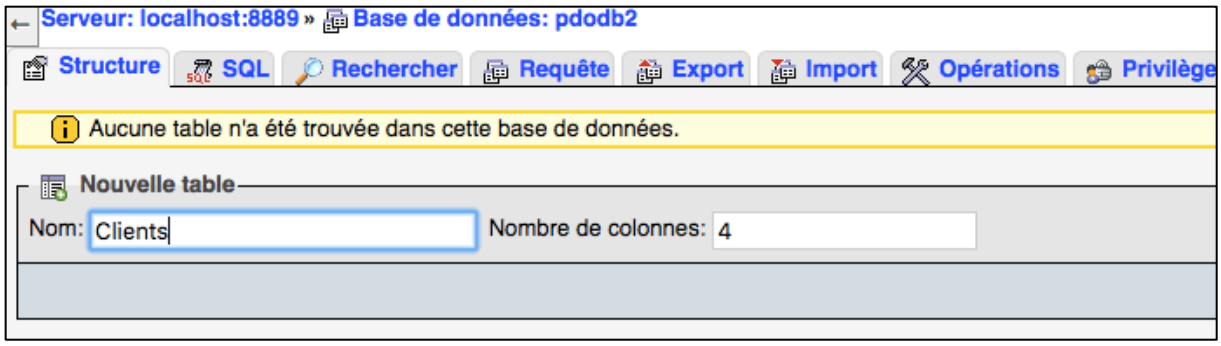


Une fois arrivé ici, renseignez le nom de la base de données que vous souhaitez créer (on peut par exemple l'appeler « pdodb2 ») et cliquez sur « créer ».

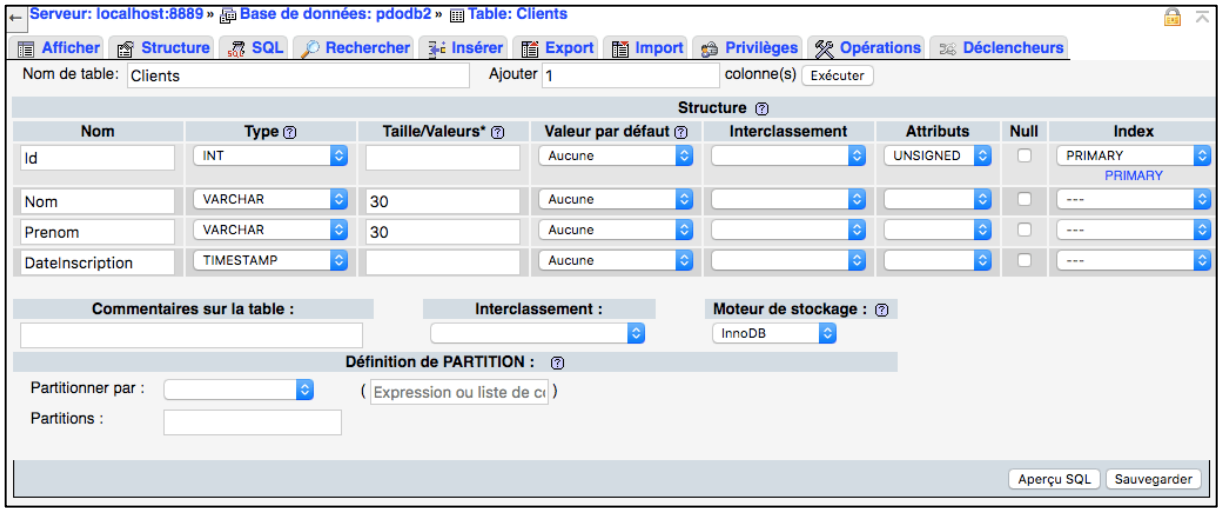


phpMyAdmin crée la table et vous amène sur une page vous permettant déjà de créer une première table. Vous pouvez revenir sur cette même page en cliquant sur le nom de la base de données créée dans la liste des bases à gauche de votre page.

Créons donc à nouveau une table « Clients » avec cette fois-ci simplement 4 colonnes pour aller plus vite.

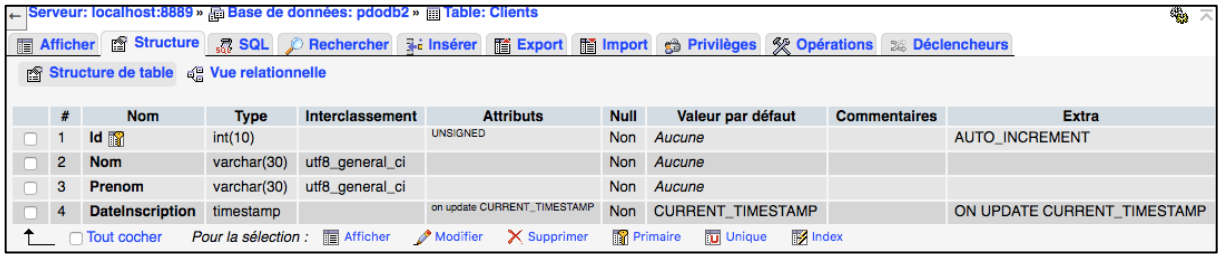


A partir de là, on vous propose de définir les colonnes de la table. Notre première colonne va se nommer Id, acceptant des données de type INT, UNSIGNED, PRIMARY KEY et AUTO_INCREMENT. Notre deuxième colonne est une colonne Nom, acceptant des données de type VARCHAR(30) et NOT NULL. Idem pour notre troisième colonne Prenom et finalement notre dernière colonne contient les dates d'inscription de nos clients.



Notez que dans phpMyAdmin, il suffit de ne pas cocher la case « Null » pour que notre colonne possède la contrainte NOT NULL.

Nous n'avons plus qu'à cliquer sur « Sauvegarder » afin que notre table et nos colonnes soient définitivement créées.



PARTIE XIV

Manipuler des données

Insérer des données dans une table MySQL

Une fois notre base de données et nos premières tables créées, nous allons pouvoir commencer à insérer des données dans ces dernières.

Dans cette nouvelle leçon, nous allons voir comment insérer une ou plusieurs entrées dans une table et allons également comprendre l'intérêt de préparer ses requêtes SQL et voir comment faire cela.

Insérer des données dans une table

Pour insérer des données dans une table, nous allons cette fois-ci utiliser l'instruction SQL `INSERT INTO` suivie du nom de la table dans laquelle on souhaite insérer une nouvelle entrée avec sa structure puis le mot clef `VALUES` avec les différentes valeurs à insérer.

Concrètement, la structure de la requête SQL va être la suivante : `INSERT INTO nom_de_table (nom_colonne1, nom_colonne2, nom_colonne3, ...) VALUES (valeur1, valeur2, valeur3, ...)`.

Il y a cependant quelques règles de syntaxe à respecter afin que cette requête fonctionne :

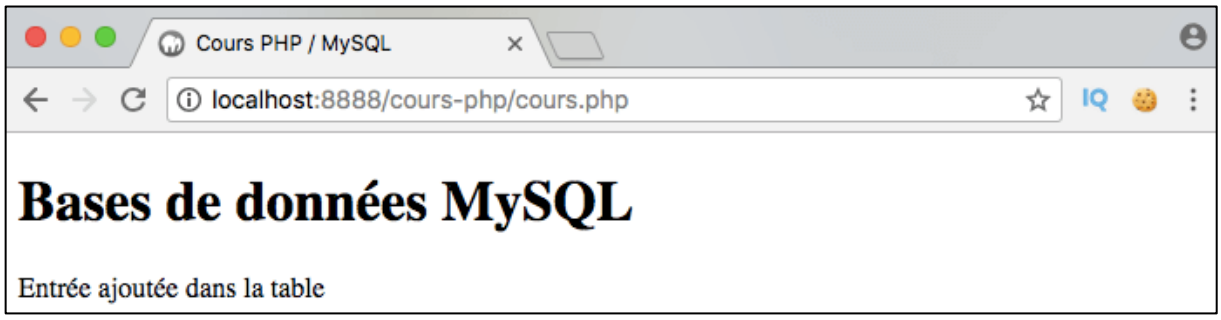
- Les valeurs de type chaine de caractère (String) doivent être placées entre apostrophes ;
- La valeur NULL ne doit pas être placée entre apostrophes ;
- Les valeurs de type numérique ne doivent pas être placées entre apostrophes.

A priori, vous devriez avoir autant de valeurs à insérer qu'il y a de colonnes dans votre table. Cependant, notez qu'il n'est pas nécessaire de préciser les colonnes possédant un attribut `AUTO_INCREMENT` ou `TIMESTAMP` ni leurs valeurs associées puisque par définition MySQL stockera automatiquement les valeurs courantes.

Reprenons par exemple notre table « Clients » créée dans la leçon précédente. Cette table possède neuf colonnes dont une colonne Id avec un attribut AUTO_INCREMENT et une colonne DateInscription qui possède un attribut TIMESTAMP.

Essayons d'insérer une première entrée dans cette table en utilisant PHP et PDO :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>

    <?php
      $servname = "localhost";
      $dbname = "pdodb";
      $user = "root";
      $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sql = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
        VALUES ('Giraud','Pierre','Quai d\'Europe','Toulon',83000,'France','pierre.giraud@edhec.com')";

        $dbco->exec($sql);
        echo "Entrée ajoutée dans la table";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

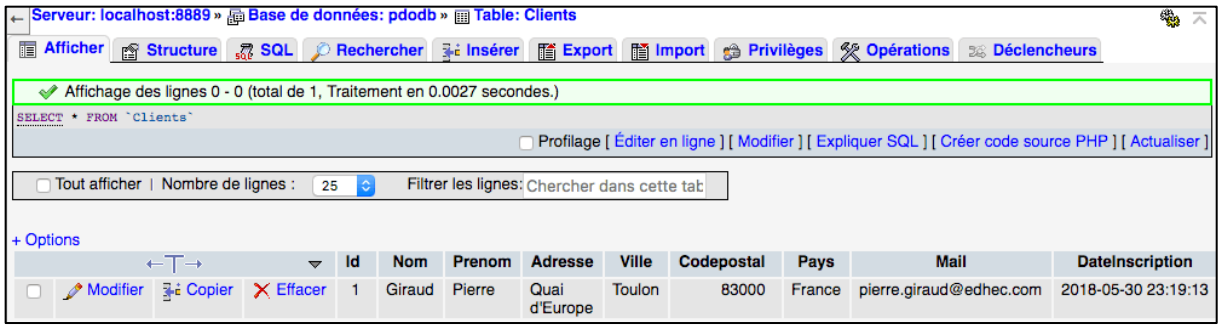


Ici, on insère dans notre table Clients une entrée. Pour cela, on utilise `INSERT INTO` et on précise le nom des colonnes pour lesquelles on doit renseigner une valeur : Nom, Prénom, Adresse, Ville, Codepostal, Pays et Mail.

Ensuite, dans la même requête SQL, on transmet les valeurs relatives à ces colonnes. Le reste du script est très classique (connexion à la base de données, exécution de notre requête SQL et gestion des exceptions).

Notez une nouvelle fois que nous n'avons pas à préciser nos colonnes Id et DateInscription ni les valeurs relatives à ces colonnes puisque celles-ci possèdent respectivement un AUTO_INCREMENT et un TIMESTAMP. Le MySQL mettra donc à jour les valeurs automatiquement par lui-même.

Une fois notre code exécuté, nous pouvons aller voir notre table dans phpMyAdmin pour voir si l'entrée a bien été ajoutée comme on le désirait.

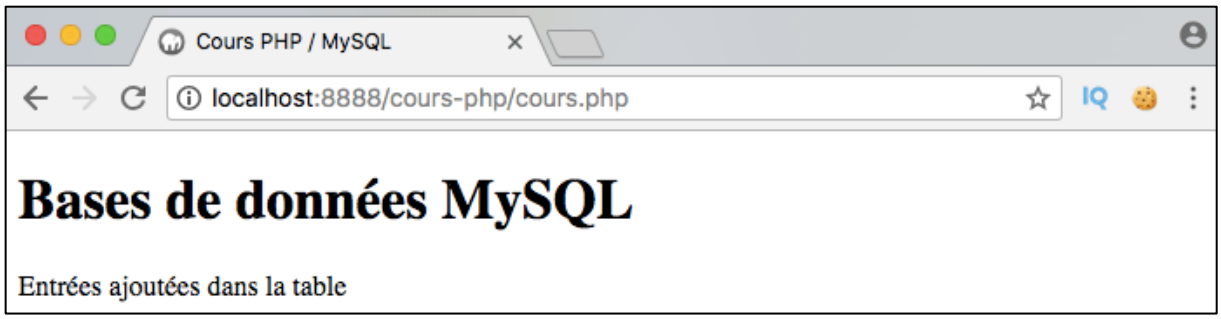


Insérer plusieurs entrées dans une table

Nous allons pouvoir insérer plusieurs entrées d'un coup dans une table de différentes façons en PDO.

Nous allons déjà tout simplement pouvoir réutiliser l'écriture précédente en la répétant plusieurs fois.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>

    <?php
      $servname = "localhost";
      $dbname = "pdodb";
      $user = "root";
      $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sql1 = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
        VALUES ('Durand','Victor','Rue des Acacias','Brest',29200,'France','v.durand@gmail.com')";
        $dbco->exec($sql1);

        $sql2 = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
        VALUES ('Julia','Joly','Rue du Hameau','Lyon',69001,'France','july@gmail.com')";
        $dbco->exec($sql2);

        echo "Entrées ajoutées dans la table";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22

L'un des gros défauts de cette méthode est que s'il y a un problème d'exécution en cours du script, certaines entrées vont être insérées et pas d'autres et certaines entrées pourraient ne pas avoir toutes leurs données insérées.

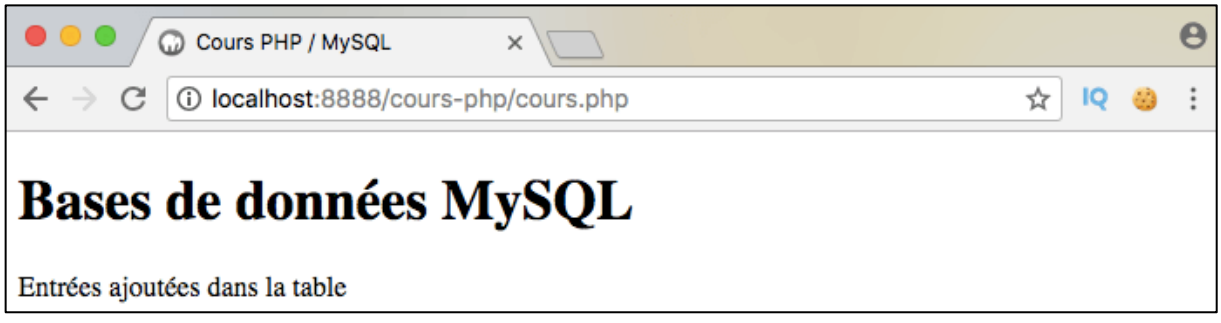
Pour éviter cela, nous pouvons ajouter les méthodes `beginTransaction()`, `commit()` et `rollBack()` dans notre code.

La méthode `beginTransaction()` permet de démarrer ce qu'on appelle une transaction et de désactiver le mode `autocommit`. Concrètement, cela signifie que toutes les manipulations faites sur la base de données ne seront pas appliquées tant qu'on ne mettra pas fin à la transaction en appelant `commit()`.

La méthode `commit()` sert donc à valider une transaction, c'est-à-dire à valider l'application d'une ou d'un ensemble de requêtes SQL. Cette méthode va aussi replacer la connexion en mode `autocommit`.

La méthode `rollBack()` sert à annuler une transaction si l'on s'aperçoit d'une erreur. Cette méthode restaure le mode `autocommit` après son exécution.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>

    <?php
      $servname = "localhost";
      $dbname = "pdodb";
      $user = "root";
      $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $dbco->beginTransaction();

        $sql1 = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
        VALUES ('Doe','John','Rue des Lys','Nantes',44000,'France','j.doe@gmail.com')";
        $dbco->exec($sql1);

        $sql2 = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
        VALUES ('Dupont','Jean','Bvd Original','Bordeaux',33000,'France','jd@gmail.com')";
        $dbco->exec($sql2);

        $dbco->commit();
        echo "Entrées ajoutées dans la table";
      }

      catch(PDOException $e){
        $dbco->rollBack();
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22
6	Doe	John	Rue des Lys	Nantes	44000	France	j.doe@gmail.com	2018-06-02 09:04:46
7	Dupont	Jean	Bvd Original	Bordeaux	33000	France	jd@gmail.com	2018-06-02 09:04:46

Dans le code ci-dessus, on commence par désactiver l'`autocommit` avec `beginTransaction()`.

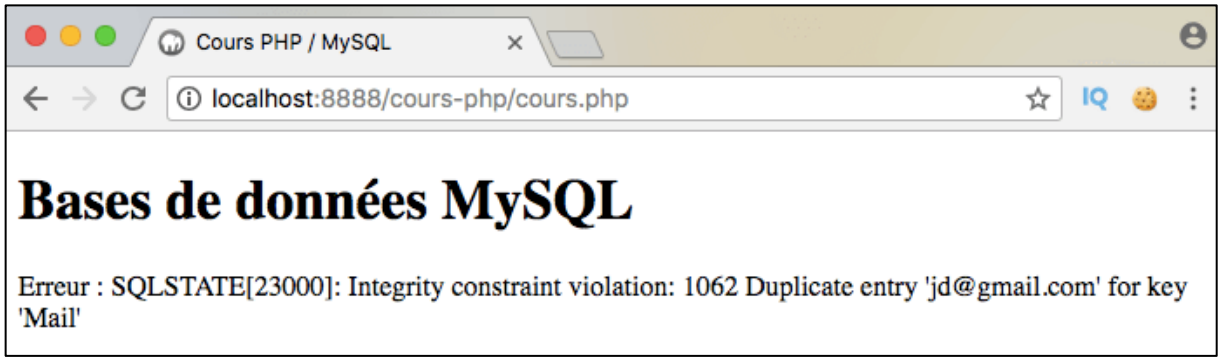
Si aucune erreur n'est détectée, la méthode `commit()` s'exécute après nos requêtes SQL, valide donc les transactions et replace la connexion en mode `autocommit`.

Note : Ici, les deux dernières entrées dans ma table possèdent les id 6 et 7 et non pas 4 et 5 tout simplement car j'ai effectué un petit test entre temps et que j'ai supprimé les deux lignes portant les id 4 et 5 que j'avais inséré durant ce test.

Si en revanche une exception est lancée, alors la méthode `rollBack()` s'exécute et annule toutes les transactions avant de restaurer le mode `autocommit`.

Par exemple, si je tente de ré-exécuter mon code en modifiant la valeur « mail » de la première entrée mais pas celle de la seconde, une erreur va être lancée car j'ai une contrainte `UNIQUE` sur le champ Mail de ma table. La méthode `rollBack()` va donc s'exécuter et aucune transaction ne va être validée.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>

    <?php
      $servname = "localhost";
      $dbname = "pdodb";
      $user = "root";
      $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $dbco->beginTransaction();

        $sql1 = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
        VALUES ('Doe','John','Rue des Lys','Nantes',44000,'France','mod@gmail.com')";
        $dbco->exec($sql1);

        $sql2 = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
        VALUES ('Dupont','Jean','Bvd Original','Bordeaux',33000,'France','jd@gmail.com')";
        $dbco->exec($sql2);

        $dbco->commit();
        echo "Entrées ajoutées dans la table";
      }

      catch(PDOException $e){
        $dbco->rollBack();
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22
6	Doe	John	Rue des Lys	Nantes	44000	France	j.doe@gmail.com	2018-06-02 09:04:46
7	Dupont	Jean	Bvd Original	Bordeaux	33000	France	jd@gmail.com	2018-06-02 09:04:46

Cette deuxième façon de procéder est déjà meilleure que la première. Cependant, en pratique, nous utiliserons plutôt les requêtes préparées pour insérer plusieurs entrées d'un coup dans nos tables, notamment lorsque les données seront fournies par les utilisateurs.

Les requêtes préparées

Les requêtes préparées correspondent à une façon de créer et d'exécuter nos requêtes selon trois étapes : une étape de préparation, une étape de compilation et finalement une dernière étape d'exécution.

Préparer ses requêtes comporte des avantages notables notamment dans le cas où l'on doit exécuter un grand nombre de fois une même requête ou si l'on doit insérer des données envoyées par les utilisateurs.

Préparer ses requêtes : comment ça marche ?

Si vous devez exécuter des requêtes similaires plusieurs fois d'affilée, il va alors être très intéressant d'utiliser ce qu'on appelle des requêtes préparées.

Les requêtes préparées sont des requêtes qui vont être créées en trois temps : la préparation, la compilation et l'exécution.

Tout d'abord, une première phase de préparation dans laquelle nous allons créer un template ou schéma de requête, en ne précisant pas les valeurs réelles dans notre requête mais en utilisant plutôt des marqueurs nommés (sous le forme `:nom`) ou des marqueurs interrogatifs (sous la forme `?`).

Ces marqueurs nommés ou interrogatifs (qu'on peut plus globalement nommer marqueurs de paramètres) vont ensuite être remplacés par les vraies valeurs lors de l'exécution de la requête. Notez que vous ne pouvez pas utiliser les marqueurs nommés et les marqueurs interrogatifs dans une même requête SQL, il faudra choisir l'un ou l'autre.

Une fois le template créé, la base de données va analyser, compiler, faire des optimisations sur notre template de requête SQL et va stocker le résultat sans l'exécuter.

Finalement, nous allons lier des valeurs à nos marqueurs et la base de données va exécuter la requête. Nous allons pouvoir réutiliser notre template autant de fois que l'on souhaite en liant de nouvelles valeurs à chaque fois.

Utiliser des requêtes préparées va nous offrir deux principaux avantages par rapport à l'exécution directe de requêtes SQL :

- Nous allons gagner en performance puisque la préparation de nos requêtes ne va être faite qu'une fois quel que soit le nombre d'exécutions de notre requête ;
- Le risque d'injection SQL est minimisé puisque notre requête est pré-formatée et nous n'avons donc pas besoin de protéger nos paramètres ou valeurs manuellement.

Un premier point sur l'injection SQL

Jusqu'à présent, nous avons fourni nous-mêmes les valeurs à insérer en base de données. Cependant, en pratique, nous allons très souvent stocker et manipuler des données envoyées directement par les utilisateurs.

Le gros souci par rapport aux données envoyées par les utilisateurs est que vous devez toujours vous en méfier : vous n'êtes jamais à l'abri d'une étourderie ou d'un comportement volontairement malfaisant.

Jusqu'ici, nos requêtes n'étaient pas du tout protégées contre ce type de comportements. Pour bien comprendre cela, imaginez que vous récupériez les valeurs à insérer dans notre table Clients créée précédemment à partir d'un formulaire.

Vous demandez donc aux utilisateurs de rentrer leur nom, prénom, adresse, etc. Sans plus de vérification, rien n'empêche un utilisateur d'envoyer une valeur qui va soit faire planter notre script soit éventuellement altérer notre base de données.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>

    <?php
      $servname = "localhost";
      $dbname = "pdodb";
      $user = "root";
      $pass = "root";

      /*Imaginons que l'on récupère les valeurs de ces variables à partir
       *d'un formulaire rempli par nos utilisateurs
       */
      $nom = "Richard";
      $prenom = "Pierre";
      $adresse = "Rue de la chèvre";
      $ville = "Toulon";
      $cp = "83000";
      $pays = "France";
      $mail ="gg@gmail.com'),('a','b','c','d',1,'e','f";


      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sql = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
        VALUES ('$nom','$prenom','$adresse','$ville',$cp,'$pays','$mail')";
        $dbco->exec($sql);

        echo "Entrée ajoutée dans la table";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22
6	Doe	John	Rue des Lys	Nantes	44000	France	j.doe@gmail.com	2018-06-02 09:04:46
7	Dupont	Jean	Bvd Original	Bordeaux	33000	France	jd@gmail.com	2018-06-02 09:04:46
10	Richard	Pierre	Rue Jean Aicard	Toulon	83000	France	gg@gmail.com	2018-06-02 10:00:21
11	a	b	c	d	1	e	f	2018-06-02 10:00:21

Ici, nous avons précisé une valeur habile dans notre variable `$mail` qui nous a permis d'insérer deux entrées d'un coup en utilisant une syntaxe non recommandée mais qui fonctionne toujours qui précise les différentes valeurs des entrées à insérer en séparant les groupes par des virgules.

En effet, regardons plus attentivement ce que ça donne lorsqu'on remplace notre variable par son contenu dans le code ci-dessus.

```
$sql = "INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
VALUES ('$nom','$prenom','$adresse','$ville',$cp,'$pays','gg@gmail.com'),
       ('a','b','c','d',1,'e','f')";
$dbco->exec($sql);
```

Dans ce cas-là, deux nouvelles entrées vont être insérées dans notre table. Ici, nous restons dans la catégorie des cas « gênants » mais non dangereux.

Cependant, rien n'aurait empêché cet utilisateur d'insérer une autre commande SQL pour lire le contenu de notre base de données, la modifier ou encore la supprimer !

Pour cette raison, vous devez une nouvelle fois faire toujours très attention dès que vous recevez des données utilisateur et ajouter différents niveaux de sécurité (sécuriser ses formulaires en utilisant les regex, neutraliser les injections en PHP, préparer ses requêtes, etc.).

Les méthodes execute(), bindParam() et bindValue()

Pour exécuter une requête préparée, nous allons cette fois-ci devoir utiliser la méthode `execute()` et non plus `exec()` comme on utilisait depuis le début de cette partie.
En utilisant des marqueurs dans nos requêtes préparées, nous allons avoir deux grandes options pour exécuter la méthode `execute()` :

- On va pouvoir lui passer un tableau de valeurs de paramètres (uniquement en entrée) ;
- On va pouvoir d'abord appeler les méthodes `bindParam()` ou `bindValue()` pour respectivement lier des variables ou des valeurs à nos marqueurs puis ensuite exécuter `execute()`.

Pas d'inquiétude, je vous explique immédiatement les différences concrètes entre ces méthodes et les cas d'utilisation !

Commencez déjà par noter que passer un tableau directement en valeur de `execute()` devrait être considéré comme la méthode par défaut puisque c'est finalement la plus simple et que tout va fonctionner normalement dans l'immense majorité des cas.

En fait, `execute(array)` est une méthode d'écriture raccourcie ; l'idée derrière cela est qu'une boucle va être exécutée en tâche de fond dont l'objet va être d'appeler `bindValue()` sur chacun des éléments du tableau.

En utilisant `execute(array)`, les valeurs vont être liées en tant que type String excepté pour le type NULL qui restera inchangé. Cela va fonctionner une nouvelle fois dans l'immense majorité des cas.

Cependant, dans de rares cas, il sera utile de définir explicitement le type de données. Les cas les plus fréquents sont les suivants :

- Notre requête contient une clause LIMIT ou toute autre clause qui ne peut pas accepter une valeur de type String et le mode émulation est activé (ON) ;
- Notre table contient des colonnes avec un type particulier qui n'accepte que des valeurs d'un certain type (comme les colonnes de type `BOOLEAN` ou `BIGINT` par exemple).

Dans ces cas-là, il sera alors préférable de lier les variables avant d'utiliser `execute()` si vous voulez avoir le résultat attendu.

Quelle différence maintenant entre `bindParam()` et `bindValue()` ?

La méthode `bindParam()` va lier un paramètre à un nom de variable spécifique et la variable va être liée en tant que référence et ne sera évaluée qu'au moment de l'appel à la méthode `execute()`.

Si la variable change de valeur entre l'appel à la méthode `bindParam()` et l'appel à la méthode `execute()`, c'est donc la dernière valeur qui sera utilisée.

La méthode `bindValue()` va elle associer directement une valeur à un paramètre.

La méthode `bindParam()` fonctionne avec deux paramètres obligatoires et trois facultatifs dont un qui va particulièrement nous intéresser :

- Un identifiant (obligatoire) qui sera de la forme :nom si on utilise des marqueurs nommés ou qui sera l'index de base 1 du paramètre si on utilise un marqueur interrogatif ;
- Le nom de la variable PHP (obligatoire) à lier au paramètre de la requête SQL ;
- Le type de données explicite pour le paramètre (facultatif) spécifié en utilisant les constantes PDO::PARAM_*constants.

Les constantes prédéfinies les plus utilisées sont les suivantes :

- PDO ::PARAM_STR, qui représente le type de données CHAR, VARCHAR et les autres types de données « chaine de caractères » SQL ;
- PDO ::PARAM_INT, qui représente le type de données SQL INTEGER (nombre entier) ;
- PDO ::PARAM_NULL, qui représente le type de données SQL NULL ;
- PDO ::PARAM_BOOL, qui représente le type de données booléen.

Pour la liste complète des constantes, vous pouvez consulter la documentation officielle ici : http://php.net/manual/fr/pdo.constants.php.

La méthode bindValue() va fonctionner également avec deux paramètres obligatoires et un facultatif :

- Un identifiant (obligatoire) qui sera de la forme :nom si on utilise des marqueurs nommés ou qui sera l'index de base 1 du paramètre si on utilise un marqueur interrogatif ;
- La valeur à associer au paramètre (obligatoire) ;
- Le type de données explicite pour le paramètre (facultatif) spécifié en utilisant les constantes PDO::PARAM_*constants.

Notez que les méthodes execute() bindParam() et bindValue() appartiennent toutes les trois à la classe PDOStatement et non pas à la classe PDO. La classe PDOStatement représente une requête préparée et, une fois exécutée, l'ensemble des résultats associés.

Exemples pratiques de requêtes préparées

Reprenons notre table Clients et tentons d'insérer de nouvelles entrées en utilisant les requêtes préparées. Nous allons passer en revue les différentes façons de faire expliquées précédemment.

Avec execute(array) et des marqueurs nommés

Commençons déjà en préparant une requête avec des marqueurs nommés puis en l'exécutant avec execute(array) :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $nom = "Flo";
        $prenom = "Dechand";
        $adresse = "Rue des Moulins";
        $ville = "Marseille";
        $cp = 13001;
        $pays = "France";
        $mail = "flodc@gmail.com";

        //$sth appartient à la classe PDOStatement
        $sth = $dbco->prepare("
            INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
            VALUES (:nom, :prenom, :adresse, :ville, :cp, :pays, :mail)
        ");
        $sth->execute(array(
                            ':nom' => $nom,
                            ':prenom' => $prenom,
                            ':adresse' => $adresse,
                            ':ville' => $ville,
                            ':cp' => $cp,
                            ':pays' => $pays,
                            ':mail' => $mail));
        echo "Entrée ajoutée dans la table";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22
6	Doe	John	Rue des Lys	Nantes	44000	France	j.doe@gmail.com	2018-06-02 09:04:46
7	Dupont	Jean	Bvd Original	Bordeaux	33000	France	jd@gmail.com	2018-06-02 09:04:46
10	Richard	Pierre	Rue Jean Aicard	Toulon	83000	France	gg@gmail.com	2018-06-02 10:00:21
11	a	b	c	d	1	e	f	2018-06-02 10:00:21
12	Flo	Dechand	Rue des Moulins	Marseille	13001	France	flodc@gmail.com	2018-06-02 15:17:41

Ici, on commence par préparer notre requête SQL grâce à la méthode `prepare()` qui appartient à la classe `PDOStatement`. On place le résultat dans un objet `$sth`.

Notez qu'on remplace bien les valeurs dans notre requête SQL par nos marqueurs nommés (sans les entourer d'apostrophes).

On appelle ensuite `execute()` en lui passant un tableau composé de nos marqueurs et des variables associées.

Avec execute(array) et des marqueurs interrogatifs

Le principe va être relativement similaire à ce que l'on vient de faire, à part que nous allons cette fois-ci utiliser des marqueurs interrogatifs :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $nom = "Tom";
        $prenom = "Dubois";
        $adresse = "Rue du Chene";
        $ville = "Nice";
        $cp = 06000;
        $pays = "France";
        $mail = "duboistom@gmail.com";

        //$sth appartient à la classe PDOStatement
        $sth = $dbco->prepare("
            INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
            VALUES (?, ?, ?, ?, ?, ?, ?)
        ");
        $sth->execute(array($nom, $prenom, $adresse, $ville, $cp, $pays, $mail));
        echo "Entrée ajoutée dans la table";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22
6	Doe	John	Rue des Lys	Nantes	44000	France	j.doe@gmail.com	2018-06-02 09:04:46
7	Dupont	Jean	Bvd Original	Bordeaux	33000	France	jd@gmail.com	2018-06-02 09:04:46
10	Richard	Pierre	Rue Jean Aicard	Toulon	83000	France	gg@gmail.com	2018-06-02 10:00:21
11	a	b	c	d	1	e	f	2018-06-02 10:00:21
12	Flo	Dechand	Rue des Moulins	Marseille	13001	France	flodc@gmail.com	2018-06-02 15:17:41
14	Tom	Dubois	Rue du Chene	Nice	6000	France	duboistom@gmail.com	2018-06-02 15:24:00

En utilisant bindValue et des marqueurs nommés

Nous allons cette fois-ci associer des valeurs à des paramètres en utilisant `bindValue()` et des marqueurs nommés :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $nom = "Laura";
        $prenom = "Dubois";
        $adresse = "Rue du Chene";
        $ville = "Nice";
        $cp = 06000;
        $pays = "France";
        $mail = "lauradb@gmail.com";

        //$sth appartient à la classe PDOStatement
        $sth = $dbco->prepare("
            INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
            VALUES (:nom, :prenom, :adresse, :ville, :cp, :pays, :mail)
        ");
        //La constante de type par défaut est STR
        $sth->bindValue(':nom', $nom);
        $sth->bindValue(':prenom', $prenom);
        $sth->bindValue(':adresse', $adresse);
        $sth->bindValue(':ville', $ville);
        $sth->bindValue(':cp', $cp, PDO::PARAM_INT);
        $sth->bindValue(':pays', $pays);
        $sth->bindValue(':mail', $mail);
        $sth->execute();
        echo "Entrée ajoutée dans la table";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22
6	Doe	John	Rue des Lys	Nantes	44000	France	j.doe@gmail.com	2018-06-02 09:04:46
7	Dupont	Jean	Bvd Original	Bordeaux	33000	France	jd@gmail.com	2018-06-02 09:04:46
10	Richard	Pierre	Rue Jean Aicard	Toulon	83000	France	gg@gmail.com	2018-06-02 10:00:21
11	a	b	c	d	1	e	f	2018-06-02 10:00:21
12	Flo	Dechand	Rue des Moulins	Marseille	13001	France	flodc@gmail.com	2018-06-02 15:17:41
14	Tom	Dubois	Rue du Chene	Nice	6000	France	duboistom@gmail.com	2018-06-02 15:24:00
16	Laura	Dubois	Rue du Chene	Nice	6000	France	lauradb@gmail.com	2018-06-02 15:43:15

On ajoute une étape ici en utilisant `bindValue()` pour lier des valeurs à des paramètres. Une fois toutes les valeurs liées, nous n'avons plus qu'à appeler `execute()` pour exécuter notre requête.

