

# PHP Premiers pas

Support de cours Version 1.1

## Sommaire

Sommaire .....	2
1 Introduction.....	6
2 Premiers pas avec PHP .....	9
2.1 Avec quels outils démarrer ?.....	9
2.2 Choisir un bon IDE .....	14
2.3 Choisir un stack PHP .....	16
2.4 Choisir un outil de test .....	19
3 Découverte du langage.....	20
3.1 Test du stack PHP .....	20
3.2 Premier « hello world » .....	24
3.3 Quelques astuces à connaître .....	32
3.3.1 Astuces d'un haricot du net .....	32
3.3.2 Index ou pas index ? .....	33
3.4 Mon « hello world » dans mon smartphone.....	34
3.5 Second « hello world » avec du style .....	37
3.6 Troisième « hello world » pour découvrir les variables .....	40
3.7 Quatrième « hello world », avec un peu de JS.....	42
3.8 Dans les coulisses d'un « hello world » .....	43
3.9 Déconstruisons notre « hello world » .....	50
4 Les bases du langage PHP.....	55
4.1 Les chaînes de caractères.....	55
4.2 Introduction aux variables, constantes, conditions, boucles .....	58
4.2.1 Variables, types, conditions.....	58
4.2.2 Constantes.....	66
4.2.3 Tableaux .....	66
4.2.4 Fonctions, Commentaires.....	74
4.2.5 Include et Require .....	77
4.4 Les formulaires et les tableaux \$_GET et \$_POST .....	79
4.4.1 Le tableau \$_GET .....	79

4.4.2 Le tableau <code>\$_POST</code> .....	84
4.4.3 Un formulaire plus intelligent .....	92
4.4.4 Un formulaire robuste et sécurisé.....	112
4.4.5 Générateur de formulaire .....	116
4.5 PHP et le SQL .....	127
4.5.1 Présentation de PDO .....	127
4.5.2 Premiers exemples avec PDO.....	128
4.5.3 Les différents modes Fetch de PDO .....	129
4.5.4 SQL et sécurité.....	134
4.5.5 Connexion à différents SGBD .....	136
4.6 Les objets en PHP .....	138
4.7 Principes d'architecture .....	139
4.8 Les fichiers .....	140
4.8.1 Les fichiers texte.....	140
4.8.2 Les fichiers CSV .....	141
4.8.3 Les fichiers XML.....	142
4.9 Le CRUD .....	147
5 Annexe.....	148
5.1 Opérateurs.....	148
5.2 Préparation d'un jeu de données SQL avec Excel .....	151
5.3 Tables des pays au format SQL.....	153
5.4 Mesure de performances .....	159
5.5 Les filtres .....	160
5.6 Les pièges du « select multiple ».....	163
5.7 Composer .....	168
5.8 Bibliographie.....	170
5.x PHP et PostgreSQL .....	171
5.x PHP et MongoDB .....	177
6 Changelog .....	178



Notes de l'auteur :

Je m'appelle Grégory Jarrige.

Je suis développeur professionnel depuis 1991. Après avoir longtemps travaillé sur des gros systèmes et des langages et technos propriétaires, j'ai fait le pari de me former aux technos et langage open source vers 2005-2006. J'ai commencé à développer des applications webs professionnelles à partir de 2007, avant d'en faire mon activité principale à partir de 2010. L'arrivée du HTML5 dans la même période a été pour moi une véritable bénédiction, et surtout un formidable terrain d'expérimentation (avec des API comme Canvas, WebAudio, etc...).

En plus de mon activité de développeur freelance, je suis également formateur - sur des sujets tels que PHP, HTML5, Javascript, SQL – tantôt en entreprise, tantôt dans le cadre de programmes de reconversion (GRETA notamment).

Ce support est une création réalisée en juin 2017 en vue de proposer une introduction rapide au langage Javascript pour des personnes débutant dans ce langage, mais ayant déjà des bases d'algorithmique, et/ou des bases de programmation dans d'autres langages (PHP ou autres...). On retrouvera dans ce support certaines techniques Javascript que j'avais utilisées lors d'ateliers d'initiation donnés dans le cadre du meetup « Creative Coding Paris » : <https://www.meetup.com/fr-FR/CreativeCodeParis>

Le présent document est publié sous Licence Creative Commons n° 6.

Il est disponible en téléchargement libre sur mon compte Github :

<https://github.com/gregja/PHPCorner>

Ce document se situe dans la continuité des supports « HTML5 – Premiers pas », « CSS3 – Premiers pas » et « Javascript – Premiers pas » qui sont disponibles dans le dépôt Github suivant :

<https://github.com/gregja/JSCorner>

## 1 Introduction

PHP est un vieux langage, si l'on considère qu'il est apparu au milieu des années 90, à peu près en même temps que Ruby et Javascript.

PHP a été développé par Rasmus Lerdorf, qui souhaitait développer un langage facilitant la gestion de pages HTML. Dans un premier temps, Rasmus a utilisé PHP pour gérer un compteur de visite sur les pages de son site. Mais très rapidement, Rasmus a étendu les possibilités du langage pour en faire un véritable couteau suisse, particulièrement bien adapté à la production de code HTML. Comme le HTML, ce n'est pas jamais que du texte, il s'avère à l'usage que PHP est une formidable boîte à outils pour la manipulation et la production de tout type de fichier texte (ce que nous étudierons aussi dans le cadre de ce cours).

Ce court historique emprunté à Wikipédia est intéressant :

*Rasmus décida alors en 1995 de publier son code, pour que tout le monde puisse l'utiliser et en profiter. PHP s'appelait alors PHP/FI (pour Personal Home Page Tools/Form Interpreter). En 1997, deux étudiants, Andi Gutmans et Zeev Suraski, redéveloppèrent le cœur de PHP/FI. Ce travail aboutit un an plus tard à la version 3 de PHP, devenu alors PHP: Hypertext Preprocessor. Peu de temps après, Andi Gutmans et Zeev Suraski commencèrent la réécriture du moteur interne de PHP. Ce fut ce nouveau moteur, appelé Zend Engine — le mot Zend est la contraction de Zeev et Andi — qui servit de base à la version 4 de PHP.*

Source : <https://fr.wikipedia.org/wiki/PHP>

On peut compléter cette précisant que, quelques années plus tard, Zend réécrivit de nouveau le moteur de PHP, de manière à implémenter un modèle objet plus complet que celui proposé par PHP4, et plus proche du modèle objet de Java. Cette réécriture aboutit à la publication de PHP5, en 2004. La version 5.3 est généralement considérée comme la version qui a fait entrer PHP dans le mode du développement professionnel, avec de nombreuses nouveautés intéressantes (tels que les espaces de nommage, les fonctions anonymes, etc...).

Parmi les points qui ont contribué au succès du langage :

- sa gratuité
- sa simplicité de mise en œuvre, avec un mode de fonctionnement s'appuyant sur le serveur d'application Apache, et simplifiant toute la mécanique d'échange entre serveur et navigateur (via le protocole HTTP) : PHP est en effet configuré pour produire du HTML par défaut, sauf si on lui demande de produire autre chose (XML, texte, etc...).

- La facilité avec laquelle on peut combiner code HTML et code PHP au sein d'un même code source (comme on le verra dans quelques exemples), a aussi contribué à populariser PHP,
- La disponibilité d'un connecteur gratuit avec la base de données MySQL (elle aussi gratuite), suivi très rapidement de connecteurs gratuits pour la plupart des autres bases de données du marché

Les hébergeurs ont très tôt compris le potentiel de PHP, et se sont livrés à une concurrence féroce en proposant des serveurs mutualisés préconfigurés avec Apache, PHP et MySQL. Et cela pour des tarifs très avantageux, pour le client final. On trouve dans ce domaine des acteurs bien établis, comme OVH, Gandi, 1&1, etc... et on voit encore régulièrement de nouveaux challengers apparaître. Mais ceci est une autre histoire...

Le langage PHP a continué à évoluer lentement. On notera une tentative avortée d'écrire un PHP6 : il s'agissait de réécrire l'interpréteur de PHP de manière à ce qu'il supporte nativement la norme d'encodage UTF-8. Mais ce projet était trop ambitieux, et la communauté des développeurs du langage PHP s'est épuisée pendant une période sur ce projet, avant de l'abandonner.

Vers la fin des années 2000, la société Facebook - qui utilisait massivement le langage PHP pour son application phare (réseau social) - décida d'écrire son propre interpréteur PHP, afin d'obtenir de meilleures performances que celles obtenues avec l'interpréteur standard développé par Zend. Cela aboutit à deux projets distincts mais étroitement liés :

- le projet HHVM (Hip Hop Virtual Machine) : <http://hhvm.com/>
- le projet Hack, qui est un langage PHP à la sauce Facebook : <http://hacklang.org/>

Facebook est même allée – pour le développement de Hack – jusqu'à écrire une spécification du langage PHP, ce que Zend n'avait jamais pris la peine de faire :

<https://github.com/php/php-langspec>

Facebook publant régulièrement des benchmarks dans lesquels elle affirmait obtenir de meilleures performances que l'interpréteur PHP standard, la société Zend se vit obligée de réagir, en publiant en 2014 le projet PHPNG (NG pour « New Generation »). Il semble que les projets HHVM (qui continue à être utilisé par Facebook) et PHPNG (qui deviendra par la suite PHP7) soient relativement proches en termes de performance.

On notera que Zend a été rachetée en 2015 par la société Rogue Wave Software, qui a repris le catalogue de solutions proposées par Zend et continue à les faire évoluer.

Le langage PHP continue à évoluer, sous la houlette d'un groupe d'experts du langage, et on peut suivre cette évolution en consultant la changelog de PHP7 :

<http://php.net/ChangeLog-7.php>

## 2 Premiers pas avec PHP

### 2.1 Avec quels outils démarrer ?

Nous avons vu dans les cours HTML5 et Javascript que l'on peut utiliser des solutions en ligne très pratiques pour développer et prototyper du code HTML5 et JS, telles que les plateformes Codepen et Codecircle.

- Codepen : <https://codepen.io/>
- Codecircle : <https://live.codecircle.com/>

Malheureusement, il n'existe pas de solutions aussi souples pour PHP, et c'est bien dommage. On peut cependant utiliser le site internet « PHP Sandbox » (traduisez : « bac à sable PHP ») :

<http://sandbox.onlinephpfunctions.com/>

The screenshot shows the homepage of Online PHP Functions. At the top, there's a navigation bar with links for Home, Sandbox, PHP Functions (with a dropdown menu), Custom Functions, Tutorials, PHP Benchmarks, and About. Below the navigation, a banner reads "Online PHP function(s){ #Test PHP Functions online}".

The main content area is titled "PHP Sandbox" and contains the sub-instruction "Test your PHP code with this code tester". It says "You can test your PHP code here on many php versions." Below this, there's a text input field labeled "Your script:" containing a sample PHP code snippet:

```

1  <?php
2      //Enter your code here, enjoy!
3
4  $array = array("1" => "PHP code tester Sandbox Online",
5                 "foo" => "bar", 5 , 5 => 89009,
6                 "case" => "Random Stuff: " . rand(100,999),
7                 "PHP Version" => phpversion()
8                 );
9
10
11
12
13
14

```

Très pratique, ce « bac à sable » permet de saisir et tester du code PHP en ligne. Cela ne permet pas de développer des applications, mais pour tester quelques fonctions ou de petits bouts de code « maison », c'est vraiment sympa.

A signaler toutefois l'existence d'une plateforme dédiée à l'apprentissage de PHP, dont je découvre l'existence au moment même où je rédige ces pages :

<http://codeclassroom.net/>

... mais il semble que la plateforme ne soit pas encore opérationnelle, à suivre donc...



Tools to learn to code. For schools and beginners.

LEARN & CODE AT THE SAME TIME

DOWNLOAD for WINDOWS  
version 14.1 alpha - soon



STUDENTS

Enter the library  
and learn something

TEACHERS

Open your office, create courses  
and enrich the library

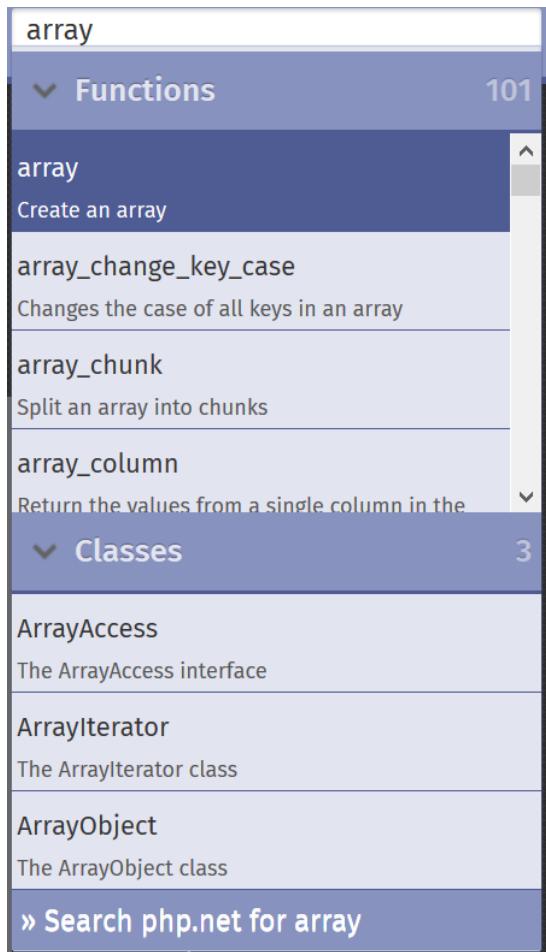
```
1 <!DOCTYPE html>
2 <html>
3   <head>
4   </head>
5   <body>
6     <p>
7       <?php
8         echo "Hello";
9       ?>
10      </p>
11    </body>
12  </html>
```

A complete and fully interactive environment to help students (kids or adults) to learn code and teachers to create and share courses.

PHP HTML CSS JS SQL ...

Dès que nous allons commencer à développer, nous allons avoir besoin de documentation.  
La documentation officielle de PHP est un incontournable :  
<http://php.net/>

La fonction de recherche du site permet de retrouver tout ce qu'on veut très rapidement :



La documentation du site [php.net](http://php.net) est très complète, et le site est traduit en plusieurs langues.

Dans l'exemple ci-dessous, on est en train de consulter la documentation de la fonction PHP « `array_column` », que l'on peut faire défiler et dans laquelle on trouvera un descriptif détaillé suivi de nombreux exemples.

Dans la partie de droite (sur fond noir), on peut faire défiler la liste de toutes les fonctions dont le nom commence par « `array` » (fonctions dédiées à la manipulation de tableaux), et vous verrez qu'il y en a une tétrach... enfin bref, un gros paquet, quoi ☺.

The screenshot shows a screenshot of the PHP documentation website. At the top, there's a navigation bar with links for 'Downloads', 'Documentation' (which is currently selected), 'Get Involved', and 'Help'. A search bar is also at the top right. Below the navigation, a breadcrumb trail shows the path: 'Manuel PHP' > 'Référence des fonctions' > 'Extensions relatives aux variables et aux types' > 'Les tableaux' > 'Fonctions sur les tableaux'. The main content area is titled 'array\_column'. It includes a 'Description' section with the PHP code signature: `array array_column ( array $array , mixed $column_key [, mixed $index_key = null ] )`. To the right of the main content, there's a sidebar titled 'Fonctions sur les tableaux' which lists various array-related functions like 'array\_change\_key\_case', 'array\_chunk', etc. A language selection dropdown is also visible in the top right corner of the main content area.

Autre site de référence intéressant pour l'étude de PHP, W3Schools, qui consacre une partie de son site à ce langage :

<https://www.w3schools.com/php/default.asp>

En termes de logiciels, nous allons avoir besoin de 2 choses importantes, et même d'une troisième :

- un bon éditeur de code, ou IDE (pour Integrated Development Environment)
- un stack PHP comprenant Apache, PHP et MySQL
- un (ou plusieurs) outil(s) de test

Nous allons préciser ces 3 points dans les chapitres suivants.

Mais j'aimerais vous montrer que vous pouvez faire énormément de choses à l'intérieur même de votre navigateur préféré, sans utiliser le moindre outil extérieur.