En utilisant bindValue et des marqueurs interrogatifs

Même principe que précédemment mais cette fois-ci avec des marqueurs interrogatifs :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $nom = "Mathilde";
        $prenom = "Palaz";
        $adresse = "Rue des Cerisiers";
        $ville = "Rouen";
        $cp = 76000;
        $pays = "France";
        $mail = "mathplz@gmail.com";

        //$sth appartient à la classe PDOStatement
        $sth = $dbco->prepare("
            INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
            VALUES (?, ?, ?, ?, ?, ?, ?)
        ");
        //La constante de type par défaut est STR
        $sth->bindValue(1, $nom);
        $sth->bindValue(2, $prenom);
        $sth->bindValue(3, $adresse);
        $sth->bindValue(4, $ville);
        $sth->bindValue(5, $cp, PDO::PARAM_INT);
        $sth->bindValue(6, $pays);
        $sth->bindValue(7, $mail);
        $sth->execute();
        echo "Entrée ajoutée dans la table";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22
6	Doe	John	Rue des Lys	Nantes	44000	France	j.doe@gmail.com	2018-06-02 09:04:46
7	Dupont	Jean	Bvd Original	Bordeaux	33000	France	jd@gmail.com	2018-06-02 09:04:46
10	Richard	Pierre	Rue Jean Aicard	Toulon	83000	France	gg@gmail.com	2018-06-02 10:00:21
11	a	b	c	d	1	e	f	2018-06-02 10:00:21
12	Flo	Dechand	Rue des Moulins	Marseille	13001	France	flodc@gmail.com	2018-06-02 15:17:41
14	Tom	Dubois	Rue du Chene	Nice	6000	France	duboistom@gmail.com	2018-06-02 15:24:00
16	Laura	Dubois	Rue du Chene	Nice	6000	France	lauradb@gmail.com	2018-06-02 15:43:15
17	Mathilde	Palaz	Rue des Cerisiers	Rouen	76000	France	mathplz@gmail.com	2018-06-02 15:52:14

Notez qu'on précise cette fois ci l'index de base 1 du paramètre dans notre méthode `bindValue()`.

En utilisant bindParam et des marqueurs nommés ou interrogatifs

On peut finalement lier nos variables à des marqueurs avec `bindParam()` dans nos requêtes préparées.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $nom = "Jean";
        $prenom = "Bombeur";
        $adresse = "Rue des Bouchers";
        $ville = "Toulouse";
        $cp = 31000;
        $pays = "France";
        $mail = "jbb@gmail.com";

        $sth = $dbco->prepare("
            INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
            VALUES (:nom, :prenom, :adresse, :ville, :cp, :pays, :mail)
        ");
        //La constante de type par défaut est STR
        $sth->bindParam(':nom', $nom);
        $sth->bindParam(':prenom', $prenom);
        $sth->bindParam(':adresse', $adresse);
        $sth->bindParam(':ville', $ville);
        $sth->bindParam(':cp', $cp, PDO::PARAM_INT);
        $sth->bindParam(':pays', $pays);
        $sth->bindParam(':mail', $mail);
        $cp = 31001;
        $sth->execute();
        echo "Entrée ajoutée dans la table";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22
6	Doe	John	Rue des Lys	Nantes	44000	France	j.doe@gmail.com	2018-06-02 09:04:46
7	Dupont	Jean	Bvd Original	Bordeaux	33000	France	jd@gmail.com	2018-06-02 09:04:46
10	Richard	Pierre	Rue Jean Aicard	Toulon	83000	France	gg@gmail.com	2018-06-02 10:00:21
11	a	b	c	d	1	e	f	2018-06-02 10:00:21
12	Flo	Dechand	Rue des Moulins	Marseille	13001	France	flodc@gmail.com	2018-06-02 15:17:41
14	Tom	Dubois	Rue du Chene	Nice	6000	France	duboistom@gmail.com	2018-06-02 15:24:00
16	Laura	Dubois	Rue du Chene	Nice	6000	France	lauradb@gmail.com	2018-06-02 15:43:15
17	Mathilde	Palaz	Rue des Cerisiers	Rouen	76000	France	mathplz@gmail.com	2018-06-02 15:52:14
18	Jean	Bombeur	Rue des Bouchers	Toulouse	31001	France	jbb@gmail.com	2018-06-02 16:04:41

Ici, on utilise des marqueurs nommés. Notez bien une nouvelle fois que les variables sont ici liées en tant que références et ne sont évaluées qu'au moment où on appelle `execute()`.

Ainsi, si on modifie la valeur stockée dans une variable entre le moment où on appelle `bindParam()` et celui où on appelle `execute()`, c'est bien la dernière valeur qui va être retenue au contraire de si on utilisait `bindValue()`.

On peut également utiliser `bindParam` avec des marqueurs interrogatifs :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $nom = "Gérard";
        $prenom = "Philippe";
        $adresse = "Impasse des sans Noms";
        $ville = "Nantes";
        $cp = 44000;
        $pays = "France";
        $mail = "philou@gmail.com";

        $sth = $dbco->prepare("
            INSERT INTO Clients(Nom,Prenom,Adresse,Ville,Codepostal,Pays,Mail)
            VALUES (?, ?, ?, ?, ?, ?, ?)
        ");
        $sth->bindParam(1, $nom);
        $sth->bindParam(2, $prenom);
        $sth->bindParam(3, $adresse);
        $sth->bindParam(4, $ville);
        $sth->bindParam(5, $cp, PDO::PARAM_INT);
        $sth->bindParam(6, $pays);
        $sth->bindParam(7, $mail);
        $sth->execute();
        echo "Entrée ajoutée dans la table";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Id	Nom	Prenom	Adresse	Ville	Codepostal	Pays	Mail	DateInscription
1	Giraud	Pierre	Quai d'Europe	Toulon	83000	France	pierre.giraud@edhec.com	2018-05-30 23:19:13
2	Durand	Victor	Rue des Acacias	Brest	29200	France	v.durand@gmail.com	2018-06-02 08:54:22
3	Julia	Joly	Rue du Hameau	Lyon	69001	France	july@gmail.com	2018-06-02 08:54:22
6	Doe	John	Rue des Lys	Nantes	44000	France	j.doe@gmail.com	2018-06-02 09:04:46
7	Dupont	Jean	Bvd Original	Bordeaux	33000	France	jd@gmail.com	2018-06-02 09:04:46
10	Richard	Pierre	Rue Jean Aicard	Toulon	83000	France	gg@gmail.com	2018-06-02 10:00:21
11	a	b	c	d	1	e	f	2018-06-02 10:00:21
12	Flo	Dechand	Rue des Moulins	Marseille	13001	France	flodc@gmail.com	2018-06-02 15:17:41
14	Tom	Dubois	Rue du Chene	Nice	6000	France	duboistom@gmail.com	2018-06-02 15:24:00
16	Laura	Dubois	Rue du Chene	Nice	6000	France	lauradb@gmail.com	2018-06-02 15:43:15
17	Mathilde	Palaz	Rue des Cerisiers	Rouen	76000	France	mathplz@gmail.com	2018-06-02 15:52:14
18	Jean	Bombeur	Rue des Bouchers	Toulouse	31001	France	jbb@gmail.com	2018-06-02 16:04:41
19	Gérard	Philippe	Impasse des sans Noms	Nantes	44000	France	philou@gmail.com	2018-06-02 16:10:24

Voilà tout pour les requêtes préparées ! Si ces nouvelles choses vous semblent floues, prenez le temps de relire ce chapitre et de refaire les exemples.

Dans tous les cas, nous allons beaucoup nous resservir des requêtes préparées par la suite donc vous devriez les assimiler et les maitriser rapidement.

Modifier les données ou la structure d'une table MySQL

Dans de nombreux cas, nous devrons mettre à jour les données dans nos bases de données. Ce sera par exemple le cas lorsqu'un utilisateur va mettre à jour une adresse de livraison ou une adresse mail, ou encore lorsqu'une valeur de type date doit être remplacée régulièrement comme une date d'expiration ou la date de dernier achat d'un client.

Il va être beaucoup plus rare d'intervenir directement sur la structure d'une table qui devrait normalement être fixe mais cependant cela peut arriver dans des cas de refonte ou de mise en conformité avec de nouvelles fonctionnalités, lois, etc.

Nous allons apprendre à faire tout cela dans cette nouvelle leçon !

Mettre à jour des données dans une table

Nous allons utiliser l'instruction SQL `UPDATE` suivie du nom de la table pour mettre à jour des données dans une table.

Cette instruction va toujours être accompagnée de `SET` qui va nous servir à préciser la colonne à mettre à jour ainsi que la nouvelle valeur pour la colonne.

En s'arrêtant là, en effet, nous allons mettre à jour toutes les valeurs d'une colonne d'un coup ! Ce sera très rarement ce que nous voudrons faire en pratique, et c'est pour cela que nous allons généralement également utiliser la clause `WHERE` pour spécifier quelles entrées doivent être mises à jour.

Prenons immédiatement pour voir en pratique comment nous allons pouvoir mettre à jour des données dans une table en utilisant PDO.

Pour cet exemple, je vais cette fois-ci m'appuyer sur une table nommée « Users » qui appartient à ma base de données « pdodb » et contient 4 colonnes :

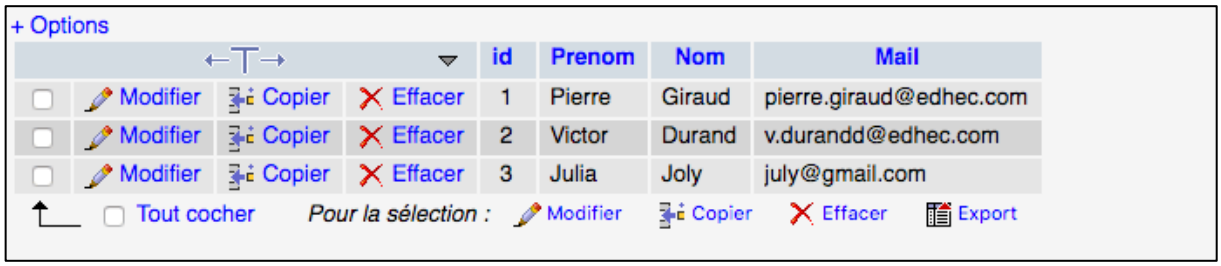
- Une colonne « Id », type INT, UNISGNED, PRIMARY KEY, AUTO_INCREMENT
- Une colonne « Prenom », type VARCHAR(30) NOT NULL
- Une colonne « Nom », type VARCHAR(30) NOT NULL
- Une colonne « Mail », type VARCHAR(30) NOT NULL

Nous allons pour le moment nous contenter d'ajouter 3 entrées dans cette table.

```php
<?php
  $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

  try{
    $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
    $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    //Crée la table Users
    $sql = "CREATE TABLE Users (
      id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      Prenom VARCHAR(30) NOT NULL,
      Nom VARCHAR(30) NOT NULL,
      Mail VARCHAR(50) NOT NULL
    )";
    $dbco->exec($sql);

    //On prépare la requête et on lie les paramètres
    $sth = $dbco->prepare("
      INSERT INTO Users (Prenom, Nom, Mail)
      VALUES (:prenom, :nom, :mail)
    ");
    $sth->bindParam(':prenom', $prenom);
    $sth->bindParam(':nom', $nom);
    $sh->bindParam(':mail', $mail);

    //Insère une première entrée
    $prenom = "Pierre"; $nom = "Giraud"; $mail = "pierre.giraud@edhec.com";
    $sth->execute();

    //Insère une deuxième entrée
    $prenom = "Victor"; $nom = "Durand"; $mail = "v.durandd@edhec.com";
    $sth->execute();

    //Insère une troisième entrée
    $prenom = "Julia"; $nom = "Joly"; $mail = "july@gmail.com";
    $sth->execute();

    echo "Parfait, tout s'est bien passé";
  }

  catch(PDOException $e){
    echo "Erreur : " . $e->getMessage();
  }
?>
```

Et voici ce que vous devriez donc avoir en visualisant la structure et le contenu de votre table via phpMyAdmin :

Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
id	int(6)		UNSIGNED	Non	*Aucune*		AUTO_INCREMENT
Prenom	varchar(30)	utf8_general_ci		Non	*Aucune*		
Nom	varchar(30)	utf8_general_ci		Non	*Aucune*		
Mail	varchar(50)	utf8_general_ci		Non	*Aucune*		



On s'aperçoit qu'il y a un « d » en trop dans l'adresse mail de notre utilisateur « Victor Durand », utilisons donc `UPDATE` pour `SET` une nouvelle valeur pour la colonne mail de cet utilisateur.

Pour ne mettre à jour que la valeur du mail correspondant à cette entrée, nous allons également utiliser `WHERE` en donnant une condition sur l'id.

Bon à savoir : Nous ne sommes pas obligés d'utiliser la clause `WHERE` sur la colonne « id », nous pouvons tout aussi bien donner une condition sur n'importe quelle autre colonne. Cependant, en pratique, nous nous appuierons très souvent sur cette fameuse colonne « id » car c'est un moyen simple et infaillible d'isoler une entrée en particulier.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //On prépare la requête et on l'exécute
        $sth = $dbco->prepare("
          UPDATE Users
          SET mail='v.durand@edhec.com'
          WHERE id=2
        ");
        $sth->execute();

        //On affiche le nombre d'entrées mise à jour
        $count = $sth->rowCount();
        print('Mise à jour de ' .$count. ' entrée(s)');
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Le script ci-dessus est assez transparent : on prépare notre requête pour mettre à jour l'adresse mail de l'utilisateur portant l'id 2 dans notre table puis on exécute cette requête.

On utilise ensuite la méthode `rowCount()` pour obtenir le nombre d'entrées affectées par notre dernière requête. En effet, `rowCount()` retourne le nombre de lignes affectées par la dernière requête `DELETE`, `INSERT` ou `UPDATE` exécutée par l'objet de la classe `PDOStatement` correspondant (en l'occurrence ici `$sth`.

On peut également aller vérifier dans phpMyAdmin que notre la valeur mail de notre entrée à bien été mise à jour.

Modifier la structure d'une table

Pour modifier la structure d'une table en soi, nous allons utiliser l'instruction SQL `ALTER TABLE`.

Cette commande va nous permettre d'ajouter, de supprimer ou de modifier une colonne dans une table.

Ajouter une colonne dans une table

Pour ajouter une colonne, nous allons également devoir utiliser `ADD` avec le nom de la colonne à ajouter et le type de données attendu.

Par exemple, on pourrait ajouter une colonne « DateInscription » dans notre table Users qui stockerait automatiquement la date d'inscription de nos utilisateurs.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Le changement de structure d'une table ne dépendra jamais des
         *utilisateurs, pas la peine donc d'utiliser de requête préparée*/
        $sql = "
          ALTER TABLE Users
          ADD DateInscription TIMESTAMP
        ";

        $dbco->exec($sql);
        echo 'Colonne ajoutée';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

id	Prenom	Nom	Mail	DateInscription
1	Pierre	Giraud	pierre.giraud@edhec.com	2018-06-03 11:14:16
2	Victor	Durand	v.durand@edhec.com	2018-06-03 11:14:16
3	Julia	Joly	july@gmail.com	2018-06-03 11:14:16

Notez que sans plus d'informations, les entrées déjà présentes dans la table vont recevoir la date correspondant à la création de la colonne.

Supprimer une colonne dans une table

Pour maintenant supprimer une colonne dans une table, nous allons cette fois-ci utiliser `ALTER TABLE` de concert avec l'instruction `DROP COLUMN`.

Attention ici : à la différence de l'instruction SQL `ADD` + nom de colonne pour ajouter une colonne, il faut bien pour supprimer une colonne utiliser l'instruction `DROP COLUMN` + le nom de la colonne.

En revanche, il n'y a bien évidemment pas besoin de préciser le type de données de la colonne lorsqu'on souhaite la supprimer.

On peut par exemple essayer de supprimer la colonne « DateInscription » que nous venons juste de créer dans notre table Users.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Le changement de structure d'une table ne dépendra jamais des
         *utilisateurs, pas la peine donc d'utiliser de requête préparée*/
        $sql = "
          ALTER TABLE Users
          DROP COLUMN DateInscription
        ";

        $dbco->exec($sql);
        echo 'Colonne supprimée';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

id	Prenom	Nom	Mail
1	Pierre	Giraud	pierre.giraud@edhec.com
2	Victor	Durand	v.durand@edhec.com
3	Julia	Joly	july@gmail.com

Modifier une colonne dans une table

Pour finalement modifier le type de donnée d'une colonne dans une table, il faudra utiliser `ALTER TABLE` avec l'instruction `MODIFY COLUMN` si vous évoluez dans un environnement MySQL (la syntaxe de cette commande n'est pas encore standardisée et peut changer selon le système de bases de données utilisé).

Notez ici que s'il y a incompatibilité entre le type de données de départ et le nouveau type de données que doit accepter la colonne, votre table peut se trouver totalement dégradée ou corrompue.

Il faut donc faire très attention lors de la modification d'une colonne. En pratique, cette opération n'est pas recommandée et on ne l'utilisera qu'en dernier recours sauf cas particulier, par exemple pour « étendre » un type de données qu'une colonne peut accepter (passer d'un texte court à un texte long, d'un petit nombre à un nombre plus grand, etc.).

Essayons par exemple de modifier la colonne « Prenom » de notre table « Users » pour qu'elle puisse accepter des valeurs allant jusqu'à 50 caractères (pour le moment celle-ci est limitée à 30).

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Le changement de structure d'une table ne dépendra jamais des
         *utilisateurs, pas la peine donc d'utiliser de requête préparée*/
        $sql = "
          ALTER TABLE Users
          MODIFY COLUMN Prenom VARCHAR(50)
        ";

        $dbco->exec($sql);
        echo 'Colonne mise à jour';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	**id**	int(6)		UNSIGNED	Non	*Aucune*		AUTO_INCREMENT
2	**Prenom**	varchar(50)	utf8_general_ci		Oui	*NULL*		
3	**Nom**	varchar(30)	utf8_general_ci		Non	*Aucune*		
4	**Mail**	varchar(50)	utf8_general_ci		Non	*Aucune*		

Supprimer des données, une table ou une base

Dans cette nouvelle leçon, nous allons apprendre à supprimer des données précises d'une table ou toutes les données d'une table ou encore à supprimer complètement une table ou une base de données.

De manière pratique, il est essentiel de savoir comment supprimer d'une table, au cas où un utilisateur voudrait faire jouer son droit à l'effacement de ses données personnelles par exemple.

Il nous arrivera cependant beaucoup moins souvent d'avoir à supprimer une table ou une base de données. Il est toutefois bon de savoir le faire dans le cas d'une refonte d'un site entre autres.

Supprimer des données d'une table

Supprimer une ou plusieurs entrées choisies d'une table

Pour supprimer des données d'une table, nous allons utiliser l'instruction SQL `DELETE FROM`.

Pour préciser quelles entrées doivent être supprimées, nous allons accompagner `DELETE FROM` d'une clause `WHERE` nous permettant de cibler des données en particulier dans notre table.

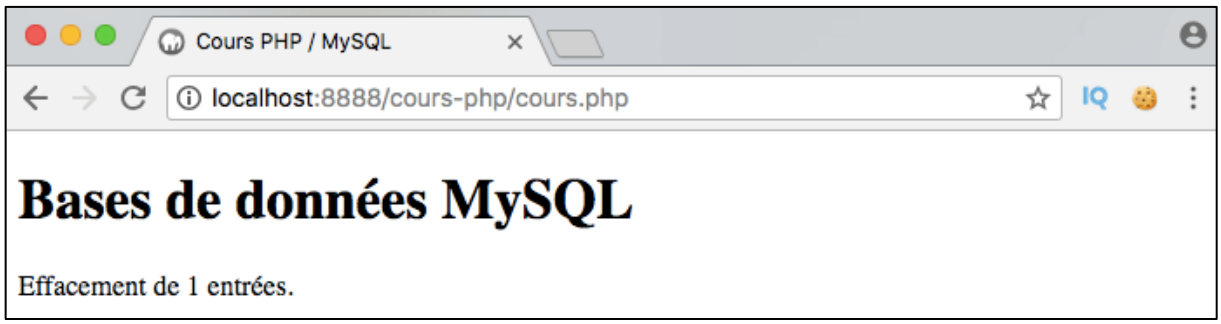
Pour tester cette instruction, nous allons utiliser la table « Users » créée précédemment (table contenant 4 colonnes et 3 entrées).

En pratique, pour supprimer une entrée en particulier, nous utiliserons la clause `WHERE` sur une colonne « id » en ciblant un « id » précis.

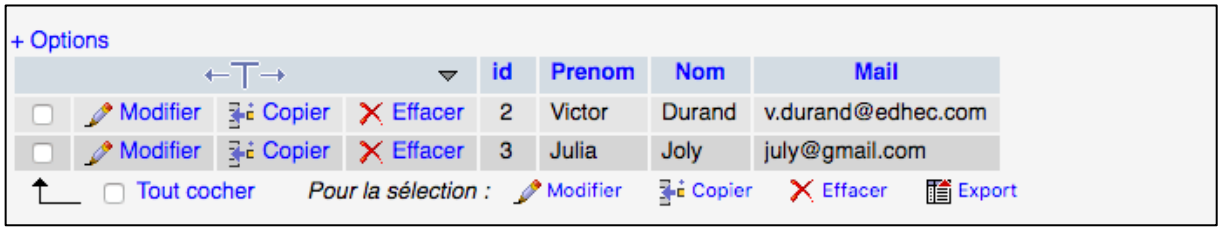
Nous pouvons également supprimer plusieurs entrées en donnant une inégalité en condition de la clause `WHERE` (cibler tous les « id » supérieurs à 5 par exemple) ou en ciblant un autre type de données (supprimer toutes les entrées dont la valeur dans la colonne « Prenom » est « Pierre » par exemple).

Ici, nous allons vouloir supprimer tous les utilisateurs dont le nom est « Giraud ».

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sql = "DELETE FROM Users WHERE nom='Giraud'";
        $sth = $dbco->prepare($sql);
        $sth->execute();

        $count = $sth->rowCount();
        print('Effacement de ' .$count. ' entrées.');
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



On peut aller vérifier dans phpMyAdmin que notre entrée a bien été effacée :

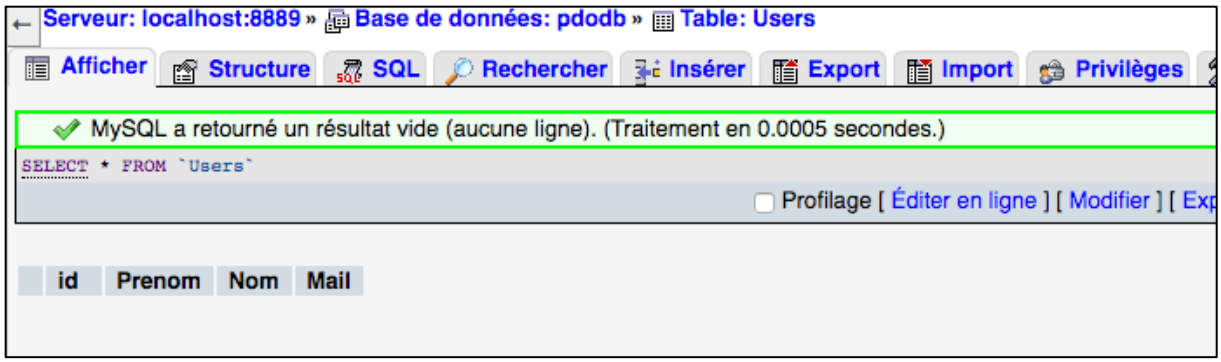


Supprimer toutes les données d'une table

Pour supprimer toutes les données d'une table sans pour autant supprimer la table ni sa structure, c'est très simple, il suffit d'utiliser l'instruction SQL `DELETE FROM` sans préciser de clause `WHERE`.

Essayons par exemple d'effacer toutes les données de la table « Users » d'un coup.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sql = "DELETE FROM Users";
        $sth = $dbco->prepare($sql);
        $sth->execute();

        $count = $sth->rowCount();
        print('Effacement de ' .$count. ' entrées.');
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



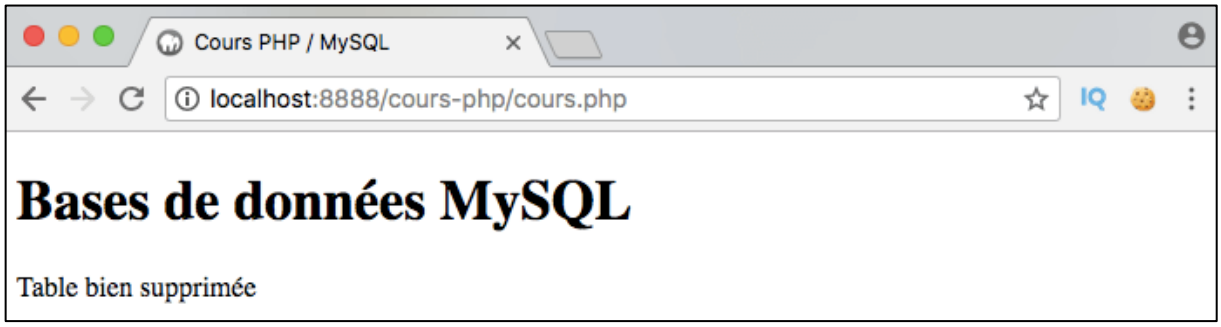
Attention lorsque vous effacez des données : n'oubliez pas que cette action est irréversible. Réfléchissez donc bien avant d'exécuter ce genre d'action et faites un backup de votre base de données.

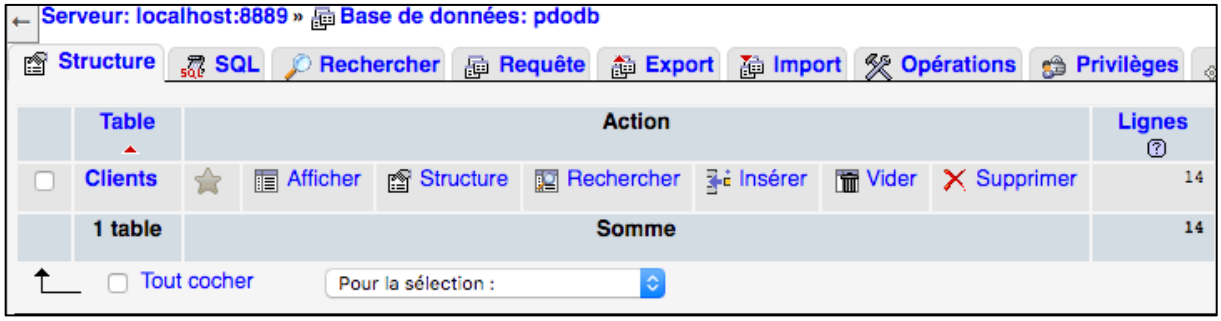
Supprimer complètement une table de la base de données

Pour supprimer complètement une table, nous allons cette fois-ci utiliser l'instruction SQL `DROP TABLE` suivie du nom de la table que l'on souhaite supprimer.

Essayons par exemple de supprimer complètement notre table « Users ». Pas besoin d'utiliser de requête préparée ici car la suppression de tables ne sera jamais (ou ne devrait jamais être du moins !) à l'initiative de vos utilisateurs.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sql = "DROP TABLE Users";
        $dbco->exec($sql);

        echo 'Table bien supprimée';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```





Une nouvelle fois, faites bien attention avant d'effectuer ce genre d'opération : supprimer une table supprimera toutes les données qu'elle contient et cette action est irréversible. Pensez donc bien toujours à sauvegarder les données avant si vous pensez en avoir besoin un jour.

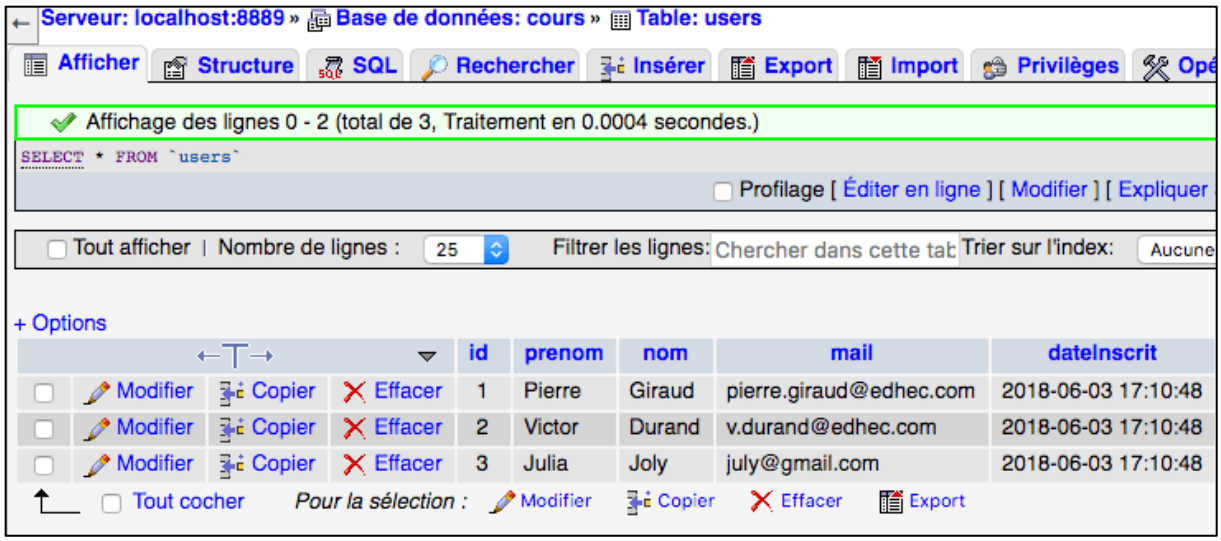
Supprimer complètement une base de données

Pour supprimer une base de données, nous utiliserons l'instruction SQL `DROP DATABASE` suivie du nom de la base de données que l'on souhaite supprimer.

Une nouvelle fois, faites bien attention avant d'exécuter ce genre d'opération : supprimer une base de données supprimera de manière irréversible toutes les données qu'elle contient.

Il est temps de dire adieu à notre base de données « pdodb » !

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "pdodb"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sql = "DROP DATABASE pdodb";
        $dbco->exec($sql);

        echo 'Base de données bien supprimée';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Vous pouvez aller vérifier dans phpMyAdmin, notre base de données n'existe plus.

Sélection simple de données dans une table via PHP

Dans les leçons précédentes, nous avons appris à insérer, modifier ou supprimer des données dans des tables.

Il ne nous reste donc plus qu'une opération de base à voir : la sélection ou récupération de données dans une base de données.

La sélection de données va être l'une des opérations fondamentales et les plus courantes que nous allons avoir à effectuer. En effet, nous allons devoir sélectionner ou récupérer des données en base notamment pour les comparer aux données envoyées par un utilisateur lors d'une tentative de connexion à son espace personnel sur notre site par exemple.

La sélection simple de données dans une base de données

Pour sélectionner des données dans une base de données, nous allons utiliser l'instruction SQL `SELECT... FROM`

Pour tester cette nouvelle instruction, il va avant tout nous falloir une base de données avec au moins une table et des données à l'intérieur. Pour cela, je vous invite à créer une nouvelle base qu'on appellera « cours » contenant une table nommée « users ».

Cette nouvelle table « users » va contenir 5 colonnes :

- Une colonne « id », type INT, UNISGNED, PRIMARY KEY, AUTO_INCREMENT
- Une colonne « prenom », type VARCHAR(30) NOT NULL
- Une colonne « nom », type VARCHAR(30) NOT NULL
- Une colonne « mail », type VARCHAR(50)
- Une colonne « dateInscrit », type TIMASTAMP.

Nous allons également en profiter pour insérer 3 entrées dans cette table. Voici le script qui va nous permettre de faire tout ça en une fois :

```php
<body>
  <h1>Bases de données MySQL</h1>
  <?php
    $servname = "localhost"; $user = "root"; $pass = "root";

    try{
      $dbco = new PDO("mysql:host=$servname", $user, $pass);
      $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

      $createDB = "CREATE DATABASE cours";
      $dbco->exec($createDB);

      //On utilise la base tout juste créée pour créer une table dedanss
      $createTb = "use cours";
      $dbco->exec($createTb);

      $createTb = "CREATE TABLE users(
        id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        prenom VARCHAR(30) NOT NULL,
        nom VARCHAR(30) NOT NULL,
        mail VARCHAR(50),
        dateInscrit TIMESTAMP)";
      $dbco->exec($createTb);

      $sth = $dbco->prepare("
        INSERT INTO users(prenom, nom, mail)
        VALUES (:prenom, :nom, :mail)
      ");
      $sth->bindParam(':prenom', $prenom);
      $sth->bindParam(':nom', $nom);
      $sth->bindParam(':mail', $mail);

      $prenom = "Pierre"; $nom = "Giraud"; $mail = "pierre.giraud@edhec.com";
      $sth->execute();
      $prenom = "Victor"; $nom = "Durand"; $mail = "v.durand@edhec.com";
      $sth->execute();
      $prenom = "Julia"; $nom = "Joly"; $mail = "july@gmail.com";
      $sth->execute();
    }

    catch(PDOException $e){
      echo "Erreur : " . $e->getMessage();
    }
  ?>
</body>
```

Notez ici l'utilisation de use qui va nous permettre de spécifier dans quelle base de données notre table doit être créée. En effet, au début du script, vous pouvez remarquer qu'on ne se connecte pas à une base de données en particulier tout simplement car celle-ci n'est pas encore créée.

Pour créer une base de données et une table en même temps, il va donc falloir préciser qu'on souhaite créer notre table dans la base de données tout juste créée.

A partir de là, nous pouvons tester notre instruction `SELECT...FROM` en sélectionnant par exemple tous les prénoms et adresses mail de notre table « users ».

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Sélectionne les valeurs dans les colonnes prenom et mail de la table
         *users pour chaque entrée de la table*/
        $sth = $dbco->prepare("SELECT prenom, mail FROM users");
        $sth->execute();

        /*Retourne un tableau associatif pour chaque entrée de notre table
         *avec le nom des colonnes sélectionnées en clefs*/
        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        /*print_r permet un affichage lisible des résultats,
         *<pre> rend le tout un peu plus lisible*/
        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Ici, nous utilisons les requêtes préparées pour sélectionner toutes les valeurs dans les colonnes « prenom » et « mail » de notre table « users ».

Une fois les valeurs sélectionnées, nous utilisons la méthode `fetchAll()` qui est une méthode de la classe `PDOStatement` qui va retourner un tableau contenant toutes les lignes de résultats.

Ici, nous passons un argument (facultatif) à `fetchAll()` qui est le « fetch_style ». La valeur `FETCH_ASSOC` va faire que le tableau retourné sera un tableau multidimensionnels contenant des tableaux indexés avec le nom des colonnes dont on récupère les données en index.

Notez que la valeur par défaut du « fetch_style » de `fetchAll()` est `ATTR_DEFAULT_FETCH_MODE` qui va lui-même prendre sa valeur par défaut de `FETCH_BOTH`.

La valeur `FETCH_BOTH` va faire que le résultat retourné va être un tableau multidimensionnel contenant des tableaux indexés et par le nom des colonnes et par les numéros des colonnes (chaque valeur va donc être spécifiés deux fois).

Si vous ne souhaitez récupérer qu'une ligne de résultats, notez également que vous pouvez utiliser la méthode `fetch()` à la place de `fetchAll()` qui va se manipuler de façon similaire.

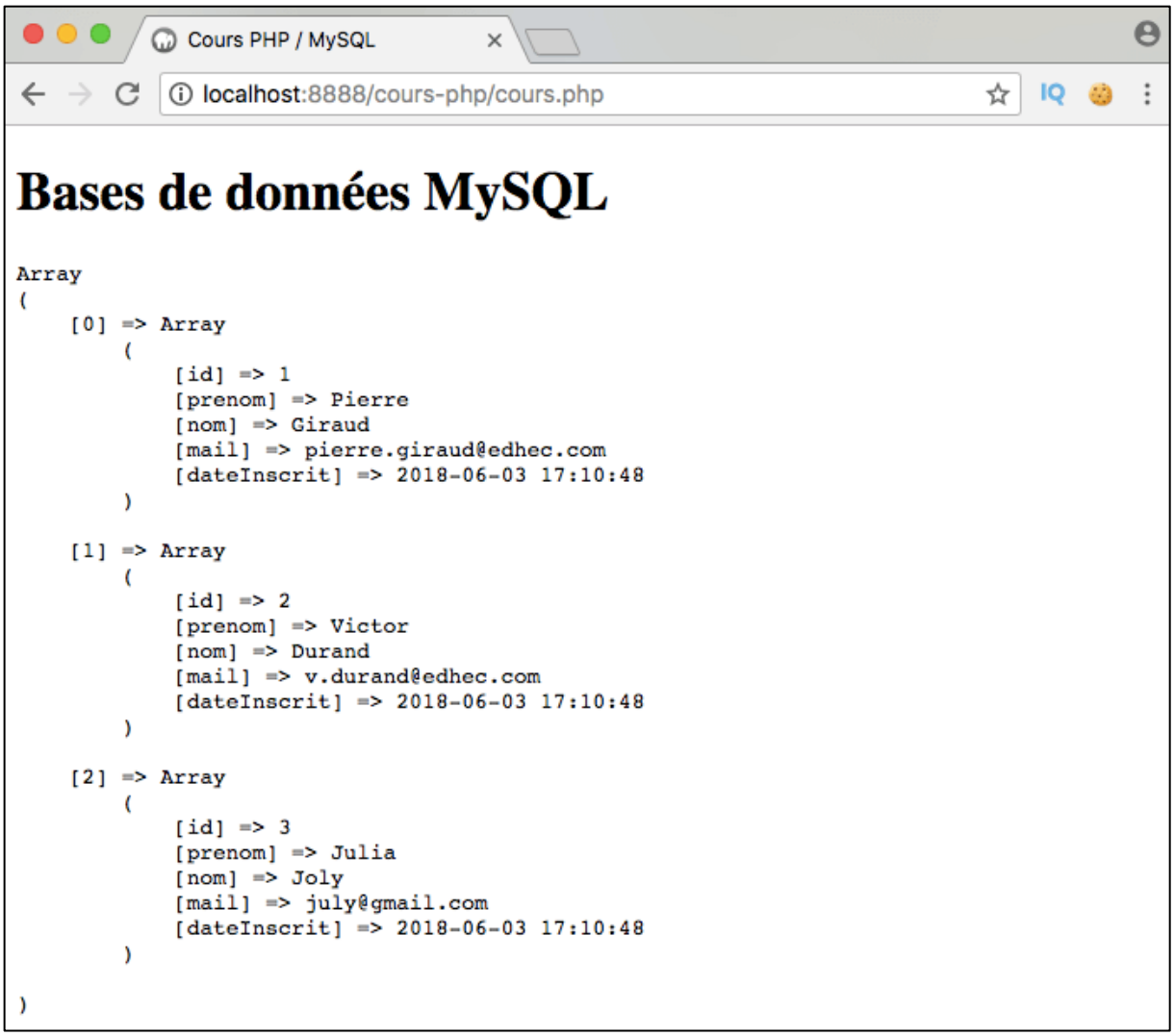
Pour plus d'informations sur ces deux méthodes, je vous invite à lire la documentation officielle de fetch() et de fetchAll().

Finalement, on affiche rapidement notre tableau avec `print_r()` et on utilise également l'élément HTML `pre` afin d'avoir un résultat un peu plus agréable à lire.

Récupérer toutes les valeurs dans une table

Pour récupérer toutes les valeurs dans une table d'un coup, nous allons simplement utiliser le sélectionneur `*` (étoile) qui, en SQL comme dans beaucoup d'autres langages informatiques, est un sélecter universel (sélecteur qui sert à tout sélectionner) avec l'instruction `SELECT... FROM`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Sélectionne toutes les valeurs dans la table users*/
        $sth = $dbco->prepare("SELECT * FROM users");
        $sth->execute();

        /*Retourne un tableau associatif pour chaque entrée de notre table
         *avec le nom des colonnes sélectionnées en clefs*/
        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        /*print_r permet un affichage lisible des résultats,
         *<pre> rend le tout un peu plus lisible*/
        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Ne récupérer que les valeurs uniques (par colonne) dans une table

Parfois, nous ne voudrons récupérer que les valeurs distinctes d'une colonne dans une table parmi toutes les valeurs.

Nous allons pouvoir faire cela en utilisant l'instruction SQL `SELECT DINSTINCT`. Cette instruction ne va retourner qu'une seule fois un même résultat.

Pour tester cette nouvelle instruction, commençons déjà par ajouter une quatrième entrée dans notre table « users ». Notre quatrième utilisateur s'appellera également « Pierre ».

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sth = $dbco->prepare("
          INSERT INTO users (prenom, nom, mail)
          VALUES ('Pierre', 'Rigaud', 'prgd@gmail.com')
          ");
        $sth-> execute();

        echo "Entrée bien insérée";
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Dès que c'est fait, nous n'avons plus qu'à tester notre instruction `SELECT_DISTINCT` de manière très classique en utilisant les requêtes préparées :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sth = $dbco->prepare("SELECT DISTINCT prenom FROM users");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        //<pre> permet un affichage lisible des résultats affichés avec print_r
        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Mettre en ordre les valeurs récupérées dans une table

Jusqu'à présent, nous avons retourné les données sélectionnées dans notre base de données selon l'ordre de leur écriture.

Nous allons cependant facilement pouvoir trier les données renvoyées selon un ordre croissant (de la plus petite valeur numérique à la plus grande, ou de A à Z) ou décroissant grâce à l'instruction SQL `ORDER BY`.

Pour signifier que l'on souhaite trier selon un ordre croissant, nous utiliserons le mot `ASC` (pour « ascending », l'équivalent anglais de « croissant »). Si l'on souhaite un ordre décroissant, nous utiliserons plutôt `DESC` (« descending » ou « décroissant » en français).

De plus, nous allons pouvoir indiquer plusieurs colonnes sur lesquelles notre tri doit être fait dans le cas où nous aurions beaucoup de fois la même valeur renvoyée. La première colonne indiquée sera considérée comme le filtre de tri primaire, la deuxième secondaire, etc.

Sélectionnons immédiatement tous les prénoms et noms de notre table « users » et trions les résultats renvoyés selon l'ordre croissant des prénoms en tri principal puis selon l'ordre décroissant des noms en tri secondaire.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //Tri croisant par prénoms puis décroissant par noms
        $sth = $dbco->prepare("
          SELECT prenom, nom
          FROM users
          ORDER BY prenom ASC, nom DESC
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

De manière concrète, les tris par ordre croissant ou décroissant vont s'avérer utiles lorsqu'il s'agira de trier une liste de commandes par prix par exemple ou encore les utilisateurs par pays.

Utiliser des critères pour effectuer des sélections conditionnelles

Dans la leçon précédente, nous avons appris à sélectionner des données dans une table grâce à l'instruction SQL `SELECT… FROM`.

Un des grands intérêts des bases de données est que les données sont organisées et classées. Lorsque nous sélectionnons des données dans nos bases de données, nous voudrons souvent tirer avantage de cette organisation et ainsi sélectionner des données précises.

Les critères de sélection vont nous aider à créer des requêtes SQL de sélection puissantes et précises. Dans cette leçon, nous allons nous intéresser aux critères de sélection SQL suivants :

- WHERE ;
- AND, OR et NOT ;
- LIMIT ;
- LIKE et les jokers (wildcards) ;
- IN et BETWEEN ;
- EXISTS ;
- ANY et ALL.

La clause SQL WHERE

Nous connaissons déjà cette clause et je ne vais donc pas beaucoup m'étendre sur son fonctionnement.

La clause `WHERE` va nous permettre de rajouter un conditionnement à une requête SQL. On va ainsi pouvoir ne sélectionner que des valeurs égales, supérieures ou inférieures à une certaine valeur.

Notez que la clause `WHERE` peut être utilisée dans des requêtes de sélection SQL aussi bien que dans des requêtes de suppression, de mises à jour, etc.

Dans notre table « users » contenant 4 entrées, nous allons par exemple pouvoir sélectionner tous les utilisateurs dont le prénom est « Pierre ».

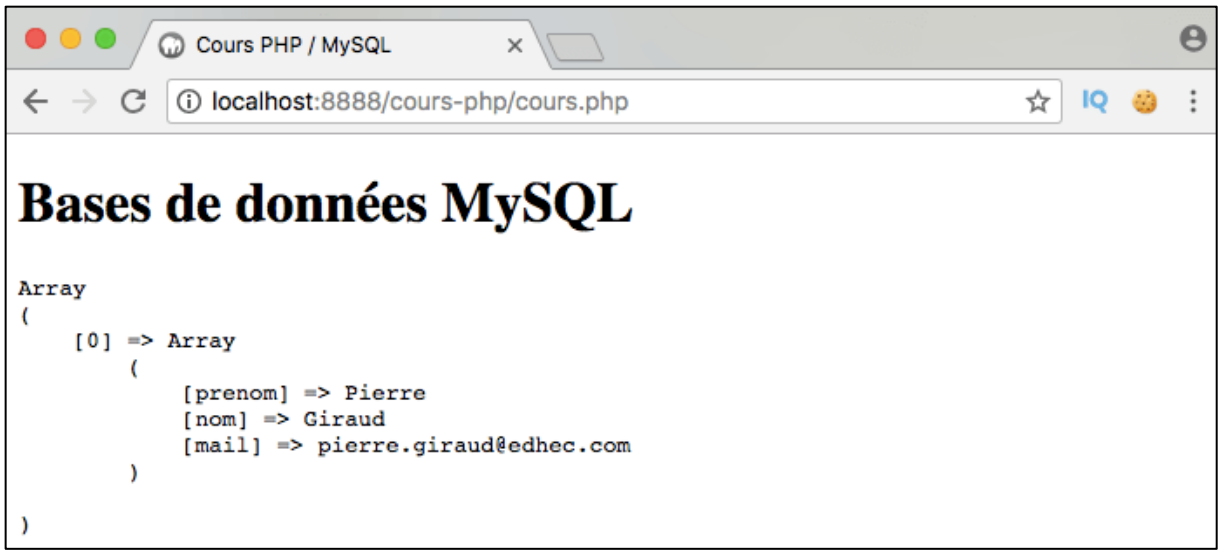
```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //On sélectionne tous les users dont le nom = Pierre
        $sth = $dbco->prepare("
          SELECT prenom, nom, mail
          FROM users
          WHERE prenom = 'Pierre'
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Les opérateurs SQL AND, OR et NOT

Nous allons pouvoir étendre les possibilités de la clause SQL `WHERE` grâce aux opérateurs `AND`, `OR` et `NOT`.

L'opérateur `AND` va nous permettre de rajouter des conditions supplémentaires. Seuls les résultats satisfaisant à toutes les conditions seront sélectionnés. Notez que vous pouvez utiliser autant de `AND` que vous souhaitez dans une requête SQL.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*On sélectionne tous les users dont le prenom = Pierre et
         *le nom = Giraud*/
        $sth = $dbco->prepare("
          SELECT prenom, nom, mail
          FROM users
          WHERE prenom = 'Pierre' AND nom = 'Giraud'
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



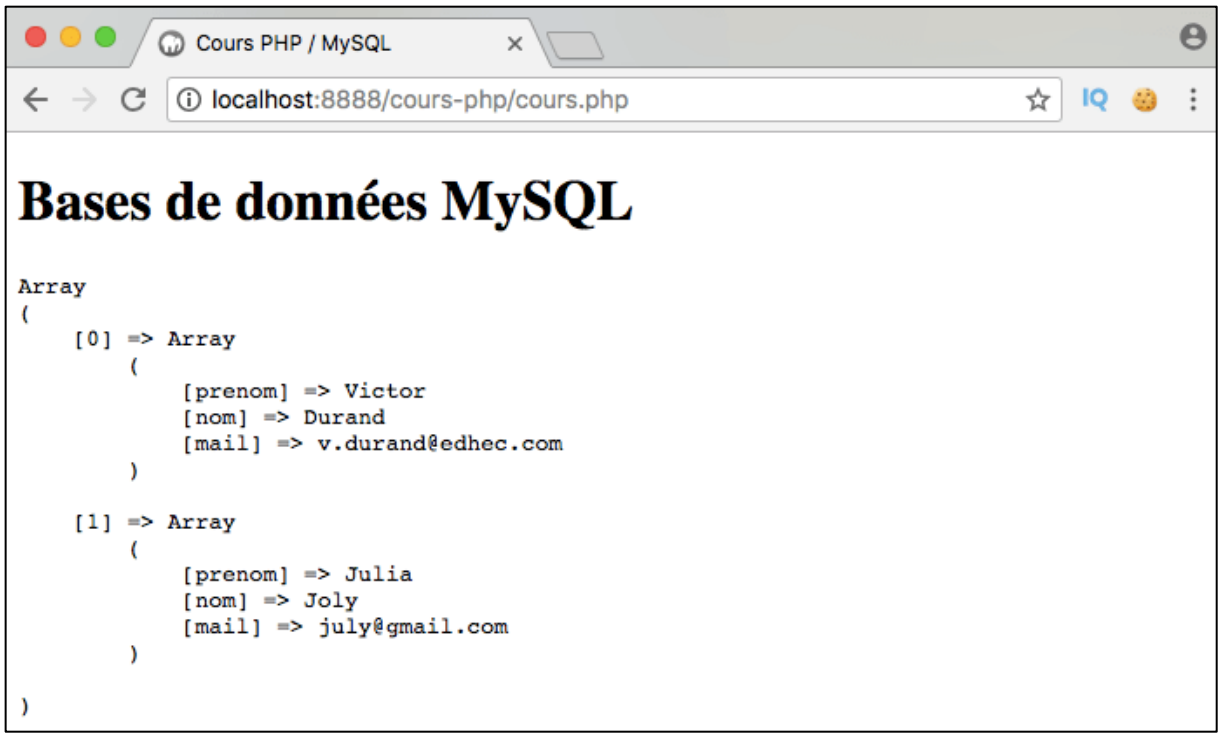
L'opérateur SQL `OR` va lui nous permettre d'élargir notre condition de base en rajoutant d'autres conditions. A la différence de `AND`, tous les résultats satisfaisants au moins l'une des conditions mentionnées seront affichés.

Nous allons également pouvoir ajouter autant de `OR` qu'on le souhaite dans une requête SQL.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*On sélectionne tous les users dont le prenom = Pierre ou
         *le nom = Joly*/
        $sth = $dbco->prepare("
          SELECT prenom, nom, mail
          FROM users
          WHERE prenom = 'Pierre' OR nom = 'Joly'
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Finalement, l'opérateur SQL `NOT` va nous permettre d'afficher tous les résultats ne satisfaisant pas une condition. On peut par exemple afficher tous les utilisateurs dont le prénom n'est pas « Pierre ».

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //On sélectionne tous les users dont le prenom n'est pas Pierre
        $sth = $dbco->prepare("
          SELECT prenom, nom, mail
          FROM users
          WHERE NOT prenom = 'Pierre'
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Notez que nous allons également pouvoir utiliser plusieurs opérateurs ensemble dans une requête SQL.

Nous allons ainsi par exemple pouvoir sélectionner tous les utilisateurs de notre table dont l'id est supérieur à 1 et dont le prénom n'est pas « Pierre » ou dont le nom est « Giraud ».

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Sélectionne tous les users dont l'id est spérieur à 1
         *ET dont le prénom n'est pas pierre OU dont le nom est Giraud*/
        $sth = $dbco->prepare("
          SELECT prenom, nom, mail
          FROM users
          WHERE id > 1 AND NOT prenom = 'Pierre' OR nom = 'Giraud'
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Ici, faites bien attention à l'ordre des opérations et à bien écrire / lire votre requête afin qu'elle affiche les résultats souhaités.

Notre requête va sélectionner tous les utilisateurs dont l'id est supérieur à 1 ET dont SOIT le prénom n'est pas « Pierre », SOIT le nom est « Giraud ». C'est pour cela que le résultat « Pierre Giraud » est renvoyé.

La clause LIMIT

La clause SQL `LIMIT` est généralement utilisée pour limiter le nombre de résultats retournés.

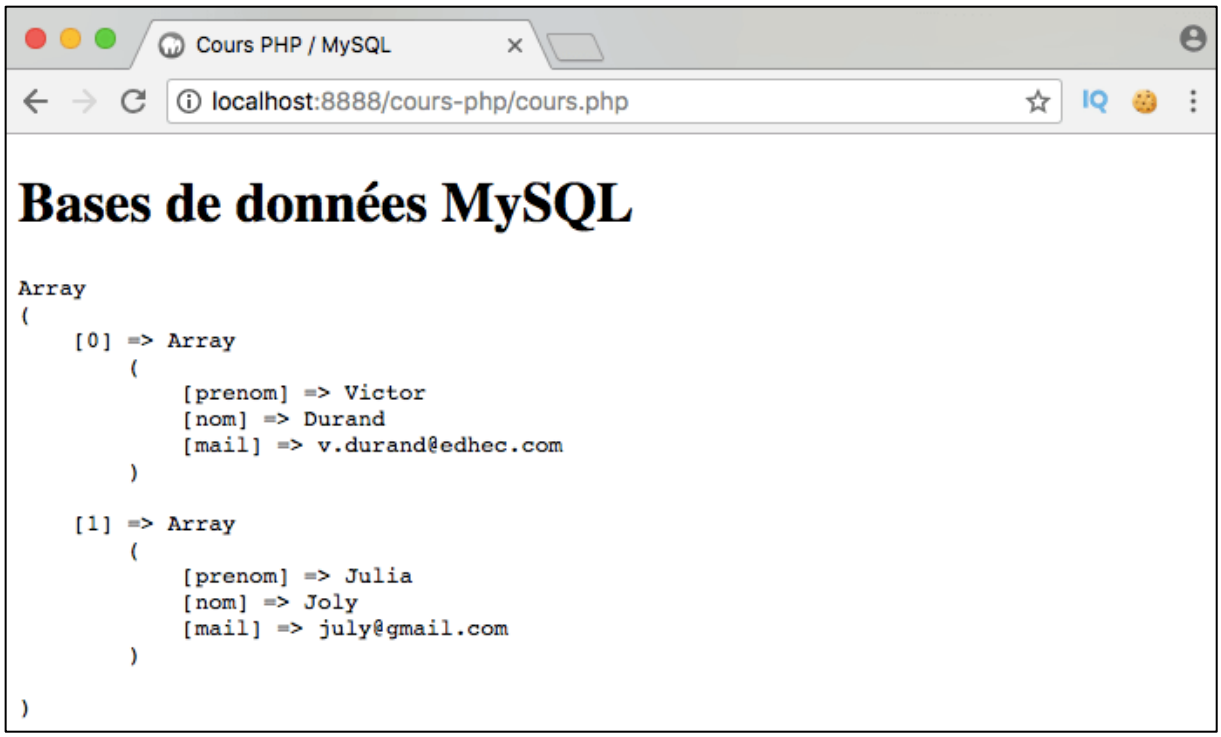
En effet, cette clause va nous permettre d'indiquer un nombre de résultats maximum à retourner. Cela peut être utile pour optimiser la performance de votre script dans le cas où un très grand nombre de résultats seraient retournés.

Par défaut, la clause `LIMIT` va sélectionner des résultats dans l'ordre des entrées de votre table. Cependant, on va pouvoir spécifier à partir de quelle entrée on souhaite commencer à récupérer des résultats grâce au mot `OFFSET`.

On peut ainsi par exemple récupérer deux résultats dans notre table « users » à partir de la deuxième entrée (comprenez bien « la deuxième entrée satisfaisant notre condition s'il y en a une ») de cette manière :

Notez que le premier résultat dans l'ordre des entrées de la table satisfaisant notre requête correspond à `OFFSET 0`. Pour commencer à récupérer des résultats à partir du deuxième résultat dans l'ordre des entrées de la table satisfaisant notre requête il faudra donc préciser `OFFSET 1` et etc.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Sélectionne 2 résultats à partir de la 2è entrée de la table*/
        $sth = $dbco->prepare("
          SELECT prenom, nom, mail
          FROM users
          LIMIT 2 OFFSET 1
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



L'opérateur SQL LIKE et les jokers (wildcards)

Nous allons utiliser l'opérateur SQL `LIKE` conjointement avec une clause `WHERE` afin de chercher un schéma spécifique dans une colonne.

Nous allons également généralement utiliser l'opérateur `LIKE` avec des jokers ou wildcards en anglais.

Les jokers sont des caractères de substitution qui vont nous permettre de rechercher un schéma précis. Vous pouvez comparer les wildcards aux différents caractères que nous utilisions dans nos expressions régulières plus tôt dans ce cours.

Il existe deux jokers que nous allons pouvoir utiliser avec `LIKE` :

- Le signe `%` qui va représenter zéro, un o plusieurs caractères ;
- Le signe `_` qui va représenter un caractère exactement.

Nous allons bien entendu pouvoir combiner les jokers entre eux dans nos requêtes.

Pour bien comprendre l'utilisation de `LIKE` et des jokers, voici quelques exemples de parties de requêtes ainsi que leur signification :

Requête	Signification
WHERE users LIKE 'p%'	Cherche les valeurs qui commencent par un « p »

WHERE users LIKE ‘%e’	Cherche les valeurs qui se terminent par « e »
WHERE users LIKE ‘%e%’	Cherche les valeurs qui possèdent un « e »
WHERE users LIKE ‘p%e’	Cherche les valeurs qui commencent par « p » et se terminent par « e »
WHERE users LIKE ‘p____e’	Cherche des valeurs de 6 caractères exactement qui commencent par « p » et se terminent par « e »
WHERE users LIKE ‘p_%’	Cherche des valeurs de 2 caractères ou plus qui commencent par « p »

On va ainsi par exemple pouvoir ne sélectionner que les utilisateurs dont le nom contient un « r » dans notre table « users » :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //Cherche tous les noms contenant un "r"
        $sth = $dbco->prepare("
          SELECT prenom, nom, mail
          FROM users
          WHERE nom LIKE '%r%'
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Les opérateurs SQL IN et BETWEEN

L'opérateur SQL `IN` va s'utiliser conjointement avec une clause `WHERE`. Cet opérateur va nous permettre de préciser une liste de données parmi lesquelles nous devons sélectionner nos données.

Utiliser `IN` revient finalement à utiliser plusieurs conditions `OR` mais avec une notation allégée et plus rapide.

On va ainsi par exemple pouvoir facilement sélectionner tous les utilisateurs dont le prénom est « Pierre » ou « Victor » dans notre table « users ».

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //Cherche tous les utilisateurs dont le prénom est Pierre ou Victor
        $sth = $dbco->prepare("
          SELECT prenom, nom, mail
          FROM users
          WHERE prenom IN ('Pierre', 'Victor')
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

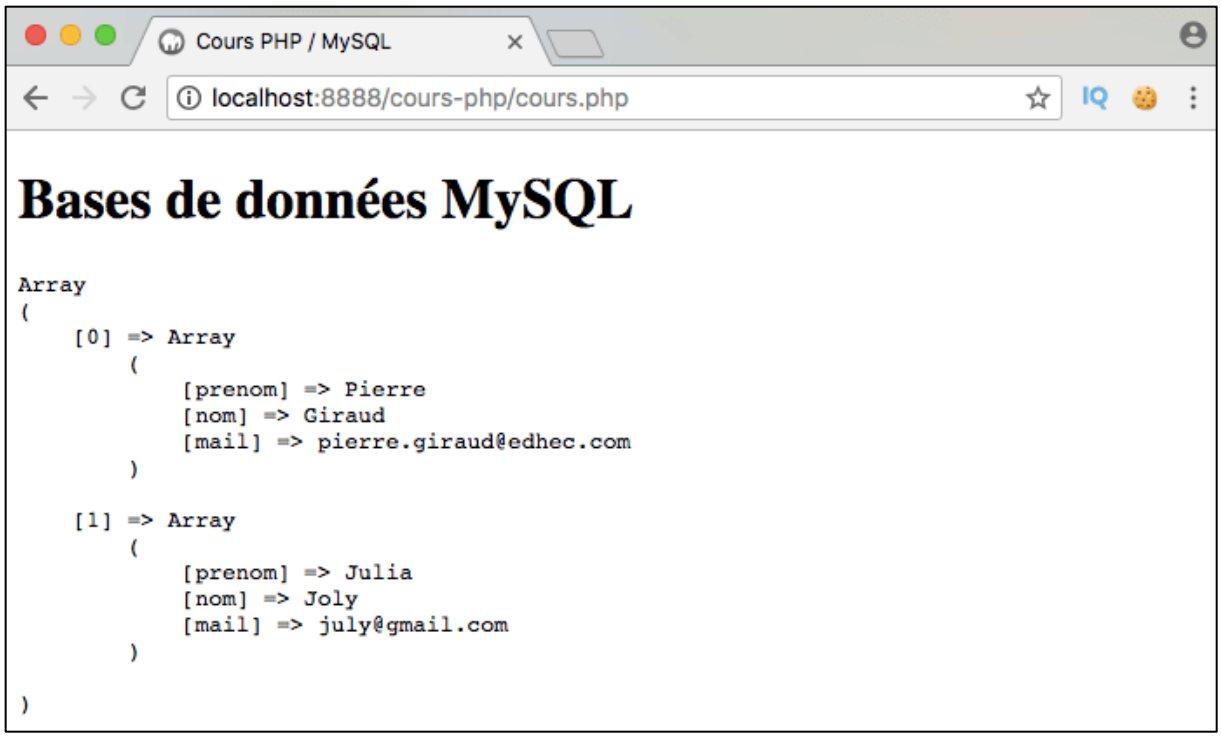
Notez qu'on va bien évidemment pouvoir combiner les différents opérateurs SQL entre eux et ainsi par exemple pouvoir utiliser `IN` avec `NOT` pour exclure certaines valeurs spécifiques de notre sélection.

L'opérateur SQL `BETWEEN` va lui nous permettre de sélectionner ou d'exclure d'une sélection des données dans un certain intervalle. Notez que cet opérateur est inclusif (la première et la dernière valeur font partie de l'intervalle).

Les valeurs d'intervalle peuvent être des nombres, des textes ou des dates.

On va par exemple pouvoir sélectionner tous les utilisateurs dont le nom se trouve entre « F » et « Joly », ces deux valeurs étant donc incluses dans notre sélection.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //Sélectionne les utilisateurs dont le nom se trouve entre F et Joly
        $sth = $dbco->prepare("
          SELECT prenom, nom, mail
          FROM users
          WHERE nom BETWEEN 'F' AND 'Joly'
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```


Cours PHP / MySQL
localhost:8888/cours-php/cours.php
Bases de données MySQL
Array
(
[0] => Array
(
[prenom] => Pierre
[nom] => Giraud
[mail] => pierre.giraud@edhec.com
)
[1] => Array
(
[prenom] => Julia
[nom] => Joly
[mail] => july@gmail.com
)
)

Les fonctions d'agrégation et scalaires

Le SQL et le MySQL possèdent de nombreuses fonctions prêtes à l'emploi qui vont nous permettre de retourner certaines données en particulier ou d'effectuer des manipulations sur les données récupérées depuis notre base de données.

Parmi toutes ces fonctions, deux grands groupes de fonctions se dessinent : les fonctions d'agrégation et les fonctions scalaires.

Les fonctions d'agrégation

Les fonctions d'agrégation vont être utilisées pour réaliser des opérations à partir de différentes valeurs dans une colonne et ne vont retourner qu'une valeur. Ces fonctions « agrègent » donc plusieurs valeurs d'entrée en une valeur de sortie.

Dans cette leçon, nous allons étudier les fonctions d'agrégation suivantes :

- Les fonctions `min()` et `max()` ;
- La fonction `count()` ;
- La fonction `avg()` ;
- La fonction `sum()`.

Les fonctions min() et max()

La fonction SQL `min()` va retourner la valeur la plus petite dans une colonne sélectionnée.

La fonction SQL `max()` va elle retourner la valeur la plus grande dans une colonne sélectionnée.

Nous pouvons utiliser ces fonctions pour voir par exemple quelle est la plus grosse commande passée sur notre site, quel est l'utilisateur le plus âgé, etc. Notez que ces fonctions vont également fonctionner sur des chaines de caractères. Dans ce cas, le « a » sera considéré plus petit que « b » et etc.

Pour tester les différentes fonctions de cette partie, nous allons à nouveau utiliser notre table « users » dans laquelle nous allons ajouter une colonne « age » et également modifier l'âge de nos utilisateurs déjà inscrits.

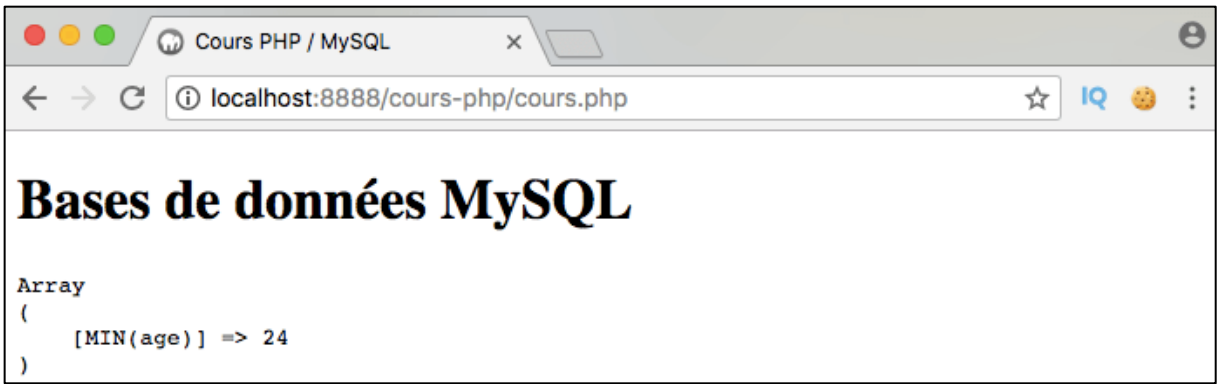
Vous pouvez faire cela directement dans phpMyAdmin en ajoutant une colonne dans notre table à partir de l'onglet « Structure » puis ne modifiant la valeur de la colonne de chaque ligne.

Voici ce qu'on obtient :

				id	prenom	nom	mail	age	dateInscrit
☐	Modifier	Copier	Effacer	1	Pierre	Giraud	pierre.giraud@edhec.com	28	2018-06-10 10:02:52
☐	Modifier	Copier	Effacer	2	Victor	Durand	v.durand@edhec.com	27	2018-06-10 10:03:01
☐	Modifier	Copier	Effacer	3	Julia	Joly	july@gmail.com	24	2018-06-10 10:03:08
☐	Modifier	Copier	Effacer	4	Pierre	Rigaud	prgd@gmail.com	36	2018-06-10 10:03:14

Nous pouvons ensuite par exemple sélectionner le plus petit âge de notre colonne avec la fonction min(). Attention, cette fonction ne va retourner que la valeur la plus petite de la colonne et non pas afficher toutes les informations relatives à l'entrée possédant cette valeur.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //Sélectionne la valeur la plus petite dans la colonne age
        $sth = $dbco->prepare("
          SELECT MIN(age)
          FROM users
        ");
        $sth-> execute();

        $resultat = $sth->fetch(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Notez ici qu'on utilise `fetch()` et non pas `fetchAll()` pour afficher les données tout simplement car les fonctions d'agrégat ne renvoient qu'une valeur en résultat.

La fonction count()

La fonction SQL `count()` va retourner le nombre d'entrées d'une colonne. Nous l'utiliserons généralement avec une clause `WHERE` pour qu'elle retourne le nombre d'entrées qui vont satisfont à une certaine condition.

Cette fonction va être très utile d'un point de vue statistique, pour savoir par exemple combien de vos clients sont des hommes, ou habitent à Paris, etc.

Nous allons par exemple pouvoir savoir combien de nos utilisateurs ont plus de 30 ans :

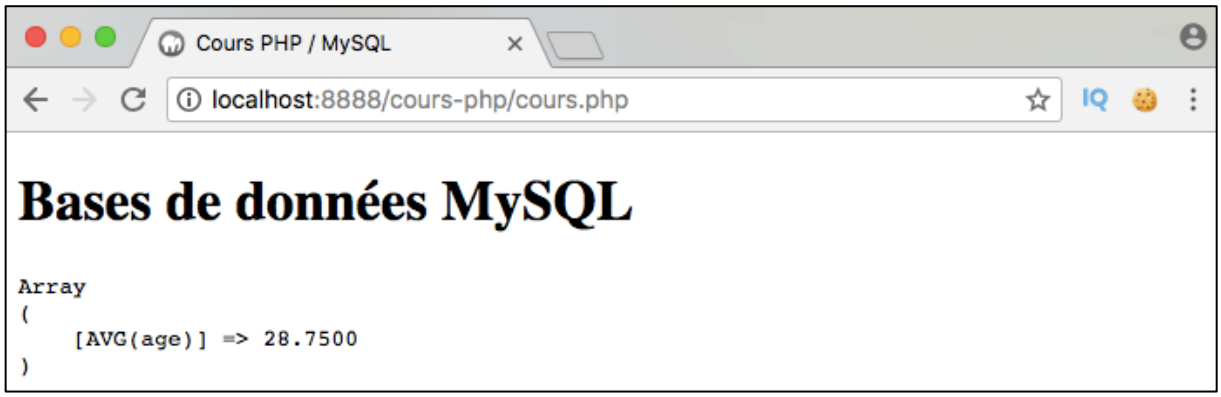
```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //Retourne le nb d'utiilsateurs qui ont plus de 30 ans
        $sth = $dbco->prepare("
          SELECT COUNT(age)
          FROM users
          WHERE age > 30
        ");
        $sth-> execute();

        $resultat = $sth->fetch(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

La fonction avg()

La fonction SQL avg() retourne la valeur moyenne d'une colonne contenant des valeurs numériques.