Nous aurons cependant besoin, dans la suite de ce cours, d'une page HTML de base, la plus simple possible, histoire de démarrer notre apprentissage sur de bonnes bases. Le plus simple, pour obtenir cette page HTML, c'est d'en créer une. Voici son code source, vous pouvez le copier-coller dans un éditeur quelconque, en prenant soin de le sauvegarder avec l'extension « .html ». Vous pouvez par exemple l'appeler « mapage.html » :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>ma page de test</title>
</head>
<body>
    <div id="cible">ceci est le contenu d'une balise "div" de test</div>
</body>
</html>
```

Si vous ne comprenez pas certains éléments du code ci-dessus, je vous recommande d'étudier le cours « HTML5 – Premiers pas », qui est disponible dans le même projet Github.

## 2.2 Choisir un bon IDE

En termes d'éditeurs et d'environnement de développement, on dispose d'un panel de solutions assez large.

L'éditeur Notepad++ est un bon éditeur de code, très polyvalent, qui propose une coloration syntaxique immédiate, dès qu'il détecte que le fichier ouvert est un source HTML ou PHP (il le détermine grâce à l'extension du fichier) :

<https://notepad-plus-plus.org/fr/>

Pour nos premiers tests en PHP, nous utiliserons Notepad++. Il est agréable à utiliser, léger, peu gourmand en ressources, bref, c'est bon pour la planète 😊.

SublimeText est un autre éditeur de code très puissant et très polyvalent, qui est tout à fait adapté à la maintenance de code PHP, à condition de lui ajouter un certain nombre de plugins :

<https://www.sublimetext.com/>

On trouve sur le web, et sur youtube, des tutos indiquant quels plugins installer pour travailler avec PHP.

Netbeans est un IDE gratuit (open source) proposé par Oracle. Très complet, il est parfaitement adapté à la maintenance de code PHP, Javascript et HTML5 :

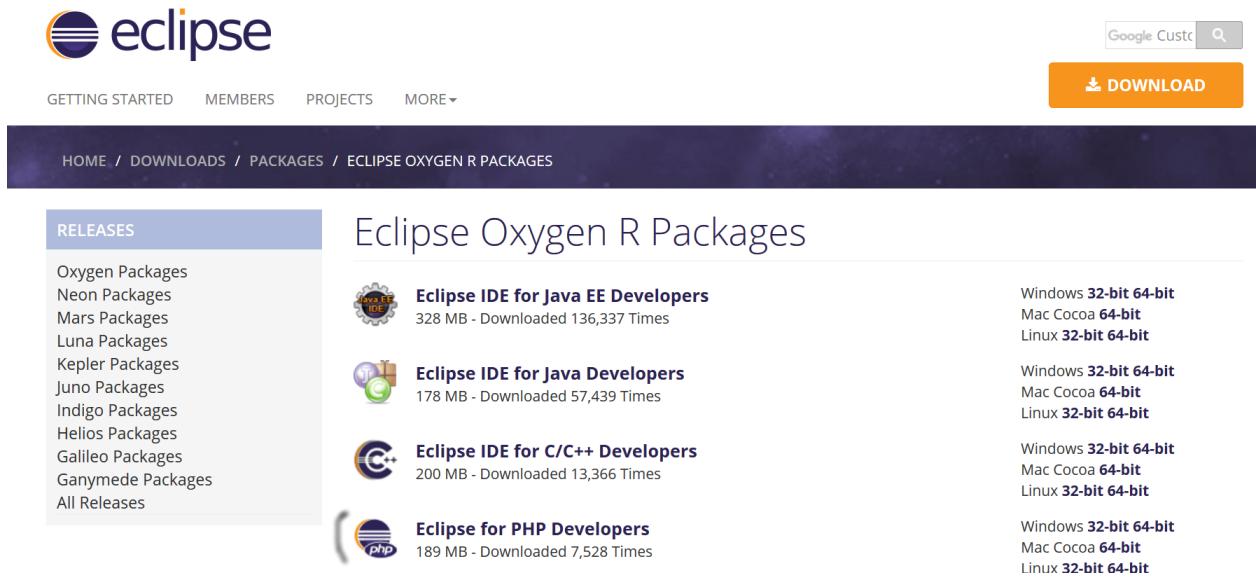
<https://netbeans.org/>

On peut télécharger une version personnalisée en fonction du type de code que l'on souhaite développer (HTML5 et PHP uniquement ? Java ?) :

The screenshot shows the NetBeans IDE 8.2 Download page. At the top, there are fields for Email address (optional), IDE Language (English), and Platform (Windows). Below these are checkboxes for Newsletter subscription (Monthly or Weekly) and a note about unsupported technologies for Windows. The main section is titled "NetBeans IDE Download Bundles" and displays a grid of supported technologies across different IDE components: Java SE, Java EE, HTML5/JavaScript, PHP, C/C++, and All. The grid shows that most technologies are supported across all components, except for some specific ones like Java ME and GlassFish Server. At the bottom, there are download links for various platforms: Download (for Java SE), Download x86, Download x86 (for Java EE), Download x64, Download x64 (for HTML5/JavaScript), Download x86 (for PHP), Download x64 (for C/C++), and Download (for All). Below the download links, there are file sizes: Free, 95 MB; Free, 197 MB; Free, 108 - 112 MB; Free, 108 - 112 MB; Free, 107 - 110 MB; and Free, 221 MB.

La version HTML5 et PHP consomme moins de mémoire, qu'une version plus complète, ça aussi c'est bon pour la planète.

Eclipse est aussi un IDE très complet, gratuit, qui se décline en différentes versions :  
<http://www.eclipse.org/downloads/packages/>



The screenshot shows the Eclipse website's navigation bar with links for 'GETTING STARTED', 'MEMBERS', 'PROJECTS', 'MORE...', 'Google Custom Search', and a large orange 'DOWNLOAD' button. Below the navigation is a breadcrumb trail: 'HOME / DOWNLOADS / PACKAGES / ECLIPSE OXYGEN R PACKAGES'. The main content area has a title 'Eclipse Oxygen R Packages' and a sidebar titled 'RELEASES' listing versions from 'Oxygen Packages' down to 'All Releases'. Four package cards are displayed: 'Eclipse IDE for Java EE Developers' (328 MB, 136,337 downloads), 'Eclipse IDE for Java Developers' (178 MB, 57,439 downloads), 'Eclipse IDE for C/C++ Developers' (200 MB, 13,366 downloads), and 'Eclipse for PHP Developers' (189 MB, 7,528 downloads). Each card includes download links for 'Windows 32-bit 64-bit', 'Mac Cocoa 64-bit', and 'Linux 32-bit 64-bit'.

Enfin, PHPStorm est un excellent IDE développé par la société Jetbrains. Très complet, il est très apprécié des développeurs. Malheureusement il n'est pas gratuit, et le coût de sa licence annuelle est peut être un peu élevé, si l'on est développeur amateur, ou étudiant :

<https://www.jetbrains.com/phpstorm/>

## 2.3 Choisir un stack PHP

Il existe de nombreux stacks PHP.

Ce terme de « stack » peut surprendre. Il signifie « pile » en anglais, et un environnement PHP, ce n'est jamais qu'une pile de logiciels, tels que : Apache, MySQL et PHP. L'ensemble de ces 3 logiciels est généralement désigné par le sigle « AMP ».

Si l'on rajoute devant ce sigle le « W » (pour « Windows »), ou le « L » (pour « Linux »), ou le « M » (pour « Mac »), vous comprenez maintenant pourquoi on parle souvent de WAMP, LAMP ou MAMP pour désigner un stack PHP.

Bon, mais c'est quoi au juste, un stack « AMP » :

- Apache, c'est le serveur d'application. Dans le contexte de notre stack PHP, Apache est une sorte de tour de contrôle, qui intercepte les requêtes HTTP provenant de l'internet (ou d'un réseau informatique interne à une entreprise, car tout ne passe pas forcément par internet). Apache fait office de routeur, il reçoit des requêtes HTTP lui demandant de transmettre une information à une application (par exemple de transmettre à un script PHP les données provenant d'un formulaire HTML). Apache s'exécute, et dès que la réponse produite par l'interpréteur PHP est disponible, Apache se charge de retransmettre cette réponse au client (généralement un navigateur internet, mais cela peut aussi être un autre type de logiciel).
- PHP, c'est l'interpréteur PHP : quand Apache reçoit une requête lui demandant d'appeler un script PHP, il vérifie si cette ressource existe, puis fait appel à l'interpréteur PHP. En très résumé, l'interpréteur PHP est un logiciel qui transforme le code source PHP en un code compréhensible par l'ordinateur. Une fois que le script PHP a terminé son travail, l'interpréteur PHP transmet à Apache la réponse produite par ce script. Le plus souvent, la réponse est au format HTML, mais cela peut aussi être un autre format, comme du XML, ou du JSON (nous étudierons ces 2 formats par la suite).
- MySQL, c'est la base de données. Comme son nom l'indique, c'est une base de données qui s'appuie sur un vieux langage d'interrogation, le SQL. Ce vieux langage est toujours beaucoup utilisé pour le stockage de données, tant il est pratique et puissant. Certains développeurs considèrent que le SQL est obsolète, et qu'il n'y a point de salut en dehors des bases de données de type NoSQL (comme MongoDB et quelques autres). C'est un débat stérile, les bases SQL et NoSQL ont toutes deux des avantages et des inconvénients qu'il serait trop long de développer ici. En ce qui nous concerne, nous travaillerons beaucoup avec MySQL, mais nous jetterons aussi un coup d'œil à PostgreSQL et à MongoDB.

La réalité de ce sigle « AMP » est aujourd’hui battue en brêche, par le fait que ce stack peut être complètement remanié en fonction des besoins et des contraintes d’une entreprise (ou d’un projet) :

- Apache est de plus en plus souvent remplacé par NGinx, projet concurrent souvent présenté comme plus performant
- MySQL est de plus en plus souvent remplacé par MariaDB, qui est un clone open source de MySQL (rappelons simplement que MySQL appartient aujourd’hui à Oracle). MySQL et MariaDB ont le même premier caractère, mais on peut leur préférer d’autres bases de données, SQL ou NoSQL. Parmi les bases de données SQL concurrentes, on trouve PostgreSQL (open source), SQLite (open source), SQL Server (Microsoft), DB2 (IBM), Oracle (Oracle), etc...
- Bon, le seul élément stable dans tout ça, c’est PHP (encore qu’il pourrait être remplacé par Hack, mais seul Facebook fait réellement ça).

Bon, dans un contexte d’apprentissage, un bon vieux stack « AMP » est très satisfaisant.

S’il est possible de télécharger chacun des éléments du stack sur les sites officiels respectifs des projets Apache, MySQL et PHP, et de les installer soi-même manuellement, on peut s’affranchir de cette tâche laborieuse en utilisant un stack prêt à l’emploi... et il en existe plusieurs.

Dans le contexte de Windows, on trouve notamment les stacks suivants :

- Wampserver : <http://www.wampserver.com/>
- Xampp : <https://www.apachefriends.org/fr>
- EasyPHP : <http://www.easypg.org/>
- UWamp : <https://www.uwamp.com/fr/>

Pour ma part, j’aime bien utiliser Xampp, je le trouve très pratique, et je ne suis pas le seul. Xampp existe pour Windows, pour Mac (OS X) et pour Linux.

Sur Mac, on trouve aussi le projet Mamp, qui remplit le même rôle que Xampp :

<https://www.mamp.info/>

Concernant UWamp : le « U » vient de « USB », car ce stack léger est conçu pour fonctionner à partir d’une simple clé USB, ce qui peut être pratique si on est amené à travailler sur plusieurs machines (ou si on est embêté par des droits administrateur trop restrictifs).

A noter que le projet EasyPHP se décline en 2 projets distincts : EasyPHP WebServer et EasyPHP DevServer.

A noter aussi, le stack WAPP, proposé par la société Bitnami, dans lequel MySQL a été remplacé par PostgreSQL :

<https://bitnami.com/stack/wapp>

Zend, qui est désormais une « Rogue Wave Company » propose son propre stack PHP, le Zend Server :

[http://www.zend.com/en/products/zend\\_server#editions](http://www.zend.com/en/products/zend_server#editions)

Zend Server est un stack PHP de niveau professionnel, contenant un ensemble de composants de haut niveau (des outils de déploiement, des outils pour la gestion de travaux asynchrones, des outils pour l'analyse de performance, etc...). Malheureusement, la version « Community Edition » - qui a été proposée pendant une période, et qui était gratuite - n'existe plus, et les licences proposées sont trop chères pour un développeur débutant, étudiant ou pas.

Tous ces stacks PHP ont leurs points forts et leurs faiblesses. Ce support de cours n'est pas un comparatif et n'a pas vocation à le devenir ☺.

Une alternative au fait d'installer un stack PHP sur son PC consiste à monter un stack PHP dans une VM (Virtual Machine) sous Linux. Pour créer une VM, le logiciel Virtual Box d'Oracle est un incontournable (et en plus il est gratuit) :

<https://www.virtualbox.org/>

Le fait de travailler avec une VM offre plusieurs avantages :

- On évite de polluer sa machine (sous Windows) avec un stack PHP, ce dernier étant circonscrit dans la VM, il est invisible pour la machine hôte
- La VM étant sous Linux, elle est plus proche d'un environnement de production, étant donné que la plupart des environnements de production sont sous Linux. Les tests effectués dans la VM sont donc plus proches du contexte réel d'utilisation de l'application
- Cela permet aussi de se familiariser avec l'environnement Linux, même si sa machine tourne sous Windows

La création d'une VM n'est pas une opération très compliquée, mais cela prend du temps.

On trouve de nombreux tutos sur le web, ainsi que des vidéos sur Youtube, expliquant comment configurer une VM et y installer un stack PHP.

## 2.4 Choisir un outil de test

Nous n'allons pas tarder à programmer, et nous allons faire beaucoup de tests manuels.

Nous verrons que certains tests peuvent être laborieux à réaliser, particulièrement si l'on est amené à les réaliser souvent.

Et il y a un test que nous allons effectuer souvent, c'est celui consistant à lancer des requêtes HTTP. Ce sera particulièrement vrai dès que nous commencerons à étudier le développement d'API avec PHP. Pour nous simplifier la vie, nous utiliserons le logiciel Postman, qui connaît un succès mérité auprès de la communauté des développeurs, tant il est pratique. Nous en reparlerons en temps et en heure, mais vous pouvez d'ores et déjà le télécharger et l'installer (ou vous pouvez le faire plus tard, quand nous aborderons le sujet) :

<https://www.getpostman.com/>



TODO : Tester les perfs avec un générateur de requêtes http (retrouver le nom)

## 3 Découverte du langage

AVERTISSEMENT : dans les nombreux exemples qui vont suivre, j'utilise le langage PHP pour générer du code HTML (et aussi dans quelques cas, du CSS et du Javascript). Si vous ne vous sentez pas à l'aise avec le HTML, ou si vous ne savez tout simplement pas de quoi il s'agit, il me semble judicieux que vous vous penchiez sur le support de cours « HTML5 – Premiers pas » qui se trouve dans le dépôt Github suivant :

<https://github.com/gregja/JSCorner>

### 3.1 Test du stack PHP

Je pars du principe que vous avez installé un stack PHP sur votre ordinateur. Pour ma part, j'utilise Xampp. La plupart des informations qui vont suivre sont valables avec les autres stacks PHP. Si certaines informations sont très spécifiques à Xampp, je le préciserai au fil de l'eau.

Pour vérifier si mon stack PHP fonctionne, c'est facile, je saisis cette URL dans mon navigateur :  
<http://localhost/>

On voit que l'URL est aussitôt modifiée pour devenir :

<http://localhost/dashboard/>

Et dans la fenêtre du navigateur, je vois apparaître la page ci-dessous, dont je ne vous ai mis qu'un extrait :



### Welcome to XAMPP for Windows 7.1.6

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the [FAQs](#) section or check the [HOW-TO Guides](#) for getting started with PHP applications.

La plupart des stacks PHP fournissent un tableau de bord (en anglais « dashboard ») qui ressemble de près ou de loin à celui de Xampp.

On notera que, si je saisis cette autre URL, cela fonctionne aussi :

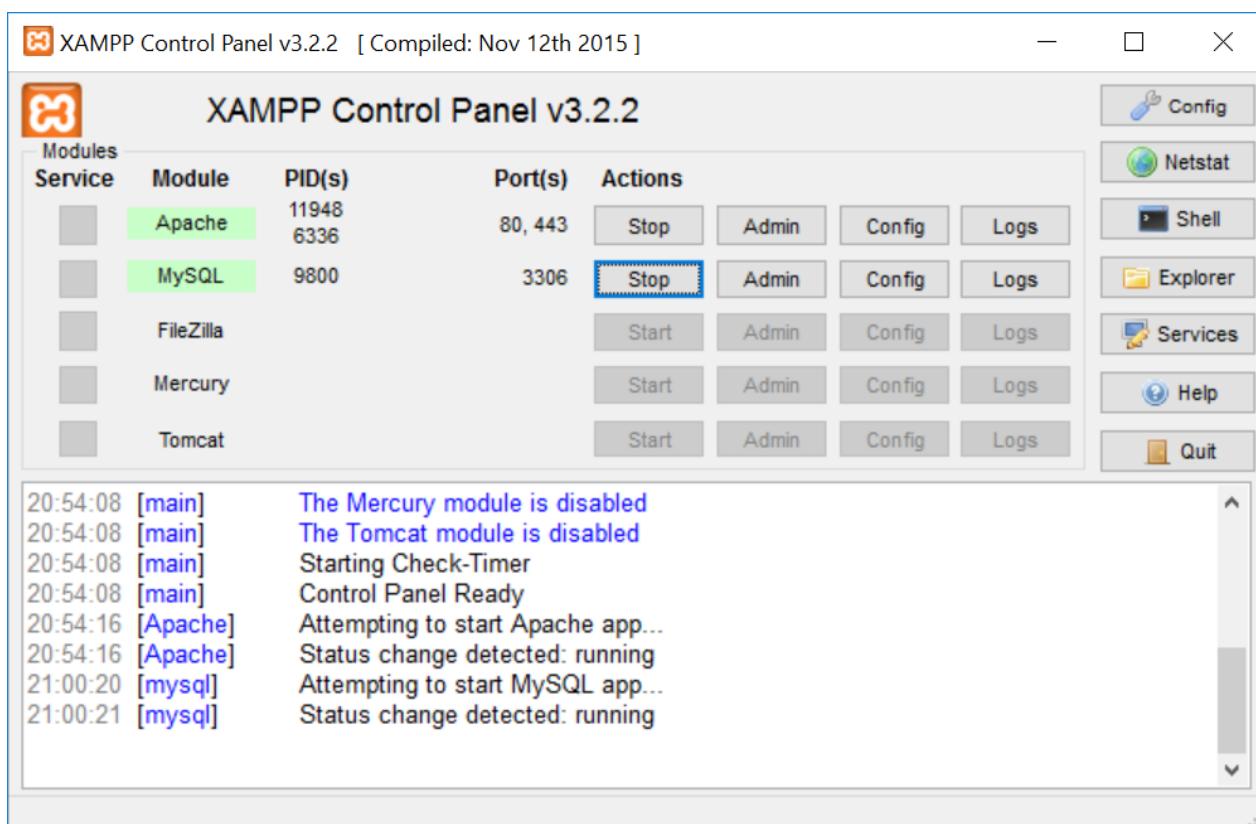
<http://127.0.0.1/>

En effet, dans la configuration du système, l'adresse « localhost » et l'adresse IP « 127.0.0.1 » fonctionnent comme des synonymes. La plupart des systèmes fonctionnent sur ce principe, le vôtre ne devrait pas faire exception.

Tiens, ça ne marche pas sur votre machine ? Que se passe-t-il donc ?

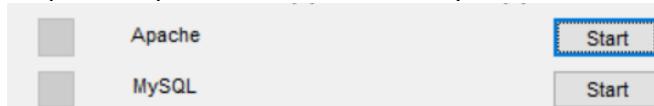
Tout d'abord, est-ce que Xampp est bien démarré ? Vous repérez facilement l'application au moyen de son logo : 

Si l'application Xampp est active, regardez ce qui apparaît dans son « control panel ».



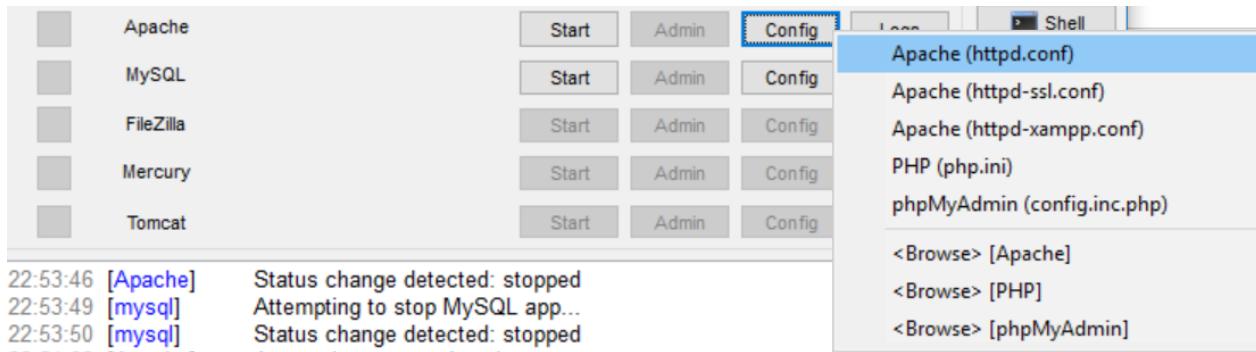
Est-ce que Apache apparaît en vert ? Si oui, tout va bien.

Si Apache apparaît en gris, alors il convient de le démarrer en cliquant sur « Start ». Vous pouvez en profiter pour démarrer aussi MySQL, mais nous n'en aurons pas besoin tout de suite.



Si Apache apparaît en rouge, cela signifie qu'il rencontre un souci au démarrage, il se peut que ce soit dû à un conflit avec un autre logiciel. Une cause possible de conflit, c'est le fait que Apache utilise par défaut le port 80, sur l'adresse IP 127.0.0.1. Si vous avez un autre logiciel qui utilise ce même port sur cette même adresse IP, alors vous avez un conflit.

Si cela arrive, vous pouvez modifier le port en passant par les options de configuration du « control panel » :



Sélectionnez l'option « Apache (httpd.conf) », puis faites défiler l'éditeur jusqu'à la ligne commençant par « Listen 80 » :

The screenshot shows a Windows Notepad window titled "httpd.conf - Bloc-notes". The file contains the Apache configuration file content. The "Listen 80" directive is highlighted in blue. The file also includes comments about listening on specific IP addresses and dynamic shared object support.

```

#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 80

#
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built as a DSO you

```

Modifier la valeur du port, par exemple, vous pouvez remplacer « 80 » par « 81 ». Sauvegardez, fermez l'éditeur, puis essayez de relancer Apache.

Si cela fonctionne, Apache va apparaître en vert.

Dans ce cas, vous pouvez relancer le tableau de bord de Xampp, mais en précisant le port 81 dans l'URL saisie dans le navigateur :

<http://localhost:81/>

Si cela ne fonctionne pas, alors votre problème se situe à un autre niveau, et je vous recommande dans ce cas de lire attentivement la log des erreurs d'Apache (cf. option « Apache error.log » ).



Si vous parvenez à identifier une erreur ou un message suspect, parcourez la FAQ (Foire aux Questions) du projet Xampp :

[https://www.apachefriends.org/faq\\_windows.html](https://www.apachefriends.org/faq_windows.html)

... ainsi que le forum sur lequel vous pouvez poser des questions :

<https://community.apachefriends.org/f/>

Attention : avant de poser une question sur un forum, quel qu'il soit, prenez soin de vérifier si la question que vous souhaitez poser n'a pas déjà été posée. Car si c'est le cas, vous risquez de vous faire « incendier » 😞.

Au risque de me répéter, tout ce que je viens d'indiquer sur Xampp est valable avec les autres stacks PHP. Si les « control panels » des autres stacks se présentent différemment, on y retrouve à peu près les mêmes options :

- Pour l'arrêt et le redémarrage d'Apache et de MySQL
- Pour l'accès aux options de configuration (notamment au fichier httpd.conf)
- Pour l'accès aux logs (notamment celles répertoriant les erreurs)

A propos de log, vous avez sans doute remarqué que nous avions deux logs pour Apache (access.log et error.log) et une pour PHP (php\_error.log) :

Apache (access.log)

Apache (error.log)

PHP (php\_error\_log)

Au fait, d'où vient ce terme de « log ». D'après ce que j'ai pu voir sur Wikipédia, un log « serait un terme de marine désignant le livre de bord d'un bâtiment ». Je ne sais pas si le terme utilisé en informatique vient de là, mais un fichier de log, pour un développeur, c'est un fichier d'historique (historique de connexion, comme pour le fichier « access.log », historique des erreurs, comme pour le fichier « error.log »).

Nous utiliserons ces fichiers de log de temps en temps, ils contiennent beaucoup d'informations intéressantes.

### 3.2 Premier « hello world »

Je pars du principe que votre stack PHP est opérationnel.

Mais au fait, où place-t-on un fichier PHP, au sein du stack PHP, pour pouvoir l'exécuter ?  
En fait, l'emplacement diffère légèrement d'un stack à l'autre.

Sur la plupart des stacks PHP, l'emplacement dans lequel on doit déposer les sources PHP est un sous-répertoire du stack lui-même, qui s'appelle « www » ou « htdocs ».

Dans le cas de Xampp, l'emplacement où déposer vos sources PHP est un sous-répertoire « htdocs » se trouvant à l'intérieur du répertoire « Xampp ».

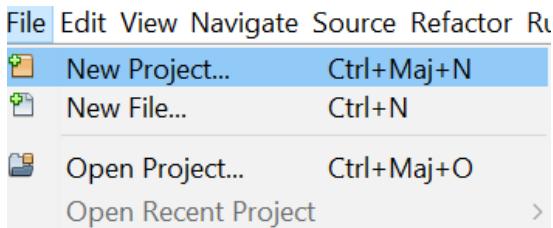
Je vous propose de créer un premier script PHP, tout petit, juste histoire de dire « Bonjour le monde ». Presque tous les développeurs ont démarré l'apprentissage de la programmation par un « hello world », ce serait dommage que vous ratiez ça 😊.

Je vous disais que le répertoire où placer notre code était de type « www » ou « htdocs », selon le stack PHP utilisé. Mais nous n'allons pas placer nos scripts PHP à la racine de ce répertoire, sinon cela va devenir très vite un énorme bazar. Je vous propose donc de créer un sous-répertoire « essaiphp » à l'intérieur de votre répertoire « www » ou « htdocs ». C'est dans ce sous-répertoire « essaiphp » que nous placerons nos premiers scripts PHP. Nous allons considérer que ce sous-répertoire « essaiphp » est notre premier projet PHP. Par la suite, vous pourrez créer d'autres sous-répertoires, sur le même principe.

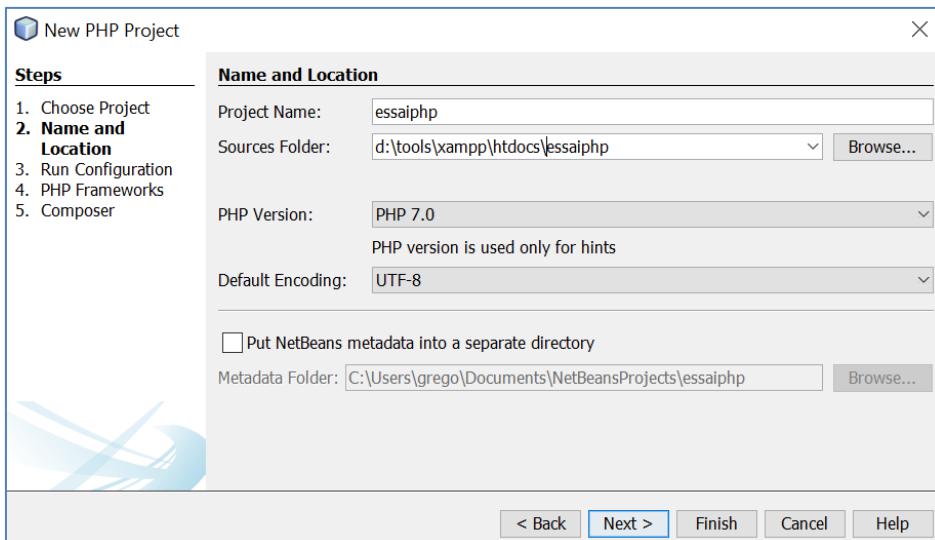
On peut bien évidemment créer le sous-répertoire « essaiphp » à la main, mais nous pouvons aussi le faire au travers d'un éditeur (comme Notepad++ ou SublimeText) ou d'un IDE (comme Netbeans ou Eclipse). Je vous propose de voir comment nous pouvons créer ce répertoire avec Netbeans, vous pourrez facilement transposer la méthode sur d'autres IDE.

Netbeans fonctionne selon une logique de projet. C'est très pratique pour compartimenter les choses, et éviter de mélanger les choux avec les carottes. A chaque projet correspond un répertoire, qui servira de conteneur pour tous les fichiers (scripts PHP, fichiers HTML, CSS, Javascript, etc..) et sous-répertoires du projet.

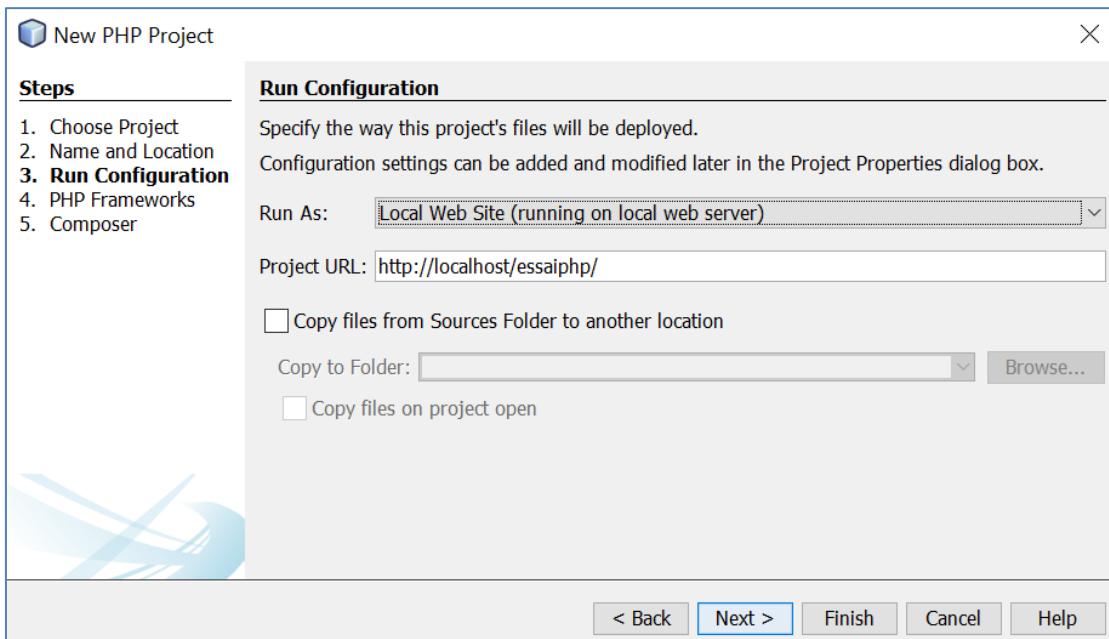
Dans Netbeans, je sélectionne donc l'option « new project » qui se trouve dans le menu « File ».



Puis j'indique le nom du projet, et je modifie le paramètre « sources folder », de manière à pointer sur le sous-répertoire « xampp/htdocs » de mon stack PHP :



Dans la fenêtre qui suit, Netbeans nous indique les paramètres qu'il retient pour la configuration du projet, par rapport au stack PHP en place. Ces paramètres nous permettront de lancer le projet directement à partir de Netbeans :

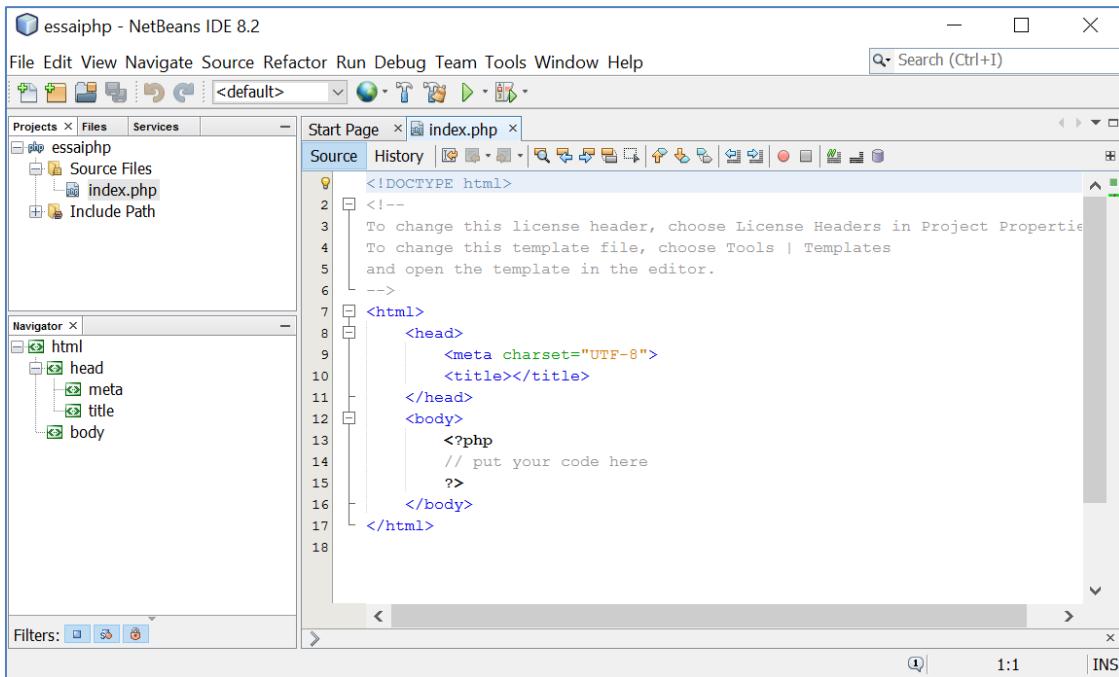


Si vous avez rencontré des conflits au niveau du port, lors de la configuration de votre stack PHP, et que vous avez été obligé de modifier ce port (par exemple en sélectionnant le port 81), alors vous devrez le préciser dans le paramètre Project URL, comme ceci :

<http://localhost:81/essaiphp>

Cliquez sur « Finish », et patientez...

Si tout s'est bien passé, vous devriez obtenir à peu près ceci :



L'écran est découpé en 3 vues. La vue en haut à gauche présente les projets, la vue de droite présente un script PHP qui a été généré automatiquement par Netbeans (il est trop cool le Netbeans), et en bas à gauche, on a une vue « navigator » qui se rapproche beaucoup de la représentation du DOM que nous propose un navigateur web quand on utilise l'inspecteur d'élément.

Le fichier généré automatiquement par Netbeans s'appelle « index.php ». Ce n'est pas innocent, mais je préfère ne pas en parler maintenant. On sait que le fichier généré est un script PHP, du fait de la présence du suffixe « .php » dans le nom du fichier. On reviendra sur ce sujet plus tard également.

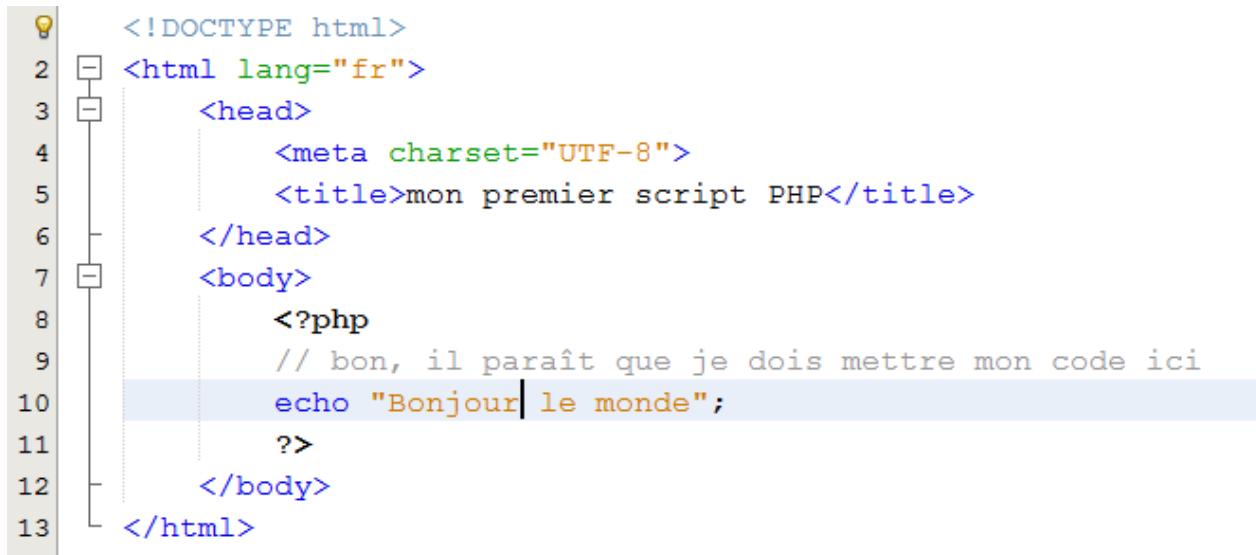
Pour l'instant, je vous demande juste de retenir que le fait que ce fichier soit un script PHP nous autorise à placer du code PHP à l'intérieur.

Pour insérer du code PHP dans un script, on utilise les balises suivantes :

```
<?php
// put your code here
?>
```

Ce sont d'ailleurs ces balises que Netbeans a inséré dans le squelette de script qu'il a généré pour nous. Il les a placées à l'intérieur de la balise « body » du code HTML.

Comme ce script contient des commentaires HTML qui ne servent pas à grand-chose, je vous propose d'élaguer un peu le code pour aboutir au résultat suivant :



```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="UTF-8">
        <title>mon premier script PHP</title>
    </head>
    <body>
        <?php
            // bon, il paraît que je dois mettre mon code ici
            echo "Bonjour le monde";
        ?>
    </body>
</html>
```

Pour vous aider à y voir clair, j'ai effectué les modifications suivantes :

- J'ai ajouté un attribut « lang » sur la balise « html »
- J'ai ajouté un titre dans la balise HTML « title »
- J'ai ajouté l'instruction PHP « echo » à l'intérieur des balises PHP

On peut dès maintenant tester ce script. Pour ce faire, ouvrez votre navigateur préféré, et saisissez l'URL suivante :

<http://localhost/essaiphp>

... ou celle-ci si vous avez changé votre configuration d'Apache pour le port 81 par exemple :

<http://localhost:81/essaiphp>

Si tout s'est bien passé, vous devriez voir apparaître le message suivant dans la fenêtre de votre navigateur :

Bonjour le monde

Si cela ne fonctionne pas, je vous invite à relire le chapitre 3.1 qui je l'espère, vous aidera à identifier l'origine de la panne.

Si vous avez bien obtenu votre message « Bonjour le monde », alors je vous invite à regarder à quoi ressemble le code HTML que le navigateur a reçu. C'est facile, nous l'avons fait souvent dans le cours « HTML5 – Premiers pas », mais je rappelle quand même la méthode : faites un clic-droit sur le fond de la page web, et sélectionnez l'option « affichage du code source ». Vous devriez obtenir ceci :

```

1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="UTF-8">
5     <title>mon premier script PHP</title>
6   </head>
7   <body>
8     Bonjour le monde      </body>
9 </html>
10

```

Bon, j'espère que ça vous rappelle des souvenirs, parce que dans le cas contraire, il faut vraiment que vous fassiez un break et que vous lisiez le cours « HTML5 – Premier pas ».

Dans le code ci-dessus, vous voyez que les balises PHP ont disparu. Pour bien comprendre ce qui s'est passé, je vous propose de mettre côté à côté le code PHP (à gauche) et le code HTML produit par ce même code PHP (à droite) :

<pre> 1 &lt;!DOCTYPE html&gt; 2 &lt;html lang="fr"&gt; 3   &lt;head&gt; 4     &lt;meta charset="UTF-8"&gt; 5     &lt;title&gt;mon premier script PHP&lt;/title&gt; 6   &lt;/head&gt; 7   &lt;body&gt; 8     &lt;?php 9       // bon, il paraît que je dois mettre mon code ici 10      echo "Bonjour le monde"; 11    ?&gt; 12   &lt;/body&gt; 13 &lt;/html&gt; </pre>	<pre> 1 &lt;!DOCTYPE html&gt; 2 &lt;html lang="fr"&gt; 3   &lt;head&gt; 4     &lt;meta charset="UTF-8"&gt; 5     &lt;title&gt;mon premier script PHP&lt;/title&gt; 6   &lt;/head&gt; 7   &lt;body&gt; 8     Bonjour le monde      &lt;/body&gt; 9 &lt;/html&gt; 10 </pre>
--	---

Quand l'interpréteur PHP démarre l'exécution du script « index.php », il commence par ouvrir un « buffer ». Ce terme barbare – le « buffer » - désigne une mémoire tampon, dans laquelle l'interpréteur va envoyer tout le texte qu'on lui demande de produire en sortie.

Parmi les éléments que l'interpréteur envoie dans ce « buffer », on trouve :

- le code HTML « brut », tel qu'il est contenu dans le script PHP, du moment qu'il est placé en dehors des balises PHP : c'est le cas dans notre exemple de la majeure partie de l'entête de la page HTML, et des balises de fermeture « /body » et « /html ».
- Le code se trouvant entre les balises PHP va être analysé et exécuté, et si des instructions telles que « echo » ou « print » s'y trouvent, elles vont envoyer, elles aussi, du texte dans le buffer HTML.

Fort de toutes ces explications, je vous invite à comparer attentivement les 2 codes (PHP et HTML) dans la page qui précède.

Tiens, vous avez sans doute remarqué, que dans le code source HTML généré, la balise « /body » se trouve placée bizarrement. Cela n'est pas très grave, car le code HTML est correct, mais c'est vrai que ça fait bizarre. On peut corriger cela facilement en modifiant légèrement l'instruction « echo ». Essayez ceci :

```
<?php  
// bon, il paraît que je dois mettre mon code ici  
echo "Bonjour le monde" . PHP_EOL ;  
?>
```

Explication : le point placé entre « Bonjour le monde » et PHP\_EOL, c'est le symbole de concaténation en PHP. Je rappelle qu'en Javascript on utilise un « + », mais en PHP c'est un « . ». C'est comme ça, ne me demandez pas pourquoi, je n'y suis pour rien ☺. Pour les grands débutants, je rappelle que le principe de la concaténation, c'est de coller bout à bout des informations pour produire une chaîne de caractères.

Le mot clé PHP\_EOL, c'est une constante PHP. Le « EOL » correspond à « End Of Line », soit en français « fin de ligne ». Cette constante a donc pour effet de générer un caractère de saut de ligne. Vous ne le savez probablement pas, mais cette constante PHP\_EOL nous simplifie la vie, car le caractères de saut de ligne n'est pas le même selon les systèmes d'exploitation :

\r = CR (Carriage Return) // saut de ligne sur Mac OS, avant la génération OS X  
\n = LF (Line Feed) // saut de ligne sur Linux/Unix/Mac OS X  
\r\n = CR + LF // saut de ligne sur Windows

Ce n'est pas stratégique, mais c'est intéressant de connaître la signification de ces caractères :

- « line feed » signifie en anglais « saut de ligne »
- « carriage return » signifie « retour chariot » : le retour chariot, c'est une opération manuelle que l'on devait effectuer sur les vieilles machines à écrire mécanique, chaque fois que l'on souhaitait changer de ligne. Je dois dire que c'est vraiment amusant de taper du texte sur une machine à écrire mécanique. Si vous n'avez jamais eu l'occasion d'en voir fonctionner, je vous recommande de visionner le film "Populaire" de Régis Roinsard, avec Romain Duris en tête d'affiche (le film est sorti en 2012).

Pourquoi sous Windows faut-il utiliser le double jeu de caractères \r\n ? A priori, il s'agissait pour Windows de maintenir une compatibilité avec les vieux systèmes d'exploitation MS-DOS et CP/M. Si le sujet vous intéresse, vous pouvez trouver pas mal d'infos sur cette page :

<https://softwareengineering.stackexchange.com/questions/29075/difference-between-n-and-r-n>

Donc, en résumé, la constante PHP\_EOL nous simplifie la vie car on n'a pas besoin de se préoccuper de savoir quel jeu de caractère utiliser pour générer un saut de ligne sur notre environnement PHP (PHP le gère pour nous au travers de cette constante).

Sauvegardez votre modification et rafraîchissez la page dans le navigateur. En façade, ça n'a rien changé, mais dans les coulisses, le code HTML est mieux formaté :

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="UTF-8">
5     <title>mon premier script PHP</title>
6   </head>
7   <body>
8     Bonjour le monde
9   </body>
10 </html>
```

Voilà, vous venez de découvrir la concaténation en PHP, plus une première constante PHP (PHP\_EOL). Nous en découvrirons d'autres au fil de l'eau.

Dans les chapitres qui suivent, nous allons continuer avec notre « hello world », mais avant ça, je vais vous montrer quelques astuces sympas dans les 2 prochains chapitres.

### 3.3 Quelques astuces à connaître

#### 3.3.1 Astuces d'un haricot du net

Nous allons parler ici de quelques astuces propres à Netbeans.

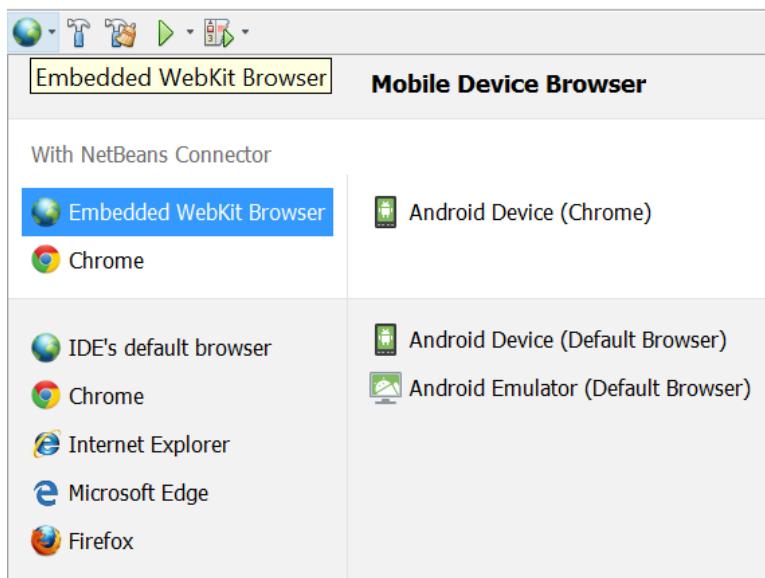
Il y a quelques instants, je vous invitais à lancer votre navigateur et à taper l'URL de votre projet « essaiphp ».

En fait, vous disposez d'un raccourci dans Netbeans pour lancer votre application.

Pour cela, il vous suffit de cliquer sur l'icône en forme de mappemonde, qui correspond à un moteur de rendu HTML, qui s'appelle WebKit, et qui est intégré à Netbeans. WebKit est un moteur très utilisé, et vous pouvez trouver des informations à son sujet sur Wikipédia :

<https://fr.wikipedia.org/wiki/WebKit>

Vous n'êtes pas limité à WebKit, vous pouvez aussi sélectionner Chrome, Firefox, IE, Edge... donc à peu près tous les navigateurs qui sont embarqués sur votre PC.



Sélectionner un navigateur ne suffit pas, il faut ensuite cliquer sur l'icône ➡ pour déclencher l'exécution du script sélectionné. Si vous avez demandé le « Embedded WebKit Browser », alors la page s'affiche dans une vue de Netbeans, dans le cas contraire le navigateur que vous avez sélectionné est « appelé » automatiquement.

### 3.3.2 Index ou pas index ?

Peut être vous êtes-vous demandé pourquoi en saisissant l'URL suivante :

<http://localhost/essaiphp>

... on lance le script index.php.

En fait, si on avait utilisé l'URL suivante, on aurait abouti au même résultat :

<http://localhost/essaiphp/index.php>

Mais par quelle magie ? En fait, Apache est configuré d'une manière particulière :

- Si on lui demande d'exécuter un script particulier à l'intérieur du répertoire « essaiphp », Apache fait le job
- Si on n'indique pas à Apache quel script on souhaite exécuter à l'intérieur de « essaiphp », alors il recherche si un script « index.php » existe dans ce même répertoire. Si c'est le cas, il demande à l'interpréteur PHP de l'exécuter, et il envoie au navigateur la réponse produite par le PHP. Si ce n'est pas le cas, il recherche s'il existe un fichier « index.html ». Si c'est le cas, il l'envoie au navigateur.

Et s'il n'existe pas non plus de fichier « index.html » ? Que se passe-t-il ?

... je vous propose de faire l'expérience. Dans votre IDE, faites un clic-droit sur le fichier « index.php » et sélectionnez l'option « rename ». Changez le nom du fichier en « helloworld.php », puis validez.

Rafraîchissez la page de votre navigateur qui pointe sur l'URL suivante :

<http://localhost/essaiphp>

Que remarquez-vous ?

<b>Index of /essaiphp</b>			
<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">helloworld.php</a>	2017-07-07 14:07	292	
 <a href="#">nbproject/</a>	2017-07-07 00:09	-	

*Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/7.1.6 Server at localhost Port 80*

La configuration d'Apache sur votre environnement de développement est ainsi faite que

Apache vous propose une vue sur le contenu du répertoire. Ce comportement très pratique sur un environnement de développement n'est pas souhaitable sur un environnement de production, pour des raisons de sécurité (on n'a pas envie qu'une application donne aussi facilement aux hackers des informations sur le contenu des répertoires). Du coup, sur une application de production, Apache sera généralement configuré pour renvoyer, soit une page blanche, soit un message d'erreur indiquant que le répertoire n'est pas accessible. Bref, l'Apache de votre stack PHP de développement est bavard comme un vieil indien qui n'aurait pas vu passer de caravane depuis longtemps, et qui taperait la « discute » avec le premier venu. Pas grave, surtout qu'en environnement de développement, ce comportement « bavard » nous arrange.

### **3.4 Mon « hello world » dans mon smartphone**

Vous avez vu votre page « hello world » fonctionner avec le navigateur de votre PC... Cela vous dirait de voir cette même page s'afficher dans le navigateur de votre smartphone ? Vous ne pensiez pas que c'était possible ? Pourtant je vous garantis que vous pouvez le faire :

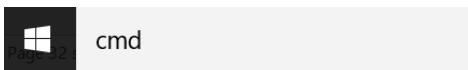


Bon, pour y arriver, il y a un peu de configuration à faire.

En premier lieu, il faut que votre PC (sur lequel « tourne » le stack PHP) et votre smartphone soient connectés à la même borne Wi-fi.

Ensuite vous avez besoin de connaître l'adresse IP de votre PC, car c'est elle que nous allons indiquer dans l'URL du navigateur du smartphone. Par exemple, mon adresse IP ici est 192.168.1.13. Si on mettait « localhost » dans le navigateur du smartphone, on tournerait en rond à l'intérieur du smartphone..., donc il nous faut l'adresse de la machine distante (la machine « remote », comme disent nos amis anglophones).

Pour connaître l'adresse IP de notre PC, nous devons utiliser la ligne de commande du PC. Pour lancer la ligne de commande, nous allons utiliser la commande « cmd » :

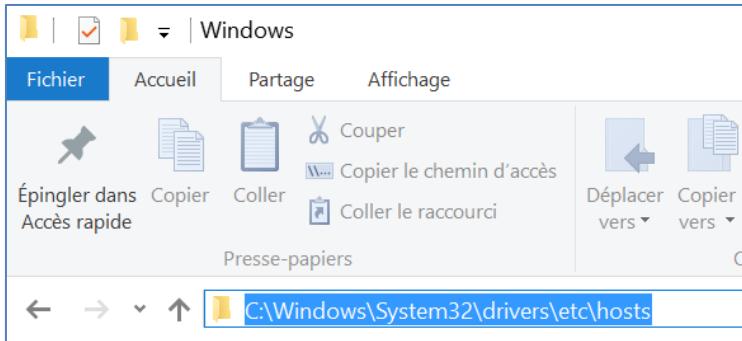


Dès que la fenêtre « invite de commandes » apparaît, saisissez la commande « ipconfig » (puis pressez la touche « Entrée »).

Comme notre connexion se fait via le Wi-fi, c'est l'adresse IP qui se trouve dans la partie « carte réseau sans fil Wi-Fi » qui nous intéresse :

Notez précieusement cette adresse IP, car on va en avoir besoin en fin de parcours.

Vous pouvez d'ores et déjà faire un essai de connexion sur le navigateur de votre smartphone, car selon la configuration de votre PC, il est possible que cela fonctionne tout de suite. Mais si ça ne fonctionne, il faut jeter un coup d'œil à votre fichiers « hosts ». Ce fichier est assez facile à trouver :



C:\Windows\System32\drivers\etc\hosts

```
1 # Copyright (c) 1993-2009 Microsoft Corp.
2 #
3 # This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
4 #
5 # This file contains the mappings of IP addresses to host names. Each
6 # entry should be kept on an individual line. The IP address should
7 # be placed in the first column followed by the corresponding host name.
8 # The IP address and the host name should be separated by at least one
9 # space.
10 #
11 # Additionally, comments (such as these) may be inserted on individual
12 # lines or following the machine name denoted by a '#' symbol.
13 #
14 # For example:
15 #
16 #      102.54.94.97      rhino.acme.com          # source server
17 #              38.25.63.10      x.acme.com            # x client host
18 #
19 # localhost name resolution is handled within DNS itself.
20 #      127.0.0.1      localhost
21 #              ::1            localhost
```

A la fin du fichier « hosts », ajoutez les 2 lignes en gris ci-contre.

Au moment de sauvegarder vos modifications, il y a de grandes chances pour que votre système vous dise que votre éditeur n'a pas le niveau d'autorisation nécessaire pour modifier le fichier « hosts ». Si c'est le cas, il devrait vous proposer de rouvrir l'éditeur en mode « administrateur »... faites-lui plaisir, répondez « oui ». Cela réglera votre problème.

Après toute modification du fichier « hosts », il est recommandé de rebooter, pour que la modification soit prise en compte par le système.

A partir de là, il n'y a plus qu'à lancer le navigateur du smartphone, et à croiser les doigts :



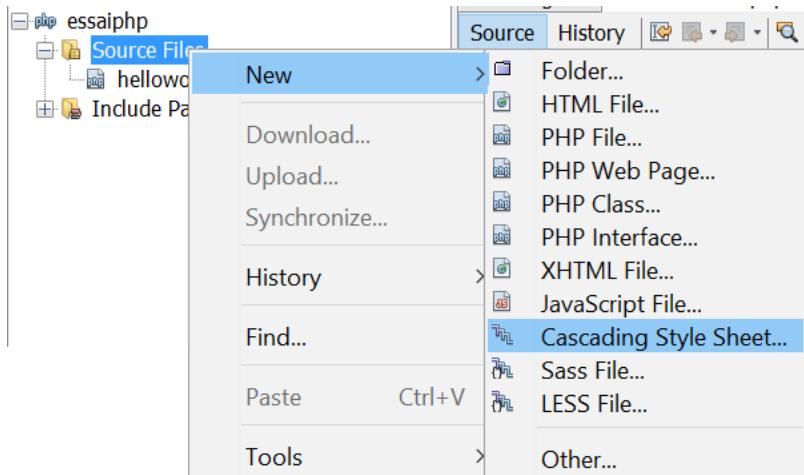
Désormais, vous êtes en mesure de tester tous vos développements sur votre PC, mais aussi sur tout appareil distant (smartphone, tablette, autre PC) qui se trouve connecté à la même borne Wi-Fi. Plutôt cool, non ?

Bon, mais il est toujours aussi moche ce « hello world »... on se le refait ?

### 3.5 Second « hello world » avec du style

Pour donner du style à notre « hello world », nous devons passer par du CSS.

Je vous propose d'ajouter un fichier CSS à notre projet « essaiphp », que nous appellerons « helloworld.css ». Pour cela, un clic-droit dans Netbeans, et le tour est joué :



Dans le fichier CSS créé, saisissez le code suivant :

```
.annonce {  
    font-size: 5em;  
    text-align: center;  
    color: lightblue;  
    text-shadow: 2px 2px 4px #000000;  
    width: 50%;  
    margin: auto;  
}
```

Modifiez le code du script PHP, ou mieux, copiez-le et créez un nouveau script « helloworld2.php ». Voici son code :

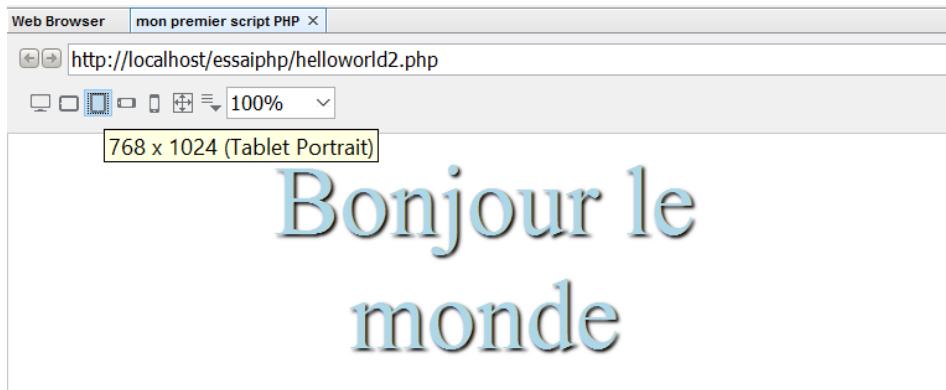
```
<!DOCTYPE html>  
<html lang="fr">  
    <head>  
        <meta charset="UTF-8">  
        <title>mon premier script PHP</title>  
        <link rel="stylesheet" href="helloworld.css" media="screen">  
    </head>  
    <body>  
        <div class="annonce">  
            <?php  
                // bon, il paraît que je dois mettre mon code ici  
                echo "Bonjour le monde" . PHP_EOL ;  
            ?>  
        </div>  
    </body>  
</html>
```

Ah, là c'est vraiment la classe !!! 😊

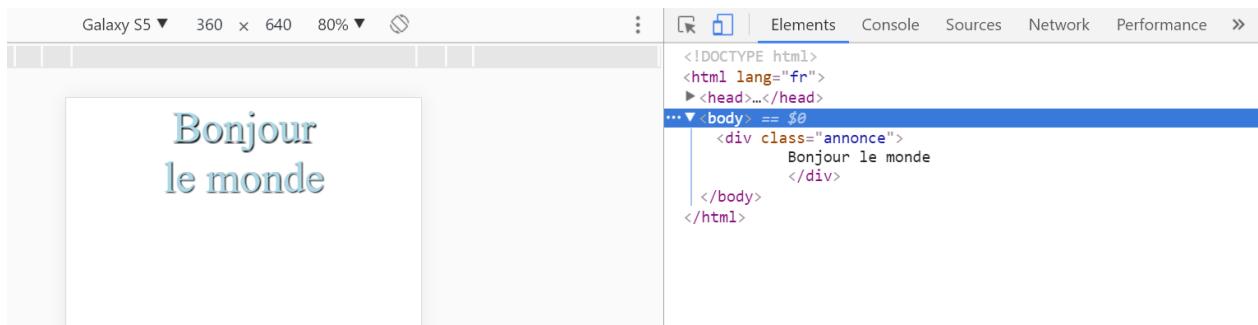


Bonjour le  
monde

Une autre petite astuce à noter au sujet de Netbeans : si vous utilisez le WebKit, vous remarquerez qu'il vous offre une série de boutons permettant de changer la résolution de la vue de simulation :



Je rappelle aussi un point évoqué en annexe du cours « CSS3 – Premiers pas » : le navigateur Chrome est doté d'un simulateur de smartphone, que l'on peut activer en passant par les outils de développement :



Mais nous avons vu dans le chapitre précédent, que nous pouvons tester notre application via notre propre smartphone, et ça c'est vraiment cool !!! 😊

### 3.6 Troisième « hello world » pour découvrir les variables

Notre précédent « hello world » était très basique. Nous avions 2 blocs de code HTML, et un bloc de code PHP glissé entre les deux. En fait, nous avons la possibilité de placer plusieurs blocs de PHP, disséminés dans le code HTML. Nous pouvons par exemple avoir un bloc de code, placé au tout début du script, et qui sert à initialiser des variables. Il pourrait ressembler à ceci :

```
<?php
    $titre = 'Mon 3ème Helloworld';
    $encodage = 'UTF-8';
    $message1 = "Bonjour le monde";
    $message2 = "jouons avec les variables";
    $taille_message2 = 28;
?>
```

Les variables en PHP sont toujours préfixées par un dollar.

Dans l'exemple ci-dessus, nous avons déclaré 4 variables de type chaîne de caractères (en anglais : « string »), et une variable de type nombre entier (en anglais : « integer »).

Vous remarquerez qu'une chaîne de caractères peut être délimitée par des apostrophes ou des guillemets, nous y reviendrons plus tard.

Nous pouvons disséminer les variables un peu partout dans notre code HTML, à condition de toujours les encadrer par les balises PHP. Voici un exemple de ce que cela peut donner :

```
<?php
    $titre = 'Mon 3ème Helloworld';
    $encodage = 'UTF-8';
    $message1 = "Bonjour le monde";
    $message2 = "jouons avec les variables";
    $taille_message2 = 28;
?><!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="<?php echo $encodage; ?>">
        <title><?php echo $titre; ?></title>
        <link rel="stylesheet" href="helloworld.css" media="screen">
    </head>
    <body>
        <div class="annonce">
            <?php
                // bon, il paraît que je dois mettre mon code ici
                echo $message1 . PHP_EOL ;
                echo '<br>' . PHP_EOL ;
            ?>
            <span style="font-size:<?php echo $taille_message2; ?>px">
                <?php
                    echo $message2 . PHP_EOL ;
                ?>
            </span>
        </div>
    </body>
</html>
```

Bon, si vous trouvez que le script PHP de la page précédente est un vrai bazar, je ne peux pas vous contredire. Dans les débuts de PHP, la plupart des scripts PHP ressemblaient à ça. Mais assez rapidement, certaines techniques ont été utilisées pour mieux structurer le code des scripts, et éviter d'aboutir à ce joyeux bazar (et nous allons étudier certaines de ces techniques au fil de l'eau).