On va donc pouvoir d'un coup d'œil connaitre l'âge moyen de nos utilisateurs.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //Retourne l'âge moyen de nos utilisateurs
        $sth = $dbco->prepare("
          SELECT AVG(age)
          FROM users
        ");
        $sth-> execute();

        $resultat = $sth->fetch(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



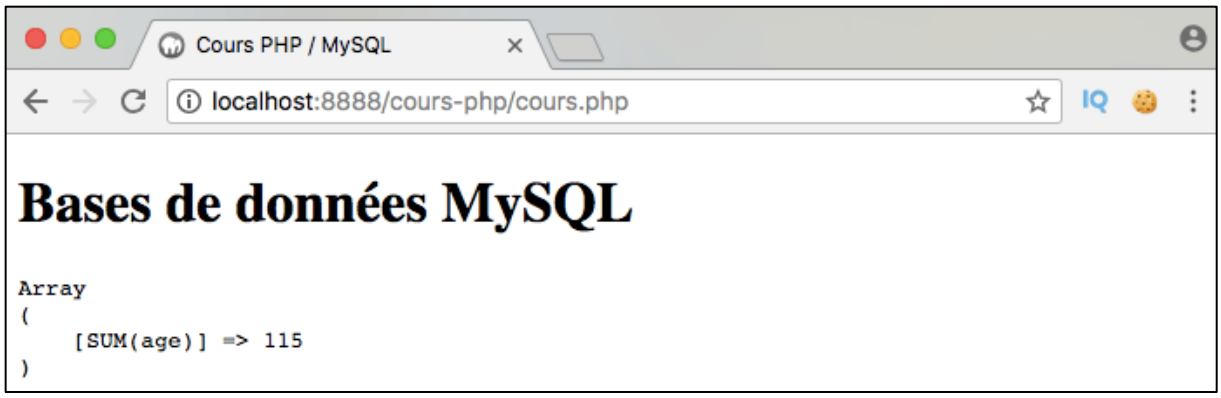
La fonction sum()

La fonction SQL `sum()` retourne la somme des valeurs d'une colonne contenant des valeurs numériques.

Cette fonction va être utile pour faire un inventaire des produits vendus par exemple.

Pour la tester, nous pouvons dans notre table additionner les âges même si, je dois le reconnaitre, ça ne sert pas à grand-chose en pratique !

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //Retourne la somme des ages de nos utilisateurs
        $sth = $dbco->prepare("
          SELECT SUM(age)
          FROM users
        ");
        $sth-> execute();

        $resultat = $sth->fetch(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Les fonctions scalaires

Les fonctions scalaires sont un sous type de UDF (« User Defined Functions » ou « Fonctions définies par l'Utilisateur » en français) en ce sens où ces fonctions sont basées sur les données envoyées par l'utilisateur.

Nous allons découvrir les fonctions scalaires suivantes :

- Les fonctions `lcase()` et `ucase()` ;
- La fonction `length()` ;
- La fonction `round()` ;
- La fonction `now()`.

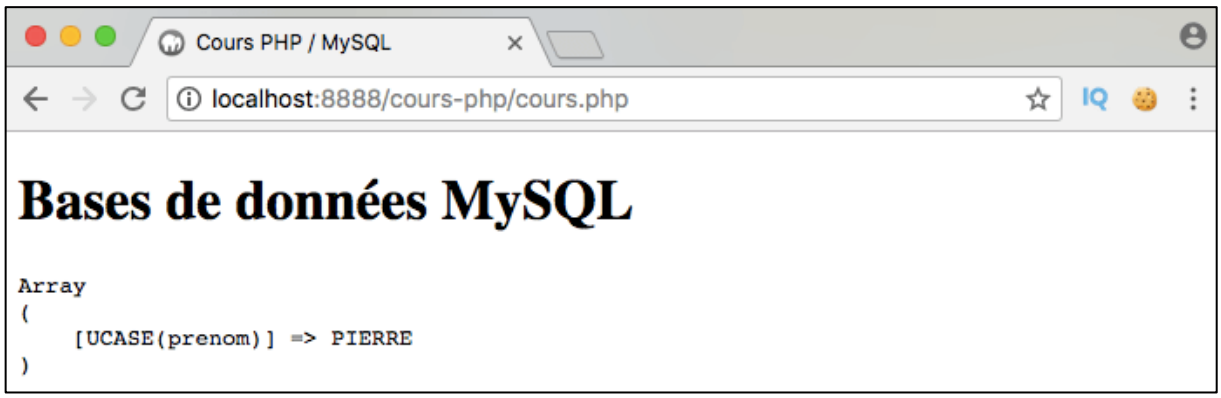
Les fonctions lcase() et ucase()

Les fonctions SQL `lcase()` et `ucase()` servent respectivement à convertir une chaine de caractères en minuscules ou en majuscules.

Ces fonctions peuvent être utiles pour pré-formater des résultats qu'on souhaite manipuler par la suite.

Nous allons ainsi par exemple pouvoir récupérer tous les prénoms de notre table « users » en majuscule.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //Retourne les prénoms en majuscules
        $sth = $dbco->prepare("
          SELECT UCASE(prenom)
          FROM users
        ");
        $sth-> execute();

        $resultat = $sth->fetch(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



La fonction length()

La fonction MySQL `length()` permet de calculer la longueur d'une chaîne de caractères.

Attention : notez que cette fonction se base sur le nombre de Bytes ou d'octets (1 Byte = 8 bits = 1 octet) pour le calcul et va bien retourner une taille en Bytes. Ainsi, un caractère multi-Bytes (comme les caractères accentués par exemple) est compté comme plus de 1 Byte.

On peut ainsi par exemple retourner le prénom de notre premier utilisateur ainsi que la longueur de celui-ci :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Retourne le prénom et la longueur du prénom
         *de la première entrée de la table*/
        $sth = $dbco->prepare("
          SELECT prenom, LENGTH(prenom)
          FROM users
          WHERE id = 1
        ");
        $sth-> execute();

        $resultat = $sth->fetch(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

La fonction round()

La fonction SQL `round()` va nous permettre d'arrondir une valeur en choisissant le nombre de décimales voulues.

Cette fonction va donc prendre en paramètres une valeur à arrondir (ou le nom d'une colonne) ainsi que le nombre de décimales auxquelles on souhaite arrondir la ou les valeurs.

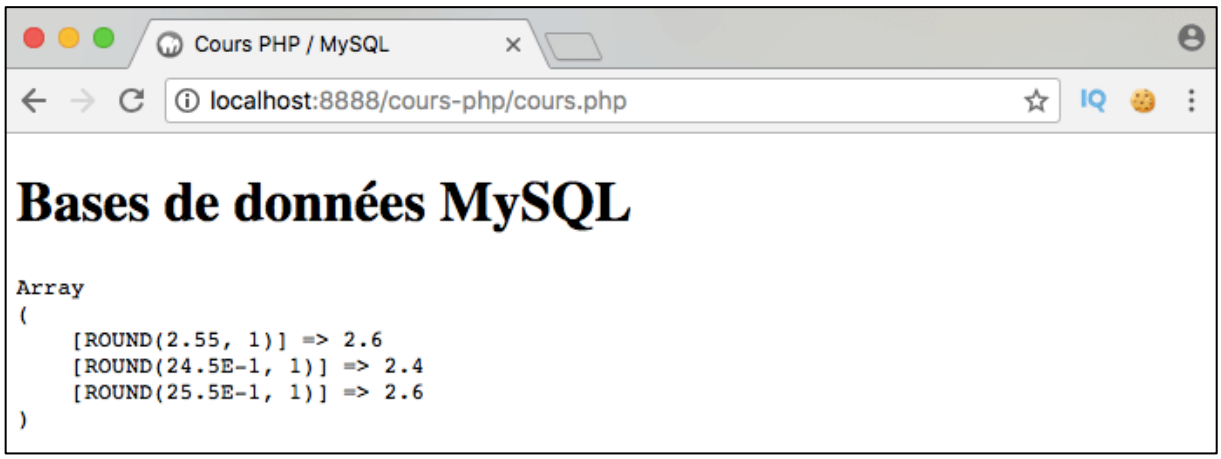
Attention cependant : les règles d'arrondis (pour les valeurs médianes comme 0.5) vont être différentes selon les situations :

- Si on passe un chiffre exact, la règle d'approximation pour une commande SQL de type `SELECT` va être d'arrondir à la valeur supérieure ;
- Si on passe une « valeur approximative » (expression utilisant une exponentielle par exemple), la règle d'arrondi va dépendre de la librairie C utilisée. Généralement, la règle sera d'arrondir au nombre pair le plus proche pour une commande SQL de type `SELECT` ;
- Dans le cas d'une commande de type `INSERT`, alors la règle d'arrondi sera d'arrondi au nombre le plus éloigné de zéro dans tous les cas.

Testons cela immédiatement en passant une valeur exacte et deux valeurs approximatives à `round()` et en lui demandant d'arrondir ces valeurs à 1 décimale.

Pour cette fois, nous ne prendrons pas de valeurs dans notre table mais allons passer les valeurs directement à notre fonction.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Retourne une valeur exacte arrondie (1 décimale) et une
        valeur approximative arrondie (1 décimale)*/
        $sth = $dbco->prepare("
          SELECT ROUND(2.55, 1), ROUND(24.5E-1, 1), ROUND(25.5E-1, 1)
        ");
        $sth-> execute();

        $resultat = $sth->fetch(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Comme vous pouvez le voir, dans le cas de notre valeur exacte (2,55), round() arrondi à la valeur supérieure (2,6). En revanche, dans le cas de nos valeurs approximatives,

l'arrondi va être fait au nombre pair le plus proche. Pour cette raison, 24,5E-1 est arrondi à 2,4 et 25,5E-1 est arrondi à 2,6.

Note : Pensez bien à utiliser des points et non pas des virgules pour représenter des nombres décimaux : on travaille toujours avec les notations anglo-saxonnes en informatique.

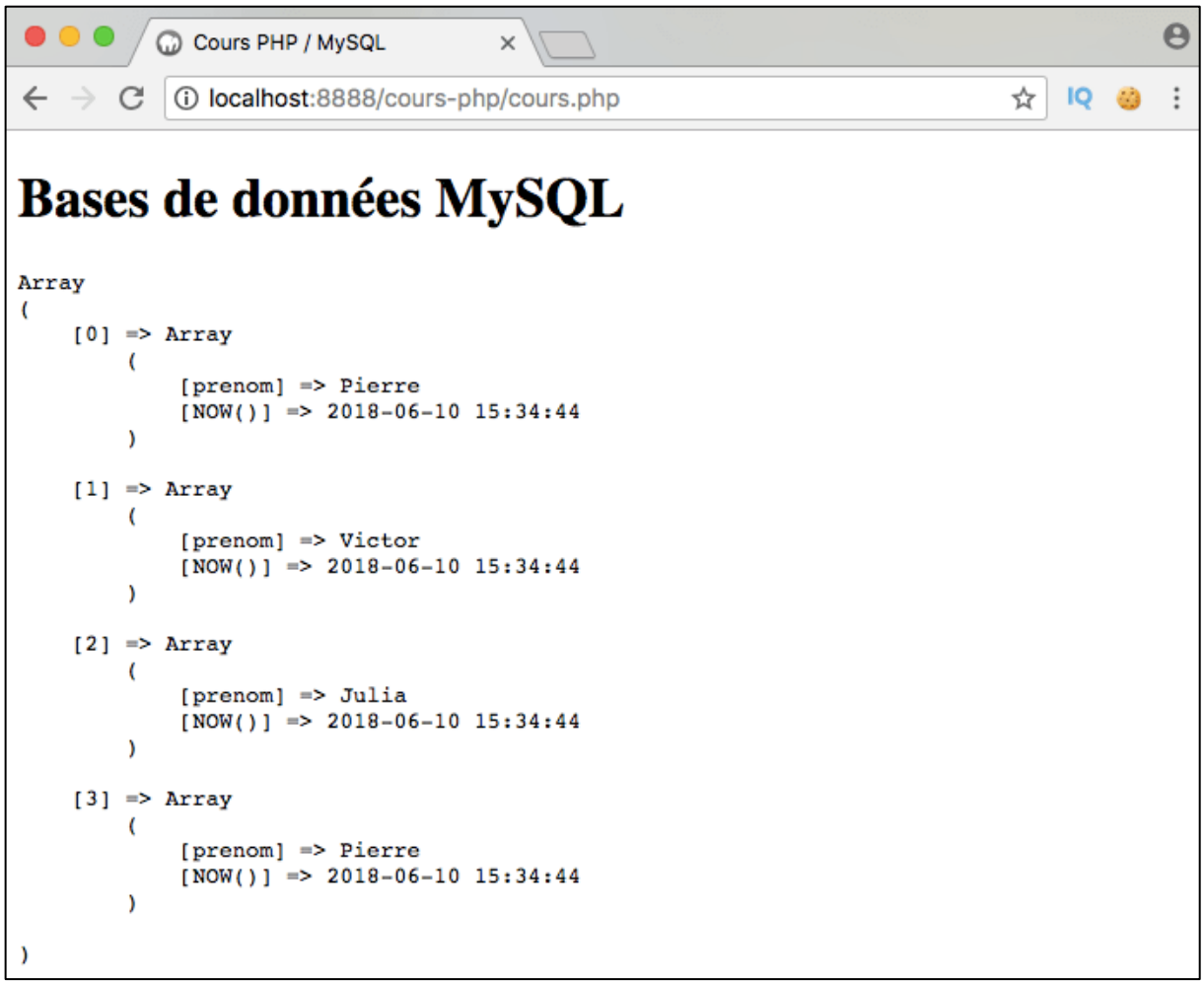
La fonction now()

La fonction SQL `now()` est utilisée pour retourner la date courante.

Cette fonction va être utile pour contextualiser une sélection de données en datant la date d'export par exemple.

On peut par exemple sélectionner tous les prénoms de notre table « users » en précisant la date de sélection :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Retourne les prénoms avec la date d'extraction*/
        $sth = $dbco->prepare("
          SELECT prenom, NOW() FROM users
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Les fonctions d'agrégation et les critères de sélection

Il existe deux autres critères de sélection et de tri très courants dont je n'ai pas encore parlé jusqu'à présent pour la simple et bonne raison qu'on va principalement les utiliser avec les fonctions d'agrégation et qui sont `GROUP BY` et `HAVING`.

L'instruction GROUP BY

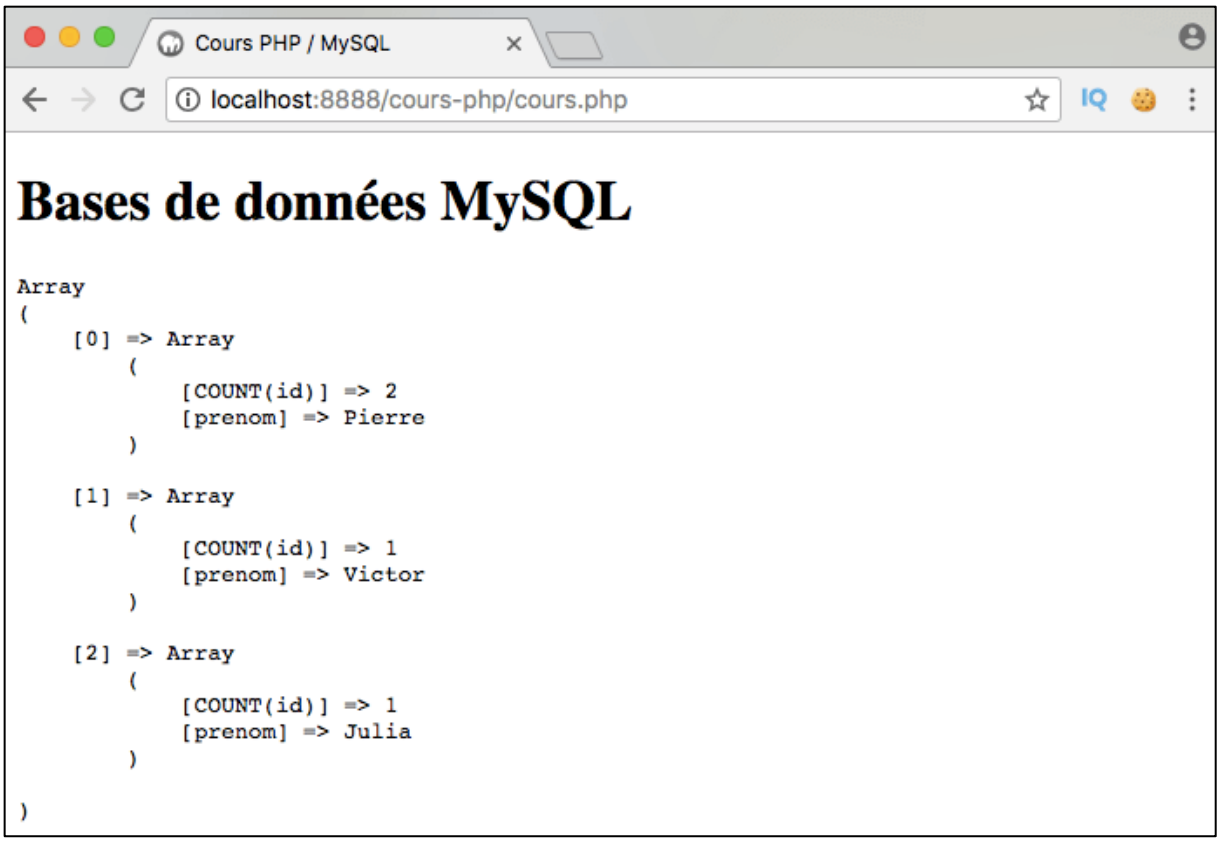
L'instruction SQL `GROUP BY` va généralement être utilisée avec les fonctions d'agrégation et en combinaison avec `ORDER BY`.

Cette instruction va nous permettre de grouper les résultats renvoyés selon une colonne.

En pratique, cette instruction va être utile pour regrouper des clients selon leur pays par exemple, ou encore selon le montant de leurs commandes, etc.

Dans notre table « users », nous allons ainsi par exemple pouvoir regrouper nos utilisateurs selon leur prénom et afficher le nombre de fois où un même prénom a été trouvé en utilisant la fonction `count()`.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Retourne tous les prénoms différents de la table users
         *ainsi que le nombre de fois où un prénom a été trouvé*/
        $sth = $dbco->prepare("
          SELECT COUNT(id), prenom FROM users
          GROUP BY prenom
          ORDER BY COUNT(id) DESC
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Ici, on commence par compter combien d'id (et donc combien d'entrées) nous avons dans la table avec `COUNT(id)`. On regroupe ensuite les résultats sélectionnés selon la valeur dans la colonne « prenom » de notre table. Finalement, on organise ces résultats selon le nombre de fois où un même prénom a été trouvé du plus grand nombre de fois au plus petit nombre de fois.

L'ordre des instructions est très important en SQL : si vous indiquez les différentes instructions dans n'importe quel ordre, vous avez de grandes chances pour que votre requête ne fonctionne pas du tout !

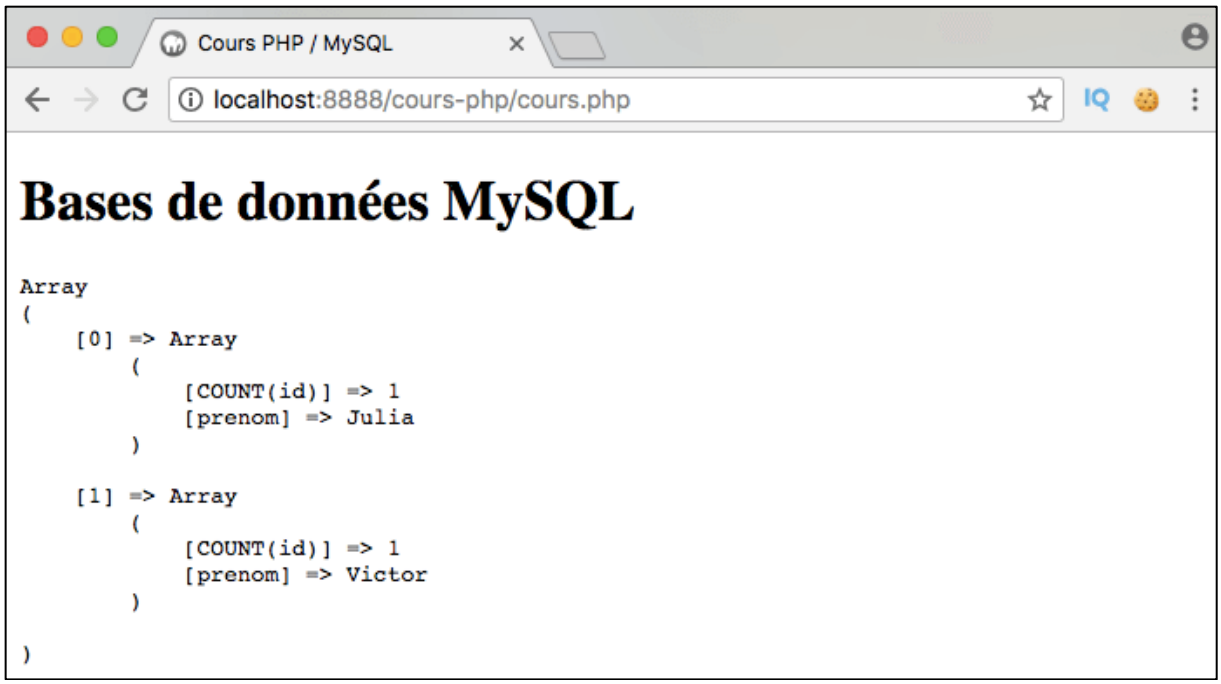
La clause SQL HAVING

La clause SQL `HAVING` remplace la clause `WHERE` dans le cas d'une utilisation avec les fonctions d'agrégation.

En effet, on ne peut tout simplement pas utiliser de clause `WHERE` avec des fonctions d'agrégation car la clause `WHERE` fonctionne avec les données de chaque ligne mais pas avec des données agrégées.

Nous allons par exemple pouvoir sélectionner uniquement les prénoms qui ne sont présents qu'une seule fois dans notre table et organiser le tout selon l'alphabet.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*N'affiche que les prénoms uniques dans notre table*/
        $sth = $dbco->prepare("
          SELECT COUNT(id), prenom FROM users
          GROUP BY prenom
          HAVING COUNT(id) < 2
          ORDER BY prenom
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```


Cours PHP / MySQL
localhost:8888/cours-php/cours.php
Bases de données MySQL
Array
(
[0] => Array
(
[COUNT(id)] => 1
[prenom] => Julia
)
[1] => Array
(
[COUNT(id)] => 1
[prenom] => Victor
)
)

PARTIE XV

Jointures, union & sous requêtes

Présentation des jointures SQL

Dans les leçons précédentes, nous avons effectué de nombreuses manipulations sur les bases de données mais n'avons jamais travaillé que sur les données d'une table à la fois.

Un des grands intérêts des bases de données et du SQL va être de pouvoir manipuler des données dans plusieurs tables à la fois.

Nous allons pouvoir faire cela grâce entre autres à la clause `JOIN` et à ce qu'on appelle en français les jointures SQL.

Manipuler des données de plusieurs tables : quel intérêt ?

Si votre base de données est bien construite, vous devriez normalement avoir de nombreuses tables contenant chacune des informations cohérentes et liées à un objet.

Pour un site e-commerce, vous allez probablement avoir une table « clients » qui contient des informations liées à vos clients (nom, prénom, adresse, adresse mail, mot de passe, etc.), une table « commandes » qui va lister les différentes commandes avec des informations liées à celle-ci (date, prix, nom du client, etc.) et etc.

Dans le cadre d'un blog, vous aurez certainement une table « utilisateur » dans laquelle vous allez stocker des informations liées aux utilisateurs, une table « commentaires » dans laquelle vous allez stocker le texte, la date, l'auteur etc. de chaque commentaire et etc.

Cette segmentation des données est ce qui fait tout l'intérêt et toute la puissance des bases de données en cela que nos informations sont organisées et accessibles de la meilleure façon qu'il soit et que les ressources sont optimisées.

En effet, imaginez qu'une personne commente régulièrement sur votre blog avec son compte créé chez vous. Il est hors de question de stocker toutes les informations liées à cet utilisateur à chaque fois qu'elle poste un nouveau commentaire dans la table « commentaires », cela serait une aberration en termes d'optimisation des ressources !

Nous allons stocker les informations du compte utilisateur une et une seule fois dans une table « utilisateurs » et ensuite les informations liées à chaque nouveau commentaire dans une table « commentaires ».

Vous pouvez ainsi vous douter qu'il va être souvent intéressant de puiser des informations dans plusieurs tables à la fois pour retourner des informations complètes et pertinentes. On va par exemple souvent vouloir récupérer les informations liées à un client et à une commande correspondante, ne serait-ce que pour remplir le bon de commande avec l'adresse du client (qui devrait se trouver dans la table « cilents ».

Principe de fonctionnement des jointures

Nous allons utiliser une clause de type `JOIN` pour combiner des entrées entre plusieurs tables.

Cependant, pour pouvoir « lier » plusieurs tables entre elles, il va nous falloir un liant, c'est-à-dire un point commun entre ces tables. Ce point commun sera une colonne qui contient les mêmes données à travers plusieurs tables.

En reprenant notre exemple de site e-commerce, par exemple, il semble tout à fait logique que nos tables « clients » et « commandes » possèdent toutes les deux une colonne contenant les noms de chaque client. Cela pourrait donc être notre colonne commune à partir de laquelle faire la jointure.

Cependant, en pratique, cette façon de faire serait très mauvaise tout simplement car un nom n'est pas quelque chose d'unique. Pour cette raison, nous allons souvent utiliser les colonnes id pour lier les données de plusieurs tables entre elles.

Pour faire cela, il va nous falloir donc dans le cas de notre site e-commerce une colonne « IDClient » dans chacune de nos deux tables. Si cela parait une nouvelle fois logique dans le cadre de notre table « clients », cela semble moins évident d'inclure une telle colonne dans notre table « commandes » à priori.

C'est bien là, et j'insiste sur ce point, que réside toute la difficulté de bien construire sa base de données : il faut bien réfléchir aux différentes situations que l'on pourrait rencontrer et aux opérations qu'on va vouloir effectuer dessus !

Les alias en SQL

Les jointures SQL vont nous permettre de travailler avec plusieurs tables en même temps. Nous allons donc être amenés à manipuler différentes colonnes appartenant à différente tables et donc il va être important de ne pas se perdre dans les différents noms. Les alias vont nous aider à cela.

En effet, les alias vont être utilisés pour donner à une colonne ou à une table complète un nom temporaire.

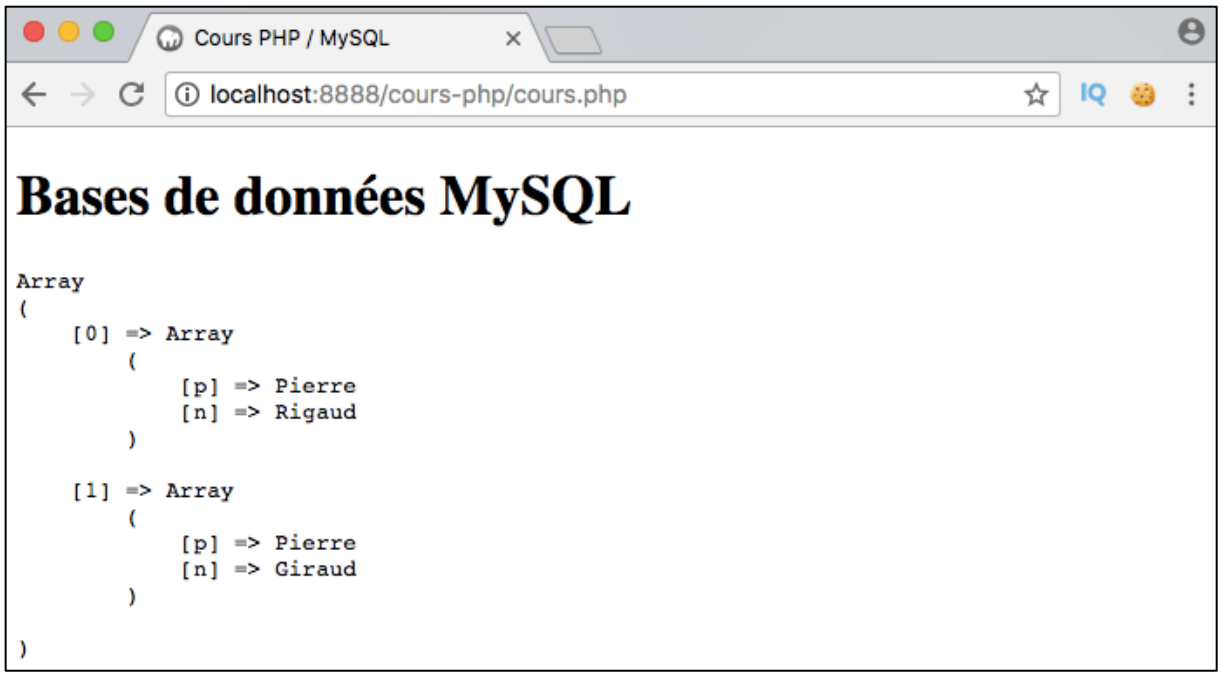
Pour créer un alias, nous utiliserons le mot `AS` (« en tant que » en français) en SQL.

Les alias vont ainsi nous permettre de renommer temporairement des tables ou des colonnes afin de les rendre plus faciles à manipuler et de rendre nos requêtes SQL plus lisibles.