Regardons ce que cela donne dans le navigateur :



Nous pouvons demander au navigateur de nous montrer le code source qu'il a reçu :

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="UTF-8">
5     <title>Mon 3ème HelloWorld</title>
6     <link rel="stylesheet" href="helloworld.css" media="screen">
7   </head>
8   <body>
9     <div class="annonce">
10       Bonjour le monde
11     <br>
12     <span style="font-size:28px">
13       jouons avec les variables
14     </span>
15   </div>
16 </body>
17 </html>
18
```

Vous voyez que toutes les variables PHP ont été remplacées par leurs valeurs respectives, il ne reste dans le code source HTML aucune trace du code PHP d'origine. L'interpréteur PHP a donc bien fait son travail.

### 3.7 Quatrième « hello world », avec un peu de JS

Je vous propose de copier le script « helloworld3.php » pour créer un « helloworld4.php ». L'objectif ici, c'est de voir comment se passe l'inclusion de code Javascript dans une page HTML générée via un script PHP.

Nous allons créer un fichier Javascript dans notre répertoire/projet « essaiphp ». Je vous propose de l'appeler « helloworld.js » dont voici le code source :

```
var span_a_cacher = document.querySelector('span');
span_a_cacher.addEventListener('click', function(evt) {
    evt.target.style = "display:none";
}, false);
```

C'est vraiment un code hyper basique, nous avons étudié des choses plus sophistiquées que cela dans le cours « Javascript – Premier pas ». En substance, nous cherchons dans le DOM une balise « span ». Nous prenons la première que nous trouvons et nous lui affectons un écouteur d'événement de type « click » qui va avoir pour effet de faire disparaître la balise « span » et son contenu, dès que nous cliquerons dessus.

Dans notre script « helloworld4.php », ajoutons la ligne ci-dessous juste avant la balise « /body » :

```
<script src="helloworld.js"></script>
```

Il ne reste plus qu'à tester :

Bonjour le monde

jouons avec les variables

Cliquez sur « jouons avec les variables ». Si vous n'avez pas commis d'erreur, cette « span » doit disparaître.

# Bonjour le monde

## 3.8 Dans les coulisses d'un « hello world »

Avant de nous attaquer à une étude plus approfondie du PHP, ce que nous ferons dans la suite de ce cours, je vous propose de faire une pause, et de regarder plus précisément ce qui se passe quand vous saisissez une URL dans le navigateur, et que cette URL a pour effet de déclencher l'exécution d'un script PHP. Pour ce faire, nous allons étudier en profondeur ce qui se passe dans les coulisses de notre quatrième « hello world ».

Quand je saisis dans mon navigateur l'URL suivante :

<http://localhost/essaiphp/helloworld4.php>

... le navigateur déclenche une requête HTTP vers le serveur qui a pour nom « localhost ». Dans une configuration « locale » comme la nôtre, ce « localhost » correspond à l'adresse IP « 127.0.0.1 ».

Si l'URL fait référence à un nom autre que localhost, comme par exemple :

<https://www.meetup.com>

... cette adresse « meetup.com » n'étant pas définie comme une adresse locale sur mon PC, une requête HTTP va être lancée sur le réseau internet, pour rechercher le serveur auquel correspond l'URL demandée.

Notre environnement web local est assez proche d'un environnement web véritable, si ce n'est que tout se passe à l'intérieur de ma machine.

Pour espionner le déroulement des opérations côté navigateur, je vous propose d'activer les outils de développement du navigateur (touche F12 sur certains navigateurs), et d'ouvrir l'onglet « réseau » (ou « network » selon les navigateurs).

Si vous êtes sur Firefox, l'onglet « réseau » est matérialisé par cette icône : 

Commençons par demander au navigateur d'afficher une page avec une URL qui n'existe pas sur notre serveur, comme par exemple :

<http://localhost/essaiphp/helloworldx.php>

Voici ce que nous indique Firefox :

The screenshot shows the Firefox developer tools Network tab. A red box highlights the status bar at the bottom which reads "HTTP/1.1 404 Not Found 210 ms".

Name	Status	Type	Initiator	Size	Time
helloworldx.php	404	doc...	Other	1.5 ...	5 ...

Et voici le résultat pour la même URL dans Chrome :

The screenshot shows the Chrome developer tools Network tab. A red box highlights the status bar at the bottom which reads "HTTP/1.1 404 Not Found 210 ms".

Name	Status	Type	Initiator	Size	Time
helloworldx.php	404	doc...	Other	1.5 ...	5 ...

En tapant l'URL précédente, nous avons lancé une requête HTTP qui a été dirigée vers le serveur web local piloté par notre stack PHP. Ce serveur web a renvoyé une réponse, qui apparaît ici en rouge.

Si l'affichage diffère un peu, on retrouve les mêmes informations, dont une doit vous sauter aux yeux : la colonne « statut » ou « état », indique le code 404. Ce code 404 correspond à l'un des codes retour qu'un serveur web peut renvoyer en réponse à une requête HTTP.

Mais c'est quoi HTTP ?

Le sigle HTTP signifie « Hypertext Transfer Protocol ». Il s'agit d'un « protocole de communication », qui s'appuie sur une norme définissant la manière dont les ordinateurs communiquent sur le réseau internet. Vous pouvez trouver une présentation détaillée sur Wikipédia, que je vous invite à lire (quand vous aurez le temps, ce n'est pas urgent) :

[https://fr.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol)

Pour l'heure, nous nous avons juste besoin de savoir à quoi correspond ce code retour 404.

Il existe plusieurs codes retours, mais les plus importants (à connaître) sont les suivants :

- Code retour 200 : la ressource demandée par la requête est disponible, la requête HTTP a bien abouti et le navigateur a reçu la réponse attendue
- Code retour 404 : la ressource demandée n'est pas disponible (elle a peut être été supprimée, ou peut être n'a-t-elle jamais existé), la requête HTTP n'a pas abouti et le navigateur n'a pas reçu de réponse qu'il soit en mesure de traiter

OK, lançons maintenant la requête correspondant à notre 4<sup>ème</sup> helloworld :

<http://localhost/essaiphp/helloworld4.php>

Voici ce que ça donne dans Firefox :

The screenshot shows the Firefox Developer Tools Network tab. At the top, there's a preview of the page content with the text "Bonjour le monde" and "jouons avec les variables". Below the preview, the Network tab lists three requests:

Etat	Méth...	Fichier	Domaine	So...	Ty...	Tr...	Tai...
200	GET	helloworld4.php	localhost	docu...	html	443 o	443 o
200	GET	helloworld.css	localhost	styl...e...	css	mis en ...	153 o
200	GET	helloworld.js	localhost	script	js	mis en ...	345 o

At the bottom of the Network tab, there's a list of network logs:

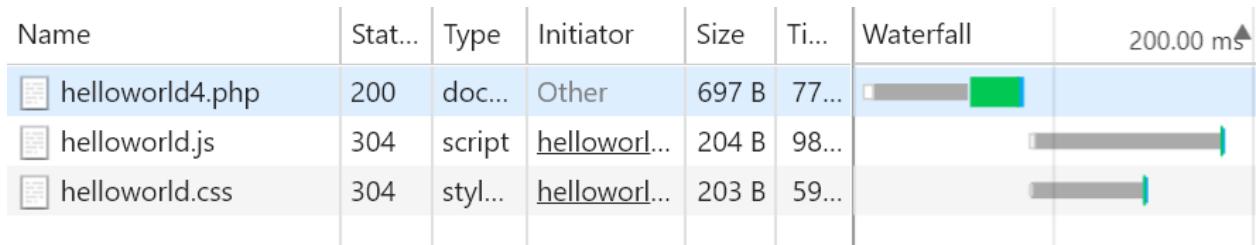
- GET http://localhost/essaiphp/helloworld4.php [HTTP/1.1 200 OK 12 ms]
- GET http://localhost/essaiphp/helloworld.css [HTTP/1.1 200 OK 0 ms]
- GET http://localhost/essaiphp/helloworld.js [HTTP/1.1 200 OK 0 ms]

Et voici la même réponse obtenue dans Chrome :

The screenshot shows the Google Chrome DevTools Network tab. At the top, there's a preview of the page content with the text "Bonjour le monde" and "jouons avec les variables". Below the preview, the Network tab lists three requests:

Name	Stat...	Type	Initiator	Size	Ti...	Waterfall	200.00 ms
helloworld4.php	200	doc...	Other	697 B	77...	███████	███████
helloworld.js	304	script	helloworld...	204 B	98...	████████	████████
helloworld.css	304	styl...	helloworld...	203 B	59...	████████	████████

Je vous propose de regarder plus attentivement le petit tableau proposé par Chrome :



Ce tableau est intéressant car il nous montre, dans la partie de droite, une ligne de temps, dans laquelle on voit que le chargement de la page s'est fait en plusieurs temps :

- Temps 1a : le navigateur lance une requête HTTP vers un serveur (dont l'adresse IP n'apparaît pas ici) pour demander à ce qu'on lui envoie la ressource « helloworld4.php ».
- Temps 1b : le navigateur reçoit une réponse avec un code retour « 200 » qui signifie « tout va bien ».
- Temps 1c : le navigateur analyse le contenu de la réponse, il s'agit de code HTML, comme nous le vérifierons dans un instant.
- Temps 2 : Si vous vous souvenez du code produit par notre script « helloworld4.php », vous savez que le code HTML produit par le script PHP fait référence à 2 fichiers externes (« helloworld.js » et « helloworld.css ») qui se trouve dans le même répertoire que le script PHP lui-même. C'est ce que découvre le navigateur en analysant le code HTML renvoyé par la ressource « helloworld4.php ». Du coup le navigateur lance 2 nouvelles requêtes HTTP pour demander au serveur de lui envoyer les ressources « helloworld.js » et « helloworld.css ».

Les étapes que je viens de décrire se sont déroulées en moins d'un 1/5<sup>ème</sup> de seconde, ce qui explique que vous ayiez l'impression que le chargement de la page est instantané. Et c'est vrai qu'à notre échelle, il est instantané.

Je vous propose de regarder quelques points plus en détail.

Cliquez sur la ligne correspondant à la ressource « helloworld4.php ». Vous voyez apparaître plein d'informations, en fait il s'agit d'une présentation détaillée de la requête HTTP.

Name

- helloworld4.php**
- helloworld.css
- helloworld.js

Headers

Request URL: http://localhost/essaiphp/helloworld4.php

Request Method: GET

Status Code: 200 OK

Remote Address: [::1]:80

Referrer Policy: no-referrer-when-downgrade

Response Headers

Connection: Keep-Alive

Content-Length: 443

Content-Type: text/html; charset=UTF-8

Date: Mon, 10 Jul 2017 09:02:30 GMT

Keep-Alive: timeout=5, max=99

Server: Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/7.1.6

X-Powered-By: PHP/7.1.6

Request Headers

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8

Accept-Encoding: gzip, deflate, br

3 requests | 696 B transferred ...

Il y a plein d'informations à « gratter » là-dedans, mais hormis quelques points de détail sur lesquels je reviendrai à l'occasion, pour le reste c'est assez barbant. Mais il y a un truc que je veux absolument vous montrer. Cliquez sur l'onglet « response » ;

Name

- helloworld4.php**
- helloworld.css
- helloworld.js

Headers

Preview

Response

Cookies

Timing

```

1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="UTF-8">
5     <title>Mon 4ème Helloworld</title>
6     <link rel="stylesheet" href="helloworld.css" type="text/css" />
7   </head>
8   <body>
9     <div class="annonce">
10       Bonjour le monde
11       <br>
12       <span style="font-size:28px">
13         jouons avec les variables
14       </span>
15     </div>
16     <script src="helloworld.js"></script>
17   </body>
18 </html>

```

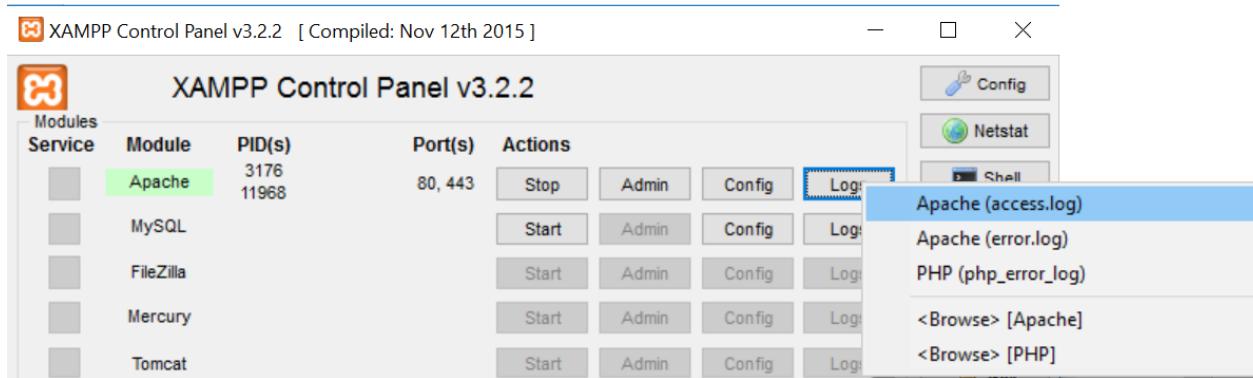
Vous le reconnaissiez ? C'est le code HTML produit par le script PHP. Tout est là. C'est ce bloc de

code HTML que le navigateur va exploiter pour construire la page. C'est aussi en analysant ce code que le navigateur va comprendre qu'il doit lancer des requêtes HTTP pour demander la livraison de ressources complémentaires (les fameux fichiers « helloworld.js » et « helloworld.css »).

Voilà, vous savez tout ou presque, sur ce qui se passe côté navigateur (côté « client » comme on le dit souvent).

Mais que se passe-t-il dans le même temps côté serveur, donc en particulier du côté d'Apache. Eh bien, nous pouvons savoir beaucoup de choses en analysant les fichiers de log générés par Apache.

Nous allons nous intéresser en particulier au fichier « access.log ». Sur Xampp, nous pouvons y accéder en cliquant sur le bouton « Logs » :



Nous allons prélever quelques échantillons de ce fichier :

- le 10 juillet à 10h42, requête vers la ressource « helloworldx.php » qui n'existe pas (on retrouve le fameux code erreur 404)

```
127.0.0.1 - - [10/Jul/2017:10:21:49 +0200] "GET /essaiphp/helloworldx.php HTTP/1.1" 404 1149 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:54.0) Gecko/20100101 Firefox/54.0"
```

- Un peu plus tard le même jour, nouvelle requête vers la ressource « helloworld4.php » (avec code retour 200), suivi de 2 requêtes complémentaires (pour les fichiers CSS et JS) :

```
127.0.0.1 - - [10/Jul/2017:10:42:10 +0200] "GET /essaiphp/helloworld4.php HTTP/1.1" 200 443 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:54.0) Gecko/20100101 Firefox/54.0"
::1 - - [10/Jul/2017:10:42:41 +0200] "GET /essaiphp/helloworld4.php HTTP/1.1" 200 443 "-" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
::1 - - [10/Jul/2017:10:42:41 +0200] "GET /essaiphp/helloworld.css HTTP/1.1" 304 - "http://localhost/essaiphp/helloworld4.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
::1 - - [10/Jul/2017:10:42:41 +0200] "GET /essaiphp/helloworld.js HTTP/1.1" 304 - "http://localhost/essaiphp/helloworld4.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
```

Concernant la première ressource, le fameux fichier « helloworld4.php », Apache a détecté grâce à l'extension du fichier, qu'il s'agissait d'un script PHP. Apache a vérifié que cette ressource était bien présente sur le serveur. S'il ne l'avait pas trouvée, il aurait renvoyé un code retour 404, mais comme la ressource est bien présente, Apache a demandé à l'interpréteur PHP de l'exécuter.

L'interpréteur PHP a exécuté le script et renvoyé le bloc de code HTML produit par le script PHP. Apache a pris ce bloc de code, en a fait un joli paquet cadeau avec un ruban portant la mention « réponse HTTP », et l'a envoyé au client (notre navigateur). C'est cette fameuse réponse que nous regardions en détail, si vous prenez la peine de remonter de 2 pages en arrière.

Voilà, vous savez tout, ou presque, sur la manière dont fonctionne un échange de type « client-serveur » basé sur le protocole HTTP. Nous explorerons quelques aspects complémentaires (que j'ai volontairement laissés de côté) quand nous aborderons la gestion des formulaires.

Tout ça peut sembler un peu compliqué. J'espère avoir néanmoins réussi à vous présenter la chose de manière pas trop barbante.

C'est très intéressant de connaître les mécanismes que nous venons de décrire. C'est bien sûr intéressant d'un point de vue intellectuel, mais c'est surtout très utile quand vous commencez à rencontrer des difficultés, et que vous ne parvenez pas à comprendre d'où vient le problème. Est-ce que la requête HTTP est mal formatée ? Il se peut tout simplement que l'URL soit fausse, et que la requête ne parvienne pas au serveur. Ou alors la requête est correcte, mais il y a un problème côté serveur. Grâce aux connaissances que vous venez d'acquérir, vous arriverez à comprendre ce qui se passe, et vous ne serez jamais pris au dépourvu.

### 3.9 Déconstruisons notre « hello world »

Nous avons vu avec le 4<sup>ème</sup> Hello world, que le code commençait à devenir confus. Je vous le remets ci-dessous, pour vous rafraîchir la mémoire :

```
<?php
    $titre = 'Mon 4ème Helloworld';
    $encodage = 'UTF-8';
    $message1 = "Bonjour le monde";
    $message2 = "jouons avec les variables";
    $taille_message2 = 28;
?><!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="<?php echo $encodage; ?>">
        <title><?php echo $titre; ?></title>
        <link rel="stylesheet" href="helloworld.css" media="screen">
    </head>
    <body>
        <div class="annonce">
            <?php
                // bon, il paraît que je dois mettre mon code ici
                echo $message1 . PHP_EOL ;
                echo '<br>' . PHP_EOL ;
            ?>
            <span style="font-size:<?php echo $taille_message2; ?>px">
                <?php
                    echo $message2 . PHP_EOL ;
                ?>
            </span>
        </div>
        <script src="helloworld.js"></script>
    </body>
</html>
```

... Non, franchement, ce code ne fait que 29 lignes, et on se croirait déjà dans l'auberge espagnole 😊. Comment pourrait s'y prendre pour que ce code soit moins confus ?

Nous pourrions commencer par circonscrire le code HTML dans des variables.

Regardons ce que cela pourrait donner. Voici le script « helloworld4a.php » :

```

<?php
$titre = 'Mon 4ème Helloworld revisité';
$encodage = 'UTF-8';
$message1 = "Bonjour le monde";
$message2 = "jouons avec les variables";
$taille_message2 = 28;

$doctype = '<!DOCTYPE html>' . PHP_EOL;
$page_debut = '<html lang="fr">' . PHP_EOL;
$page_fin = '</html>';
$entete_debut = '<head>' . PHP_EOL ;
$entete_fin = '</head>' . PHP_EOL;
$entete_bloc = '<meta charset="' . $encodage . '">' . PHP_EOL .
    '<title>' . $titre . '</title>' . PHP_EOL .
    '<link rel="stylesheet" href="helloworld.css" media="screen">' . PHP_EOL ;

$corps_entete = '<body>' . PHP_EOL ;
$corps_bloc = '<div class="annonce">' . PHP_EOL .
    $message1 . PHP_EOL .
    '<br>' . PHP_EOL .
    '<span style = "font-size:' . $taille_message2. 'px" >' . $message2 .
    PHP_EOL . '</span >' . PHP_EOL;

$corps_bloc .= '</div >' . PHP_EOL . '<script src="helloworld.js" ></script
>' . PHP_EOL ;
$corps_fin = '</body >' ;

// Génération du code HTML avec les "echo" ci-dessous
echo $doctype;
echo $page_debut;
echo $entete_debut;
echo $entete_bloc;
echo $entete_fin;
echo $corps_entete;
echo $corps_bloc;
echo $corps_fin;
echo $page_fin;

```

Bon, quelques explications s'imposent :

- Premier détail, vous remarquez qu'il n'y a plus qu'une seule balise d'ouverture du code PHP, celle qui se trouve au tout début : **<?php**
- Du coup on ne passe plus son temps à ouvrir et fermer les balises PHP comme on le faisait dans la version antérieure
- Dans cette nouvelle version, j'ai créé tout un jeu de variables de type « chaînes de caractères », variables que j'ai alimentées avec des bouts de code HTML (une pour le

- doctype, une pour l'entête de page, etc...)
- Pour générer les différents blocs de code HTML, j'ai usé, et même abusé, de la concaténation PHP, dont je rappelle qu'elle se fait avec le point ( . )
  - Pour produire un code HTML lisible, j'ai aussi utilisé la constante PHP\_EOL, dont je rappelle ici qu'elle permet de générer un saut de ligne à l'intérieur du code HTML.

Si vous oubliez d'insérer des sauts de ligne à l'intérieur de votre code HTML, ce n'est pas dramatique, cela ne change rien pour le navigateur et pour le rendu final de la page. En revanche, un code HTML sans saut de ligne est plus difficile à lire, et il est du même coup plus difficile de repérer des erreurs s'il y en a.

Voici un exemple de code source HTML produit par une version du script PHP n'utilisant pas la constante PHP\_EOL (donc pas de saut de ligne) :

```
<!DOCTYPE html><html lang="fr"><head><meta charset="UTF-8"><title>Mon 4ème Helloworld revisité</title><link rel="stylesheet" href="helloworld.css" media="screen"></head><body><div class="annonce">Bonjour le monde<br><span style = "font-size:28px" > jouons avec les variables</span></div><script src="helloworld.js" ></script ></body > </html>
```

Argh !! c'est imbuvable !!

Voici le même code HTML avec des sauts de ligne :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<title>Mon 4ème Helloworld</title>
<link rel="stylesheet" href="helloworld.css" media="screen">
</head>
<body>
<div class="annonce">
Bonjour le monde
<br>
<span style = "font-size:28px" > jouons avec les variables
</span >
</div >
<script src="helloworld.js" ></script >
</body > </html>
```

Ah, là on respire mieux 😊.

Tiens, pendant que vous saisissez votre code, peut être avez-vous rencontré quelques erreurs. Je vais partir du principe que vous êtes très fort, et que vous n'avez commis aucune erreur. Aussi, comme je souhaite vous montrer quelques types d'erreur que vous êtes susceptible de rencontrer, je vais modifier la fin de mon script PHP en introduisant 2 erreurs que j'ai indiquées avec des commentaires :

```
// Génération du code HTML avec les "echo" ci-dessous
echo $doctype;
echo $page_debut;
echo $entete_debut;
echo entete_bloc; // aïe, j'ai oublié le dollar devant le nom de la variable
echo $entetex_fin; // aïe, le nom de la variable est erroné, elle n'existe pas
echo $corps_entete;
echo $corps_bloc;
echo $corps_fin;
echo $page_fin;
```

Voyons ce que ça donne dans le navigateur :

**Notice:** Use of undefined constant entete\_bloc - assumed 'entete\_bloc'  
in D:\Tools\xampp\htdocs\essaiphp\helloworld4ax.php on line 30  
entete\_bloc

**Notice:** Undefined variable: entetex\_fin  
in D:\Tools\xampp\htdocs\essaiphp\helloworld4ax.php on line 31

Il s'agit de messages de type « notice », donc des alertes. Nous avons ici 2 alertes, produites par le code se trouvant sur les lignes 30 et 31, que j'ai modifiées il y a quelques instants. A noter que l'on retrouve ces alertes dans le fichier « php\_error.log », que vous pouvez consulter dans Xampp en passant par ce menu :



Bon, c'est quoi une alerte de type « notice » ? Il s'agit d'une anomalie potentielle ou avérée (l'interpréteur PHP ne prend pas parti, c'est à vous d'apprecier la gravité du problème).

Les alertes de type « notice » apparaissent car la configuration de PHP dans mon stack PHP est faite de telle manière que ces alertes sont envoyées directement dans le flux de données destiné au navigateur.

En revanche, sur un environnement de production, l'affichage de ces erreurs est bloqué pour ne pas perturber l'utilisateur. Mieux vaut s'intéresser au plus tôt à ces erreurs, et les éliminer dès qu'elles apparaissent.

J'ai trouvé dans la doc officielle de PHP cet extrait que je trouve intéressant :

*Activer le rapport d'erreur de niveau E\_NOTICE durant le développement a des avantages. En terme de débogage, les messages d'alertes vous signalent des bogues potentiels dans votre code. Par exemple, l'utilisation de valeurs non initialisées est signalée. Il est aussi plus pratique pour trouver des coquilles, et, ainsi, gagner du temps. Les messages NOTICE vous signaleront aussi les mauvaises pratiques de codage. Par exemple \$arr[item] doit toujours être écrit \$arr['item'] car PHP va considérer "item" comme une constante, au premier abord. Si cette constante n'est pas définie, alors il va l'utiliser comme une chaîne.*

Source : <http://php.net/manual/fr/errorfunc.configuration.php#ini.error-reporting>

Bon, mais franchement, vous en pensez quoi de ce code ? Est-il vraiment plus lisible que le précédent ? N'y aurait-il pas mieux à faire ? Eh bien, si bien sûr, en fait il y a plein de manières différentes de faire, mais pour nous devons au préalable en savoir un peu plus sur le fonctionnement des chaînes de caractères en PHP. C'est ce que je vous propose d'aborder dans le chapitre suivant (et nous en profiterons pour voir une dernière version de « hello world »).

## 4 Les bases du langage PHP

### 4.1 Les chaînes de caractères

Il existe 4 manières de créer des chaînes de caractères en PHP.

1. Chaîne délimitée par des apostrophes
2. Chaîne délimitée par des guillemets
3. Syntaxe Heredoc
4. Syntaxe Nowdoc (apparue sur PHP 5.3)

Entre les techniques 1 et 2, il y a une différence importante, que je me propose d'expliquer en prenant un petit bout de notre dernier « hello world ».

`<span style="font-size:28px">jouons avec les variables</span>`

Pour créer cette « span » avec des apostrophes, on peut s'y prendre de plusieurs manières.

On peut par exemple initialiser une variable, puis la compléter par petits bouts, en utilisant le raccourci « .= » qui signifie grosso modo « je te colle au derrière le bout de code qui se trouve à droite du égal » :

```
$mon_message = '<span style="font-size:' ;
$mon_message .= $taille_message2 ;
$mon_message .= 'px">';
$mon_message .= $message2;
$mon_message .= '</span>' ;
```

Mais on peut aussi faire du « tout en un » :

```
$mon_message = '<span style="font-size:' .
    $taille_message2.'px">'.$message2.'</span>' ;
```

Les chaînes délimitées avec des guillemets ont une particularité que celles avec apostrophes n'ont pas : on peut y insérer des variables, ce qui va déclencher un mécanisme d'interpolation ayant pour effet de remplacer les variables par leur contenu.

Exemple :

```
$nom1 = 'titi';
$nom2 = 'gros minet';
echo "$nom1 a piégé $nom2"; //=> titi a piégé gros minet
echo '$nom1 a piégé $nom2'; //=> $nom1 a piégé $nom2
```

Si on reprend notre exemple de la balise « span », cela donne :

```
$mon_message = "<span style=\"$taille_message2px\">
$message2</span>" ;
```

On peut aussi délimiter les variables interpolées en utilisant des accolades (personnellement j'aime bien, mais ce n'est pas obligatoire) :

```
$mon_message = "<span style=\"$font-size:{$taille_message2}px\">
$message2</span>" ;
```

A l'arrivée, on va bien obtenir ceci :

`<span style="font-size:28px">jouons avec les variables</span>`

Mais au fait, pourquoi ai-je ajouté des barres obliques qu'on appelle « antislash » ? Il s'agit d'un caractère qu'on appelle « caractère d'échappement ». Il nous sert à indiquer que le caractère qui le suit - en l'occurrence il s'agit d'un guillemet - ne doit pas être interprété par l'interpréteur PHP. S'il était interprété, il serait considéré à tort comme caractère de fermeture de la chaîne qui a été « ouverte » avec ce même caractère guillemet.

Nous avons vu les chaînes délimitées par des guillemets et des apostrophes.

Bon, et la syntaxe HEREDOC, c'est quoi au juste ?

Le plus simple c'est de l'étudier au travers d'un exemple. La syntaxe HEREDOC se reconnaît facilement car elle début avec ce symbole <<<, suivi du nom du bloc (et là vous pouvez vous lâcher, c'est « open-bar ») :

```
$varbidon = 'je veux';

$mon_texte = <<<BLOC_TEXTE
j'écris ce que je veux ici, c'est génial,
pas besoin de s'embêter avec des "caractères d'échappement"
et je peux placer des variables interpolées où $varbidon
BLOC_TEXTE;
```

La syntaxe HEREDOC, honnêtement j'aime beaucoup ☺. Il y a juste une petite contrainte, c'est que le nom du bloc que j'ai appelé ici « BLOC\_TEXTE » doit impérativement être écrit – sur la ligne délimitant la fin du bloc – à partir du premier caractère de la ligne. Et il ne faut mettre aucun caractère (même pas un blanc), après le point virgule de cette même ligne. A part ça ce petit détail, c'est impeccable.

La syntaxe NOWDOC, apparue avec PHP 5.3, pour l'instant je vous avoue que je ne lui ai pas trouvé beaucoup d'utilité. En résumé, c'est la même chose que HEREDOC, mais sans l'interpolation de variable.

Fort de toutes ces connaissances, nous pouvons réécrire notre « hello world » en exploitant la technique d'interpolation et la syntaxe HEREDOC. Voici une possibilité d'implémentation :

```
<?php
$titre = 'Mon 4ème Helloworld';
$encodage = 'UTF-8';
$message1 = "Bonjour le monde";
$message2 = "jouons avec les variables";
$taille_message2 = 28;

$mon_message = "<span style=\"font-size:{$taille_message2}px\>{$message2}</span>" ;

$mon_html = <<<BLOC_HTML
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset=\"$encodage\">
        <title>$titre</title>
        <link rel="stylesheet" href="helloworld.css" media="screen">
    </head>
    <body>
        <div class="annonce">
            $message1
            <br>
            $mon_message
        </div>
        <script src="helloworld.js"></script>
    </body>
</html>
BLOC_HTML;

echo $mon_html;
```

Si vous souhaitez approfondir vos connaissances sur les chaînes de caractères PHP :  
<http://php.net/manual/fr/language.types.string.php>

## 4.2 Introduction aux variables, constantes, conditions, boucles

Dans les chapitres précédents, et en particulier dans certains exemples de « hello world », nous avons utilisé quelques variables et quelques constantes. Nous allons dans ce nouveau chapitre approfondir ces notions, et quelques autres.

Pour tester les différents exemples présentés dans ce chapitre, vous pouvez détourner un des « hello world » que nous avons créé dans les chapitres précédents, ou vous pouvez effectuer vos tests directement sur le site « PHP Sandbox » dont je rappelle le lien ci-dessous :

<http://sandbox.onlinephpfunctions.com/>

### 4.2.1 Variables, types, conditions

En PHP, comme sur la plupart des langages de programmation, vous avez besoin de conserver temporairement des informations, en vue de pouvoir les utiliser à différentes étapes du script durant son exécution. Imaginez les variables comme les tiroirs d'une grande armoire, cette grande armoire étant la mémoire de votre ordinateur (ou plus exactement de la mémoire allouée à l'interpréteur PHP par votre système).

Une variable en PHP se définit par un nom précédé du symbole dollar (\$). Nous en avons déjà vu quelques exemples, tels que :

```
$titre = 'Mon 4ème Helloworld';
$encodage = 'UTF-8';
$message1 = "Bonjour le monde";
$message2 = "jouons avec les variables";
$taille_message2 = 28;
```

PHP fonctionne selon le principe du typage faible, notamment comme Javascript. Cela signifie que le type d'une variable dépend de son contenu. Si ce contenu change, le type de la variable change aussi, et ces changements peuvent intervenir au sein d'un même script :

```
$taille_message2 = '28'; // variable de type string (chaîne)
$taille_message2 = 28; // variable de type integer (nombre entier)
```

Dans des langages utilisant un typage fort, comme Java, le type de la variable est précisé dès sa création, et on n'a pas le droit de le modifier en cours de route.

En plus des variables personnalisées (celle que vous créez vous-même), PHP fournit un certain nombre de variables prédéfinies. Comme pour les variables personnalisées, le nom des variables prédéfinies commence par un dollar. Mais le nom lui-même est en majuscule. On trouve par exemple `$_POST`, `$_GET`, `$_SESSION`, `$_GLOBALS`, etc... Vous trouverez un descriptif détaillé sur le lien suivant, mais n'y passez pas trop de temps, nous en étudierons un certain nombre au fil de l'eau :

<http://php.net/manual/fr/reserved.variables.php>

Je rappelle donc que PHP fonctionne sur le principe du typage faible, que les anglophones désignent par « weakly typed » (ou « loosely typed »). Du coup, il est parfois – et même souvent – nécessaire de connaître le type d'une variable, pour déterminer de quelle manière on va pouvoir la manipuler (si c'est une chaîne, on n'emploiera pas les mêmes fonctions que s'il s'agit d'un nombre, ou on pourra être amené à effectuer une conversion de type, en préalable à un traitement).

Heureusement, on dispose d'un certain nombre de fonctions bien pratiques, permettant de connaître et manipuler le type et le contenu des variables. Parmi ces fonctions, on trouve : `gettype()`, `settype()`, `isset()`, `unset()`, `empty()`, plus un jeu de fonctions telles que `is_int()`, `is_array()`, `is_numeric()`, etc...

Avant de présenter certaines de ces fonctions, il me semble utile de préciser que plusieurs de ces fonctions renvoient une valeur booléenne, qui peut être « vrai » (en anglais : « true »), ou « faux » (en anglais : « false »). Ce résultat « vrai » ou « faux » peut être stocké dans une variable, qui va du coup être de type « booléen ». Tout cela se raccroche bien évidemment à l'algèbre booléenne, qui tire son nom d'un célèbre logicien, mathématicien et philosophe britannique, j'ai nommé Georges Boole : [https://fr.wikipedia.org/wiki/George\\_Boole](https://fr.wikipedia.org/wiki/George_Boole)

Certains langages de programmation avec lesquels on travaillait dans les années 80 et 90 ne savaient pas gérer le type booléen. Du coup, chaque fois que l'on avait besoin de stocker le résultat d'un test, on utilisait traditionnellement une variable numérique de type « integer » (nombre entier) dans laquelle on stockait 0 (zéro) pour la valeur « faux » et 1 pour la valeur « vrai ». Si je vous en parle ici, c'est pour vous montrer qu'il reste un peu de cette logique dans le PHP. Si j'affiche une variable booléenne contenant « true », je ne vais pas obtenir « vrai », mais « 1 ». Et pour « false », c'est encore pire, car j'obtiens un blanc (même pas un zéro). Pour clarifier mon propos, j'ai mis un petit de code PHP en exemple sur la page suivante. Je vous invite à tester ces bouts de code sur « PHP Sandbox » ou dans votre stack PHP local.

```

$test = true;
echo $test . PHP_EOL; // affiche 1
if ($test) {
    echo "vrai".PHP_EOL;
} else {
    echo "ce texte ne devrait pas s'afficher";
}

$test = false;
echo $test . PHP_EOL; // affiche un blanc, et non pas un 0
if (!$test) {
    echo "faux".PHP_EOL;
} else {
    echo "ce texte ne devrait pas s'afficher";
}

```

On voit que les valeurs booléennes « true » et « false » ne doivent en aucun cas être utilisées directement pour de l'affichage. Il faut toujours passer par un test comme ceux que vous pouvez lire ci-contre.

Tiens, en parlant de tests, vous allez revoir cette structure que l'on appelle « test » à plusieurs reprises, alors autant en parler un petit peu maintenant.

Quand je souhaite déterminer si une condition est remplie (par exemple si 1 est bien égal à 1), je peux écrire ceci :

```

if (1==1) {
    echo "vrai".PHP_EOL;
} else {
    echo "faux".PHP_EOL; // ne se produira jamais compte tenu de la condition
}

```

En notation algorithmique on écrirait ceci :

```

Si 1==1
    Alors
        Afficher "vrai" suivi d'un saut de ligne
    Sinon
        Afficher "faux" suivi d'un saut de ligne
FinSi

```

Le test ci-dessus est carrément idiot, c'était juste pour vous montrer le principe. Dans la « vraie vie » vous serez parfois confronté au besoin d'effectuer plusieurs fois un même test. Pour éviter une répétition pénible, qui peut être source d'erreur, vous aurez intérêt à stocker le résultat de ce test dans une variable. Vous pourrez ainsi évaluer le contenu de cette variable (pour voir s'il est à « vrai » ou « faux»), au moment où cela vous arrange. Dans le cas de notre test précédent, qui est toujours aussi idiot, cela donnerait cela :

```
$test = 1==1; // stocke la valeur "true" dans la variable $test
if ($test == true) {
    echo "vrai".PHP_EOL;
} else {
    echo "faux".PHP_EOL;
}
```

PHP nous permet de simplifier l'écriture ci-dessus, en écrivant la condition ci-dessous, qui est strictement équivalente :

```
if ($test) {
    echo "vrai".PHP_EOL;
} else {
    echo "faux".PHP_EOL;
}
```

Dans un test, nous pouvons souhaiter privilégier le cas où la condition renvoie « faux », ou en tout cas nous pouvons

```
if ($test == false) {
    echo "false".PHP_EOL;
} else {
    echo "vrai".PHP_EOL;
}
```

PHP nous permet - là encore - de simplifier l'écriture ci-dessus, en écrivant la condition ci-dessous (avec le point d'exclamation, qui peut se lire « non vrai ») :

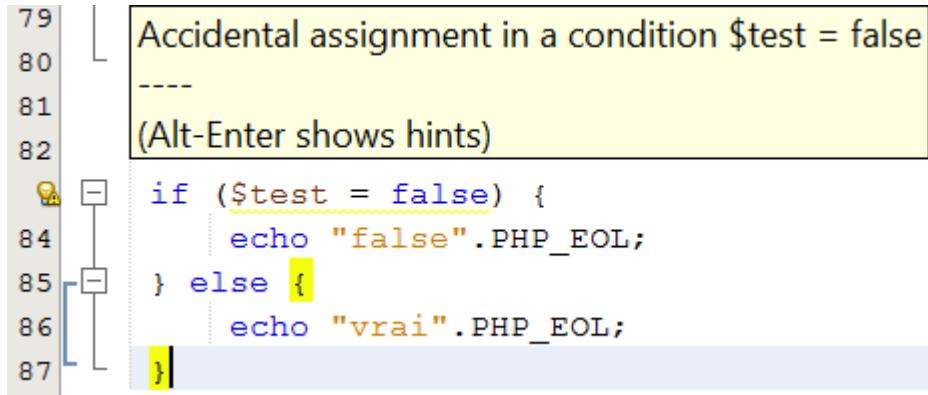
```
if ( !$test) {
    echo "false".PHP_EOL;
} else {
    echo "vrai".PHP_EOL;
}
```

Vous avez sans doute remarqué que nous avons effectué le test d'égalité avec un double égal (==). En effet, le simple égal (=) correspond à une instruction d'affectation. Si dans un moment d'inattention, vous écrivez ceci :

```
if ($test = false) {
    echo "false".PHP_EOL;
} else {
    echo "vrai".PHP_EOL;
}
```

... en réalité vous venez d'écrire : j'affecte la valeur « false » à la variable \$test, et comme cette affectation s'est bien passée (il n'y a d'ailleurs aucune raison pour qu'elle se passe mal), cette opération d'affectation renvoie implicitement la valeur « true » (donc « vrai ») : donc notre test est faussé. Je vous garantis que ce genre de bêtise arrive souvent, et même à des développeurs chevronnés.

Certains IDE fournissent une assistance qui permet de se prémunir contre ce genre d'erreur. Par exemple, Netbeans fait ça plutôt bien. Dans l'exemple ci-dessous, j'ai placé le pointeur de la souris sur le picto , qui se trouve au début de la ligne du « if », et un message est venu m'avertir du fait que j'avais probablement écrit une ânerie :



```

79
80
81
82
83
84
85
86
87

```

```

if ($test = false) {
    echo "false".PHP_EOL;
} else {
    echo "vrai".PHP_EOL;
}

```

Dans l'annexe 5.1, vous trouverez quelques approfondissements autour des opérateurs PHP. Je vous laisse le soin d'étudier ce chapitre à tête reposée.

Je vous propose maintenant d'étudier quelques fonctions PHP qui sont bien pratiques pour connaître et manipuler le type et le contenu des variables. Nous les avons listées quelques pages plus haut, mais je les rappelle ici : [gettype\(\)](#), [settype\(\)](#), [isset\(\)](#), [unset\(\)](#), [empty\(\)](#), plus un jeu de fonctions telles que [is\\_int\(\)](#), [is\\_array\(\)](#), [is\\_numeric\(\)](#), etc...

### **gettype()**

`gettype()` détermine le type d'une variable reçue en paramètre, et renvoie une chaîne qui peut prendre l'une des valeurs suivante : integer, double, string, array, object, boolean, resource, NULL et unknown.

```

$mavar = '123';
if(gettype($mavar) == "string") {
    echo ($mavar);
}

```

### **settype()**

`settype()` définit explicitement le type d'une variable (transmise en premier paramètre), par rapport à un type transmis sous forme de chaîne (en second paramètre). Les valeurs possibles sont : integer, double, string, array, object. Si le type est correctement défini, alors `SetType()` renvoie true, sinon la fonction retourne faux.

```
$mavar = '123';
if(settype($mavar, "integer")) {
    echo("Cette variable est bien de type entier");
}
```

### isset()

isset () est utilisée pour déterminer si une variable est définie (renvoie "true") ou pas (renvoie "false").

```
$mavar = '123';
if( isset($mavar) ) {
    echo("Cette variable a bien une valeur définie");
}
```

On peut aussi se servir de la fonction isset() pour tester la présence d'un poste dans un tableau, mais nous en reparlerons dans le chapitre dédié aux tableaux.

### unset()

unset () est utilisée pour détruire une variable transmise en paramètre (et ça fonctionne aussi pour supprimer un poste de tableau).

```
unset($mavar);
if(isset($mavar)) {
    echo("Ce texte ne s'imprimera jamais !");
}
```

### empty()

empty () est l'exact opposé de la fonction isset (). Elle retourne « vrai » si la variable n'est pas définie, et « faux » si elle est définie. Dans le cas de la fonction « isset() » de l'exemple précédent, on pourrait obtenir strictement le même résultat en écrivant ceci :

```
unset($mavar);
if( !empty($mavar) ) {
    echo("Ce texte ne s'imprimera jamais !");
}
```

**is\_[datatype] ()**

Les fonctions `is_int()`, `is_integer()`, et `is_long()` vérifient si la variable transmise en paramètre est de type "entier".

Le type de données "décimal" dispose également de trois fonctions : `is_double()`, `is_float()`, et `is_real()`. La fonction `is_string()` indique si la variable est une chaîne.

Les fonctions `is_array()` et `is_object()` fonctionnent sur le même principe, avec leurs types de données respectifs.

```
$somevar = "ceci est une chaîne";
if(is_string($somevar)) {
    echo("Cette variable est une chaîne");
}
```

Pour vous aider à y voir clair :

- Si vous souhaitez vous assurer qu'une variable est bien un nombre, utilisez la fonction `is_numeric()` pour vérifier sa valeur.
- Vous pouvez forcer une variable à « adopter » le type « nombre entier » en utilisant la fonction `intval()`, ou en plaçant `(int)` devant la variable (cf. exercice ci-après).
- Si la valeur doit être un tableau, utilisez `is_array()`.
- Pour tester si une variable est une chaîne, utilisez `is_string()`. Pour la forcer à devenir une chaîne, utiliser `strval()`.
- Si la valeur devrait être nulle, utiliser `is_null()`.
- Si la valeur doit être définie, utilisez `isset()`.

Je vous propose un petit exercice : lisez le code ci-dessous, essayez de déterminer lesquels des messages 1, 2, 3 et 4 vont s'afficher, puis testez le :

```
$var1 = '123a';
$var2 = 123;
if( $var1 == $var2) {
    echo("Ces variables contiennent la même valeur (1)".PHP_EOL);
}
if( $var1 === $var2) {
    echo("Ces variables contiennent la même valeur (2)".PHP_EOL);
}
if( (int)$var1 == (int)$var2) {
    echo("Ces variables contiennent la même valeur (3)".PHP_EOL);
}
if( (int)$var1 === (int)$var2) {
    echo("Ces variables contiennent la même valeur (4)".PHP_EOL);
}
```

Si vous avez du mal à comprendre ce qui se passe, testez le code suivant :

```
echo (int)$var1;
```

Dans ce chapitre d'introduction aux variables, nous avons présenté quelques types de données très simples. Dans un prochain chapitre, nous étudierons certains types particuliers comme les tableaux. Mais avant cela, je vous propose de faire d'abord un petit détour par les constantes.

## 4.2.2 Constantes

Les constantes fonctionnent elles-aussi comme les tiroirs de la même grande armoire (relative à la mémoire allouée à PHP par le système). Mais des tiroirs dont le contenu, une fois défini, n'est pas modifiable.

La déclaration d'une constante en PHP se fait au moyen de la fonction define() :

```
define('MA_DATA1', 23);
define("MA_DATA2", 'hello!!!!');
echo MA_DATA1;
echo MA_DATA2;
```

On voit que les constantes n'utilisent pas le caractère dollar.

Par convention on déclare les constantes avec un nom en majuscule. Cela permet de les repérer plus facilement à l'intérieur du code.

Pour approfondir la notion de constante PHP :

<http://php.net/manual/fr/function.define.php>

En plus des constantes personnalisées, PHP fournit un nombre assez important de constantes standard. Vous connaissez déjà PHP\_EOL, vous aurez peut être besoin de temps à autre d'utiliser PHP\_VERSION, pour connaître la version courante de PHP et ainsi savoir si vous pouvez utiliser une instruction PHP standard, ou si vous devez faire appel à une solution de substitution (si la fonction qui vous intéresse n'existe pas dans certaines versions de PHP).

Je vous invite à jeter un coup d'œil à la liste des constantes prédéfinies (on dit aussi « constantes réservées ») présentée sur cette page :

<http://php.net/manual/fr/reserved.constants.php>

## 4.2.3 Tableaux

Un tableau c'est une liste ou collection de données. Si l'on reprend l'idée de « grande armoire », que nous avions évoquée lors de notre introduction aux variables, un tableau ce serait un classeur à l'intérieur d'un tiroir de notre grande armoire. Ce classeur peut contenir des données très structurées (exemple : une collection de timbre), ou au contraire pas structurées du tout (exemple : toutes les bricoles qu'un enfant ramène dans ses poches à la fin d'une journée de promenade, comme des cailloux, des trombones, des craies, des bonbons, etc...).

Pour initialiser un tableau en PHP, on peut utiliser l'une des 2 syntaxes suivantes (elles sont strictement équivalentes) :

```
$tableau = [];
$tableau = array();
```

En PHP, il existe deux grands types de tableaux :

- Les tableaux simples
- Les tableaux associatifs

Commençons par les tableaux simples.

Voici un exemple de tableau contenant une liste d'APIs spécifiques à la norme HTML5 :

```
$apis_HTML5 = array('Canvas', 'SVG', 'Touch', 'WebAudio', 'WebMIDI',
    'LocalStorage', 'SessionStorage', 'WebWorker', 'WebSocket', 'File');
```

On aurait pu obtenir strictement le même résultat en écrivant ceci :

```
$apis_HTML5 = array();
$apis_HTML5[] = "Canvas";
$apis_HTML5[] = "SVG";
$apis_HTML5[] = "Touch";
$apis_HTML5[] = "WebAudio";
$apis_HTML5[] = "WebMIDI";
$apis_HTML5[] = "LocalStorage";
$apis_HTML5[] = "SessionStorage";
$apis_HTML5[] = "WebWorker";
$apis_HTML5[] = "WebSocket";
$apis_HTML5[] = "File";
```

Le fait d'utiliser les crochets vides [], juste après le nom de la variable indique à PHP que l'on souhaite ajouter un nouvel élément à la fin du tableau. C'est l'équivalent de la méthode push() en Javascript.

On aurait pu également numérotter explicitement chaque poste du tableau, mais je ne recommande pas cette méthode :

```
$apis_HTML5 = [];
$apis_HTML5[0] = "Canvas";
$apis_HTML5[1] = "SVG";
$apis_HTML5[2] = "Touch";
$apis_HTML5[3] = "WebAudio";
$apis_HTML5[4] = "WebMIDI";
$apis_HTML5[5] = "LocalStorage";
$apis_HTML5[6] = "SessionStorage";
$apis_HTML5[7] = "WebWorker";
$apis_HTML5[8] = "WebSocket";
$apis_HTML5[9] = "File";
```

PHP fournit un impressionnant jeu de fonctions pour manipuler les tableaux, vous pouvez consulter la liste détaillée sur cette page :

<http://php.net/manual/fr/ref.array.php>

Supposons que nous souhaitons trouver quel poste de notre tableau contient l'API SVG, nous pouvons utiliser la fonction array\_search(), comme ceci :

```
$numero = array_search('SVG', $apis_HTML5);
echo $numero; // renvoie 1 car le tableau est numéroté à partir de 0
```

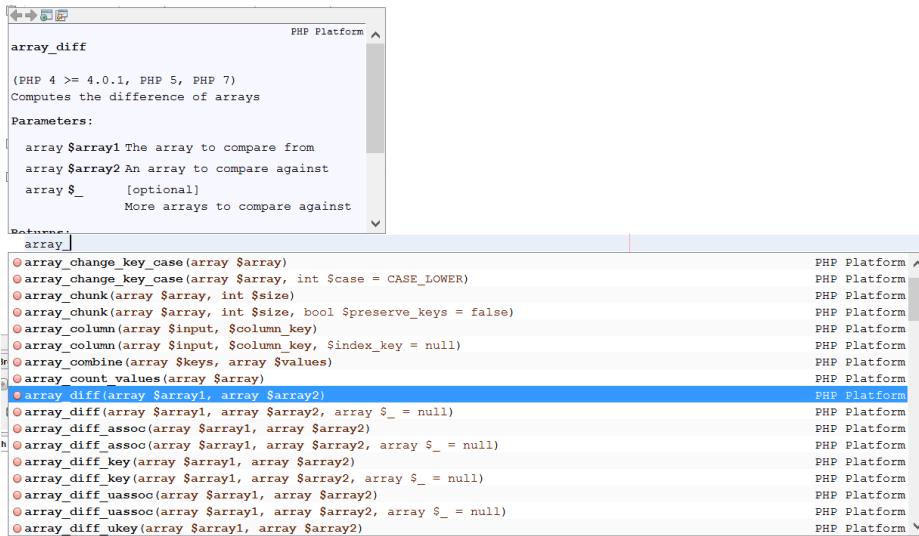
Quelques fonctions PHP intéressantes à connaître :

- count() : renvoie le nombre de postes d'un tableau
- implode() : renvoie une chaîne contenant les postes du tableau collés bout à bout, séparés par un caractère indiqué en premier paramètre de la fonction
- explode() : transforme une chaîne en tableau, en fonction d'un caractère séparateur... implode() et explode() sont vraiment très pratiques
- sort() : tri le contenu du tableau

Attention : en PHP, il y a un manque de cohérence criant en ce qui concerne les fonctions. Beaucoup de fonctions de manipulation de tableaux ont des noms qui commencent par « array\_ », ce qui n'est pas le cas des fonctions indiquées ci-dessus, pourtant elles aussi spécialisées dans la manipulation de tableaux.

De plus, certaines fonctions reçoivent le nom du tableau en premier, second, voire troisième paramètre (peu de cohérence de ce côté-là également), et renvoient un résultat en sortie. C'est le mode de fonctionnement notamment des fonctions count() et implode(), ... tandis que d'autres fonctions ne renvoient rien en sortie et modifient le contenu du tableau qu'elle ont reçu en paramètre, c'est le cas de la fonction sort().