Notez que l'alias, c'est-à-dire le nom donné de manière temporaire à nos colonnes ou à nos tables, n'existera que le temps de notre requête. Nos colonnes et tables ne sont bien entendus pas renommées pour toujours !

Voici un exemple concret d'utilisation des alias avec notre table « users » :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sth = $dbco->prepare("
          SELECT prenom AS p, nom as n FROM users AS U
          WHERE prenom = 'Pierre'
          ORDER BY n DESC
        ");
        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Dans l'exemple ci-dessus, nous donnons l'alias « p » à notre colonne « prenom », « n » à notre colonne « nom » et « U » à notre table « users ».

Ici, cependant, comme nous n'allons puiser des résultats que dans une table et comme notre requête est relativement simple l'intérêt des alias est limité.

Les alias vont ainsi s'avérer particulièrement utiles dans les cas où nous devons manipuler plusieurs tables à la fois, ou si le nom de nos colonnes ou tables est long ou peu pertinent par rapport aux données qu'elles contiennent, ou encore lorsque nous combinons plusieurs colonnes en une le temps d'une requête.

Attention, nous n'allons toutefois pas pouvoir utiliser les alias avec n'importe quelle clause en MySQL. En particulier, il est interdit d'utiliser les alias au sein d'une clause `WHERE` car en SQL standard, lorsqu'une clause `WHERE` est évalué la valeur de la colonne en soi peut ne pas encore avoir été déterminée.

Nous allons pouvoir utiliser les alias sans soucis avec les clauses `GROUP BY`, `ORDER BY` et `HAVING`.

Création de jointures

Nous allons pouvoir effectuer différents types de jointures SQL qui vont nous permettre de récupérer plus ou moins de données dans une table ou dans une autre.

Dans cette deuxième leçon consacrée aux jointures, nous allons découvrir les différents types de jointures à notre disposition en MySQL et comprendre comment et quand utiliser une jointure plutôt qu'une autre.

Les différents types de jointures

Il existe différents types de jointures en SQL mais tous ne sont pas supportés par le MySQL. Nous allons donc simplement nous concentrer sur les types de jointures suivants dans ce cours :

- L'`INNER JOIN` ;
- Le `LEFT (OUTER) JOIN` ;
- Le `RIGHT (OUTER) JOIN` ;
- Le `CROSS JOIN` ;
- Le `SELF JOIN` ;

Nous discuterons également d'un dernier type de jointures SQL qu'est le `FULL (OUTER) JOIN` qui n'est toujours pas utilisable en l'état en MySQL mais qu'on va pouvoir émuler en utilisant d'autres ressources.

Selon l'opération que nous voulons effectuer ou les données que nous voulons récupérer, nous utiliserons un type de jointure plutôt qu'un autre.

Les jointures de type interne, par exemple, ne vont nous retourner des résultats que lorsqu'il y aura une correspondance entre les deux tables, tandis que les jointures de type externe vont nous retourner des données même lorsqu'il n'y aura pas de correspondance dans la seconde table.

Préparation à l'utilisation des jointures : création d'une deuxième table

Avant tout, pour pouvoir tester de manière concrète les différents types de jointures et en apprécier tout l'intérêt, nous allons déjà créer une deuxième table dans notre base de données « cours » : la table « comments ».

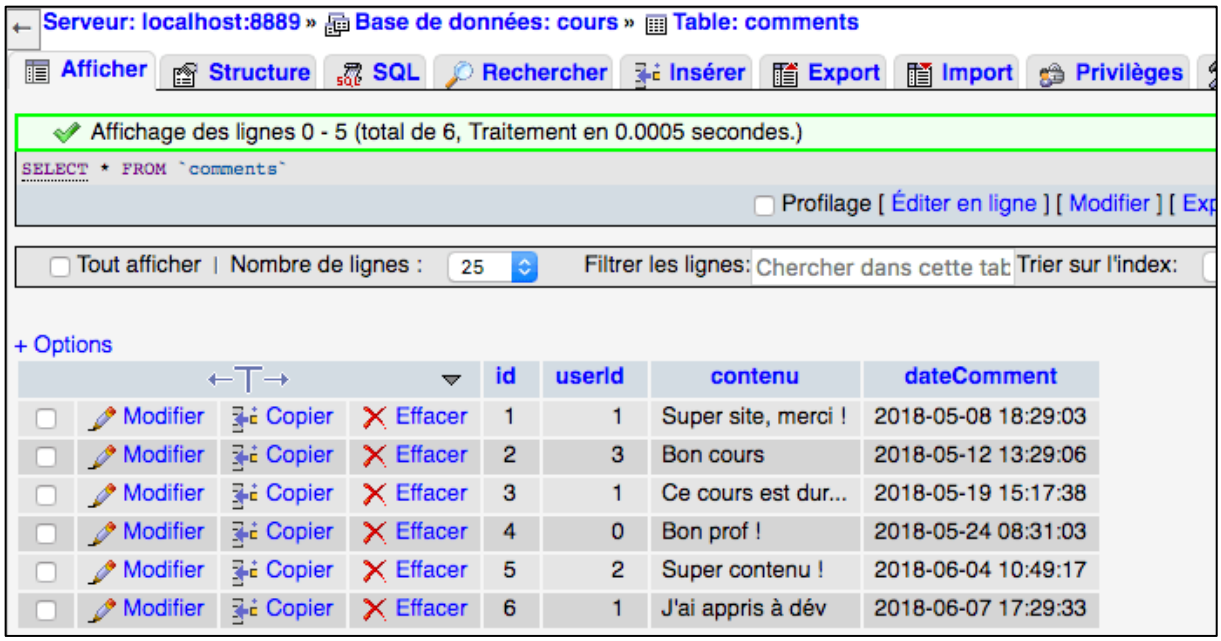
Cette table va être composée de 4 colonnes :

- Une colonne « id » INT UNSIGNED AUTO_INCREMENT PRIMARY KEY ;
- Une colonne « userId » INT DEFAULT 0 ;
- Une colonne « contenu » TEXT NOT NULL ;
- Une colonne dateComment TIMESTAMP ;

Vous pouvez trouver le script de création de la table et de l'insertion de quelques entrées ci-dessous. Notez que j'ai choisi de placer des valeurs au hasard dans la colonne « dateComment » pour plus de réalisme et pour pouvoir exploiter ces valeurs plus tard dans le script. Le format de date est ici « année-mois-jour heures:minutes:secondes ».

```php
<?php
  $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

  try{
    $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
    $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    //Crée la table comments
    $createTb = "CREATE TABLE comments(
      id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      userId INT DEFAULT 0,
      contenu TEXT NOT NULL,
      dateComment TIMESTAMP
    )";
    $dbco->exec($createTb);

    $sth = $dbco->prepare("
      INSERT INTO comments(userId, contenu, dateComment)
      VALUES (:userId, :contenu, :dateC)
      ");
    $sth->bindParam(':userId', $userId);
    $sth->bindParam(':contenu', $contenu);
    $sth->bindParam(':dateC', $dateC);

    //Insère des entrées avec des valeurs custom
    $userId = 1; $contenu = "Super site, merci !"; $dateC = "2018-05-08 18:29:03";
    $sth->execute();
    $userId = 3; $contenu = "Bon cours"; $dateC = "2018-05-12 13:29:06";
    $sth->execute();
    $userId = 1; $contenu = "Ce cours est dur..."; $dateC = "2018-05-19 15:17:38";
    $sth->execute();
    $userId = ''; $contenu = "Bon prof !"; $dateC = "2018-05-24 08:31:03";
    $sth->execute();
    $userId = 2; $contenu = "Super contenu !"; $dateC = "2018-06-04 10:49:17";
    $sth->execute();
    $userId = 1; $contenu = "J'ai appris à dév"; $dateC = "2018-06-07 17:29:33";
    $sth->execute();
  }

  catch(PDOException $e){
    echo "Erreur : " . $e->getMessage();
  }
?>
```

Et voici donc les entrées qui devraient être créées dans votre table (vous pouvez aller vérifier cela dans phpMyAdmin) :



				id	userId	contenu	dateComment
Modifier	Copier	Effacer		1	1	Super site, merci !	2018-05-08 18:29:03
Modifier	Copier	Effacer		2	3	Bon cours	2018-05-12 13:29:06
Modifier	Copier	Effacer		3	1	Ce cours est dur...	2018-05-19 15:17:38
Modifier	Copier	Effacer		4	0	Bon prof !	2018-05-24 08:31:03
Modifier	Copier	Effacer		5	2	Super contenu !	2018-06-04 10:49:17
Modifier	Copier	Effacer		6	1	J'ai appris à dév	2018-06-07 17:29:33

Nous avons donc maintenant deux tables dans notre base de données « cours » : la table « users » et la table « comments ».

Notez que chacune de ces deux tables possède une colonne avec des Id utilisateurs. Comme vous vous en doutez, nous allons nous servir de ces valeurs pour effectuer nos jointures.

L'INNER JOIN

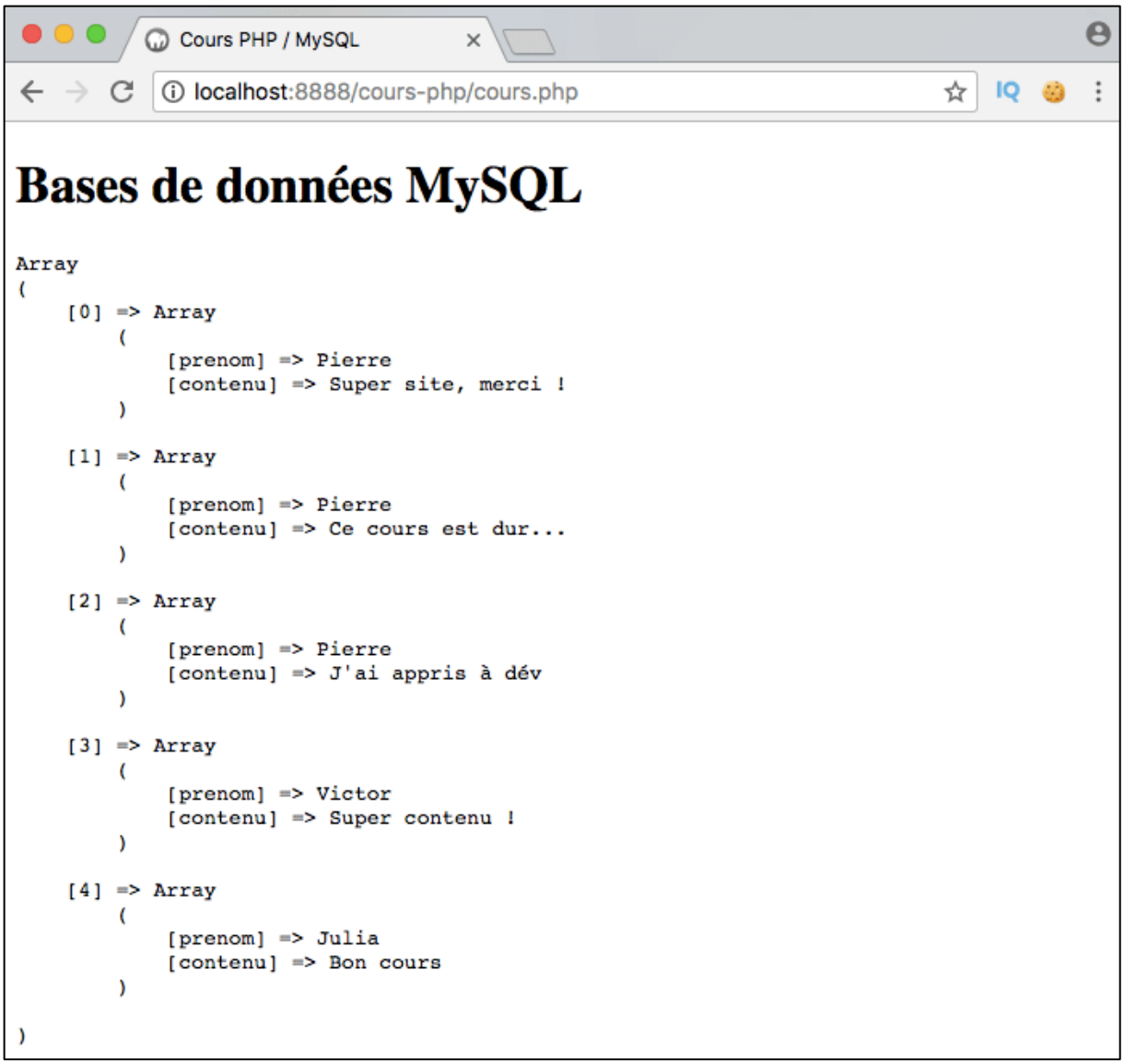
Nous allons pouvoir créer une jointure interne à l'aide du mot clef `INNER JOIN` en SQL.

Une jointure interne est un type de jointures qui va nous permettre de ne sélectionner que les donnes relatives aux entrées qui ont des valeurs identiques dans les deux tables.

Pour appliquer un `INNER JOIN`, nous allons avoir besoin d'une colonne de référence dans chacune des deux tables, c'est-à-dire de deux colonnes qui ont des valeurs qui représentent les mêmes choses.

Nous allons par exemple pouvoir récupérer tous les commentaires de tous les utilisateurs connus d'un coup :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Sélectionne tous les users qui ont commenté et tous les
         *commentaires liés à un user connu*/
        $sth = $dbco->prepare("
          SELECT users.prenom, comments.contenu
          FROM users
          INNER JOIN comments ON users.id = comments.userId
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Ici, notre requête sélectionne toutes les données des colonnes « prenom » de la table « users » et « contenu » de la table « comments » pour les entrées où la valeur dans la colonne « id » de « users » trouve un équivalent dans la colonne « userId » de « comments ».

Les commentaires sans utilisateurs ET les utilisateurs qui n'ont pas commenté seront donc exclus des résultats renvoyés.

Notez bien la nouvelle syntaxe employée dans cet exemple : maintenant que nous travaillons avec plusieurs tables, nous ne pouvons plus nous permettre de simplement indiquer le nom d'une colonne seul car il pourrait y avoir ambiguïté dans le cas où les deux tables sur lesquelles nous travaillons aient une colonne portant le même nom. Nous écrirons donc désormais « nom_de_la_table.nom_de_la_colonne » pour éviter toute ambiguïté.

Note : Dans ce cours, nous allons nous contenter d'effectuer des jointures sur deux tables. Cependant, rien ne vous empêche d'aller récupérer des données dans trois tables à la fois par exemple. Pour cela, il suffira d'effectuer deux `INNER JOIN` dans la même requête

: 1 premier sur une colonne commune aux tables 1 et 2 et un second sur une colonne commune aux tables 2 et 3.

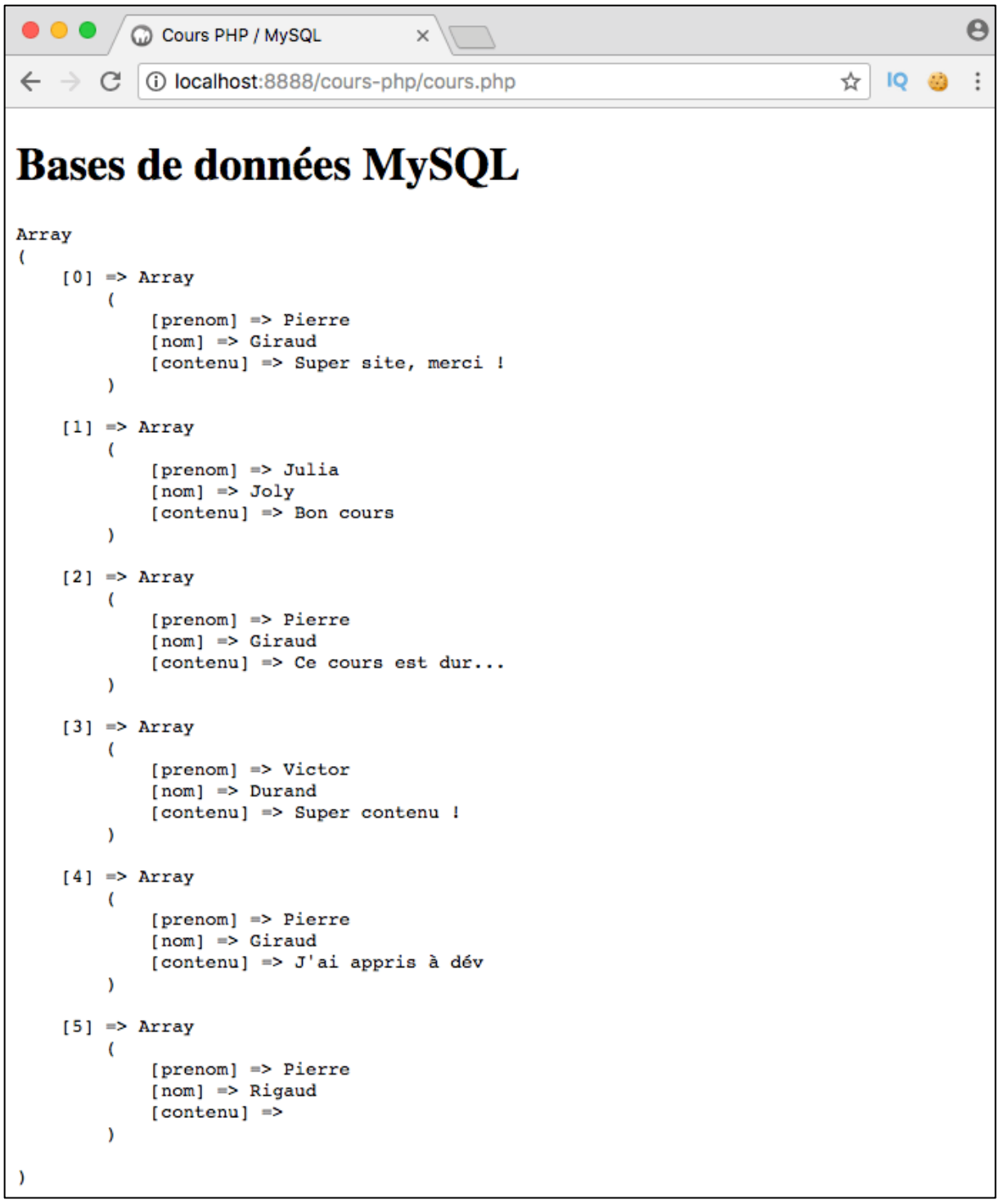
Le LEFT (OUTER) JOIN

Le `LEFT JOIN` (également appelé `LEFT OUTER JOIN`) est un type de jointures externes.

Ce type de requête va nous permettre de récupérer toutes les données (relatives aux colonnes spécifiées) de la table de gauche c'est-à-dire de notre table de départ ainsi que les données de la table de droite (table sur laquelle on fait la jointure) qui ont une correspondance dans la table de gauche.

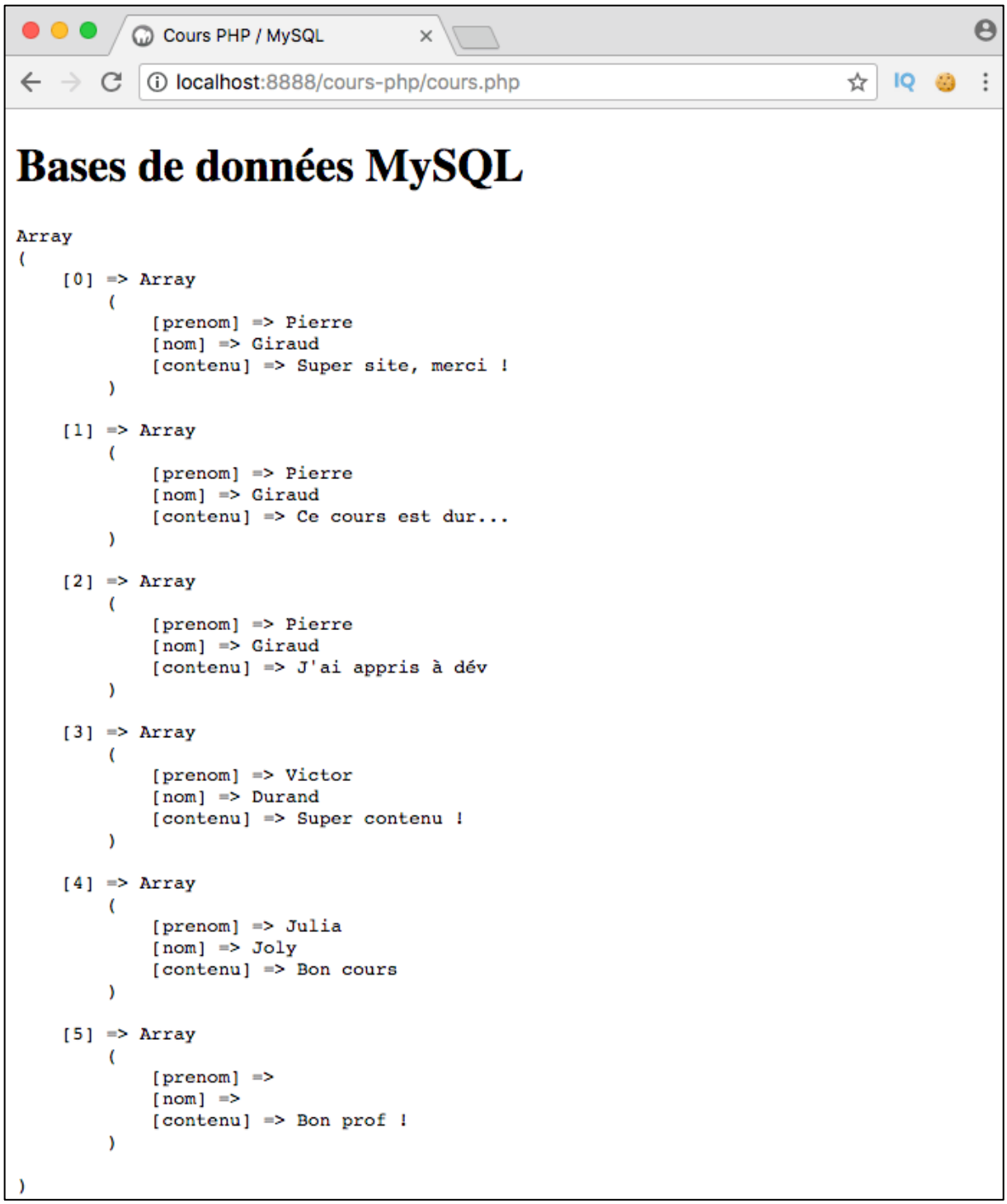
Avec ce type de requête, nous allons par exemple pouvoir récupérer tous les noms et prénoms de nos utilisateurs dans notre table « users » et SEULEMENT les commentaires liés à un utilisateur de notre table « comments ».

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Sélectionne tous les prenoms et noms dans la table users
         *ainsi que les contenus des commentaires liés à un utilisateur
         *de la table comments*/
        $sth = $dbco->prepare("
          SELECT users.prenom, users.nom, comments.contenu
          FROM users
          LEFT OUTER JOIN comments ON users.id = comments.userId
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Notez ici qu'il suffit « d'inverser » notre requête (c'est-à-dire utiliser notre table « comments » comme table de base et la table « users » comme table sur laquelle on fait la jointure) pour au contraire récupérer les contenus de tous les commentaires et UNIQUEMENT les noms et prénoms des utilisateurs qui ont commenté.

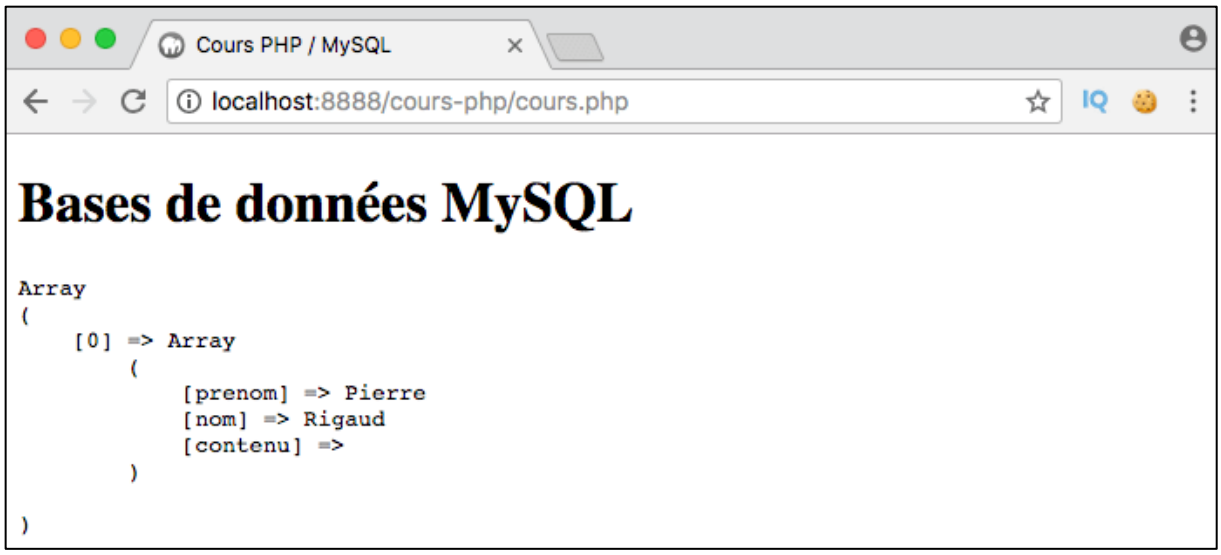
```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Comments est désormais notre table de base et on effectue notre
         *jointure sur la table users*/
        $sth = $dbco->prepare("
          SELECT users.prenom, users.nom, comments.contenu
          FROM comments
          LEFT OUTER JOIN users ON users.id = comments.userId
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Notez également qu'une petite astuce simple va nous permettre d'exclure les données qui ont une référence commune dans les deux tables et donc de ne récupérer que les données relatives à notre table de départ.

Pour cela, il va suffire d'utiliser une clause `WHERE` en demandant une valeur nulle sur la colonne de la table de droite (table sur laquelle on fait la jointure).

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Récupère les noms et prénoms des utilisateurs n'ayant pas
         *commenté*/
        $sth = $dbco->prepare("
          SELECT users.prenom, users.nom, comments.contenu
          FROM users
          LEFT OUTER JOIN comments ON users.id = comments.userId
          WHERE comments.userId IS NULL
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



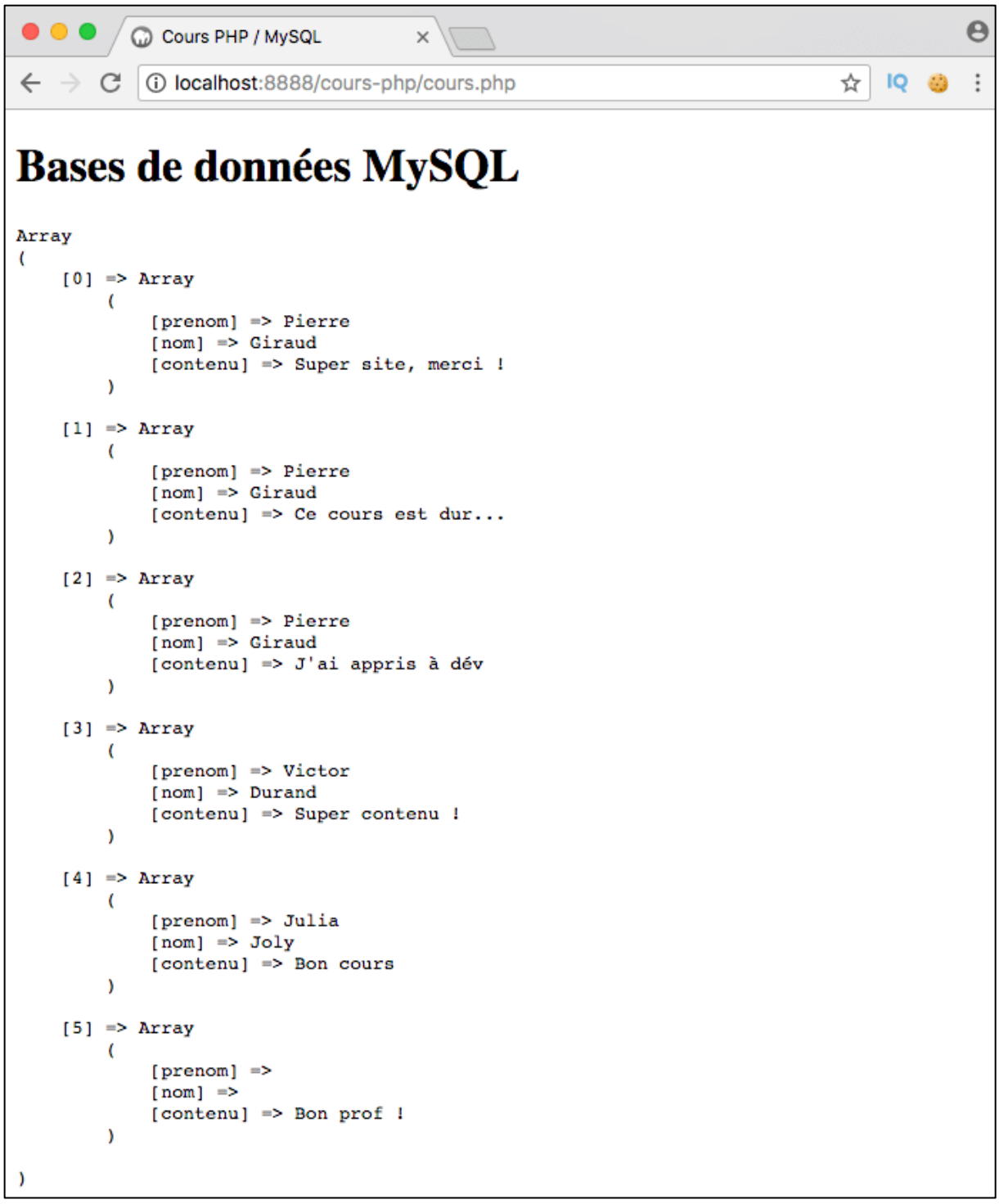
Ici, par exemple, nous récupérons seulement les noms et prénoms des utilisateurs n'ayant pas posté de commentaires.

Le RIGHT (OUTER) JOIN

Le `RIGHT JOIN`, encore appelé `RIGHT OUTER JOIN` est un autre type de jointures externes qui va fonctionner de la même façon qu'un `LEFT JOIN` à la différence que cette fois-ci ce sont toutes les données de la table de droite (table sur laquelle on effectue la jointure qui vont être récupérées tandis que seules les entrées de la table de gauche ou table de départ qui vont satisfaire à la condition de jointure seront sélectionnées.

La requête ci-dessous par exemple va bien récupérer le contenu de tous les commentaires dans la table « comments » sans exception et SEULEMENT les noms et prénoms des personnes qui ont posté un commentaire.

```php
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Récupère le contenu de tous les commentaires et seulement les
         *noms et prénoms des utilisateurs qui ont commenté*/
        $sth = $dbco->prepare("
          SELECT users.prenom, users.nom, comments.contenu
          FROM users
          RIGHT OUTER JOIN comments ON users.id = comments.userId
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Nous allons bien évidemment pouvoir effectuer un `RIGHT JOIN` qui ne va pas récupérer les données présentes dans les deux colonnes mais seulement celles dans la colonne de droite en procédant exactement de la même façon qu'avec notre `LEFT JOIN` c'est-à-dire en utilisant une clause `WHERE`.

Le FULL (OUTER) JOIN

Le `FULL JOIN` ou `FULL OUTER JOIN` est un type de jointures externes qui va récupérer toutes les données pour les colonnes sélectionnées de chacune des deux tables.

Attention cependant : ce type de jointures n'est pas supporté en par MySQL.

Nous allons toutefois pouvoir simuler le comportement d'un `FULL JOIN` en utilisant intelligemment une combinaison de `LEFT JOIN` et de `RIGHT JOIN` avec une clause `WHERE` et en utilisant l'opérateur `UNION`.

Nous verrons comment faire cela lors de la prochaine leçon quand nous apprendrons à utiliser ce nouvel opérateur.

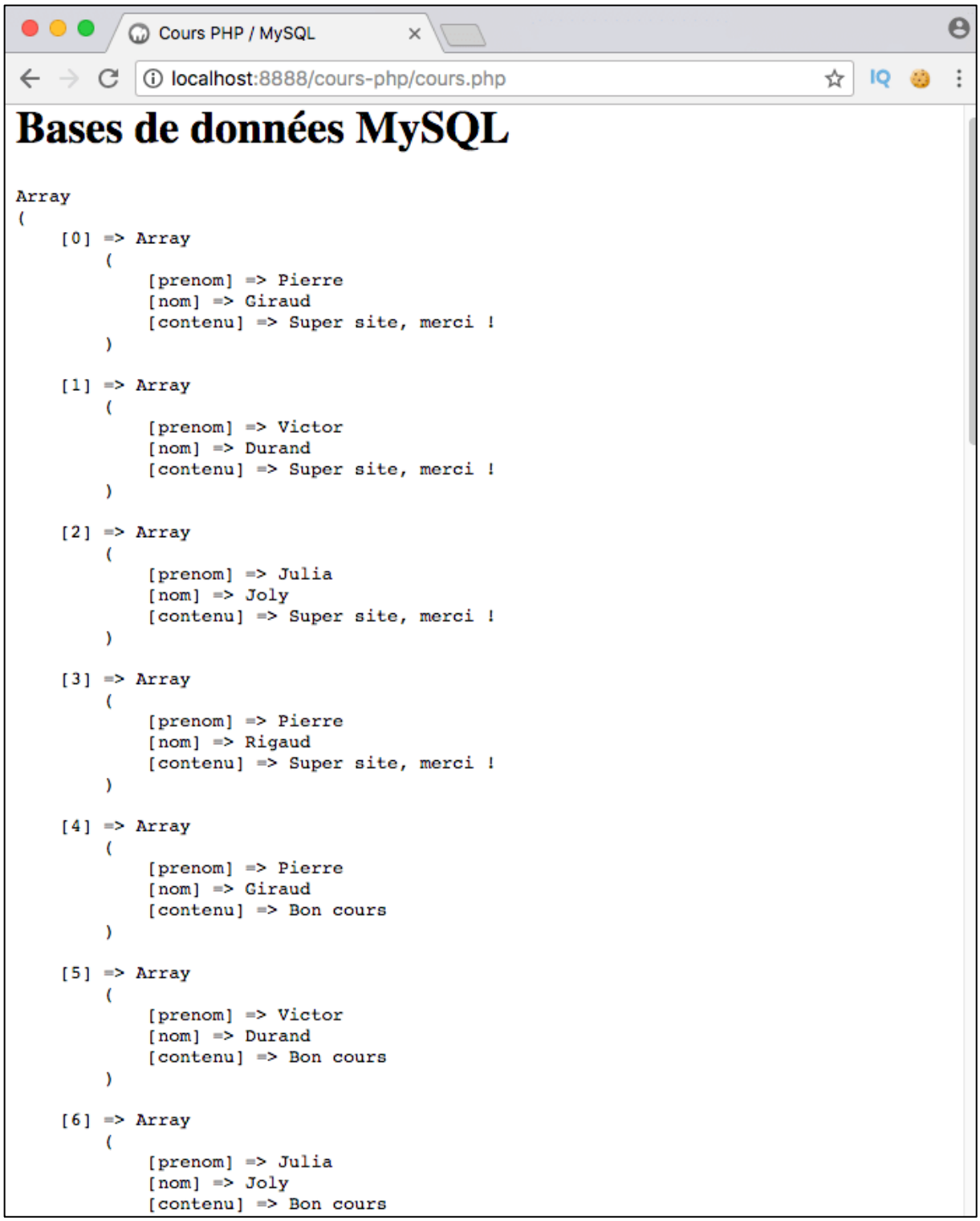
Le CROSS JOIN

Le `CROSS JOIN` en MySQL va retourner une liste de résultat qui va être le produit des entrées des deux tables jointes lorsqu'aucune clause `WHERE` n'est utilisée.

Dans une requête SQL de type `SELECT`, par exemple, chacune des lignes de la première table va être accouplée à chacune des lignes de la seconde table pour former à chaque fois une nouvelle ligne qui va être renvoyée.

En pratique, ce type de jointure est peu utilisé car souvent peu pertinent. Voyons tout de même comment il fonctionne :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*RAffiche le produit cartésien des résultats des deux tables*/
        $sth = $dbco->prepare("
          SELECT users.prenom, users.nom, comments.contenu
          FROM users
          CROSS JOIN comments
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Comme vous pouvez le voir dans cet exemple, le contenu de chaque commentaire va être joint aux noms et prénoms trouvés dans notre table « users » pour venir former un résultat qui va être affiché.

Note : Je n'ai affiché qu'un échantillon des résultats retournés en photo ici.

Le SELF JOIN

Un SELF JOIN représente une jointure d'une table avec elle-même. Le mot clef SELF JOIN n'existe pas, nous allons simplement écrire notre jointure comme on a pu le faire précédemment mais à la différence que nos deux tables seront la même.

Les SELF JOIN vont être utiles pour des tables dans lesquelles il y a un lien hiérarchique entre les données de deux colonnes (employé / patron, enfant / parent) et qui se référencent elles-mêmes.

Pour illustrer l'utilité de faire des SELF JOIN, je vous propose l'exemple suivant : imaginons une table « employes » qui contient des informations sur les employés ainsi que sur leur manager.

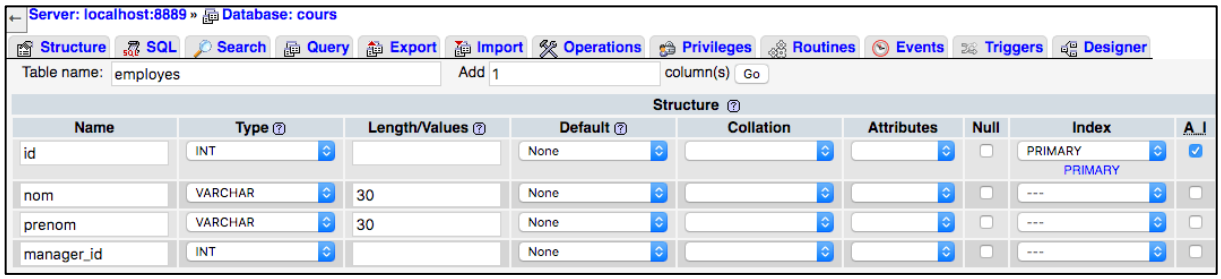
Chaque employé va avoir un manager qui va lui-même être l'employé d'un autre manager de niveau hiérarchique supérieur jusqu'à arriver au grand patron.

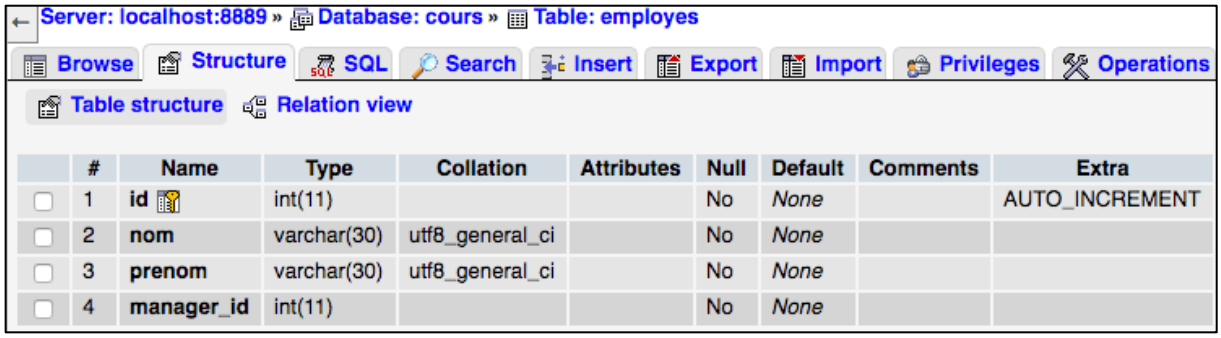
Notre table va donc contenir 4 colonnes :

- Une colonne id ;
- Une colonne nom ;
- Une colonne prenom ;
- Une colonne id_manager ;

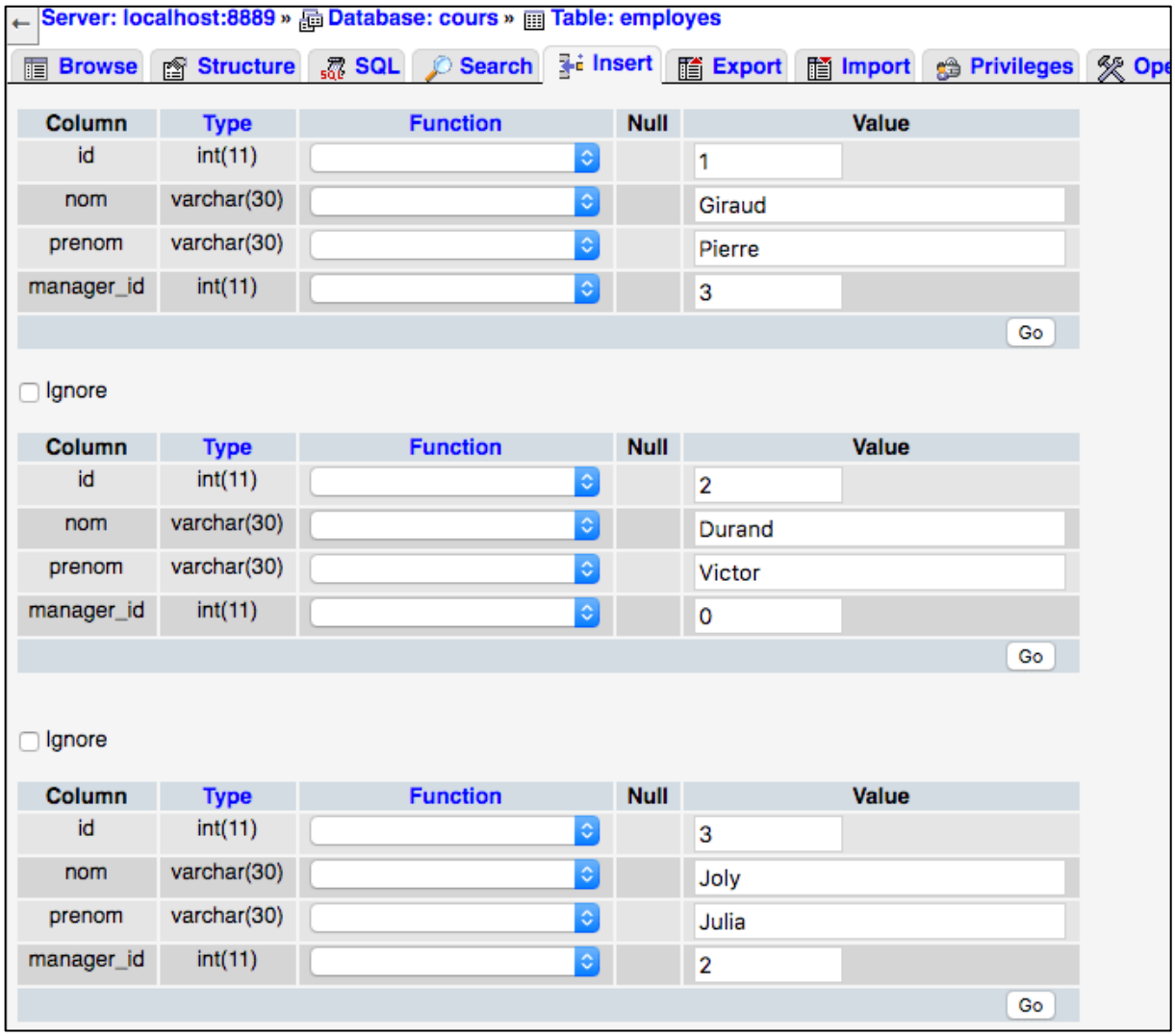
La colonne « id_manager » va faire référence à la colonne « id » puisqu'encore une fois chaque manager est également un employé.

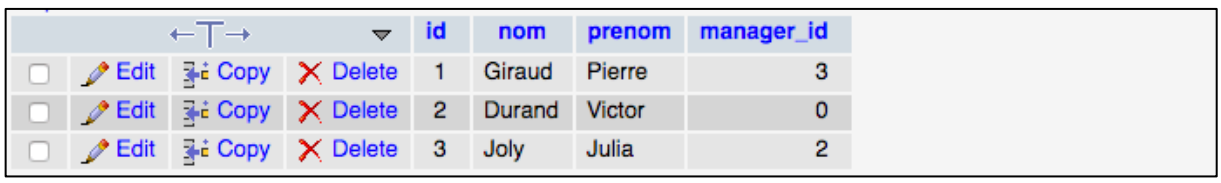
Voici à quoi ressemble la structure de notre table que vous pouvez créer comme moi directement avec phpMyAdmin :





Nous allons insérer trois entrées dans cette table pour les besoins de notre exemple :





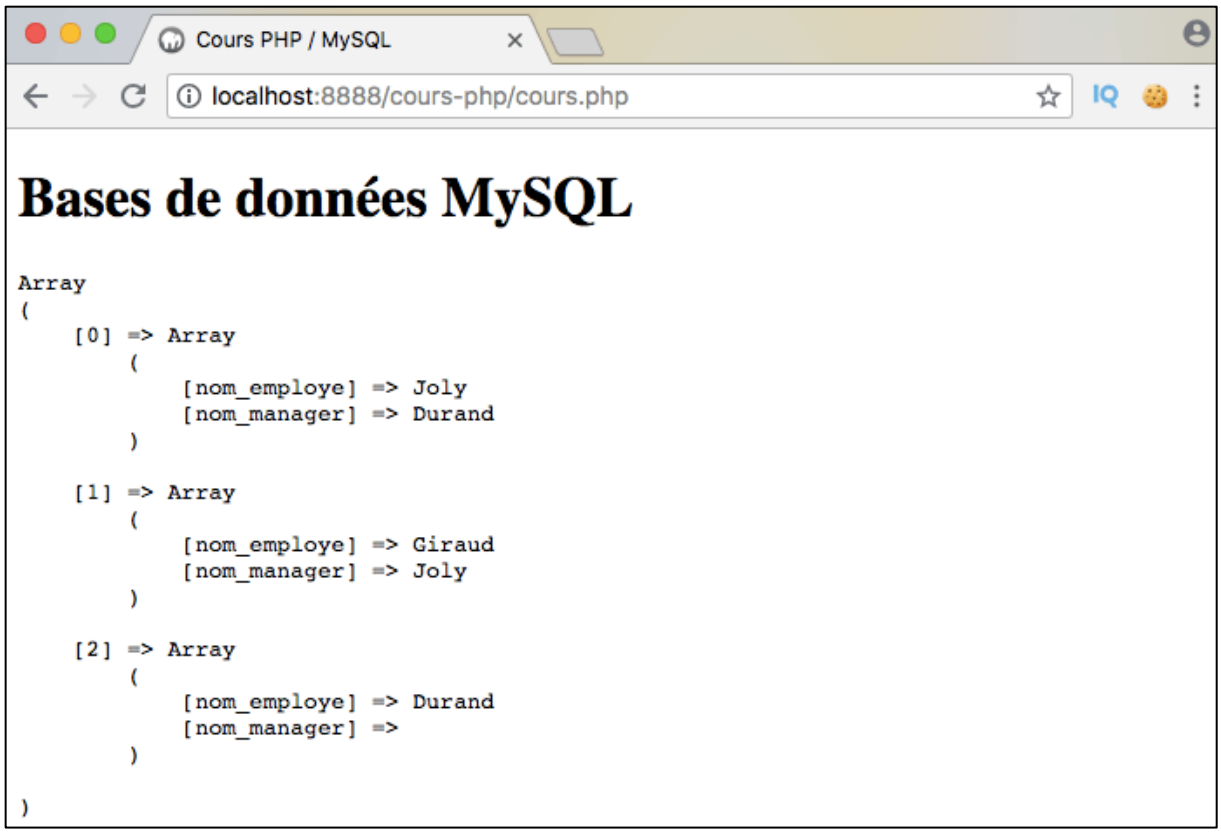
Dans notre table, ici, « Pierre Giraud » est simplement un employé mais n'est le manager d'aucune autre personne. Son manager est l'employé portant l'id = 3, c'est-à-dire « Julia Joly ».

« Julia Joly » est donc manager de « Pierre Giraud » mais est également une employée de la société. Son manager est l'employé portant l'id = 2, c'est-à-dire « Victor Durand ».

« Victor Durand » est ici le Président de l'entreprise : il n'a pas de manager. Nous avons donc mis « 0 » en manager_id afin qu'il n'y ait pas de référence dans la colonne id de notre table. Cependant, même en étant Président, « Victor Durand » reste tout de même un employé (particulier, je vous l'accorde) au sens propre de la société.

Notre but va donc être maintenant d'afficher en même temps le nom d'un employé et le nom de son manager. Pour faire cela, nous allons avoir besoin d'utiliser un SELF JOIN.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Jointure d'une table avec elle même*/
        $sth = $dbco->prepare("
          SELECT e.nom AS nom_employe, m.nom AS nom_manager
          FROM employes e
          LEFT OUTER JOIN employes m
          ON e.manager_id = m.id
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Ici, on commence par sélectionner le nom de nos employés dans la table ainsi que le nom de nos managers. Rappelez-vous bien qu'on va chercher ces noms dans la même colonne puisque les managers des uns sont également les employés des autres.

Lorsqu'on fait une jointure d'une table avec elle-même, nous sommes obligés d'utiliser des alias afin d'éviter qu'il y ait ambiguïté sur le nom de la table.

Ici, nous utilisons donc deux alias pour notre table « employes » : « e » et « m » qui représentent une nouvelle fois la même table.

Dans cet exercice, nous allons ensuite utiliser un `LEFT OUTER JOIN`. Rappelez-vous que le mot clef `SELF JOIN` n'existe pas mais sert simplement à représenter un concept. On va pouvoir faire des `SELF JOIN` en utilisant des `INNER JOIN`, `LEFT OUTER JOIN`, `RIGHT OUTER JOIN`, etc.

Ici, nous allons donc récupérer les noms de tous les employés ainsi que les noms de leur manager associé. Pour cela, on effectue notre jointure avec la condition de la similitude des valeurs dans la colonne « manager_id » et de celle de la colonne « id ».

Lorsqu'une valeur dans la colonne « manager_id » trouvera un équivalent dans la colonne « id » de notre table, on renverra la valeur dans la colonne « nom » de l'entrée associée avec la clef « nom_manager » ainsi que la valeur de la colonne nom de notre table de départ avec la clef « nom_employe ».

L’opérateur SQL UNION

Dans cette leçon, nous allons découvrir un nouvel opérateur SQL : l’opérateur `UNION` et allons apprendre à l’utiliser pour combiner les résultats de deux déclarations `SELECT` ou pour simuler le comportement d’un `FULL OUTER JOIN`.

Conditions d’utilisation de l’opérateur SQL UNION

Nous allons utiliser l’opérateur SQL `UNION` pour combiner les résultats de deux déclarations `SELECT`.

Afin que cet opérateur fonctionne correctement et que les résultats puissent bien être combinés, il va falloir respecter les règles suivantes :

- Chacun des `SELECT` dans un `UNION` doit posséder le même nombre de colonnes ;
- Les colonnes doivent posséder le même type de données une à une dans chaque `SELECT` (la première colonne du premier `SELECT` doit posséder des données de même type que la première colonne du deuxième `SELECT` et etc.).

Notez bien que les colonnes sélectionnées peuvent avoir un nom différent entre les différents `SELECT` du moment que le type de données est le même. En cas de nom différent, et si nous voulons harmoniser les noms, nous pouvons tout à fait utiliser un alias.

Par défaut, l’opérateur SQL `UNION` ne va pas sélectionner les doublons dans les valeurs (les valeurs qui sont trouvées plusieurs fois n’apparaitront qu’une seule fois).

Nous allons par exemple pouvoir sélectionner la liste des noms et prénoms dans nos tables « users » et « employes » créées précédemment :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*L'opérateur SQL UNION*/
        $sth = $dbco->prepare("
          SELECT nom, prenom FROM employes
          UNION
          SELECT nom, prenom FROM users
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Comme vous pouvez le voir, les valeurs dupliquées n'apparaissent qu'une fois. Notez bien qu'ici on sélectionne les noms et prénom dans chaque table qui vont représenter une entité et c'est bien sur cette entité nom + prénom que va être évaluée la duplication.

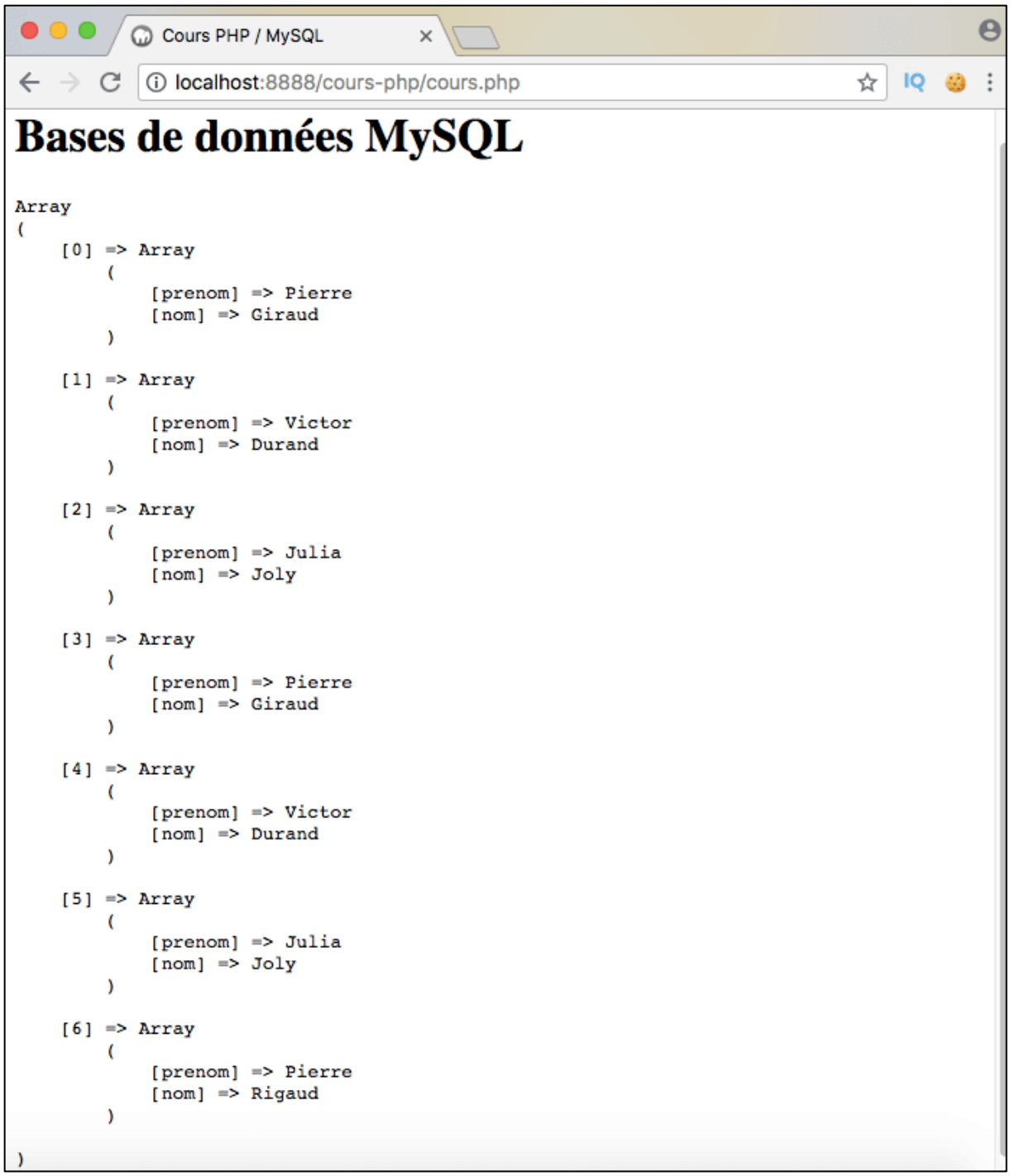
C'est pour cela qu'on a deux fois le prénom « Pierre » qui s'affiche, tout simplement car nos « Pierre » n'ont pas le même nom.

Si nous n'avions sélectionnés que les prénoms, le prénom « Pierre » ne se serait affiché qu'une fois :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*L'opérateur SQL UNION*/
        $sth = $dbco->prepare("
          SELECT prenom FROM employes
          UNION
          SELECT prenom FROM users
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Si vous souhaitez obtenir toutes les valeurs (incluant les valeurs en double et etc.) alors il faudra utiliser `UNION ALL`.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*UNION ALL*/
        $sth = $dbco->prepare("
          SELECT prenom, nom FROM employes
          UNION ALL
          SELECT prenom, nom FROM users
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Utiliser l'opérateur UNION pour simuler un FULL OUTER JOIN en MySQL

Rappel : un `FULL OUTER JOIN` est un type de jointures externes qui va récupérer toutes les données pour les colonnes sélectionnées de chacune des deux tables.

Le mot clef `FULL OUTER JOIN` n'est pas supporté en tant que tel par MySQL. Nous allons cependant pouvoir simuler un `FULL OUTER JOIN` en utilisant un `LEFT JOIN` et un `RIGHT JOIN` avec une clause `WHERE` pour ne pas avoir les données présentes dans les deux

tables en double dans notre résultat final et en combinant les résultats obtenus avec un `UNION`.

Pour rappel, un `LEFT JOIN` va nous permettre de récupérer toutes les données de notre table de gauche ainsi que les données de la table de droite qui satisfont à la condition dans la jointure.

Un `RIGHT JOIN` va lui nous permettre de récupérer toutes les données de notre table de droite ainsi que les données de la table de gauche qui satisfont à la condition dans la jointure.

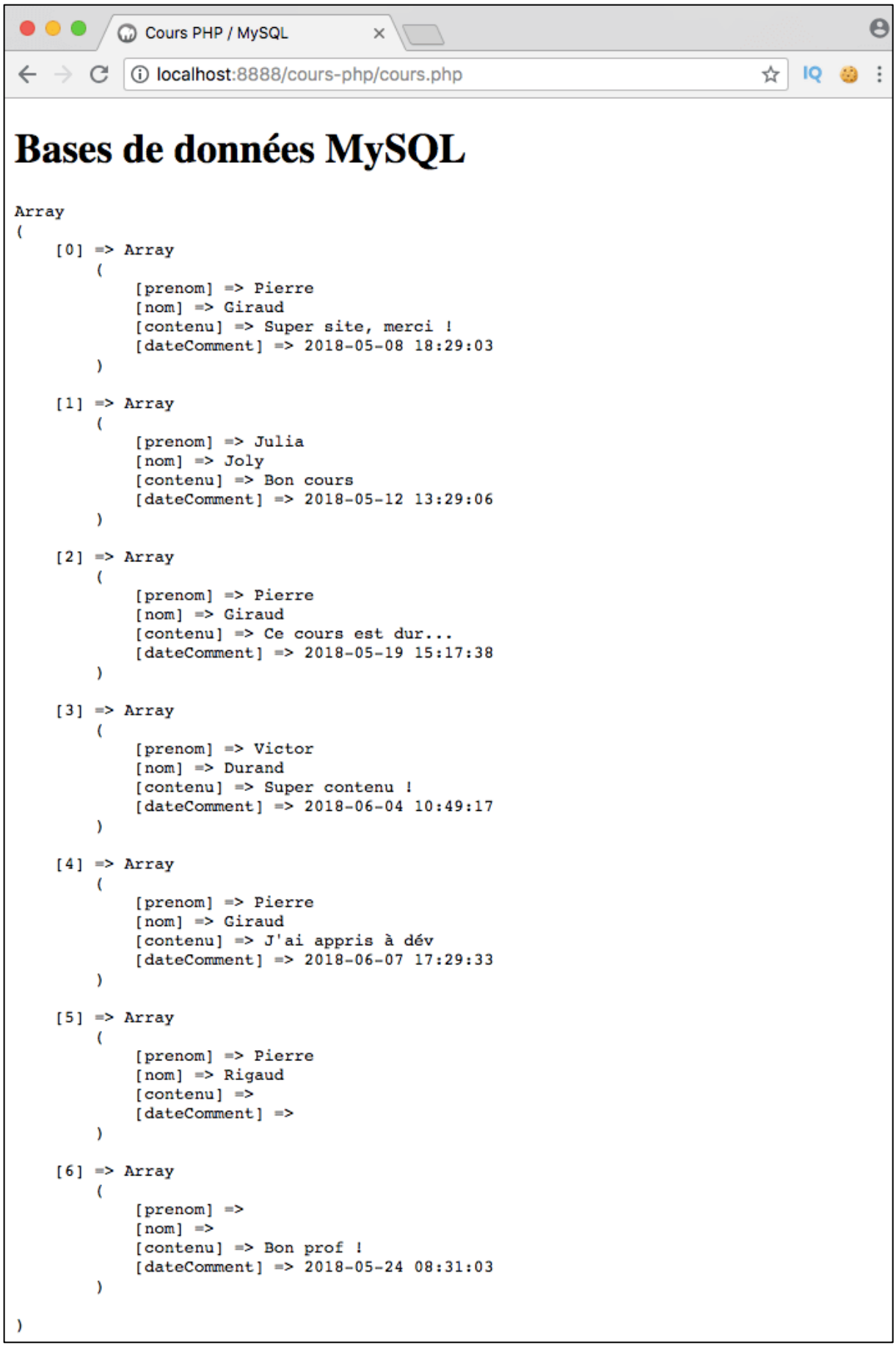
Ainsi, en s'arrêtant là, nous allons avoir les données de la table de droite qui satisfont à la condition de notre `LEFT JOIN` et les données de la table de gauche qui satisfont à la condition de notre `RIGHT JOIN` en double (puisque notre `RIGHT JOIN` récupère déjà toutes les données de la table de droite et notre `LEFT JOIN` récupère déjà toutes les données de la table de gauche).

Ainsi, nous allons utiliser une clause `WHERE` soit dans notre `LEFT JOIN` soit dans notre `RIGHT JOIN` pour ne pas sélectionner plus de données que ce qu'on doit.

Attention, on ne parle pas ici de supprimer les résultats dupliqués, on veut seulement tenter de simuler le résultat d'un `FULL OUTER JOIN` et pour cela il ne faut pas aller sélectionner deux fois les mêmes résultats. Il est possible que l'on ait des valeurs dupliquées au final s'il y en a dans nos colonnes à la base.

Essayons de simuler le comportement d'un `FULL OUTER JOIN` qui sélectionnerait tous les prénoms et noms de notre table « users » et tous les contenus et les dates des commentaires de notre tables « comments » :

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Simulation d'un FULL OUTER JOIN*/
        $sth = $dbco->prepare("
          SELECT u.prenom, u.nom, c.contenu, c.dateComment FROM users AS u
          LEFT JOIN comments AS c ON u.id = c.userId
          UNION ALL
          SELECT u.prenom, u.nom, c.contenu, c.dateComment FROM users AS u
          RIGHT JOIN comments AS c ON u.id = c.userId
          WHERE u.id IS NULL
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Bases de données MySQL

```
Array
(
    [0] => Array
        (
            [prenom] => Pierre
            [nom] => Giraud
            [contenu] => Super site, merci !
            [dateComment] => 2018-05-08 18:29:03
        )

    [1] => Array
        (
            [prenom] => Julia
            [nom] => Joly
            [contenu] => Bon cours
            [dateComment] => 2018-05-12 13:29:06
        )

    [2] => Array
        (
            [prenom] => Pierre
            [nom] => Giraud
            [contenu] => Ce cours est dur...
            [dateComment] => 2018-05-19 15:17:38
        )

    [3] => Array
        (
            [prenom] => Victor
            [nom] => Durand
            [contenu] => Super contenu !
            [dateComment] => 2018-06-04 10:49:17
        )

    [4] => Array
        (
            [prenom] => Pierre
            [nom] => Giraud
            [contenu] => J'ai appris à dév
            [dateComment] => 2018-06-07 17:29:33
        )

    [5] => Array
        (
            [prenom] => Pierre
            [nom] => Rigaud
            [contenu] =>
            [dateComment] =>
        )

    [6] => Array
        (
            [prenom] =>
            [nom] =>
            [contenu] => Bon prof !
            [dateComment] => 2018-05-24 08:31:03
        )

)
```

Ici, le premier `SELECT` sélectionne tous les prénoms et noms de notre table « users » ainsi que les contenus et dates de nos commentaires liés à un id utilisateur.

Le deuxième `SELECT` sélectionne lui uniquement les contenus et dates des commentaires qui n'ont pas d'utilisateur associé (`WHERE u.id IS NULL`).

En unissant ces deux `SELECT` avec un `UNION ALL`, cela nous permet donc d'obtenir tous les noms et prénoms de nos utilisateurs et tous les contenus et dates des commentaires.

Les opérateurs de sous requête

Une sous-requête est tout simplement une requête imbriquée dans une autre requête. Nous allons utiliser des sous requêtes lorsque nous devrons faire des opérations en plusieurs étapes, comme par exemple faire la somme des valeurs de plusieurs colonnes puis calculer la moyenne.

Dans cette leçon, nous allons découvrir de nouveaux opérateurs SQL que l'on va pouvoir utiliser spécifiquement conjointement avec des sous-requêtes.

Nous allons donc ici nous concentrer sur :

- L'opérateur SQL `EXISTS` ;
- L'opérateur SQL `ANY` ;
- L'opérateur SQL `ALL`.