Pour s'y retrouver dans ce joyeux bazar, heureusement que la documentation officielle sur php.net est bien faite, et que les IDE fournissent autocomplétion et aide en ligne, comme par exemple Netbeans, quand je commence à saisir « array\_ » :



Parcourir la globalité d'un tableau, ce n'est pas compliqué. Pour cela vous disposez des boucles « for », « foreach » et « while » :

```
for($i=0, $ilen=count($apis_HTML5); $i<$ilen; $i++) {
    echo $i . ' => ' . $apis_HTML5[$i]. PHP_EOL;
}
```

```
foreach($apis_HTML5 as $apikey=>$apival) {
    echo $apikey . ' => ' . $apival. PHP_EOL;
}
```

```
$i = 0;
$ilen = count($apis_HTML5);
while ($i<$ilen) {
    echo $i . ' => ' . $apis_HTML5[$i]. PHP_EOL;
    $i++;
}
```

Les 3 boucles ci-dessus produisent le même résultat :

```
0 => Canvas
1 => SVG
2 => Touch
3 => WebAudio
4 => WebMIDI
```

```

5 => LocalStorage
6 => SessionStorage
7 => WebWorker
8 => WebSocket
9 => File

```

Dans le cas de la boucle « for », deux points à noter :

Il y a 3 grandes parties dans les paramètres attendus par la boucle « for », parties qui sont séparées par des points virgules :

- la partie « initialisation » qui permet d'initialiser 1 ou une plusieurs variables (dans notre exemple de la page précédente, nous initialisons simultanément les variables \$i et \$ilen)
- la partie « analyse de la condition de poursuite de la boucle. Dans notre exemple, nous vérifions si \$i est bien inférieure à \$ilen. Si c'est le cas, la boucle se poursuit, dans le cas contraire la boucle s'arrête
- la partie « incrémentation », dans laquelle on incrémente le compteur en prévision de l'itération suivante (si on l'oubliait, dans notre exemple nous partirions dans une boucle sans fin). Dans cette partie, on peut gérer le « pas » d'incrémentation, et par exemple considérer qu'on souhaite traiter un élément sur 2 ou 3.

Petite précision, écrire ceci : \$i++

... équivaut à écrire ceci : \$i = \$i + 1

... ou encore ceci : \$i+=1

On notera une différence cruciale entre les boucles « for », « foreach » et « while » :

- à l'intérieur des boucles « for » et « while », il faut impérativement utiliser l'index pour adresser un élément du tableau  
`echo $i . ' => ' . $apis_HTML5[$i]. PHP_EOL;`
- à l'intérieur d'une boucle « foreach », on utilise pas directement le nom du tableau, mais le nom de la variable déclarée à droite du mot clé « as », soit ici \$apival.
- Toujours dans la boucle « foreach », vous noterez qu'on n'a pas besoin de l'index (disponible dans la variable \$apikey), pour récupérer le nom d'une API. On n'a d'ailleurs pas besoin de l'incrémenter, c'est PHP qui le fait pour nous, en récupérant l'index en cours et en le stockant dans la variable \$apikey. A noter que, si vous ne souhaitez pas

afficher ou utiliser l'index, vous pouvez simplifier l'écriture du « foreach » en écrivant ceci :

```
foreach($apis_HTML5 as $apival) {  
    echo $apival. PHP_EOL;  
}
```

Avant de passer à l'étude des tableaux associatifs, je rappelle que vous disposez des fonctions :

- `is_array()` pour vérifier si une variable est bien de type tableau.
- `count()` pour compter le nombre d'éléments d'un tableau

Ces deux fonctions sont opérationnelles aussi bien sur les tableaux simples que les tableaux associatifs.

Un tableau associatif, ce n'est pas très compliqué non plus. Voici un exemple de tableau associatif contenant un menu pas franchement équilibré :

```
$menu = array (  
    "entrée" => "terrine du chef",  
    "plat" => "entrecôte au bleu (accompagné de frites)",  
    "dessert" => "mousse au chocolat"  
) ;  
foreach($menu as $menukey=>$menuval) {  
    echo $menukey . ' : ' . $menuval. PHP_EOL;  
}
```

Résultat obtenu avec la boucle « foreach » :

```
entrée : terrine du chef  
plat : entrecôte au bleu (accompagné de frites)  
dessert : mousse au chocolat
```

Vous voyez ce n'est pas très compliqué. Il faut juste noter que les boucles « for » et « while » ne sont pas utilisables avec un tableau associatif. Seule la boucle « foreach » est véritablement adaptée à ce type de tableau.

Maintenant, supposons que nous souhaitions gérer un tableau contenant les menus de la semaine. Nous pouvons dans ce cas créer un tableau associatif de tableaux associatifs (on emploie souvent le terme de « tableaux imbriqués » pour désigner ce cas de figure).

Voici un exemple de tableau contenant des menus allant du lundi au mercredi. Chaque poste du tableau principal \$menu\_sem contient lui-même 3 postes de tableau correspondant aux 3 plats principaux d'un repas :

```
$menu_sem = [];
$menu_sem['lundi']= array (
    "entrée" => "terrine du chef",
    "plat" => "entrecôte au bleu accompagné de frites",
    "dessert" => "mousse au chocolat"
);
$menu_sem['mardi']= array (
    "entrée" => "macédoine de légumes",
    "plat" => "hamburger (accompagné de frites)",
    "dessert" => "crème caramel"
);
$menu_sem['mercredi']= array (
    "entrée" => "avocat au surimi",
    "plat" => "cordon bleu accompagné de frites",
    "dessert" => "mousse au chocolat"
);
```

Voici un exemple de script PHP avec deux boucles « foreach » imbriquées permettant de générer une liste HTML :

```
echo '<ol>' . PHP_EOL;
foreach ($menu_sem as $menu_jour_key=>$menu_jour_val) {
    echo '<li>' . $menu_jour_key . PHP_EOL;
    echo '<ul>' . PHP_EOL;
    foreach($menu_jour_val as $menu_detail_key=>$menu_detail_val) {
        echo '<li>' . $menu_detail_key . ' : ' . $menu_detail_val . '</li>' . PHP_EOL;
    }
    echo '</ul>' . PHP_EOL;
    echo '</li>' . PHP_EOL;
}
echo '</ol>' . PHP_EOL;
```

Exercice : Etudiez le code ci-dessus. Essayez de bien comprendre le fonctionnement et le rôle des 2 boucles « foreach ». Essayez de vous représenter le code HTML qui va être généré et le résultat que l'on va obtenir visuellement. Ensuite seulement, vous pouvez consulter le résultat sur la page qui suit.

Résultat :

1. lundi

- entrée : terrine du chef
- plat : entrecôte au bleu accompagné de frites
- dessert : mousse au chocolat

2. mardi

- entrée : macédoine de légumes
- plat : hamburger (accompagné de frites)
- dessert : crème caramel

3. mercredi

- entrée : avocat au surimi
- plat : cordon bleu accompagné de frites
- dessert : mousse au chocolat

Jetons un coup d'œil au code HTML généré, cela fera un bon rappel par rapport au cours « HTML5 – Premiers pas ».

```
<ol>
<li>lundi
<ul>
<li>entrée : terrine du chef</li>
<li>plat : entrecôte au bleu accompagné de frites</li>
<li>dessert : mousse au chocolat</li>
</ul>
</li>
<li>mardi
<ul>
<li>entrée : macédoine de légumes</li>
<li>plat : hamburger (accompagné de frites)</li>
<li>dessert : crème caramel</li>
</ul>
</li>
<li>mercredi
<ul>
<li>entrée : avocat au surimi</li>
<li>plat : cordon bleu accompagné de frites</li>
<li>dessert : mousse au chocolat</li>
</ul>
</li>
</ol>
```

## 4.2.4 Fonctions, Commentaires

Les fonctions PHP sont assez similaires aux fonctions que l'on trouve sur la plupart des autres langages.

La création d'une fonction se fait de la façon suivante :

```
function multiplication($var1, $var2) {
    $resultat = $var1 * $var2 ;
    return $resultat ;
}
```

Explication : notre fonction `multiplication()` reçoit 2 paramètres qu'elle multiplie ensemble. Le résultat de cette multiplication est stocké dans une variable qui est ensuite envoyé en sortie de la fonction. Le stockage dans une variable n'étant pas ici indispensable, nous aurions pu simplifier le code comme ceci :

```
function multiplication($var1, $var2) {
    return $var1 * $var2 ;
}
```

PHP offre la possibilité de définir des paramètres optionnels, on les reconnaît facilement au fait qu'ils ont une valeur par défaut, comme ici pour les variables `$var3`, `$var4` et `$var5` :

```
function test($var1, $var2, $var3='', $var4=array() , $var5=0) {
    return ... ;
}
```

Si l'on souhaite qu'une fonction puisse renvoyer plusieurs valeurs en sortie, il est possible de le faire très facilement en encapsulant les différentes valeurs dans un tableau simple ou associatif. Exemple ci-dessous avec un tableau associatif :

```
function test2($var1, $var2, $var3='', $var4=array()) {
    return ['out1'=>10, 'out2'=>$var1.$var2, 'out3'=>'titi'];
}
```

Un autre élément important à noter, c'est la portée des variables utilisées à l'intérieur des fonctions :

- Toute variable déclarée à l'intérieur d'une fonction a une portée locale, c'est-à-dire qu'elle n'est pas visible depuis l'extérieur de la fonction
- Pour pouvoir utiliser à l'intérieur d'une fonction des variables globales, c'est-à-dire des variables qui ont été déclarées en dehors de toute fonction, et en amont de l'appel de la fonction considérée. Pour utiliser une variable globale, il faut l'indiquer explicitement au moyen de l'instruction "global", comme dans l'exemple suivant :

```
function test3($var1, $var2, $var3='', $var4=array()) {  
    global $glo1, $glo2 ;  
    ...  
}
```

**AVERTISSEMENT** : je ne suis pas fan des variables globales, et je sais que je ne suis pas le seul. Pour garantir une bonne robustesse du code, je préfère que mes fonctions n'utilisent que des variables reçues en paramètres, et des variables locales. En règle générale, j'exclus toutes utilisation de variables globales au sein de fonctions, sauf cas très particulier que je prends soin de documenter dans le code, afin que les collègues qui reprendront la maintenance du code sachent précisément pourquoi j'ai fait ce choix. Tiens, puisque nous parlons de commentaires, profitons-en pour regarder comment nous pouvons insérer des commentaires dans du code PHP.

```
// ceci est un commentaire mono-ligne occupant une ligne entière  
  
$var1 = 123 ; // commentaire mono-ligne placé en bout de ligne  
  
/* ceci est aussi un commentaire ... */  
  
/* ... mais cette technique est généralement utilisée pour  
commenter plusieurs lignes.  
on s'en sert aussi pour mettre en commentaire  
provisoirement certaines portions de code */
```

En introduction de fonction, il est recommandé de placer un bloc de commentaire définissant le rôle de la fonction, ainsi que ses paramètres entrants et sortants. Les principaux IDE proposent une assistance pour la construction de ce type de commentaire.

Si l'on prend pour exemple la fonction suivante :

```
function test5($var1, $var2, $var3='', $var4=array(), $var5=0) {
    // placer le code "métier" ici
    return ['out1'=>$var1, 'out2'=>$var2.$var3];
}
```

... sur la ligne précédant le mot clé « function », avec Netbeans ou PHPStorm, saisissez `/**` puis pressez la touche Entrée :

```
/**
 * @param $var1
 * @param $var2
 * @param string $var3
 * @param array $var4
 * @param int $var5
 * @return array
 */
```

... vous voyez apparaître un bloc de commentaire « pré-mâché » contenant la liste des paramètres que l'IDE a été en mesure de déterminer via la ligne de déclaration de la fonction, ainsi que via le mot clé « return ». On peut dès lors compléter le bloc de commentaire en ajoutant un texte explicatif et en précisant le type des paramètres que l'IDE n'a pas été en mesure de déterminer de lui-même :

```
/**
 * Ceci est un bloc de commentaire placé avant la déclaration
 * d'une fonction pour préciser son utilité, ainsi que le
 * nombre et le type des paramètres entrants (avec @param)
 * et sortants (avec @return)
 * @param integer $var1
 * @param string $var2
 * @param string $var3
 * @param array $var4
 * @param int $var5
 * @return array
 */
function test5($var1, $var2, $var3='', $var4=array(), $var5=0) {
    // placer le code "métier" ici
    return ['out1'=>$var1, 'out2'=>$var2.$var3];
}
```

Il peut être intéressant dans certains cas de recourir aux fonctions suivantes :

- [func\\_num\\_args\(\)](#) : renvoie le nombre de paramètres d'une fonction
- [func\\_get\\_args\(\)](#) : renvoie un tableau contenant les paramètres d'une fonction

Je ne rentre pas dans le détail, je vous laisse le soin de consulter la documentation officielle.

Il existe un certain nombre de fonctions PHP dédiées à la gestion de fonctions :

<http://fr.php.net/manual/fr/ref.funchand.php>

#### 4.2.5 Include et Require

Les fonctions PHP `include()` et `require()` sont très utiles dans le processus de développement. Elles permettent d'inclure dans un script en cours d'exécution des déclarations de fonctions ou d'objets qui ont été créés dans d'autres scripts PHP.

En cas d'absence du fichier indiqué, « `include` » génère un warning alors que « `require` » génère une erreur fatale.

En règle générale, on préfère utiliser « `require` », car l'erreur fatale déclenchée par cette instruction nous permettra d'identifier plus rapidement une situation anormale. Or, l'absence du fichier spécifié est généralement une situation anormale (sauf cas particulier qu'il conviendra de bien documenter dans le code).

Exemple :

```
// remplace la ligne "require" par le contenu du fichier transmis en paramètre
require ("copyright.php");

// identique à "require", mais l'instruction est ignorée si le fichier considéré
// a déjà été chargé par un autre script
require_once ("header.php");

// identique à "require" mais déclenche seulement un warning en cas d'absence du
// fichier indiqué
include ("disclaimer.php");

// identique à "include" mais ignore l'instruction si le fichier a déjà été chargé
// par un autre script
include_once ("title.php");
```

On notera qu'il est possible de « remonter » dans l'arborescence d'un site pour aller chercher un script qui se trouve dans une autre branche de l'arborescence du projet. Dans l'exemple qui suit, on remonte de 2 répertoires par rapport répertoire courant, pour aller chercher le fichier « copyright.php » qui se trouve dans le sous-répertoire « lib » :

```
require("../lib/copyright.php");
```

TODO : compléter ce chapitre en développant les notion de chemin d'accès et de require « automatique ».

## 4.4 Les formulaires et les tableaux \$\_GET et \$\_POST

Au travers des chapitres précédents, nous avons étudié quelques aspects essentiels de la syntaxe du PHP. Nous n'avons pas tout vu, et en particulier nous n'avons pas parlé des classes et des objets. Mais je préfère laisser ce sujet de côté, nous en parlerons plus tard.

Car maintenant que nous connaissons les tableaux PHP, nous pouvons nous intéresser de nouveau au navigateur, et étudier de quelle manière nous pouvons faire dialoguer un navigateur (le client) avec un stack PHP (le serveur). Pour établir ce dialogue, nous allons nous intéresser :

- à la balise « a » du HTML,
- aux formulaires HTML
- aux tableaux PHP \$\_GET et \$\_POST : ces 2 tableaux sont fournis en standard par l'interpréteur PHP pour récupérer les données transmises par les requêtes HTTP de type GET et POST.

Si vous ne sentez pas à l'aise avec les tableaux HTML (ou avec la balise « a »), je vous conseille de vous reporter au support de cours « HTML5 – Premiers pas ».

### 4.4.1 Le tableau \$\_GET

Pour introduire le sujet, je vous propose de créer un script PHP que vous appellerez « pagelink.php » :

```
<?php
$titre = 'Mon premier échange client-serveur';
$encodage = 'UTF-8';
$message1 = "Page de test pour un échange client-serveur via la balise HTML \"a\"";
$mon_html = <<<BLOC_HTML
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset=\"$encodage\">
        <title>$titre</title>
    </head>
    <body>
        <div class="annonce">
            {$message1}
            <br><br>
            <a href="pagecible.php">Appel de pagecible sans paramètres</a><br>
            <a href="pagecible.php?param1=titi">Appel de pagecible avec 1 paramètre</a><br>
            <a href="pagecible.php?param1=titi&param2=gros-minet">Appel de pagecible avec 2
paramètres</a><br>
        </div>
    </body>
</html>
BLOC_HTML;
echo $mon_html;
```

Créez ensuite, dans le même répertoire, le script PHP « pagetable.php », dont voici le code source :

```
<?php

/* La balise "pre" permet de forcer un affichage en police non proportionnelle
   (très pratique pour afficher le contenu d'un tableau pendant une phase de
   mise au point) */
echo '<pre>';
var_dump($_GET); // return un "dump" de la variable $_GET
echo '</pre>';

// Si $_GET contient au moins 1 paramètre...
if (count($_GET)>0) {
    echo '$_GET contient '.count($_GET). ' paramètres<br>';
} else {
    echo '$_GET ne contient aucun paramètre<br>';
}

// 1ère technique permettant de détecter la présence d'un paramètre dans $_GET
if (isset($_GET['param1'])) {
    echo '$_GET contient le paramètre "param1" qui lui-même contient la valeur : ' .
$_GET['param1']. '<br>';
}

// 2ème technique permettant de détecter la présence d'un paramètre dans $_GET
if (array_key_exists('param2', $_GET)) {
    echo '$_GET contient le paramètre "param2" qui lui-même contient la valeur : ' .
$_GET['param2']. '<br>';
}
```

J'ai inséré des commentaires dans ce code pour vous aider à le comprendre. Si malgré tout, le fonctionnement de certaines instructions PHP vous met en difficulté, je vous encourage à parcourir la documentation du site php.net, dans laquelle vous trouverez le descriptif de chaque fonction, accompagné d'exemples.

Maintenant, à partir de votre navigateur, ou du raccourci proposé par votre IDE, lancez l'exécution de la page « pagetable.php ». L'URL de lancement ressemblera peut être à ceci, si vous avez créé – comme moi – un répertoire « coursphp » dans le répertoire « htdocs » de votre stack PHP, et si vous avez placé votre script PHP dans le sous-répertoire « testlink » :

<http://localhost/coursphp/testlink/pagetable.php>

Voici à quoi ressemble la page dans le navigateur :

Page de test pour un échange client-serveur via la balise HTML "a"

[Appel de pagetable sans paramètres](#)  
[Appel de pagetable avec 1 paramètre](#)  
[Appel de pagetable avec 2 paramètres](#)

Cliquez sur le premier lien, vous devriez voir apparaître ceci :

```
array(0) {  
}  
  
$_GET ne contient aucun paramètre
```

Cliquez sur le bouton de « retour arrière » de votre navigateur, et cliquez sur le second lien :

```
array(1) {  
    ["param1"]=>  
    string(4) "titi"  
}  
  
$_GET contient 1 paramètres  
$_GET contient le paramètre "param1" qui lui-même contient la valeur : titi
```

Retour arrière, puis clic sur le troisième et dernier lien :

```
array(2) {  
    ["param1"]=>  
    string(4) "titi"  
    ["param2"]=>  
    string(10) "gros-minet"  
}  
  
$_GET contient 2 paramètres  
$_GET contient le paramètre "param1" qui lui-même contient la valeur : titi  
$_GET contient le paramètre "param2" qui lui-même contient la valeur : gros-minet
```

Pour rappel, le dernier lien dans la page a la structure suivante :

[Appel de pageable avec 2 paramètres](pageable.php?param1=titi&param2=gros-minet)<br>

Vous voyez que dans l'attribut « href » de la balise « a », nous avons 2 paramètres placés derrière le point d'interrogation, et séparés par le symbole « & » :

?param1=titi&param2=gros-minet

Ce sont bien ces mêmes paramètres que nous retrouvons dans le tableau PHP \$\_GET, sous la forme de 2 postes distincts. Et vous voyez que \$\_GET est un tableau associatif, au même titre que ceux que nous avons créé dans le chapitre d'introduction dédié à ce sujet.

Cela fait un moment que nous ne nous sommes pas intéressés aux coulisses du navigateur. Jetons un coup d'œil à ce qui se passe dans les outils de développement, et en particulier dans l'onglet « réseau » (ou « network »).

Je vais prendre pour exemple l'onglet « network » de Chrome. Voici ce que j'obtiens après avoir cliqué sur le 3<sup>ème</sup> et dernier lien de ma page « pagelink.php » :

Name	Status	Type	Initiator	Size	Time	Waterfall	
pagecible.php?param1=titi&p...	200	document	Other	553 B	4 ms		100.00 ms

http://localhost/greta91\_2017/testlink/pagecible.php?param1=titi&param2=gros-minet

J'ai reçu un code retour 200, ce qui signifie que tout s'est bien passé.

Si je clique sur la seule ligne de l'affichage, j'obtiens des informations détaillées sur la requête HTTP qui a été envoyée au serveur, et en particulier sur les paramètres de la requête, paramètres que l'on retrouve présentés en détail tout en bas de la vue « headers ».

The screenshot shows the Network tab in the Chrome DevTools. A single request for "pagecible.php?param1=titi..." is listed with a status of 200. The Headers section shows the following:

```

Cookie: Phpstorm-4488da6a=e4a32db3-ce2b-4103-aa7e-370464f8b706; __utma=111872281.343457310.1498570338.1498570338.1498570338.1; __utmz=111872281.1.1498570338.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)
Host: localhost
Referer: http://localhost/greta91_2017/testlink/pagelink.php
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36

```

The Query String Parameters section shows:

- param1: titi
- param2: gros-minet

Si vous cliquez sur l'onglet « Response », vous allez voir apparaître le code HTML généré par le script PHP. On y retrouve la balise « pre », le résultat produit par la fonction PHP var\_dump(), et les messages complémentaires (tronqués dans la copie d'écran) générés par les tests utilisant les fonctions PHP isset() et array\_key\_exists() :

The screenshot shows the Response tab in the Chrome DevTools. The content is as follows:

```

<pre>array(2) {
  ["param1"]=>
  string(4) "titi"
  ["param2"]=>
  string(10) "gros-minet"
}
</pre>$_GET contient 2 paramètres<br>$_GET contient le paramètre "para

```

Vous vous souvenez sans doute que, côté serveur, nous avons la possibilité de consulter le fichier « access.log » d'Apache. Voici un échantillon de ce que vous pouvez y trouver :

```
::1 - - [14/Jul/2017:17:30:12 +0200] "GET
/coursphp/testlink/pagecible.php?param1=titi&param2=gros-minet HTTP/1.1" 200 299
"http://localhost/coursphp/testlink/pagelink.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"

::1 - - [14/Jul/2017:17:34:34 +0200] "GET
/coursphp/testlink/pagecible.php?param1=titi&param2=gros-minet HTTP/1.1" 200 299
"http://localhost/coursphp/testlink/pagelink.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"

::1 - - [14/Jul/2017:17:34:43 +0200] "GET
/coursphp/testlink/pagecible.php?param1=titi&param2=gros-minet HTTP/1.1" 200 299
"http://localhost/coursphp/testlink/pagelink.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
```

Nous venons d'étudier les mécanismes en jeu lors d'un échange client-serveur utilisant la méthode GET du protocole HTTP. Je rappelle en effet que la balise HTML « a » génère d'office – quand on clique dessus – une requête HTTP de type GET.

Nous allons retrouver le même principe avec les requêtes HTTP de type POST, et nous allons l'illustrer avec un exemple de formulaire.

#### 4.4.2 Le tableau \$\_POST

Pour partir à la découverte du tableau \$\_POST, je vous propose de créer le formulaire suivant dans un fichier « pageform.php » :

Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

Destination

Aéroport:

Heure d'embarquement:

Slider avec l'élément input de type range

Choisissez un nombre:

Voici la valeur choisie: 6

**Valider**

Pour créer ce formulaire, je me suis amusé à utiliser quelques-uns des nouveaux types de balises « input » du HTML5 (les types « search », « number » etc..) , ainsi que la balise « select » (en 3<sup>ème</sup> position) qui elle, est plus ancienne.

Dans cet exemple, nous avons besoin d'un fichier CSS et de 2 scripts PHP. Je vous propose de les regrouper dans un même sous-répertoire « testform », à l'intérieur de votre projet.