L'opérateur SQL EXISTS

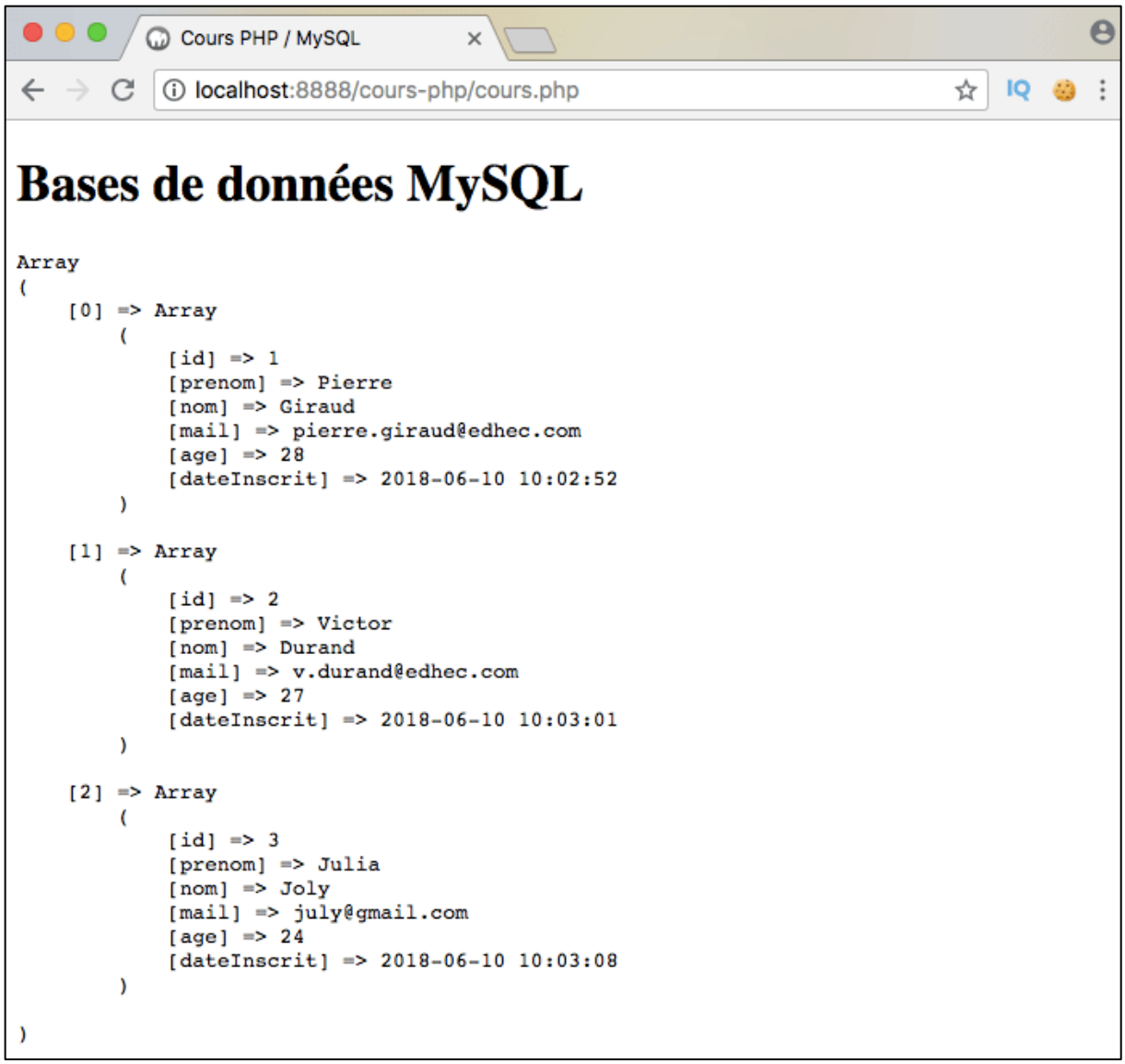
L'opérateur SQL `EXISTS` va nous servir à tester l'existence d'une entrée dans une sous requête.

Cet opérateur renvoie `TRUE` si au moins un résultat a été trouvé. La requête principale ne s'exécutera que si la sous requête retourne au moins un résultat.

En pratique, on utilisera principalement cet opérateur pour retourner des résultats à partir d'une table seulement si une certaine condition est vérifiée dans une autre table.

Nous allons par exemple pouvoir renvoyer les informations relatives à chaque utilisateur qui a commenté sur notre site. Pour cet exemple, nous nous basons à nouveau sur les tables « users » et « comments » créées précédemment.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Renvoie les informations de chaque user qui a commenté*/
        $sth = $dbco->prepare("
          SELECT * FROM users
          WHERE EXISTS (SELECT * FROM comments WHERE comments.userId = users.id)
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```



Ici, si notre sous-requête renvoie au moins un résultat, c'est-à-dire si la condition `comments.userId = users.id` (si on trouve au moins un utilisateur enregistré dans la table « users » lié à un commentaire) alors `EXISTS` va retourner `TRUE` et notre requête principale va pouvoir s'exécuter.

En d'autres mots, toutes les informations de chaque utilisateur ayant commenté seront renvoyées.

L'opérateur SQL ANY

L'opérateur SQL `ANY` va être utilisé conjointement avec une clause `WHERE` ou une clause `HAVING` dans le cas où nous utilisons une sous-requête dans notre requête.

Cet opérateur va nous permettre de comparer une valeur avec le résultat d'une sous-requête.

ANY va retourner TRUE si au moins l'une des valeurs de la sous-requête satisfait à la condition imposée et va ainsi permettre de poursuivre l'exécution de la requête principale.

Par exemple, nous allons pouvoir sélectionner les prénoms de nos utilisateurs qui ont commenté sur notre blog depuis le 18 mais 2018 à midi SI AU MOINS l'un d'entre eux a posté un commentaire depuis cette date.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Renvoie le prenom des users qui ont écrit au moins un commentaire
         *après le 18 mai 2018 à midi si au moins un user à écrit un
         *commentaire depuis cette date*/
        $sth = $dbco->prepare("
          SELECT prenom FROM users
          WHERE id =
            ANY (SELECT userId FROM comments WHERE dateComment > '2018-05-18 12:00:00')
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Ici, si un utilisateur a posté un commentaire après le 18 mais 2018 à midi alors la sous-requête renvoie `TRUE` et nous allons récupérer les prénoms des utilisateurs qui ont écrit un commentaire après cette date.

L'opérateur SQL ALL

L'opérateur SQL `ALL` va également être utilisé conjointement avec une clause `WHERE` ou une clause `HAVING` dans le cas où nous utilisons une sous-requête dans notre requête.

Cependant, cette fois-ci, cet opérateur ne va retourner `TRUE` que si toutes les valeurs de la sous-requête satisfont à la condition posée.

Par exemple, nous allons pouvoir sélectionner les prénoms de nos utilisateurs qui ont commenté sur notre blog depuis le 18 mais 2018 à midi S'ILS ONT TOUS posté un commentaire depuis cette date.

```php
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP / MySQL</title>
    <meta charset='utf-8'>
  </head>
  <body>
    <h1>Bases de données MySQL</h1>
    <?php
      $servname = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

      try{
        $dbco = new PDO("mysql:host=$servname;dbname=$dbname", $user, $pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        /*Renvoie le prenom des users qui ont écrit au moins un commentaire
         *après le 18 mai 2018 à midi si chacun d'entre eux a écrit un
         *commentaire depuis cette date*/
        $sth = $dbco->prepare("
          SELECT prenom FROM users
          WHERE id =
            ALL (SELECT userId FROM comments WHERE dateComment > '2018-05-18 12:00:00')
        ");

        $sth-> execute();

        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);

        echo '<pre>';
        print_r($resultat);
        echo '</pre>';
      }

      catch(PDOException $e){
        echo "Erreur : " . $e->getMessage();
      }
    ?>
  </body>
</html>
```

Cette fois-ci la sous-requête ne renvoie TRUE que si tous les utilisateurs ont posté un commentaire depuis le 18 mais 2018 à midi. Si ce n'est pas le cas, elle renvoie FALSE et la requête principale (la sélection des prénoms) n'est pas exécutée et donc aucun prénom n'est sélectionné.

PARTIE XVI

Gestion des formulaires

Rappels sur les formulaires HTML

L'objet de cette partie n'est pas d'apprendre à créer des formulaires en HTML ni de découvrir les différents éléments de formulaire puisque je pars du principe que vous connaissez déjà le HTML et savez donc créer un formulaire en HTML.

Pour la bonne compréhension de tous, néanmoins, nous allons dans cette première leçon effectuer quelques rappels sur l'utilité des formulaires et la création de formulaires en HTML.

Ensuite, dans le reste de cette partie, ce qui va nous intéresser va être de comprendre comment on va pouvoir récupérer les données envoyées via un formulaire en PHP et d'apprendre à les manipuler et à les stocker.

Pour faire cela, nous allons utiliser un formulaire très simple comme base de travail.

Définition et rôle des formulaires HTML

Les formulaires HTML vont nous permettre de recueillir des données envoyées par nos utilisateurs. Les formulaires vont se révéler indispensable pour tout site proposant aux utilisateurs de s'inscrire et de se connecter, et vont être l'élément privilégié pour permettre aux utilisateurs d'envoyer un message via notre site par exemple.

Les formulaires HTML vont pouvoir être composés de champs de texte (cas d'un champ de formulaire demandant à un utilisateur de renseigner son adresse mail pour se connecter ou pour s'inscrire sur le site par exemple), de listes d'options (choix d'un pays dans une liste de pays par exemple), de cases à cocher, etc.

L'intérêt et le rôle principal des formulaires HTML va résider dans le fait que les formulaires vont permettre de transmettre des données. En effet, une fois que l'utilisateur a fini de remplir un formulaire, il va pouvoir l'envoyer. Les données envoyées vont ensuite pouvoir être stockées ou traitées / manipulée (on va ainsi par exemple pouvoir vérifier que le pseudo et le mot de passe d'un utilisateur souhaitant se connecter à notre site sont valides).

L'élément form et ses attributs

Pour créer un formulaire, nous allons utiliser l'élément HTML `form`. Cet élément `form` va avoir besoin de deux attributs pour fonctionner normalement : les attributs `method` et `action`.

L'attribut `method` va indiquer comment doivent être envoyées les données saisies par l'utilisateur. Cet attribut peut prendre deux valeurs : `get` et `post`.

Que signifient ces deux valeurs et laquelle choisir ? Les valeurs `get` et `post` vont déterminer la méthode de transit des données du formulaire. En choisissant `get`, on indique que les données doivent transiter via l'URL (sous forme de paramètres) tandis

qu'en choisissant la valeur `post` on indique qu'on veut envoyer les données du formulaire via transaction post HTTP.

Concrètement, si l'on choisit l'envoi via l'URL (avec la valeur `get`), nous serons limités dans la quantité de données pouvant être envoyées et surtout les données vont être envoyées en clair. Évitez donc absolument d'utiliser cette méthode si vous demandez des mots de passe ou toute information sensible dans votre formulaire.

Cette méthode de transit est généralement utilisée lors de l'affichage de produits triés sur un site e-commerce (car oui, les options de tris sont des éléments de formulaires avant tout !). Regardez bien les URL la prochaine fois que vous allez sur un site e-commerce après avoir fait un tri : si vous retrouvez des éléments de votre tri dedans c'est qu'un `get` a été utilisé.

En choisissant l'envoi de données via post transaction HTTP (avec la valeur `post`), nous ne sommes plus limités dans la quantité de données pouvant être envoyées et les données ne sont visibles par personne. C'est donc généralement la méthode que nous utiliserons pour l'envoi véritablement de données que l'on souhaite stocker (création d'un compte client, etc.).

L'attribut `action` va lui nous servir à préciser l'adresse relative de la page qui va traiter les données. En effet, nous ne pouvons pas effectuer les opérations de traitement ou de stockage des données avec le HTML. Pour faire cela, nous devrons utiliser des langages comme le PHP (pour le traitement) et le MySQL (pour le stockage) par exemple.

Création d'un formulaire HTML

Comme énoncé précédemment, nous allons créer un petit formulaire en HTML qui va nous servir pour le reste de cette partie.

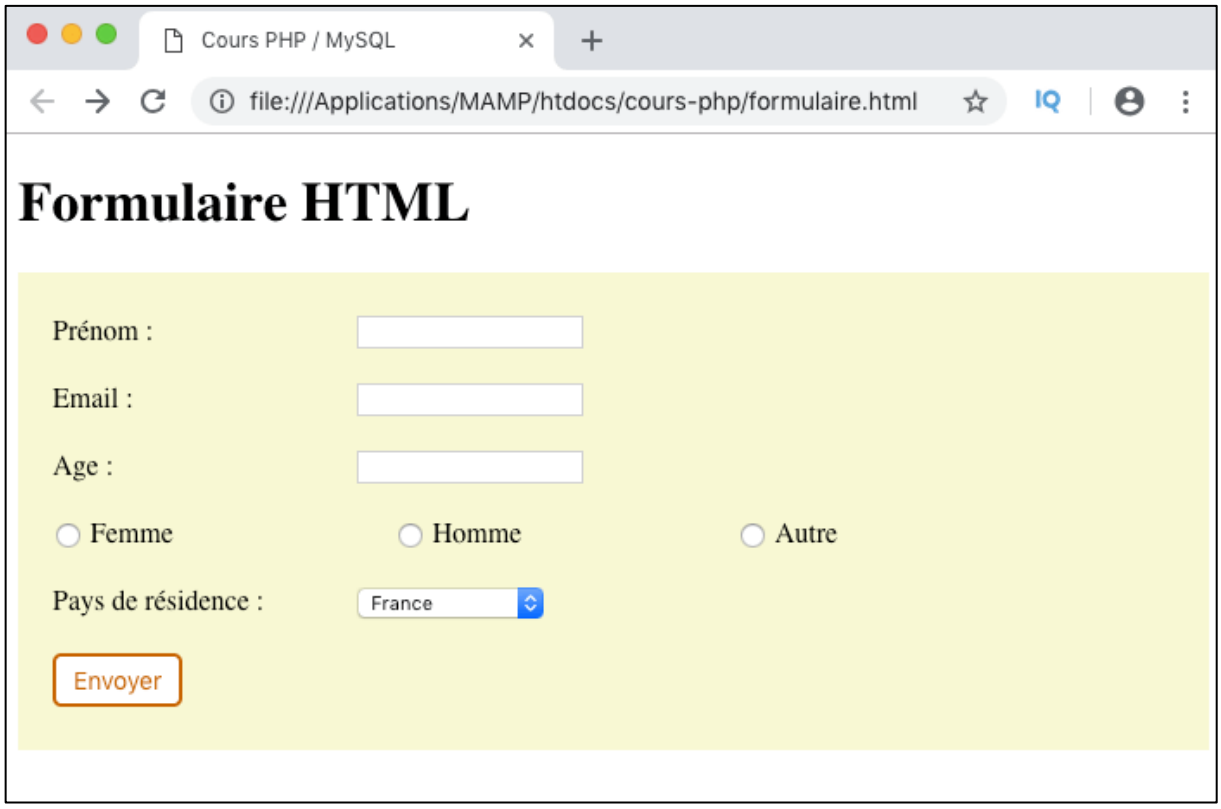
Ce formulaire va nous permettre de récupérer les informations suivantes :

- Le prénom d'un utilisateur ;
- Son adresse mail ;
- Son âge ;
- Son sexe ;
- Son pays de résidence.

Voici les codes HTML et CSS que nous allons utiliser pour ce formulaire ainsi que le résultat obtenu :

```
<form action="formulaire.php" method="post">
    <div class="c100">
        <label for="prenom">Prénom : </label>
        <input type="text" id="prenom" name="prenom">
    </div>
    <div class="c100">
        <label for="mail">Email : </label>
        <input type="email" id="mail" name="mail">
    </div>
    <div class="c100">
        <label for="age">Age : </label>
        <input type="number" id="age" name="age">
    </div>
    <div class="c100">
        <input type="radio" id="femme" name="sexe" value="femme">
        <label for="femme">Femme</label>
        <input type="radio" id="homme" name="sexe" value="homme">
        <label for="homme">Homme</label>
        <input type="radio" id="autre" name="sexe" value="autre">
        <label for="autre">Autre</label>
    </div>
    <div class="c100">
        <label for="pays">Pays de résidence :</label>
        <select id="pays" name="pays">
            <optgroup label="Europe">
                <option value="france">France</option>
                <option value="belgique">Belgique</option>
                <option value="suisse">Suisse</option>
            </optgroup>
            <optgroup label="Afrique">
                <option value="algerie">Algérie</option>
                <option value="tunisie">Tunisie</option>
                <option value="maroc">Maroc</option>
                <option value="madagascar">Madagascar</option>
                <option value="benin">Bénin</option>
                <option value="togo">Togo</option>
            </optgroup>
            <optgroup label="Amerique">
                <option value="canada">Canada</option>
            </optgroup>
        </select>
    </div>
    <div class="c100" id="submit">
        <input type="submit" value="Envoyer">
    </div>
</form>
```

```css
form{
    width: 100%;
    background-color: rgba(220,220,0,0.2);
    padding : 5px 0px;
}
.c100{
    width: 100%;
    margin: 20px;
}
label{
    display: inline-block;
    min-width: 25%;
}
input[type="submit"]{
    color: RGB(200,100,0);
    border-radius: 5px;
    padding: 5px 10px;
    font-size: 14px;
    border: 2px solid RGB(200,100,0);
}
input[type="submit"]:hover{
    background-color: RGB(200,100,0);
    color: #fff;
    cursor: pointer;
    box-shadow: 0px 0px 5px 0px #777;
}
```



Ici, on utilise des éléments input pour demander le prénom, le mail et l'âge de l'utilisateur. On ajuste la valeur de l'attribut type en fonction du type de données attendues (texte, nombre, etc.).

On utilise également un input type="radio" pour le choix du sexe. L'utilisation de boutons radio est ici toute indiquée puisque l'utilisateur ne devrait pouvoir faire qu'un choix dans la liste d'options et car cela va grandement faciliter le traitement des données par la suite puisque nous allons toujours recevoir soit la valeur « femme » soit la valeur « homme » soit « autre ».

Enfin, nous utilisons un élément select pour le choix des pays avec différents éléments option pour chaque option. Ici, nous groupons les pays par continent avec l'élément optgroup.

Notez qu'on utilise également à chaque fois des éléments label pour indiquer à l'utilisateur ce qu'il doit renseigner comme information. On lie également chaque label à son champ de formulaire grâce aux attributs for du label et id de l'input ou du select en leur donnant la même valeur.

Côté mise en forme et CSS, on va se contenter du strict minimum puisqu'encore une fois ce n'est pas l'objet de cette partie. Nous allons ici nous servir de nos div class="c100" pour mettre chaque champ de formulaire sur une nouvelle ligne et allons mettre rapidement en forme les autres éléments.

Notez bien ici les valeurs données aux attributs method et action et notre élément form car c'est cela qui va particulièrement nous intéresser.

Ici, on indique qu'on choisit d'envoyer nos données par transaction http de type post. Nous allons envoyer les données vers la page formulaire.php. C'est cette page là qui va s'occuper du traitement des données du formulaire et sur laquelle nous allons nous concentrer par la suite.

Si vous cliquez sur le bouton de validation, vous pouvez observer que vous êtes renvoyés vers cette page. Cependant, comme nous n'avons pas encore crée la page, vous devriez avoir une erreur de type « page introuvable ».

Nous avons notre base, il est donc maintenant temps d'apprendre à récupérer et à manipuler les données envoyées via nos formulaires en PHP.

Récupérer et manipuler des données de formulaire

Le HTML nous permet de créer nos formulaires. Pour récupérer et manipuler les données envoyées, cependant, nous allons devoir utiliser du PHP.

Dans cette leçon, nous allons voir comment récupérer et manipuler (afficher, stocker, etc.) les données récoltées via les formulaires.

Les superglobales $_POST et $_GET

Dans la leçon précédente, nous avons créé un formulaire dans une page qui s'appelait `formulaire.html`.

Notre élément `form` possédait les attributs suivants : `method="post"` et `action="formulaire.php"`.
Cela signifie que les données vont être envoyées via transaction post http à la page (ou au script) `formulaire.php`.

La première chose à comprendre ici est que toutes les données du formulaire vont être envoyées et être accessibles dans le script PHP mentionné en valeur de l'attribut `action`, et cela quelle que soit la méthode d'envoi choisie (`post` ou `get`).

En effet, le PHP possède dans son langage deux variables superglobales `$_GET` et `$_POST` qui sont des variables tableaux et dont le rôle va justement être de stocker les données envoyées via des formulaires.

Plus précisément, la superglobale `$_GET` va stocker les données envoyées via la méthode `get` et la variable `$_POST` va stocker les données envoyées via la méthode `post`.

Les valeurs vont être stockées sous forme d'un tableau associatif c'est-à-dire sous la forme clef => valeur où la clef va correspondre à la valeur de l'attribut `name` d'un champ de formulaire et la valeur va correspondre à ce qui a été rempli (ou coché, ou sélectionné) par l'utilisateur pour le champ en question.

A noter : On va également pouvoir utiliser la variable superglobale `$_REQUEST` pour accéder aux données d'un formulaire sans se soucier de la méthode d'envoi. Cependant, utiliser `$_REQUEST` ne présente généralement que peu d'intérêt en pratique et peut potentiellement ouvrir des failles de sécurité dans nos formulaires. C'est la raison pour laquelle je n'en parlerai pas plus dans ce cours.

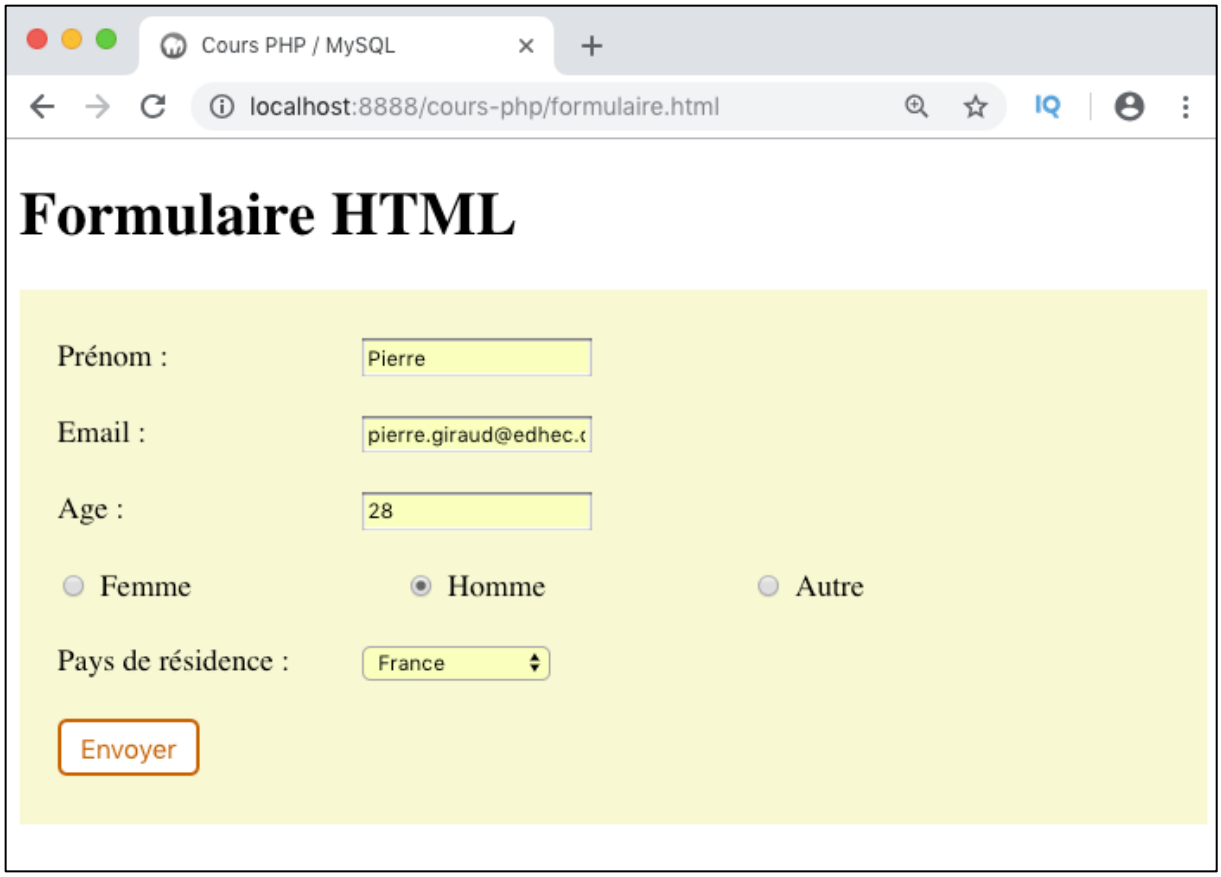
Affichage simple des données de formulaire reçues

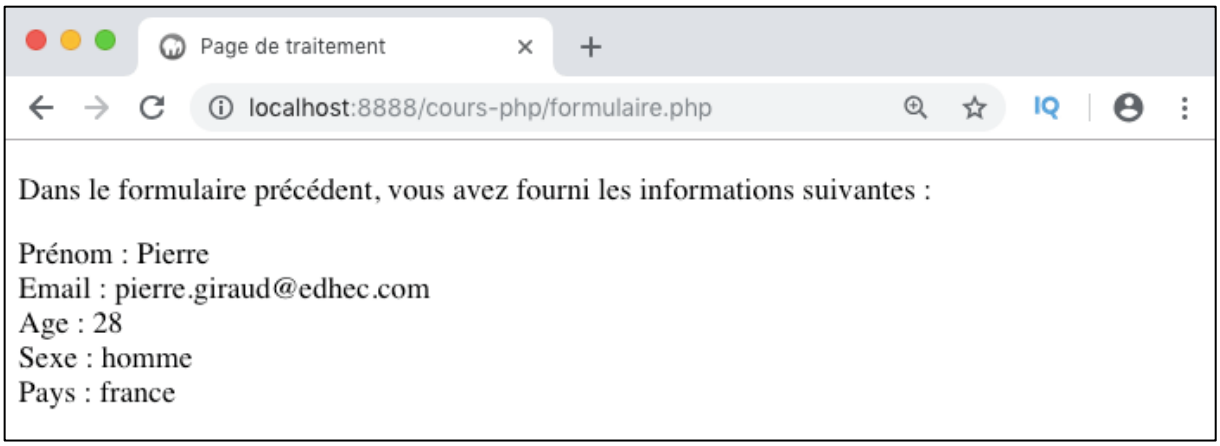
Comme `$_GET` et `$_POST` sont des variables superglobales, elles seront toujours accessibles n'importe où dans le script par définition.

On va alors très facilement pouvoir accéder aux données envoyées dans les formulaires en parcourant nos variables superglobales `$_GET` ou `$_POST`.

Par exemple, on va pouvoir très simplement afficher les données reçues à l'utilisateur. Pour cela, nous allons `echo` les valeurs contenues dans `$_POST` via notre page d'action `formulaire.php` :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Page de traitement</title>
        <meta charset="utf-8">
    </head>
    <body>
        <p>Dans le formulaire précédent, vous avez fourni les
        informations suivantes :</p>

        <?php
            echo 'Prénom : '.$_POST["prenom"].'<br>';
            echo 'Email : ' .$_POST["mail"].'<br>';
            echo 'Age : ' .$_POST["age"].'<br>';
            echo 'Sexe : ' .$_POST["sexe"].'<br>';
            echo 'Pays : ' .$_POST["pays"].'<br>';
        ?>
    </body>
</html>
```





En pratique, cependant, nous n'allons pas créer des formulaires pour afficher les données aux utilisateurs mais bien pour utiliser les données de notre côté. Généralement, donc, l'utilisateur ne verra pas la page de traitement des données et nous le redirigerons plutôt immédiatement vers une page pertinente.

Par exemple, on va renvoyer l'utilisateur vers une page de remerciement si le formulaire était un formulaire créé pour nous envoyer un message, ou vers la page d'accueil du site ou son espace client si le formulaire était un formulaire de connexion, ou encore le renvoyer vers la page où se situe le formulaire si le formulaire servait à envoyer un commentaire sur un article.

Pour renvoyer un utilisateur vers une autre page, on peut utiliser la fonction PHP `header()` à laquelle on va passer la page où l'utilisateur doit être renvoyé sous la forme `Location : adresse de ma page`.

Pour illustrer cela, on va créer deux nouvelles pages `formulaire2.php` et `form-merci.html`. Notre page `formulaire2.php` va être notre nouvelle page d'action, pensez donc bien à modifier la valeur de l'attribut `action` dans le formulaire.

Dans cette page, nous allons donc effectuer différentes manipulations de données (que l'on va se contenter d'imaginer pour le moment) et renvoyer immédiatement l'utilisateur vers une autre page qu'on appelle ici `form-merci.html` et qui va être une page de remerciement.

```php
<?php
    /*Ici, on manipule nos données à loisir
     *
     *
     *
     *...Puis on redirige l'utilisateur vers une autre page*/

    header("Location:form-merci.html");
?>
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="formulaire.css">
    </head>
    <body>
        <h1>Formulaire bien envoyé !</h1>
        <p>Merci d'avoir pris le temps de remplir le formulaire !</p>
    </body>
</html>
```

Manipulation et stockage de données

Sur les sites Internet, les formulaires sont utilisés de manière courante pour effectuer différents types d'opération comme :

- Donner la possibilité à un utilisateur de s'inscrire ;
- Donner la possibilité à un utilisateur de se connecter ;
- Permettre à un utilisateur de poster un commentaire ;
- Permettre à un utilisateur de nous envoyer un message ;
- Etc.

Dans chacun de ces cas, nous allons manipuler les données envoyées dans des buts différents : enregistrer les données pour une inscription, vérifier les données de connexion envoyées, enregistrer et afficher un commentaire, etc.

Nous n'allons bien évidemment pas ici créer un script complet pour chacune de ces situations car cela serait bien trop long et car cela nous éloignerait du sujet de cette leçon.

Cependant, je vous propose ici de voir comment on va pouvoir réceptionner les données d'un formulaire et les enregistrer en base de données pour vous donner un exemple concret.

Cela va finalement être très simple puisque nous savons que nous avons toutes les données du formulaire stockées dans notre variable `$_POST`. La vraie difficulté ici va finalement être de savoir manipuler les bases de données. Si vous n'êtes pas au point sur ce sujet, je vous renvoie aux parties précédentes !

De mon côté, je vais réutiliser ma base de données `cours` créée précédemment dans ce cours et je vais également créer une table `form` dans cette base de données.

Je vous invite à faire de même, soit en passant par phpMyAdmin, soit via un script PHP comme celui-ci :

```php
<?php
    $serveur = "localhost";
    $dbname = "cours";
    $user = "root";
    $pass = "root";

    try{
        //On se connecte à la BDD
        $dbco = new PDO("mysql:host=$serveur;dbname=$dbname",$user,$pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //On crée une table form
        $form = "CREATE TABLE form(
            id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
            prenom TEXT,
            mail TEXT,
            age INT,
            sexe TEXT,
            pays TEXT)";
        $dbco->exec($form);
    }
    catch(PDOException $e){
        echo 'Erreur : '.$e->getMessage();
    }
?>
```

Bien évidemment, on crée notre table de façon à ce qu'elle puisse recevoir les données du formulaire. Ici, nous allons donc créer des colonnes prenom (TEXT), mail (TEXT), age (TEXT), sexe (INT) et pays (TEXT) ainsi que bien sûr notre traditionnelle colonne id (INT UNSIGNED AUTO_INCREMENT PRIMARY KEY).

Nous n'avons ensuite plus qu'à insérer dans notre table les données récoltées. Pour cela, nous allons modifier le script de notre page d'action `formulaire2.php`.

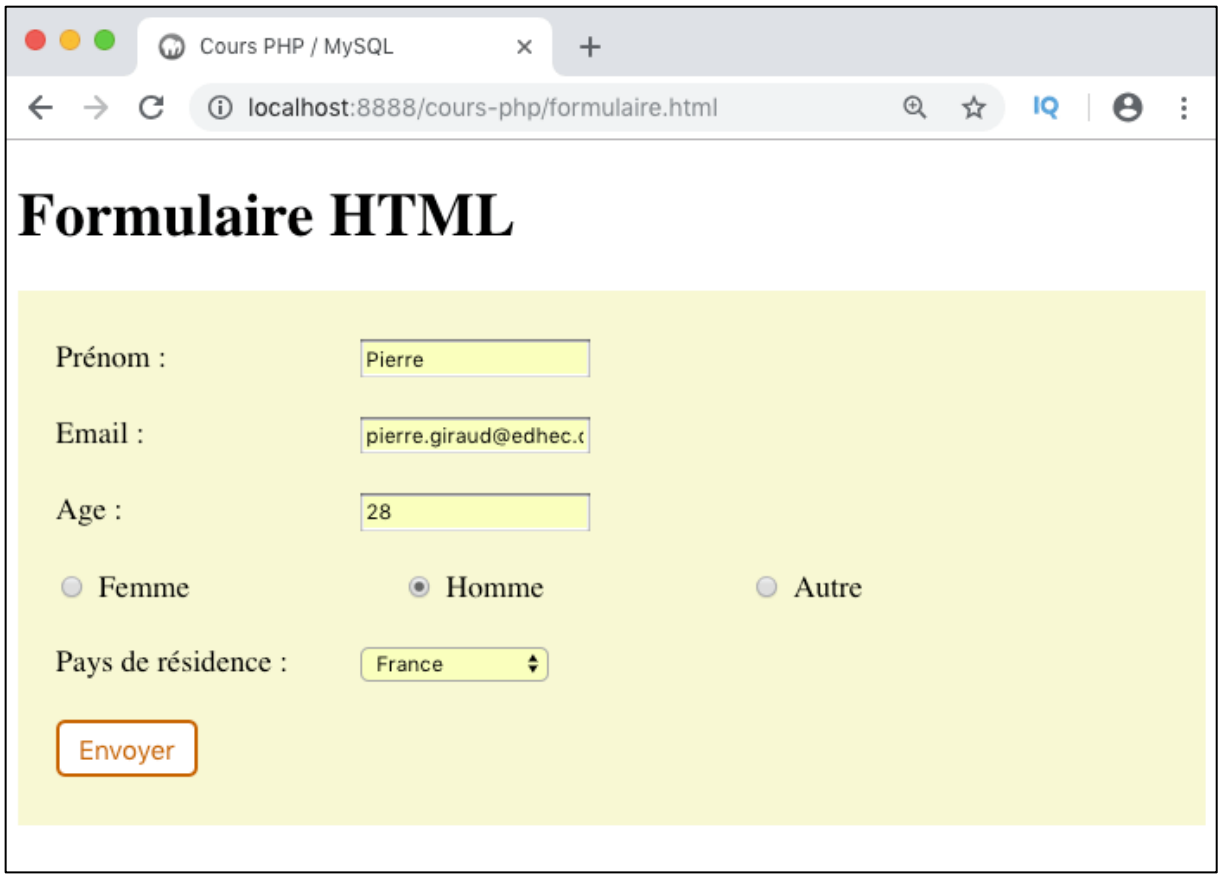
```php
<?php
    $serveur = "localhost";
    $dbname = "cours";
    $user = "root";
    $pass = "root";

    $prenom = $_POST["prenom"];
    $mail = $_POST["mail"];
    $age = $_POST["age"];
    $sexe = $_POST["sexe"];
    $pays = $_POST["pays"];

    try{
        //On se connecte à la BDD
        $dbco = new PDO("mysql:host=$serveur;dbname=$dbname",$user,$pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //On insère les données reçues
        $sth = $dbco->prepare("
            INSERT INTO form(prenom, mail, age, sexe, pays)
            VALUES(:prenom, :mail, :age, :sexe, :pays)");
        $sth->bindParam(':prenom',$prenom);
        $sth->bindParam(':mail',$mail);
        $sth->bindParam(':age',$age);
        $sth->bindParam(':sexe',$sexe);
        $sth->bindParam(':pays',$pays);
        $sth->execute();

        //On renvoie l'utilisateur vers la page de remerciement
        header("Location:form-merci.html");
    }
    catch(PDOException $e){
        echo 'Impossible de traiter les données. Erreur : '.$e->getMessage();
    }
?>
```

Ici, on commence par stocker les données contenues dans `$_POST` dans différentes variables PHP simples.

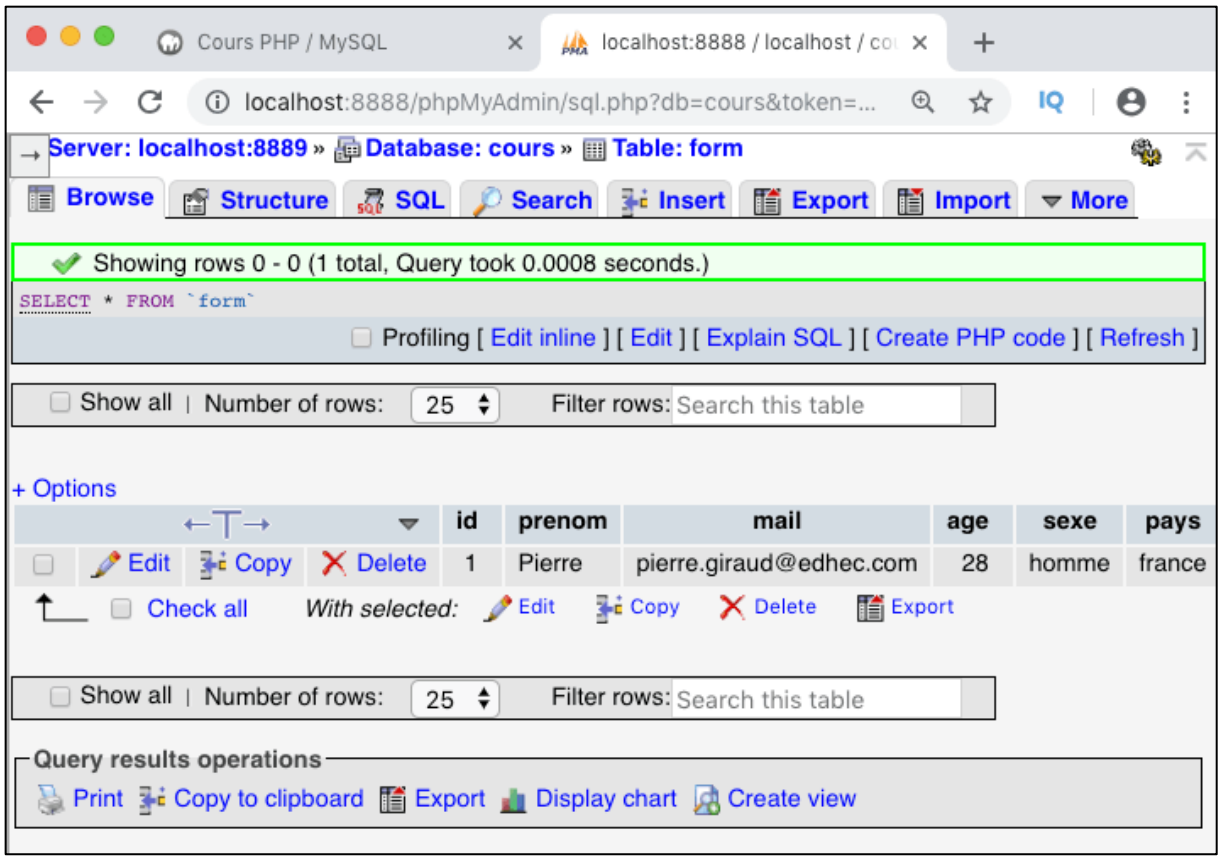
Note : J'expliquerai dans a leçon suivante pourquoi vous ne devriez jamais directement enregistrer les données reçues en base de données comme cela.

On insère ensuite les données dans notre table. Si tout se passe bien, on renvoie l'utilisateur vers la page `form-merci.html`. Dans le cas où une erreur est rencontrée, on affiche l'erreur.

Vous pouvez essayer de remplir à nouveau votre formulaire !


Cours PHP / MySQL
localhost:8888/cours-php/formulaire.html
Formulaire HTML
Prénom :
Pierre
Email :
pierre.giraud@edhec.
Age :
28
Femme
Homme
Autre
Pays de résidence :
France
Envoyer


Cours PHP / MySQL
localhost:8888 / localhost / co
localhost:8888/cours-php/form-merci.html
Formulaire bien envoyé !
Merci d'avoir pris le temps de remplir le formulaire !



Note : Bien évidemment, ce script est simplifié à l'extrême et n'est pas utilisable en l'état pour des raisons de sécurité sur les données et etc. L'idée ici est simplement de vous montrer en pratique comment différents éléments du PHP vont pouvoir fonctionner ensemble.

Sécurisation et validation des formulaires

Dans cette leçon, nous allons comprendre l'importance de la validation des données envoyées par les formulaires et allons voir comment mettre en place un premier système de validation de ces données.

Ne jamais faire confiance aux données utilisateurs

La sécurisation des formulaires est un aspect essentiel de la création de ceux-ci.

Lorsqu'on crée des formulaires, c'est généralement pour demander aux utilisateurs de nous envoyer des données. Si on ne met pas en place des systèmes de filtre sur le type de données qui peuvent être envoyées pour chaque champ et de vérification ensuite de la qualité des données envoyées, les données récoltées vont alors pouvoir être aberrantes ou même potentiellement dangereuses.

En effet, sans contrainte sur les données qui peuvent être envoyées, rien n'empêche un utilisateur d'envoyer des données invalides, comme par exemple un prénom à la place d'une adresse email ou un âge de 2000 ans ou encore de tenter de nous envoyer un script potentiellement dangereux.

Ici, il va falloir faire la différence entre deux types d'utilisateurs qui vont être gérés de façons différentes : les utilisateurs maladroits qui vont envoyer des données invalides par mégarde et les utilisateurs malveillants qui vont tenter d'exploiter des failles de sécurité dans nos formulaires pour par exemple récupérer les données personnelles d'autres utilisateurs.

Pour ce premier groupe d'utilisateurs qui ne sont pas mal intentionnés, la première action que nous allons pouvoir prendre va être d'ajouter des contraintes directement dans notre formulaire pour limiter les données qui vont pouvoir être envoyées. Pour cela, nous allons pouvoir utiliser des attributs HTML comme `min`, `max`, `required`, etc. ainsi que préciser les bons types d'`input` à chaque fois.

Nous allons ensuite également pouvoir tester que les données nous conviennent dès le remplissage d'un champ ou au moment de l'envoi du formulaire grâce au HTML ou au JavaScript (principalement) et bloquer l'envoi du formulaire si des données ne correspondent pas à ce qu'on attend.

Tout cela ne va malheureusement pas être suffisant contre les utilisateurs malintentionnés pour la simple et bonne raison que n'importe qui peut neutraliser toutes les formes de vérification effectuées dans le navigateur. Pour cela, il suffit par exemple de désactiver l'usage du JavaScript dans le navigateur et d'inspecter le formulaire pour supprimer les attributs limitatifs avant l'envoi.

Contre les utilisateurs malveillants, nous allons donc également devoir vérifier les données après l'envoi du formulaire et neutraliser les données potentiellement dangereuses. Nous allons effectuer ces vérifications en PHP, côté serveur.

Ces deux niveaux de vérifications (dans le navigateur / côté serveur) doivent être implémentés lors de la création de formulaires. En effet, n'utiliser qu'une validation dans

le navigateur laisse de sérieuses failles de sécurité dans notre formulaire puisque les utilisateurs malveillants peuvent désactiver ces vérifications.

N'effectuer qu'une série de vérifications côté serveur, d'autre part, serait également une très mauvaise idée d'un point de vue expérience utilisateur puisque ces vérifications sont effectuées une fois le formulaire envoyé.

Ainsi, que faire si des données aberrantes mais pas dangereuses ont été envoyées par un utilisateur maladroit ? Supprimer les données ? Le recontacter pour qu'il soumette à nouveau le formulaire ? Il est bien plus facile dans ce cas de vérifier directement les données lorsqu'elles sont saisies dans le navigateur et de lui faire savoir si une donnée ne nous convient pas.

Note : Dans ce cours, je n'envisage les formulaires que sous forme de code HTML avec traitement des données en PHP. Certains sites utilisent cependant également le JavaScript notamment pour actualiser les données en direct, sans avoir à recharger la page.

Cela va être le cas pour les options de tri d'un site e-commerce par exemple (qui sont également créées avec des formulaires). Dans ce cas-là, il faudra bien évidemment également sécuriser le code JavaScript.

Les failles XSS et l'injection

Un peu plus haut, j'ai parlé « d'utilisateurs malveillants » et de « données dangereuses ». La question que vous devriez vous poser est donc : comment un utilisateur peut-il exploiter mon formulaire ? Pour répondre à cela, je vais devoir vous parler des failles XSS pour « cross site scripting ».

Une attaque XSS consiste en l'injection d'un code dans le formulaire qui va permettre au hacker d'exécuter des scripts JavaScript dans le navigateur de la victime.

Ici, le hacker n'attaque pas directement sa victime qui va être un autre utilisateur du site mais exploite une faille dans le formulaire du site pour que le site lui-même délivre le code JavaScript à la victime.

Prenons immédiatement un exemple d'injection simple pour voir comment ça fonctionne. Pour cela, je vais créer une nouvelle page `xss.php` et je vais récupérer mon formulaire HTML précédent en ne gardant que les champs prenom, mail et age.

```html
<html>
    <head>
        <title>Cours PHP / MySQL</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="formulaire.css">
    </head>
    <body>
        <h1>Formulaire HTML</h1>

        <form action="xss.php" method="post">
            <div class="c100">
                <label for="prenom">Prénom : </label>
                <input type="text" id="prenom" name="prenom" style="width:30em">
            </div>
            <div class="c100">
                <label for="mail">Email : </label>
                <input type="email" id="mail" name="mail">
            </div>
            <div class="c100">
                <label for="age">Age : </label>
                <input type="number" id="age" name="age" min="12" max="99">
            </div>
            <div class="c100" id="submit">
                <input type="submit" value="Envoyer">
            </div>
        </form>

    </body>
</html>
```

L'idée va être ici de sauvegarder les données du formulaire en base de données puis de les afficher sur la même page, sous le formulaire.

Pour cela, j'indique la page actuelle en page d'action puisqu'on va effectuer ces opérations dans la même page de code et je vais également agrandir la taille de mon champ prénom pour plus de clarté sur les données que je vais insérer.

On va ensuite insérer le code PHP à la suite du formulaire. Côté PHP, on va donc déjà devoir se connecter à la base de données. Ensuite, on va vouloir insérer les nouvelles données si aucun des champs n'est vide puis récupérer toutes les données dans la table et les afficher.

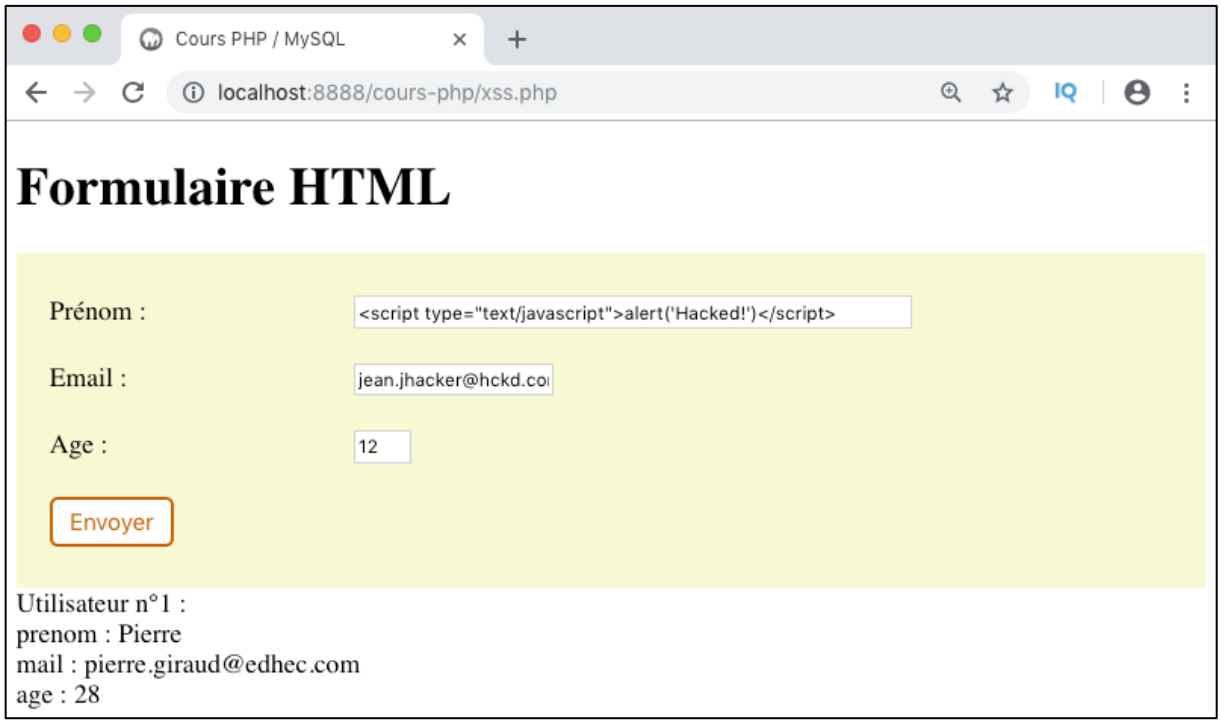
```php
<?php
    $serveur = "localhost";
    $dbname = "cours";
    $user = "root";
    $pass = "root";

    $prenom = $_POST["prenom"];
    $mail = $_POST["mail"];
    $age = $_POST["age"];

    try{
        //On se connecte à la BDD
        $dbco = new PDO("mysql:host=$serveur;dbname=$dbname",$user,$pass);
        $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        //On insère les données reçues si les champs sont remplis
        if(!empty($prenom)  && !empty($mail) && !empty($age)){
            $sth = $dbco->prepare("
                INSERT INTO form(prenom, mail, age)
                VALUES(:prenom, :mail, :age)");
            $sth->bindParam(':prenom',$prenom);
            $sth->bindParam(':mail',$mail);
            $sth->bindParam(':age',$age);
            $sth->execute();
        }

        //On récupère les infos de la table
        $sth = $dbco->prepare("SELECT prenom, mail, age FROM form");
        $sth->execute();
        //On affiche les infos de la table
        $resultat = $sth->fetchAll(PDO::FETCH_ASSOC);
        $keys = array_keys($resultat);
        for($i = 0; $i < count($resultat); $i++){
            $n = $i + 1;
            echo 'Utilisateur n°' .$n. ' :<br>';
            foreach($resultat[$keys[$i]] as $key => $value){
                echo $key. ' : ' .$value. '<br>';
            }
            echo '<br>';
        }
    }
    catch(PDOException $e){
        echo 'Impossible de traiter les données. Erreur : '.$e->getMessage();
    }
?>
```

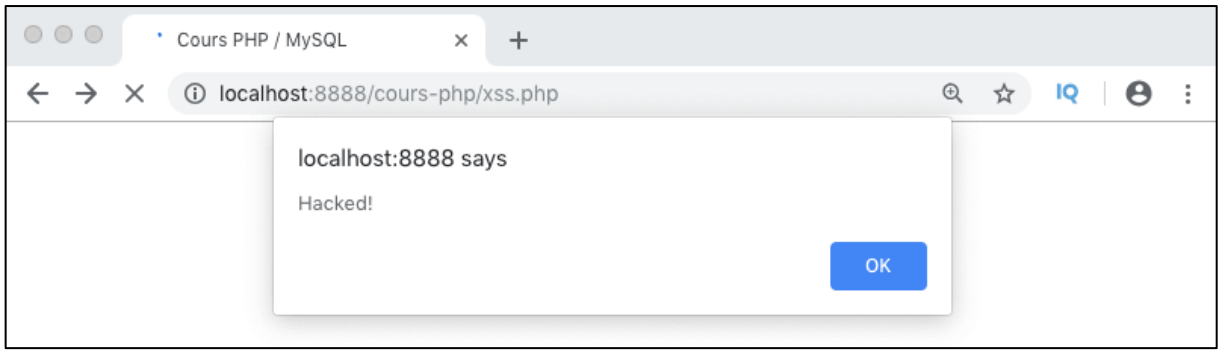
Ici, on insère les données dans la table `form` créée précédemment (les champs sexe et pays de la table seront vides pour les entrées ajoutées mais ce n'est pas grave).

Ensuite, on récupère les données dans la table et on les affiche. On récupère nos données sous forme d'un tableau associatif avec `fetchAll(PDO::FETCH_ASSOC)`.

Notre variable `$resultat` va alors être n tableau multidimensionnel qui va contenir plusieurs tableaux associatifs (un par entrée). On utilise donc une boucle `foreach` pour récupérer les données dans chaque tableau associatif et une boucle `for` pour passer d'un tableau associatif à l'autre.

Voilà pour la partie PHP de la page. Ici, notre code ne contient aucune réelle sécurisation. Un utilisateur va donc pouvoir passer plus ou moins ce qu'il veut comme données dans le champ prenom. Par exemple, il peut tout à fait passer un élément `script`.





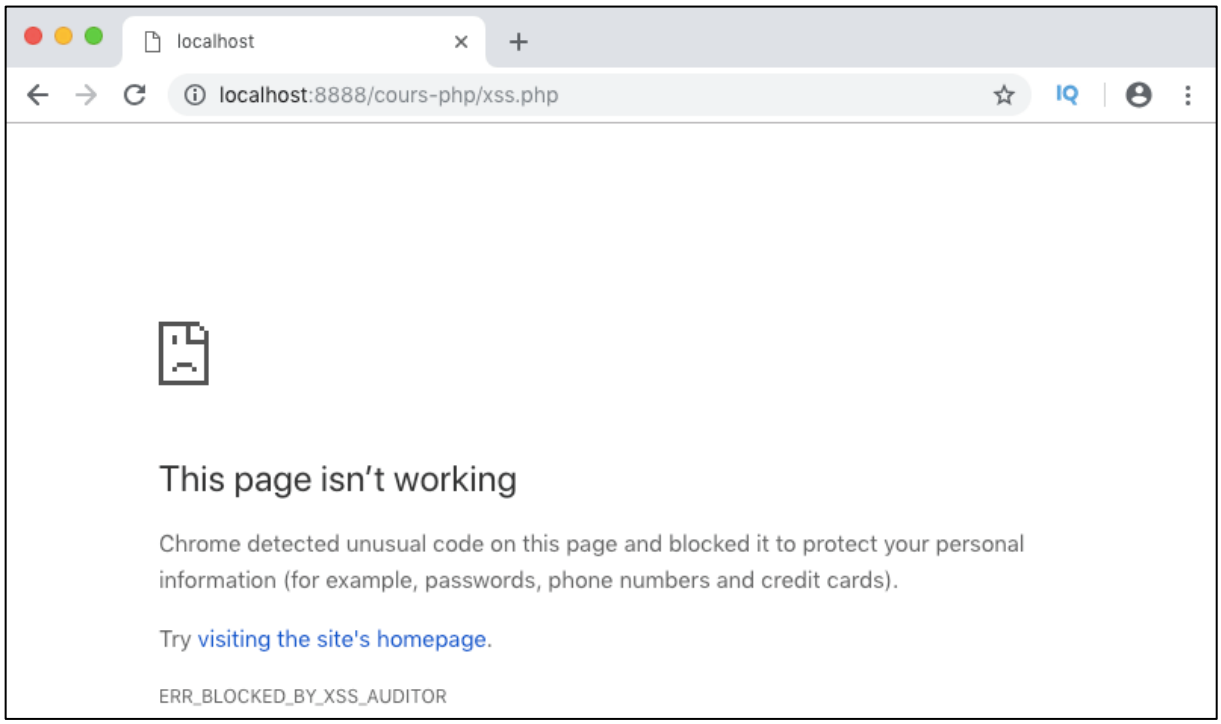
Ici, la valeur donnée dans le champ prénom va être enregistrée sans problème en base de données puis être récupérée et affichée. Cependant, lorsqu'on récupère cette valeur, le navigateur va lire l'élément `script` et donc exécuter le code JavaScript qu'il contient !

Comme les données de la table sont récupérées et affichées à chaque fois qu'un utilisateur arrive sur la page, tous les utilisateurs vont donc exécuter ce code JavaScript lorsqu'ils vont charger la page !

Dans ce cas-là, il s'agit d'une simple instruction `alert` et ce n'est donc pas dangereux en soi, juste gênant. Cependant, rien ne m'aurait empêché ici d'écrire un script qui, une fois

exécuté, pourrait m'envoyer des informations de connexion comme des mots de passe ou autre.

Note : La plupart des navigateurs sont aujourd'hui vigilants par rapport à l'injection de JavaScript dans des formulaires et donc vont bloquer l'envoi du formulaire pour éviter justement le danger comme Chrome :



Cependant, il est tout à fait possible de passer outre ces vérifications ou simplement d'utiliser une ancienne version d'un navigateur qui ne les fait pas pour injecter du JavaScript. Vous ne pouvez donc pas vous en remettre aux vérifications effectuées par les navigateurs !

Je n'irai pas plus loin dans l'explication de l'exploitation des failles XSS car c'est un sujet complexe et qui mériterait un cours à lui seul. Je voulais simplement vous montrer un exemple concret d'injection pour que vous compreniez bien les risques.

La validation des données du formulaire dans le navigateur

Les processus de validation des données que nous allons pouvoir mettre en place dans le navigateur vont s'effectuer avant ou au moment de la tentative d'envoi du formulaire.

L'objectif va être ici de bloquer l'envoi du formulaire si certains champs ne sont pas correctement remplis et de demander aux utilisateurs de remplir correctement les champs invalides.

Nous allons pouvoir faire cette vérification principalement en HTML et / ou en JavaScript.

Le HTML5 propose aujourd'hui des options de validation relativement puissantes et couvrant la majorité de nos besoins. Je vais donc ici n'utiliser que du HTML.

Notez toutefois que si vous utilisez du JavaScript dans vos formulaires pour par exemple modifier les données de la page sans avoir à la rafraichir, il faudra bien évidemment également sécuriser vos scripts JavaScript.

La validation des données en HTML va principalement passer par l'ajout d'attributs dans les éléments de formulaire. Nous allons ainsi pouvoir utiliser les attributs suivants :

Attribut	**Définition**
size	Permet de spécifier le nombre de caractères dans un champ
minlength	Permet de spécifier le nombre minimum de caractères dans un champ
maxlength	Permet de spécifier le nombre maximum de caractères dans un champ
min	Permet de spécifier une valeur minimale pour un champ de type number ou date
max	Permet de spécifier une valeur maximale pour un champ de type number ou date
step	Permet de définir un multiple de validité pour un champ acceptant des donnés de type nombre ou date. En indiquant `step="4"`, les nombres valides seront -8, -4, 0, 4, 8, etc.
autocomplete	Permet d'activer l'autocomplétion pour un champ : si un utilisateur a déjà rempli un formulaire, des valeurs lui seront proposées automatiquement lorsqu'il va commencer à remplir le champ
required	Permet de forcer le remplissage d'un champ. Le formulaire ne pourra pas être envoyé si le champ est vide
pattern	Permet de préciser une expression régulière. La valeur du champ devra respecter la contrainte de la regex pour être valide

Reprenons notre formulaire précédent et ajoutons quelques contraintes sur les données que l'on souhaite recevoir :

- Le prénom est désormais obligatoire et ne doit comporter que des lettres + éventuellement des espaces, tirets ou apostrophes. Sa taille ne doit pas accéder 20 caractères ;
- Le mail doit avoir au moins 1 caractère de type lettre ou chiffre + le symbole « @ » + à nouveau au moins 1 caractère de type lettre ou chiffre + le symbole « . » + au moins deux caractères de type lettre ou chiffre. Il est également obligatoire ;
- L'âge doit être un nombre et être compris entre 12 et 99 ans.

Nous allons donc écrire :

```
<div class="c100">
    <label for="prenom">Prénom : </label>
    <input type="text" id="prenom" name="prenom"
           required pattern="^[A-Za-z '-]+$" maxlength="20">
</div>
<div class="c100">
    <label for="mail">Email : </label>
    <input type="email" id="mail" name="mail"
           required pattern="^[A-Za-z]+@{1}[A-Za-z]+\.{1}[A-Za-z]{2,}$">
</div>
<div class="c100">
    <label for="age">Age : </label>
    <input type="number" id="age" name="age" min="12" max="99">
</div>
```

Ici, on utilise l'attribut `required` pour nos champs « prenom » et « mail » pour indiquer qu'ils doivent être automatiquement remplis.

Ensuite, on utilise `maxlength` pour limiter la taille de notre champ à 20 caractères. On utilise également `min` et `max` pour fournir un intervalle de valeurs valides pour le champ âge.

Finalement, on utilise l'attribut `pattern` pour ajouter des contraintes sur la forme des données qui doivent être renseignées pour nos champs prenom et mail en fournissant des regex adaptées.

L'idée n'est pas ici de refaire un cours sur les expressions régulières, je vous invite donc à relire la partie qui leur est dédiée dans ce cours si vous ne comprenez pas l'écriture ci-dessus. Notez simplement que nous n'avons pas besoin ici de préciser des délimiteurs pour nos regex avec l'attribut `pattern`.

La validation des données du formulaire sur serveur

La validation dans le navigateur va nous éviter une immense majorité de données invalides et donc d'avoir des données à priori exploitables.

Cependant, elle n'est pas suffisante contre des utilisateurs malveillants puisque n'importe qui peut neutraliser les attributs HTML ou le JavaScript en les désactivant dans le navigateur avant d'envoyer le formulaire.

Une validation côté serveur, en PHP, va donc également s'imposer pour filtrer les données potentiellement dangereuses.

Le PHP nous offre différentes options pour sécuriser nos formulaires en testant la validité des données envoyées : on va pouvoir utiliser des fonctions, des filtres, des expressions régulières, etc.

Ici, la première fonction que vous devez absolument connaitre est la fonction `htmlspecialchars()`. Cette fonction va permettre d'échapper certains caractères spéciaux comme les chevrons « < » et « > » en les transformant en entités HTML.

En échappant les chevrons, on se prémunit d'une injection de code JavaScript puisque les balises `<script>` et `/<script>` vont être transformées en `& <script>` et `&/<script>` et ne vont donc pas être exécutées par le navigateur.

On va ensuite pouvoir utiliser d'autres fonctions pour nettoyer les données avant de les stocker comme `trim()` qui va supprimer les espaces inutiles et `stripslashes()` qui va supprimer les antislashs que certains hackers pourraient utiliser pour échapper des caractères spéciaux.

On peut ici créer une fonction personnalisée qui va se charger d'exécuter chacune des trois fonctions ci-dessus :

```php
$prenom = valid_donnees($_POST["prenom"]);
$mail = valid_donnees($_POST["mail"]);
$age = valid_donnees($_POST["age"]);
$sexe = valid_donnees($_POST["sexe"]);
$pays = valid_donnees($_POST["pays"]);

function valid_donnees($donnees){
    $donnees = trim($donnees);
    $donnees = stripslashes($donnees);
    $donnees = htmlspecialchars($donnees);
    return $donnees;
}
```

Note : Ici, j'ai créé une nouvelle page `formulaire-valid.php` qui va être la nouvelle page d'action de ma page `formulaire.html`.

Ensuite, on peut aller plus loin en testant que les données envoyées ont bien la forme attendue en utilisant des filtres et / ou des expressions régulières.

```php
<?php
    $serveur = "localhost"; $dbname = "cours"; $user = "root"; $pass = "root";

    $prenom = valid_donnees($_POST["prenom"]);
    $mail = valid_donnees($_POST["mail"]);
    $age = valid_donnees($_POST["age"]);
    $sexe = valid_donnees($_POST["sexe"]);
    $pays = valid_donnees($_POST["pays"]);

    function valid_donnees($donnees){
        $donnees = trim($donnees);
        $donnees = stripslashes($donnees);
        $donnees = htmlspecialchars($donnees);
        return $donnees;
    }

    /*Si les champs prenom et mail ne sont pas vides et si les donnees ont
     *bien la forme attendue...*/
    if (!empty($prenom)
        && strlen($prenom) <= 20
        && preg_match("^[A-Za-z '-]+$",$prenom)
        && !empty($mail)
        && filter_var($mail, FILTER_VALIDATE_EMAIL)){

        try{
            //On se connecte à la BDD
            $dbco = new PDO("mysql:host=$eerveur;dbname=$dbname",$user,$pass);
            $dbco->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            //On insère les données reçues
            $sth = $dbco->prepare("
                INSERT INTO form(prenom, mail, age, sexe, pays)
                VALUES(:prenom, :mail, :age, :sexe, :pays)");
            $sth->bindParam(':prenom',$prenom);
            $sth->bindParam(':mail',$mail);
            $sth->bindParam(':age',$age);
            $sth->bindParam(':sexe',$sexe);
            $sth->bindParam(':pays',$pays);
            $sth->execute();
            //On renvoie l'utilisateur vers la page de remerciement
            header("Location:form-merci.html");
        }
        catch(PDOException $e){
            echo 'Erreur : '.$e->getMessage();
        }
    }else{
        header("Location:formulaire.html");
    }
?>
```

Que fait-on ici ? Lorsque le formulaire est envoyé, on commence par utiliser notre fonction `valid_donnees()` pour échapper les caractères dangereux potentiellement envoyées et effectuer un premier nettoyage des données du formulaire. On place alors le résultat dans nos variables `$prenom`, `$mail`, etc.

Ensuite, on ne va vouloir enregistrer les données en base de données que si elles nous conviennent. On va donc déjà tester les données envoyées avec un `if` : si le format est satisfaisant, alors on les enregistre et en envoie l'utilisateur vers la page de remerciement. Sinon (`else`), on renvoie l'utilisateur vers le formulaire pour qu'il le remplisse à nouveau.

Ici, notre `if` teste :

1. Que notre variable `$prenom` ne soit pas vide ;
2. ET que notre variable `$prenom` ne fasse pas plus de 20 caractères avec la fonction `strlen()` qui calcule la taille d'une chaine de caractères ;
3. ET que notre variable `$prenom` ait bien la forme attendue avec le fonction `preg_match()` à laquelle on passe une regex ;
4. ET que notre variable `$mail` ne soit pas vide ;
5. ET que notre variable `$mail` ait bien la forme attendue avec le fonction `filter_var()` et le filtre `FILTER_VALIDATE_EMAIL`.

PARTIE XVII

Conclusion du cours

Conclusion du cours

Notre tour d'horizon des notions à connaitre en PHP et en MySQL s'achève ici. Ce cours complet devrait vous avoir fourni des bases de compréhension solides des rôle et fonctionnement du langage PHP ainsi que des bases de données.

Le PHP est un langage qui s'exécute côté serveur et qui est donc à opposer aux langages s'exécutant côté client, c'est-à-dire dans le navigateur des visiteurs. C'est également un langage dynamique en opposition aux langages statiques, ce qui signifie qu'on va pouvoir l'utiliser pour fournir des pages différentes à chaque utilisateur en fonction de certaines variables.

Dans ce cours, nous avons appris à manipuler les éléments de syntaxe de base du PHP et notamment des variables, des fonctions, des boucles et des conditions.

Nous avons également découvert la partie orientée objet du PHP qui rend ce langage d'autant plus puissant et compris les concepts d'objet, de classe et d'instance.

Enfin, on a vu comment utiliser l'ensemble de ces notions PHP pour créer et se connecter à des bases de données MySQL. Les bases de données MySQL nous permettent de stocker de grandes quantités de données de manière durable tout en nous laissant l'opportunité de manipuler ces données de manière simple.

A ce stade, il est possible que vous ne maitrisiez pas encore parfaitement chaque notion du cours ou que vous n'arriviez pas à visualiser de manière concrète quels outils utiliser pour parvenir à réaliser tel ou tel projet. Cela est tout à fait normal : ce sont des choses qui ne s'acquièrent qu'avec de la pratique.

Retenez bien que vous ne deviendrez un « vrai » développeur qu'en pratiquant et en étant confronté aux difficultés. Je vous invite donc maintenant à refaire les exercices de ce cours et à vous créer vos propres défis en commençant avec des projets relativement simples et en allant vers plus de complexité. C'est le meilleur moyen pour valider ce que vous avez appris ici et pour devenir parfaitement autonome.

Bonne chance pour la suite !