Commençons par créer un fichier « pageform.css », dont voici le code source :

```
body {
    padding-top: 60px;
    padding-bottom: 40px;
    margin-left: 10px;
}
form {
    margin-left: 10px;
}
input, select {
    border-radius: 5px;
    border-color: chocolate;
}
input[type="search"] {
    border-radius: 10px;
}
input:required {
    border-color: purple ;
}
input[data-evt] {
    border-color: silver;
}
```

Dans le même répertoire que le fichier CSS précédent, créons un script PHP « pageform.php », dont voici le code source :

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title>Exemple de formulaire</title>
    <link rel="stylesheet" href="pageform.css">
</head>
<body>
<form action="pagecible.php" method="post">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                   placeholder="saisissez un critère de recherche">
        </p>
        <p>
            <label for="number_field">Exemple de champ "number"</label>
            <input type="number" name="number" id="number_field" required >
        </p>
        <p>
            <label for="select_field">Exemple de champ "select"</label>
            <select name="myselect" id="select_field">
                <option value="A">valeur A</option>
                <option value="B">valeur B</option>
                <option value="C">valeur C</option>
            </select>
        </p>
    </fieldset>
    <fieldset>
        <legend>Destination</legend>
        <p>
```

```

        <label for="airport_code">Aéroport:</label>
        <input type="text" name="airport" id="airport_code"
               list="airports" data-evt="airport" />
    </p>
    <p>
        <label>Heure d'embarquement:</label>
        <input type="datetime-local" name="totime" step=3600
               data-evt="datetime" />
    </p>
</fieldset>
<datalist id="airports">
    <option value="ATL" label="Atlanta">
    <option value="MEM" label="Memphis">
    <option value="LHR" label="London Heathrow">
    <option value="LAX" label="Los Angeles">
    <option value="FRA" label="Frankfurt">
    <option value="PAR" label="Paris">
</datalist>
<fieldset>
    <legend>Slider avec l'élément input de type range</legend>
    <label for="yournumber">Choisissez un nombre: </label>
    <input id="yournumber" name="yournumber" type="range"
           min="1" max="10" step="1" value="6"
           onchange="myOutput.value=this.value;" />
    <br>
    <label>Voici la valeur choisie: </label>
    <output name="myOutput"> 6 </output>
</fieldset>
<input type="submit" name="valid" value="Valider" />
</form>
</body>
</html>

```

Vous aurez sans doute remarqué que dans ce premier script PHP, nous n'utilisons pas les balises PHP. Nous n'en avons pas vraiment besoin dans cette première version, car pour l'instant notre formulaire ne contient pas de logique particulière. Nous aurions d'ailleurs pu créer un fichier « pageform.html », mais pour le coup cela nous aurait privé de la possibilité d'insérer du code PHP, et nous aurions dû tôt ou tard modifier l'extension du fichier en « .php ».

Si vous souhaitez vérifier que l'interpréteur PHP est bien actif malgré tout, vous pouvez insérer les balises PHP et un « echo » envoyant un message quelconque, quelque part entre les balises « body » et « /body ».

Nous intégrerons de la logique PHP dans ce script, dans la version suivante, qui sera présentée dans le prochain chapitre.

Le point important dans ce script, c'est la balise « form » ainsi que ses attributs :

```
<form action="pagecible.php" method="post">
```

L'attribut « action » définit le script PHP qui sera appelé dès que l'utilisateur cliquera sur le bouton « submit » du formulaire.

L'attribut « method » définit le type de requête HTTP qui va être utilisé par le formulaire. Nous avons le choix entre les méthodes GET et POST. Il existe d'autres méthodes au sein du protocole HTTP, comme par exemple PUT et DELETE, mais elles ne sont pas disponibles sur les formulaires HTML.

La méthode GET, c'est strictement la même que la méthode GET utilisée par la balise « a ». Ce qui signifie que, d'un point de vue du PHP, les paramètres sont récupérés dans le tableau `$_GET`.

Voici le code source du script « pageable.php », qui se trouve dans le même répertoire que notre formulaire :

```
<?php

/* La balise "pre" permet de forcer un affichage en police non proportionnelle
   (très pratique pour afficher le contenu d'un tableau pendant une phase de
   mise au point *)
echo '<pre>';
var_dump($_POST); // return un "dump" de la variable $_POST
echo '</pre>';

// Si $_POST contient au moins 1 paramètre...
if (count($_POST)>0) {
    echo '$_POST contient '.count($_POST) . ' paramètres<br>';
} else {
    echo '$_POST ne contient aucun paramètre<br>';
}

// Première technique permettant de détecter la présence d'un paramètre dans $_POST
if (isset($_POST['airport'])) {
    echo '$_POST contient le paramètre "airport" qui lui-même contient la valeur : ' .
$_POST['airport']. '<br>';
}

// Seconde technique permettant de détecter la présence d'un paramètre dans $_POST
if (array_key_exists('totime', $_POST)) {
    echo '$_POST contient le paramètre "totime" qui lui-même contient la valeur : ' .
$_POST['totime']. '<br>';
}

echo '<br>Contenu de $_POST affiché avec une boucle foreach<br>';
echo '<ul>' .PHP_EOL;
foreach($_POST as $postkey=>$postval) {
    echo "<li>{$postkey} => {$postval}</li>" .PHP_EOL ;
}
echo '</ul>' .PHP_EOL;
```

Ce script est très proche de celui que j'avais utilisé dans le chapitre précédent pour présenter le tableau `$_GET`. La différence principale, c'est la boucle « `foreach` » que j'ai ajoutée à la fin du script. Cette boucle « parcourt » le tableau `$_POST` et affiche chaque poste du tableau dans une liste HTML de type « `ul` ».

Après avoir saisi quelques infos « bidon » dans mon formulaire, et cliqué sur le bouton « valider », voici ce que j'obtiens dans la page HTML générée par le script « pagecible.php ».

```
array(7) {  
    ["search"]=>  
        string(4) "test"  
    ["number"]=>  
        string(1) "3"  
    ["myselect"]=>  
        string(1) "B"  
    ["airport"]=>  
        string(3) "MEM"  
    ["totime"]=>  
        string(16) "2017-07-27T05:00"  
    ["yournumber"]=>  
        string(1) "4"  
    ["valid"]=>  
        string(7) "Valider"  
}
```

`$_POST` contient 7 paramètres

`$_POST` contient le paramètre "airport" qui lui-même contient la valeur : MEM

`$_POST` contient le paramètre "totime" qui lui-même contient la valeur : 2017-07-27T05:00

Contenu de `$_POST` affiché avec une boucle foreach

- search => test
- number => 3
- myselect => B
- airport => MEM
- totime => 2017-07-27T05:00
- yournumber => 4
- valid => Valider

Je vous encourage à étudier ce qui se passe du côté des outils de développement du navigateur.  
L'onglet « network » est riche en informations :

\$\_POST contient 7 paramètres  
\$\_POST contient le paramètre "airport" qui lui-même contient la valeur : MEM  
\$\_POST contient le paramètre "totime" qui lui-même contient la valeur : 2017-07-21T18:00

Contenu de \$\_POST affiché avec une boucle foreach

- search => test de recherche
- number => 3
- myselect => B
- airport => MEM
- totime => 2017-07-21T18:00
- yournumber => 8
- valid => Valider

Vous voyez que nous avons affaire à une requête de type POST et que son code statut est 200. Les autres informations en dessous sont des informations contenues dans la réponse HTTP produite par notre stack PHP. La partie qui m'intéresse le plus, c'est celle qui se situe tout en bas de cette fenêtre d'informations. Faites descendre l'ascenseur jusqu'en bas :

Cache-Control: max-age=0  
Connection: keep-alive  
Content-Length: 109  
Content-Type: application/x-www-form-urlencoded  
Cookie: Phpstorm-4488da6a=e4a32db3-ce2b-4103-aa7e-370464f8b706; \_\_utma=111872281.343457310.1498570338.1498570338.1; \_\_utmz=111872281.1498570338.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)  
Host: localhost  
Origin: http://localhost  
Referer: http://localhost/greta91\_2017/testform/pageform.php  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36

Form Data

search:	test de recherche
number:	3
myselect:	B
airport:	MEM
totime:	2017-07-21T18:00
yournumber:	8
valid:	Valider

Vous voyez que nous retrouvons les champs de notre formulaire, avec pour chacun d'eux le contenu qui a été envoyé dans la requête HTTP transmise par le navigateur au stack PHP.

```
::1 - - [16/Jul/2017:06:27:17 +0200] "POST /greta91_2017/testform/pagecible.php HTTP/1.1" 200  
774 "http://localhost/greta91_2017/testform/pageform.php" "Mozilla/5.0 (Linux; Android 6.0;  
Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile  
Safari/537.36"
```

Comparons cette requête HTTP de type POST à la requête de type GET que nous avions étudiée au chapitre précédent (que je mets ci-dessous pour rappel) :

```
::1 - - [14/Jul/2017:17:30:12 +0200] "GET  
/coursphp/testlink/pagecible.php?param1=titi&param2=gros-minet HTTP/1.1" 200 299  
"http://localhost/coursphp/testlink/pagelink.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5  
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
```

... on voit que dans la requête de type POST, les paramètres correspondant aux champs de formulaire n'apparaissent pas. Ils font pourtant bien partie de la requête, mais dans une requête de type POST, ils sont encapsulés dans la requête et n'apparaissent pas dans l'URL. Or c'est cette URL que nous montre le fichier « access.log ».

Je vous encourage vivement à modifier la balise « form » et en particulier l'attribut « method », en y plaçant la valeur « get » :

```
<form action="pagecible.php" method="get">
```

Rafraîchissez la page de votre formulaire, saisissez quelques données et cliquez de nouveau sur le bouton « valider ». Vous voyez que vous obtenez une requête à rallonge, dans laquelle tous les champs du formulaire sont présents :

[http://localhost/greta91\\_2017/testform/pagecible.php?search=test+de+recherche&number=3&myselect=B&airport=MEM&totime=2017-07-26T05%3A00&yournumber=7&valid=Valider](http://localhost/greta91_2017/testform/pagecible.php?search=test+de+recherche&number=3&myselect=B&airport=MEM&totime=2017-07-26T05%3A00&yournumber=7&valid=Valider)

On retrouve ces mêmes paramètres dans la partie détaillée de la vue « Network » :

▼ **Query String Parameters** [view source](#) [view URL encoded](#)

search: test de recherche  
number: 3  
myselect: B  
airport: MEM  
totime: 2017-07-26T05:00  
yournumber: 7  
valid: Valider

Tiens au fait, la modification de la méthode d'envoi du formulaire a eu une conséquence inattendue. Regardez ce qui se passe dans la page affichée après envoi :

```
array(0) {  
}  
$_POST ne contient aucun paramètre  
  
Contenu de $_POST affiché avec une boucle foreach
```

L'interpréteur PHP n'a trouvé aucune information dans le tableau PHP `$_POST`. C'est normal, toutes les informations se trouvent maintenant dans le tableau `$_GET`, et ce cas de figure n'est pas prévu dans notre script PHP.

Des questions se posent alors :

- dans quel cas faut-il utiliser la méthode GET, dans quel cas faut-il utiliser POST ?
- ne serait-il pas plus simple d'utiliser tout le temps GET ?

En règle générale, on réservera la méthode GET aux formulaires dédiés à la recherche d'informations, sous réserve qu'ils contiennent peu de paramètres : car les requêtes HTTP de type GET ont une longueur maximale qui varie d'un navigateur à l'autre, mais qui tourne généralement autour de 1024 caractères (TODO : longueur à vérifier).

Par convention, on utilisera systématiquement la méthode POST dans les cas de figure suivants :

- pour les formulaires de mise à jour, création et suppression,
- pour les formulaires contenant beaucoup de paramètres (pas de limite de longueur avec POST, contrairement à GET)
- pour les formulaires susceptibles de contenir des caractères exotiques (chinois au autres) car les requêtes de type GET ne peuvent pas les transmettre correctement (pour de nombreuses raisons un peu longues à détailler ici), alors que les requêtes POST n'ont aucun souci avec ça.

### 4.4.3 Un formulaire plus intelligent

Nous allons maintenant étudier comment rendre notre formulaire plus intelligent.

Je vous propose de dupliquer le sous-répertoire « testform » et de créer un sous-répertoire « testform2 ».

Je vous propose également de simplifier le code des fichiers « pageform.php » et « pagecible.php », histoire de ne pas alourdir ce chapitre avec des exemples trop lourds.

Nous allons réutiliser le fichier CSS tel quel, mais :

- un formulaire avec 3 champs de saisie devrait suffire pour notre démonstration
- un script d'affichage « pagecible.php » simplifié contenant juste une boucle « foreach » nous suffira également

Voici le code source initial de « pageform.php » :

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title>Exemple de formulaire (2)</title>
    <link rel="stylesheet" href="pageform.css">
</head>
<body>
<form method="POST">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field" placeholder="saisissez un critère de recherche">
        </p>
        <p>
            <label for="number_field">Exemple de champ "number"</label>
            <input type="number" name="number" id="number_field" >
        </p>
        <p>
            <label for="select_field">Exemple de champ "select"</label>
            <select name="myselect" id="select_field">
                <option value="">Choisissez une valeur</option>
                <option value="A">valeur A</option>
                <option value="B">valeur B</option>
                <option value="C">valeur C</option>
            </select>
        </p>
    </fieldset>
    <input type="submit" name="valid" value="Valider" />
</form>
</body>
</html>
```

Et voici le code source initial de « pagecible.php » :

```
<?php
echo '<br>Contenu de $_POST affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_POST as $postkey=>$postval) {
    echo "<li>{$postkey} => {$postval} </li>" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Dans le formulaire, je n'ai conservé que les 3 premiers champs de saisie, car c'est suffisant pour étudier les différents points que je souhaite vous montrer maintenant. J'ai aussi fait quelques modifications sur les éléments conservés, modifications que je vais détailler tout de suite :

- J'ai retiré l'attribut « required » sur le second champ de formulaire, car je souhaite gérer d'une autre manière (que nous allons voir dans un instant) le fait que la saisie de ce champ est obligatoire.
- J'ai aussi ajouté une option au champ « select » qui est la suivante, je l'ai placée en première position :
 

```
<option value="">Choisissez une valeur</option>
```
- Dans le formulaire, vous aurez peut être remarqué que j'ai modifié la balise « form » de la façon suivante :
 

```
<form method="POST">
```

Explication :

- je souhaite utiliser la méthode POST, considérant que mon formulaire va être utilisé pour effectuer de la mise à jour côté serveur (même si ce n'est pas vrai pour l'instant)
- le fait d'avoir supprimé l'attribut « action » va avoir pour effet que le navigateur va « boucler » sur la même page, donc sur le même script « pageform.php ». Nous devrons donc trouver un moyen pour lancer l'exécution du script « pagecible.php » (c'est bien sûr PHP qui va nous aider dans cette tâche, nous verrons comment dans un moment).

Maintenant, je vais instaurer quelques nouvelles règles de fonctionnement pour notre formulaire :

1. tous les champs de saisie sont obligatoires, et je souhaite que le contrôle soit effectué par PHP (d'où la suppression de l'attribut « required » sur l'un des champs de saisie)
2. c'est seulement quand tous les champs de saisie seront bien remplis, que je déclencherai l'appel du script « pagecible.php ». Tant que certains champs de saisie resteront « vides », je bouclerai sur le formulaire en cours.
3. Lors du réaffichage du formulaire, les champs correctement renseignés par l'utilisateur devront conserver la valeur saisie pour ne pas obliger l'utilisateur à les saisir de nouveau.

Nous allons voir que ces règles en apparence simples vont avoir un certain impact sur le code de notre script « pageform.php ».

Pour comprendre comment agir sur notre script de gestion de formulaire, je vous propose de modifier le script « pageform.php », en ajoutant le code suivant, juste en dessous de la balise « body », et juste avant la balise « form » :

```
<?php
    echo 'Dump de $_POST :<br>';
    echo '<pre>';
    var_dump($_POST);
    echo '</pre><br>';
?>
```

Vous connaissez déjà la fonction `var_dump()`, je ne la présente pas ici. Je rappelle simplement que la balise « `pre` », influe sur la manière dont le navigateur affiche les infos renvoyées par `var_dump()`. Vous pourrez vous amuser à supprimer la balise « `pre` » pour voir la différence avant/après. La balise « `/pre` » placée après le `var_dump()` a pour effet de remettre le navigateur dans un mode d'affichage normal.

Mais revenons à nos moutons... Maintenant que nous avons ajouté la fonction `var_dump()`, je vous propose de regarder ce que ça donne dans le navigateur :

Dump de \$\_POST :

```
array(0) {}
```

— Formulaire en vrac —

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

La fonction `var_dump()` nous indique clairement que le tableau `$_POST` est vide au premier affichage de la page.

OK, saisissez maintenant quelques valeurs dans le formulaire, comme par exemple ceci :

Dump de \$\_POST :

```
array(0) {  
}
```

Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

... puis cliquez sur « valider » :

Dump de \$\_POST :

```
array(4) {  
    ["search"]=>  
        string(16) "ceci est un test"  
    ["number"]=>  
        string(1) "4"  
    ["myselect"]=>  
        string(0) ""  
    ["valid"]=>  
        string(7) "Valider"  
}
```

Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

Intéressant non ? Le tableau \$\_POST est maintenant bien rempli. Vous notez que j'ai laissé le champ « select » avec sa première option (dont la valeur est à blanc). Il apparaît bien comme tel dans le tableau.

Par contre, dans le formulaire, tous les champs sont réinitialisés, nous avons perdu notre saisie. Ou plus exactement, nous avons déporté cette saisie dans le tableau `$_POST`, mais nous n'avons rien fait pour que le formulaire soit en mesure de réafficher les valeurs saisies. Nous devons lui ajouter un peu d'intelligence.

Une première manière de faire consiste à ajouter un peu de code PHP à l'intérieur de chaque champ de saisie, au niveau de l'attribut « value », et d'y placer le contenu des postes « search », « number » et « myselect ». Dans l'exemple ci-dessous, j'ai « surligné » en gris plus foncé les parties modifiées :

```
<form method="POST">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                   placeholder="saisissez un critère de recherche"
                   value="<?php echo $_POST['search']; ?>" />
        </p>
        <p>
            <label for="number_field">Exemple de champ "number"</label>
            <input type="number" name="number" id="number_field"
                   value="<?php echo $_POST['number']; ?>" />
        </p>
        <p>
            <label for="select_field">Exemple de champ "select"</label>
            <select name="myselect" id="select_field"
                   value="<?php echo $_POST['myselect']; ?>">
                <option value="">Choisissez une valeur</option>
                <option value="A">valeur A</option>
                <option value="B">valeur B</option>
                <option value="C">valeur C</option>
            </select>
        </p>
    </fieldset>
    <input type="submit" name="valid" value="Valider" />
</form>
```

Regardons page suivante ce que cela donne côté navigateur.

Dump de \$\_POST :

```
array(0) {
}
```

Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

Oups, c'est bizarre ce qui se produit dans le champ de recherche. On ne le voit pas sur le champ « number », car comme c'est champ de type « number », le navigateur masque certaines infos, mais le code source HTML est sans appel... ça va mal sur les 3 champs de saisie !

```
<body>
Dump de $_POST :<br><pre>array(0) {
}</pre><br><form method="POST">
<fieldset>
    <legend>Formulaire en vrac</legend>
    <p>
        <label for="search_field">Exemple de champ "search"</label>
        <input type="search" name="search" id="search_field"
            placeholder="saisissez un critère de recherche"
            value=<br />
    <b>Notice</b>: Undefined index: search in <b>D:\Tools\xampp\htdocs\GRETA91_2017\testform2\pageform.php</b> on line <b>21</b><br />
    </p>
    <p>
        <label for="number_field">Exemple de champ "number"</label>
        <input type="number" name="number" id="number_field"
            value=<br />
    <b>Notice</b>: Undefined index: number in <b>D:\Tools\xampp\htdocs\GRETA91_2017\testform2\pageform.php</b> on line <b>27</b><br />
    </p>
    <p>
        <label for="select_field">Exemple de champ "select"</label>
        <select name="myselect" id="select_field"
            value=<br />
            <option value="">Choisissez une valeur</option>
            <option value="A">valeur A</option>
            <option value="B">valeur B</option>
            <option value="C">valeur C</option>
        </select>
    </p>
</fieldset>
<input type="submit" name="valid" value="Valider" />
</form>
</body>
```

On voit que sur chaque champ de saisie, l'attribut « value » contient un message de type « notice » envoyé par PHP, nous indiquant que nous lui demandons de faire n'importe quoi.

Bon, quand on y réfléchit, c'est assez logique. Lors du tout premier affichage de la page, on demande au PHP d'insérer dans le code HTML des informations qui n'existent pas encore dans le tableau `$_POST`. Ces informations ne seront présentes dans `$_POST` qu'à partir du moment où nous aurons cliqué sur le bouton « valider ». Il nous faut donc insérer du code PHP un peu plus intelligent.

Donc plutôt que d'écrire ceci :

```
<input type="search" name="search" id="search_field"
       placeholder="saisissez un critère de recherche"
       value="php echo $_POST['search']; ?&gt;" /&gt;</pre

```

... je vous propose d'écrire cela :

```
<input type="search" name="search" id="search_field"
       placeholder="saisissez un critère de recherche"
       value="php echo isset($_POST['search']) ? $_POST['search'] : ''; ?&gt;" /&gt;</pre

```

Explication : cette syntaxe bizarre que j'ai utilisée dans l'exemple précédent, et que je vous remets ci-dessous...

```
isset($_POST['search']) ? $_POST['search'] : '';
```

... revient en définitive à écrire ceci (en syntaxe algorithmique) :

```
Si le poste « search » existe dans le tableau $_POST
Alors je renvoie en sortie le contenu de $_POST['search']
Sinon je renvoie en sortie un blanc
FinSi
```

C'est en fait un test en condensé, j'aurais pu écrire un vrai test « if »... comme ceci :

```
<input type="search" name="search" id="search_field"
       placeholder="saisissez un critère de recherche"
       value="php if (isset($_POST['search'])) echo $_POST['search']; else
echo ''; ?&gt;" /&gt;</pre

```

... cela aurait fonctionné également, ça faisait juste un peu plus de code à saisir. En termes de lisibilité, je préfère la version condensée, mais dans un cas comme dans l'autre, il faut bien reconnaître que ça fait un code assez... moche (pour rester poli). Et en plus il va falloir le dupliquer sur chacun des champs... berk !!! :Q

Mais au fait, est-ce que ce code fonctionne dans tous les cas ? Je ne m'inquiète pas trop pour le champ de type « number », mais est-ce que cela va réellement fonctionner avec le champ de type « select » ? Le mieux c'est d'essayer non ?

Voici le code PHP inséré dans le code HTML du champ « myselect » :

```
<select name="myselect" id="select_field"
        value=<?php echo isset($_POST['myselect']) ? $_POST['myselect'] : '';
?>>
```

Et voici le même code en fonctionnement :

Dump de \$\_POST :

```
array(4) {
  ["search"]=>
  string(0) ""
  ["number"]=>
  string(0) ""
  ["myselect"]=>
  string(1) "B"
  ["valid"]=>
  string(7) "Valider"
}
```

Formulaire en vrac

Exemple de champ "search" (saisissez un critère de re...)

Exemple de champ "search" (saisissez un critère de re...)

Exemple de champ "number" | 154.67 x 18

Exemple de champ "select" (Choisissez une valeur ▾)

Valider

Label for"select\_field">Exemple de champ "select"</label>
... <select name= myselect id= select\_field value=B == \$0
 <option value>Choisissez une valeur</option>
 <option value=A>valeur A</option>
 <option value=B>valeur B</option>
 <option value=C>valeur C</option>
 </select>
</p>
</fieldset>
<input type="submit" name="valid" value="Valider">
</form>
</body>
</html>

C'est bizarre non ? L'inspecteur d'élément nous indique que l'attribut « value » est bien alimenté avec la valeur sélectionnée (en l'occurrence « B »), pourtant c'est toujours la première option qui s'affiche (et qui correspond au code option à blanc).

Bon, si vous avez zappé la partie de mon cours « HTML5 – premiers pas » relative au champ « select » (ou si vous l'avez lu mais un peu oublié, auquel cas je vous pardonne 😊), je rappelle ce que j'indiquais dans le cours HTML5 :

*Exemple de liste déroulante :*

```
<select name="test">
  <option value="0">Développeur</option>
  <option value="1" selected>Utilisateur</option>
</select>
```

Exemple de liste déroulante :  ▾

L'attribut « *selected* » est facultatif. Quand il est défini, il détermine quelle est la valeur sélectionnée par défaut.

Bref, en ce qui concerne notre code... c'était bien essayé 😊, mais ça ne pouvait pas fonctionner. Une solution fonctionnelle, en revanche, ce serait celle-ci :

```
<select name="myselect" id="select_field">
    <option value="">Choisissez une valeur</option>
    <option value="A" <?php echo isset($_POST['myselect']) &&
        $_POST['myselect'] == 'A'? 'selected': ''; ?>>valeur A</option>
    <option value="B" <?php echo isset($_POST['myselect']) &&
        $_POST['myselect'] == 'B'? 'selected': ''; ?>>valeur B</option>
    <option value="C" <?php echo isset($_POST['myselect']) &&
        $_POST['myselect'] == 'C'? 'selected': ''; ?>>valeur C</option>
</select>
```

La vache !!! 😱, ça commence à piquer les yeux tout ce code PHP disséminé dans le code HTML. C'est « pas glop », dirait le Pifou 😊. J'en connais d'autres qui diraient que ça sent le mois. Je vous invite quand même à vérifier que le code fonctionne (moi j'ai déjà fait le test, c'est bon en ce qui me concerne).

En fait, il existe plein de manières différentes de générer des formulaires à partir de PHP. Il existe plusieurs styles, plusieurs écoles pourrait-on dire. Je ne vais certainement pas contenter tout le monde (je ne vais même pas essayer), mais nous allons étudier 2 approches différentes qui peuvent toutes deux convenir, au moins pour les cas relativement simples comme l'est notre formulaire de test.

La première approche va consister à créer deux fonctions PHP, qui permettront d'alléger notre code HTML. Nous allons l'étudier dans la suite de ce chapitre.

La seconde approche va consister à générer la totalité d'un champ de saisie HTML à partir de code PHP. Cela nous demandera un peu plus de travail, nous l'étudierons au chapitre 4.4.5 (« générateur de formulaire »).

Voyons cette première approche, qui consiste à simplifier l'écriture du code PHP imbriqué dans le code HTML, sans pour autant tout « péter ».

Je vais prendre quelque raccourcis, en vous proposant deux fonctions toutes faites, que je vous laisse le soin d'étudier (le code n'est pas très compliqué) :

- La fonction `get_form_input_value()`, à utiliser sur les 2 premiers champs du formulaire
- La fonction `get_form_select_selected()`, à utiliser sur le champ de type « select »

```
/**
 * Renvoie en sortie la valeur d'un champ de saisie
 * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
 *   'post' (valeur par défaut)
 *   'get'
 * @param $field
 * @param string $method
 * @return string
 */
function get_form_input_value($field, $method='post') {
    // suppression des blancs parasites et forçage des valeurs en minuscule
    $field = strtolower(trim($field));
    $method = strtolower(trim($method));

    if ($method == 'get') {
        $value = isset($_GET[$field]) ? trim($_GET[$field]): '';
    } else {
        $value = isset($_POST[$field]) ? trim($_POST[$field]): '';
    }
    return $value;
}

/**
 * Renvoie en sortie la valeur "selected" pour les champs de type SELECT
 * si la valeur du champ considéré est égale au second paramètre
 * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
 *   'post' (valeur par défaut)
 *   'get'
 * @param $field
 * @param $option_key
 * @param string $method
 * @return string
 */
function get_form_select_selected($field, $option_key, $method='post') {
    // suppression des blancs parasites et forçage des valeurs en minuscule
    $field = strtolower(trim($field));
    $method = strtolower(trim($method));
    if ($method == 'get') {
        $value = isset($_GET[$field]) ? trim($_GET[$field]): '';
    } else {
        $value = isset($_POST[$field]) ? trim($_POST[$field]): '';
    }
    if ($value == $option_key) {
        return 'selected';
    }
    return '';
}
```

Je vous propose de placer la déclaration de ces 2 fonctions dans le script « pageform.html », juste après l'appel de la fonction var\_dump().

Voyons maintenant comment placer l'appel de ces fonctions dans le formulaire :

```
<form method="POST">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                   placeholder="saisissez un critère de recherche"
                   value=<?php echo get_form_input_value('search'); ?>">
        />
    </p>
    <p>
        <label for="number_field">Exemple de champ "number"</label>
        <input type="number" name="number" id="number_field"
               value=<?php echo get_form_input_value('number'); ?>">
    />
    </p>
    <p>
        <label for="select_field">Exemple de champ "select"</label>
        <select name="myselect" id="select_field">
            <option value="">Choisissez une valeur</option>
            <option value="A" ><?php echo get_form_select_selected('myselect', 'A'); ?>>valeur A</option>
            <option value="B" ><?php echo get_form_select_selected('myselect', 'B'); ?>>valeur B</option>
            <option value="C" ><?php echo get_form_select_selected('myselect', 'C'); ?>>valeur C</option>
        </select>
    </p>
    </fieldset>
    <input type="submit" name="valid" value="Valider" />
</form>
```

Bon, ce n'est pas le Pérou, mais c'est déjà mieux. Mais j'entends déjà un de mes élèves au fond de la classe : « eh m'sieur, on appelle 3 fois la fonction get\_form\_select\_selected(), ce n'est pas un peu redondant ? »

Je suis d'accord, on peut effectivement faire mieux.

On peut par exemple écrire quelque chose de ce genre :

```
<select name="myselect" id="select_field">
    <option value="">Choisissez une valeur</option>
    <?php
    foreach(array('A', 'B', 'C') as $valselect) {
        echo "<option value=\"{$valselect}\">\n"
        . get_form_select_selected('myselect', $valselect)
        . ">valeur {$valselect}</option>" . PHP_EOL;
    };
    ?>
</select>
```

Dans cet exemple, j'ai généré dynamiquement un tableau contenant les valeurs A, B et C, et je m'en suis servi pour exécuter une boucle « foreach » générant une ligne d'option HTML pour chaque poste du tableau.

Attention, le code ci-dessus est une proposition, qui a le mérite de fonctionner. Mais on peut trouver bien d'autres manières de faire.

Ceci étant, nous n'avons pas couvert toutes les règles de fonctionnement que nous nous étions fixées. Je les rappelle ci-dessous :

1. tous les champs de saisie sont obligatoires, et je souhaite que le contrôle soit effectué par PHP (d'où la suppression de l'attribut « required » sur l'un des champs de saisie)
2. c'est seulement quand tous les champs de saisie seront bien remplis, que je déclencherai l'appel du script « pagecible.php ». Tant que certains champs de saisie resteront « vides », je bouclerai sur le formulaire en cours.
3. Lors du réaffichage du formulaire, les champs correctement renseignés par l'utilisateur devront conserver la valeur saisie pour ne pas obliger l'utilisateur à les saisir de nouveau.

Nous couvrons bien fonctionnellement la 3<sup>ème</sup> exigence, mais il nous manque les 2 premières.

Pour la gestion des champs de saisie obligatoires, il y a là encore plusieurs manières de faire. On peut même dire qu'il y a deux grandes écoles :

- Soit on affiche toutes les erreurs dans un bloc de commentaires placé au dessus du formulaire
- Soit on affiche les erreurs au niveau de chaque champ de saisie concerné : dans ce cas, on place généralement le message d'erreur à droite de la balise « span », ou entre la balise « span » et la balise « input » concernée, ou quelquefois sous la balise « input ». Le choix de l'emplacement du message d'erreur dépend souvent de contraintes de place, mais aussi du type d'ergonomie recherchée, ou encore des habitudes des utilisateurs, qu'on essaie de ne pas trop perturber, quand c'est possible.

Si on décide de regrouper tous les messages d'erreur dans un bloc de commentaires situé au dessus du formulaire, alors une boucle de type « foreach » peut suffire. On pourrait faire cette boucle directement à partir du tableau \$\_POST, et afficher les erreurs au fur et à mesure, mais ce n'est pas une bonne idée, et cela pour plusieurs raisons :

- On peut souhaiter ignorer certaines données contenues dans \$\_POST, c'est le cas notamment du poste « valid » qui correspond au bouton de validation du formulaire, et qui ne nous intéresse pas. Il serait plus pratique d'effectuer une première boucle qui filtre \$\_POST et ne retient que les données significatives contenues ce tableau.
- On peut souhaiter connaître à l'avance le nombre d'erreurs, pour optimiser l'affichage. Et pour cela, il est plus facile de procéder d'abord à l'analyse de \$\_POST, de manière à pouvoir stocker les éventuelles erreurs dans un tableau secondaire. On utilisera ce tableau secondaire pour déterminer le nombre d'erreurs, et bien sûr les afficher.

Concrètement, j'aimerais obtenir un affichage de ce type :

#### Liste des erreurs

- Champ search : zone obligatoire
- Champ number : zone obligatoire
- Champ myselect : zone obligatoire

#### Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

Voici le code PHP, à insérer en remplacement de la fonction var\_dump(), cette dernière ne nous étant plus daucune utilité :

```
// Initialisation du tableau des erreurs
$erreurs = [];
// Boucle pour analyser si des erreurs sont présentes
foreach($_POST as $keypost=>$valpost) {
    if ($keypost != 'valid') {
        $valpost = trim($valpost);
        if ($valpost == '') {
            $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
        }
    }
}
// Si des erreurs sont présentes, alors on en affiche la liste
if (count($erreurs)>0) {
    echo "<fieldset style='background-color: bisque; border-radius: 5px;'>".PHP_EOL;
    echo "<legend>Liste des erreurs</legend>".PHP_EOL;
    echo "<ul>".PHP_EOL;
    foreach($erreurs as $valerr) {
        echo "<li>$valerr</li>".PHP_EOL;
    }
    echo "</ul>".PHP_EOL;
    echo "</fieldset>".PHP_EOL;
}
```

Je vous recommande de tester ce code, de le décortiquer, de regarder aussi avec la fonction d'examen du navigateur le code HTML généré. Réfléchissez aussi à la manière dont vous pourriez améliorer ce code PHP. Vous pouvez envisager de le placer dans une fonction PHP « maison ». Si vous écrivez cette fonction de façon suffisamment générique, vous pourrez la réutiliser sur plusieurs scripts PHP.

Je rappelle qu'il restait une dernière exigence, celle consistant à « débrancher » vers le script « pagecible.php », si le formulaire ne contient plus d'erreurs. Pour cela, il nous faut une fonction de redirection, permettant d'interrompre le script PHP en cours, et de réorienter le flux d'exécution du PHP vers un autre script.

Pour ce faire, PHP met à notre disposition la fonction `header()`.

Elle est pratique cette fonction `header()`, c'est une sorte de couteau suisse car elle permet de faire beaucoup de choses, mais ce qui nous intéresse ici, c'est la redirection.

Donc, pour effectuer une redirection vers notre script « `pagecible.php` », il nous suffit d'écrire ceci :

```
header('location: pagecible.php');
exit; // on stoppe l'exécution du script courant
```

Bon, en pratique, c'est légèrement plus compliqué, car nous devons déterminer s'il n'y a plus d'erreurs avant de procéder à la redirection, donc nous pourrions envisager un code de ce type (attention, ce n'est qu'une version provisoire) :

```
if (count($erreurs)==0) {
    // redirection vers "pagecible.php"
    header('location: pagecible.php');
    exit; // on stoppe l'exécution du script courant
}
```

On pourrait placer ce code en complément du code précédent, juste après la boucle d'analyse de `$_POST` qui effectue le remplissage du tableau `$erreurs`.

Attention, la documentation officielle de la fonction `header()` indique ceci :

*header() permet de spécifier l'en-tête HTTP string lors de l'envoi des fichiers HTML. /.../  
N'oubliez jamais que `header()` doit être appelée avant que le moindre contenu ne soit envoyé, soit par des lignes HTML habituelles dans le fichier, soit par des affichages PHP. Une erreur très classique est de lire un fichier avec `include` ou `require`, et de laisser des espaces ou des lignes vides, qui produiront un affichage avant que la fonction `header()` ne soit appelée. Le même problème existe avec les fichiers PHP/HTML standards.*

Source : <http://php.net/manual/fr/function.header.php>

Ce qui signifie que, en théorie, il conviendrait de placer notre redirection bien avant d'avoir généré le moindre code dans le buffer HTML (dont je rappelle qu'il commence à se remplir dès qu'on envoie du HTML brut, comme notre entête de page). Mais en pratique, je ne rencontre pas ce problème de redirection sur notre exemple, qui pourtant ne respecte pas la règle édictée ci-dessus (en rouge). J'en viens même à me demander si ce problème est réellement d'actualité, et si la doc de php.net est véritablement à jour sur cette fonction `header()`. Je conserve néanmoins le souvenir d'avoir rencontré ce problème de redirection inopérante sur certains

environnements PHP sous Linux. On va laisser ce sujet de côté, retenez simplement que, si vous recontrez le problème, il conviendra de déplacer le code à l'origine de la redirection avant tout envoi de donnée vers le buffer HTML, que ce soit du code HTML brut, ou du code généré par les fonctions PHP echo() ou print().

Je rappelle ci-dessous le code source du script « pagecible.php » :

```
<?php
echo '<br>Contenu de $_POST affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_POST as $postkey=>$postval) {
    echo "<li>{$postkey} => {$postval}</li>" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Je vous propose d'exécuter notre formulaire, de provoquer volontairement une erreur (en oubliant de remplir un champ), pour vérifier que la boucle sur le formulaire fonctionne, puis de remplir tous les champs et de recliquer sur le bouton « valider ». Vous allez voir apparaître le message suivant :

Contenu de \$\_POST affiché avec une boucle foreach

Mince ! \$\_POST est vide ? Où sont passées mes données? WTF !!!

Ben oui, vous ne pensiez quand même pas vous en tirer si facilement ! 😊  
Ce qu'il faut comprendre ici, c'est que le contenu de \$\_POST, de même que celui de \$\_GET, n'est accessible que dans la requête HTTP initiale. En cas de redirection, ces données ne « suivent » pas, elles demeurent dans le script PHP d'où est parti la redirection. Il nous faut donc trouver un moyen de les envoyer dans le script « pagecible.php ».

Heureusement, à chaque problème, il y a une solution 😊.

ATTENTION : parmi les solutions, il y en a de très mauvaises. Il se trouve que l'on raconte beaucoup d'âneries sur internet... et même dans certains forums qui ont pignon sur rue. En l'occurrence, la solution préconisée par certains développeurs, consistant à envoyer le contenu de \$\_POST sous forme de paramètres dans la redirection, c'est-à-dire concaténés derrière « pagecible.php » à la manière d'une requête HTTP de type GET, est une énorme... bêtise 😞 (j'avais pensé à un autre mot, mais je me suis censuré).

Vous avez peut être du mal à comprendre de quoi je parle, mais je ne veux pas vous montrer d'exemple de code, car un lecteur pressé pourrait croire que c'est la bonne solution. Et comme en plus, ça me hérissé le poil, je ne vous montrerai pas ça.

La bonne solution consiste à faire transiter les informations au moyen d'un tableau PHP qui s'appelle `$_SESSION`. Pour pouvoir utiliser ce tableau, nous devons impérativement démarrer la gestion de session via la fonction PHP `session_start()`. Cette fonction doit être placée au tout début du script, de préférence à la toute première ligne.

A partir du moment où le gestionnaire de session est démarré, nous pouvons alimenter le tableau `$_SESSION` comme bon nous semble. Pour pouvoir récupérer les données de `$_SESSION` dans le script « `pagecible.php` », nous devrons là aussi effectuer un démarrage du gestionnaire de session, avec la fonction `session_start()`.

Je vous propose ci-dessous une nouvelle version de notre formulaire, avec les modifications suivantes :

- Activation du gestionnaire de session
- Stockage des données du formulaire dans un tableau PHP « maison » que j'ai appelé « `gooddata` », puis copie de ce tableau dans `$_SESSION` juste avant de procéder à la redirection

Voici le code source :

```
<?php
// démarrage du gestionnaire de session
session_start();
?><!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title>Exemple de formulaire (2)</title>
    <link rel="stylesheet" href="pageform.css">
</head>
<body>
<?php
    // Initialisation du tableau des erreurs
    $erreurs = [];
    $gooddata = [];

    // Boucle pour analyser si des erreurs sont présentes
    foreach($_POST as $keypost=>$valpost) {
        if ($keypost != 'valid') {
            $valpost = trim($valpost);
            if ($valpost == '') {
                $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
            } else {
                $gooddata[$keypost] = $valpost;
            }
        }
    }

    // si le tableau $gooddata contient des données et qu'il n'y a pas d'erreurs
    if (count($gooddata)>0 && count($erreurs)==0) {
        // copie des bonnes données dans $_SESSION
        $_SESSION = $gooddata;
        // redirection vers "pagecible.php"
        header('location: pagecible.php');
    }
}
```

```

        exit; // on stoppe l'exécution du script courant
    }

    if (count($erreurs)>0) {
        echo '<fieldset style="background-color: bisque; border-radius: 5px;">' .PHP_EOL;
        echo '<legend>Liste des erreurs</legend>' .PHP_EOL;
        echo '<ul>' .PHP_EOL;

        foreach($erreurs as $valerr) {
            echo "<li>$valerr</li>" .PHP_EOL;
        }
        echo '</ul>' .PHP_EOL;
        echo '</fieldset>' .PHP_EOL;
    }

    /**
     * Renvoie en sortie la valeur d'un champ de saisie
     * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
     *   'post' (valeur par défaut)
     *   'get'
     * @param $field
     * @param string $method
     * @return string
     */
    function get_form_input_value($field, $method='post') {
        // suppression des blancs parasites et forçage des valeurs en minuscule
        $field = strtolower(trim($field));
        $method = strtolower(trim($method));

        if ($method == 'get') {
            $value = isset($_GET[$field]) ? trim($_GET[$field]): '';
        } else {
            $value = isset($_POST[$field]) ? trim($_POST[$field]): '';
        }
        return $value;
    }

    /**
     * Renvoie en sortie la valeur "selected" pour les champs de type SELECT
     * si la valeur du champ considéré est égale au second paramètre
     * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
     *   'post' (valeur par défaut)
     *   'get'
     * @param $field
     * @param $option_key
     * @param string $method
     * @return string
     */
    function get_form_select_selected($field, $option_key, $method='post') {
        // suppression des blancs parasites et forçage des valeurs en minuscule
        $field = strtolower(trim($field));
        $method = strtolower(trim($method));
        if ($method == 'get') {
            $value = isset($_GET[$field]) ? trim($_GET[$field]): '';
        } else {
            $value = isset($_POST[$field]) ? trim($_POST[$field]): '';
        }
        if ($value == $option_key) {
            return 'selected';
        }
    }
}

```

```

        }
        return '';
    }
?>
<form method="POST">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                   placeholder="saisissez un critère de recherche"
                   value=<?php echo get_form_input_value('search'); ?>" />
        </p>
        <p>
            <label for="number_field">Exemple de champ "number"</label>
            <input type="number" name="number" id="number_field"
                   value=<?php echo get_form_input_value('number'); ?>" />
        </p>
        <p>
            <label for="select_field">Exemple de champ "select"</label>
            <select name="myselect" id="select_field">
                <option value="">Choisissez une valeur</option>
                <?php
                    foreach(array('A', 'B', 'C') as $viselect) {
                        echo "<option value=\"{$viselect}\">\n".
                            . get_form_select_selected('myselect', $viselect)
                            . ">valeur {$viselect}</option>" . PHP_EOL;
                    };
                ?>
            </select>
        </p>
    </fieldset>
    <input type="submit" name="valid" value="Valider" />
</form>
</body>
</html>
```

Dans le script « pagecible.php », il y a aussi quelques modifications mineures :

- Démarrage du gestionnaire de session au tout début du script
- Remplacement de `$_POST` par `$_SESSION` dans la boucle « `foreach` ».

```
<?php
// démarrage du gestionnaire de session
session_start();

echo '<br>Contenu de $_SESSION affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_SESSION as $postkey=>$postval) {
    echo "<li>{$postkey} => {$postval} </li>" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Refaites un test complet, relisez aussi la liste des 3 exigences que nous avions fixées au début de ce chapitre, vous allez voir que tout fonctionne.

Formulaire en vrac –

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"  ▾

Contenu de `$_SESSION` affiché avec une boucle foreach

- search => test
- number => 4
- myselect => B

C'était un gros chapitre, pas vrai ?

Nous avons découvert ici beaucoup de choses :

- Comment gérer un formulaire et boucler tant qu'il contient des erreurs, sans perdre les données déjà saisies par l'utilisateur
- Comment afficher les erreurs
- Comment gérer une redirection vers un autre script – via la fonction `header()` - sans perdre les données saisies par l'utilisateur (grâce au gestionnaire de session)
- Nous avons découvert le gestionnaire de session, avec la fonction `session_start()` et le tableau associatif `$_SESSION`

Prenez le temps de bien relire ce chapitre, de bien retester les différents exemples, de bien vous approprier les techniques que nous venons d'étudier. Vous pouvez le faire au travers d'un formulaire différent si vous le jugez utile.

Si vous butez sur des problèmes relatifs à l'utilisation de `$_POST`, rappelez-vous que ce n'est rien d'autre qu'un tableau associatif, et si cela ne vous rassure pas, relisez les chapitres 4.2.3 et 4.4.2 qui parlent respectivement de tableau et de `$_POST`.

Et n'oubliez pas de faire une pause, je pense que vous l'avez bien méritée 😊.

Quelques remarques complémentaires s'imposent :

- Le code du script « pageform.php » est un peu lourd. Sa lisibilité pourrait être grandement améliorée si certains parties du code étaient déportées dans des fonctions, et placées dans des script distincts importés via la fonction PHP require().
- L'objectif visé dans ce chapitre, c'est la découverte et la compréhension des mécanismes en jeu dans les échanges entre client (navigateur) et serveur (stack PHP).
- L'optimisation du code PHP est certes souhaitable, mais elle peut être réalisée dans un second temps par l'apprenant, ce qui peut constituer un excellent exercice en soi
- Dans le chapitre 5.6 en annexe, vous trouverez une version légèrement différente des scripts « pageform.php » et « pagecible.php ». Dans cette version, le champ de type « select » a été modifié, avec l'ajout de l'attribut « multiple ». Vous verrez dans le chapitre en annexe, que cette modification en apparence mineure a un fort impact sur le code PHP.

#### 4.4.4 Un formulaire robuste et sécurisé

Ce chapitre est une très courte introduction aux problématiques de sécurité liées à la gestion des formulaires.

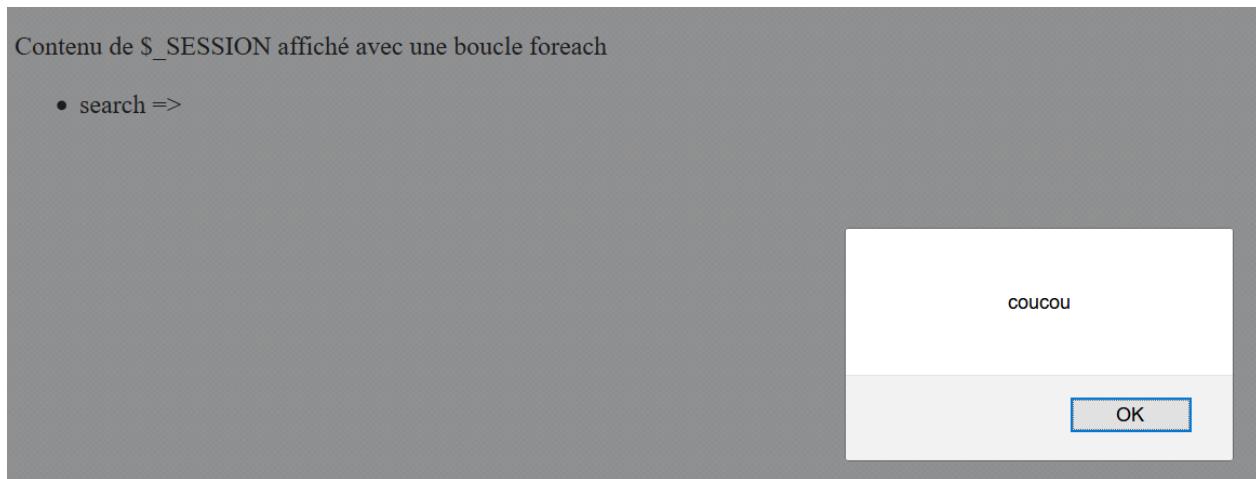
C'est souvent un sujet qui est abordé beaucoup trop tard, quand les développeurs ont pris de mauvaises habitudes. Alors, je préfère aborder le sujet, au moins partiellement, dès maintenant.

Je vous propose de commencer par un test.

Sur la version finale du formulaire du chapitre précédent, saisissez dans le premier champ de saisie (celui qui est de type « text »), le code suivant :

```
<script>alert('coucou');</script>
```

En arrivant sur la page affichée par le script « pagecible.php », on a la mauvaise surprise de voir notre code Javascript s'exécuter, simplement parce que nous avons fait un « echo » du contenu du champ « search » :



Nous venons de mettre le doigt dans un monde bizarre, celui du XSS... du quoi ?

XSS est l'acronyme déformé de « Cross Site Scripting ». En toute logique, on devrait dire CSS, mais l'acronyme CSS étant déjà pris pour autre chose, l'acronyme XSS a finalement été retenu pour éviter toute confusion.

Je vous propose de lire la définition Wikipédia :

*Le cross-site scripting (abrégé XSS), est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, permettant ainsi de provoquer des actions sur les navigateurs web visitant la page. Les possibilités des XSS sont très larges puisque l'attaquant peut utiliser tous les langages pris en charge par le navigateur (JavaScript, Java, Flash...) et de nouvelles possibilités sont régulièrement découvertes notamment avec l'arrivée de nouvelles*

*technologies comme HTML5. Il est par exemple possible de rediriger vers un autre site pour de l'hameçonnage ou encore de voler la session en récupérant les cookies.*

Source Wikipédia : [https://fr.wikipedia.org/wiki/Cross-site\\_scripting](https://fr.wikipedia.org/wiki/Cross-site_scripting)

Un peu plus bas, dans le même article, il y a un paragraphe qui nous intéresse car il explique comment se protéger contre les attaques XSS, quand on travaille avec le langage PHP. Je le cite :

- utiliser la fonction `htmlspecialchars()` qui filtre les '<' et '>' (cf. ci-dessus) ;
- utiliser la fonction `htmlentities()` qui est identique à `htmlspecialchars()` sauf qu'elle filtre tous les caractères équivalents au codage HTML ou Javascript.
- utiliser `strip_tags()` qui supprime les balises.

Utiliser la fonction `htmlentities()` pour afficher dans une page HTML des données provenant de PHP est une bonne pratique que je recommande, nous allons en reparler dans un instant.

Je vous avoue que j'ai un faible pour la fonction `strip_tags()`, qui a pour effet de dégager les balises HTML qui pourraient être contenues dans des champs de saisie. Dans le cas de notre tentative d'attaque précédente, cela donne ceci :

Avant filtrage via <code>strip_tags()</code>	Après filtrage
<code>&lt;script&gt;alert('coucou');&lt;/script&gt;</code>	<code>alert('coucou');</code>

Débarassé de la balise « script », le contenu du champ de saisie devient inoffensif.

L'imagination des pirates informatiques est sans limites quand il s'agit de mettre un site en difficulté, aussi si on peut leur compliquer la tâche, il ne faut pas s'en priver. Dans notre cas, nous pouvons placer la fonction `strip_tags()` à au moins deux niveaux :

- dans la boucle d'analyse de `$_POST`, lors du chargement des tableaux `$erreurs` et `$gooddata` :

```
// Boucle pour analyser si des erreurs sont présentes
foreach($_POST as $keypost=>$valpost) {
    if ($keypost != 'valid') {
        $valpost = strip_tags($valpost);
        $valpost = trim($valpost);
        if ($valpost == '') {
            $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
        } else {
            $gooddata[$keypost] = $valpost;
        }
    }
}
```

- dans la fonction `get_form_input_value()`, juste avant le « return » :

```
function get_form_input_value($field, $method='post') {
    // suppression des blancs parasites et forçage des valeurs en minuscule
    $field = strtolower(trim($field));
    $method = strtolower(trim($method));

    if ($method == 'get' ) {
        $value = isset($_GET[$field]) ? $_GET[$field] : '';
    } else {
        $value = isset($_POST[$field]) ? $_POST[$field] : '';
    }
    $value = trim(strip_tags($value));
    return $value;
}
```

Et puisque l'utilisation de la fonction `htmlentities()` constitue une bonne pratique, nous allons la mettre en œuvre, mais je souhaiterais auparavant vous montrer en quoi elle est utile.

Prenons l'exemple de notre champ de saisie « search », son code HTML de base est le suivant :

```
<input type="search" name="search" id="search_field"
placeholder="saisissez un critère de recherche" value="">
```

On voit donc que les valeurs contenues dans l'attribut « value » sont circonscrites par des guillemets. Supposons qu'un utilisateur saisisse un guillemet dans ce même champ, et qu'un autre champ de saisie ait été oublié, le navigateur va boucler sur le formulaire, mais l'attribut « value » de notre champ « search » va rencontrer un problème :

```
<input type="search" name="search" id="search_field"
placeholder="saisissez un critère de recherche" value ">
```

Oups, c'est « pas glop » dirait le Pifou. Il y a manifestement un problème du côté de l'attribut « value ». Si on passe en mode « édition » sur ce même champ, ça sent carrément le roussi :

```
<input type="search" name="search" id="search_field"
placeholder="saisissez un critère de recherche" value="" "=">|
```

En fait, on ne s'en rend pas compte au premier coup d'œil, mais notre formulaire est en train de « partir en sucette », tout ça à cause d'un caractère mal échappé.

Et pourtant, c'est tellement simple de se prémunir contre ce genre d'embrouille, avec la fonction `htmlentities()`.

Si à la fin de la fonction `get_form_input_value()`, juste avant le « return », j'ajoute la ligne ci-dessous :

```
$value = htmlentities($value, ENT_QUOTES, "UTF-8");
```

... dans ce cas, les apostrophes et guillemets sont bien échappés, et tout rentre dans l'ordre :

```
<input type="search" name="search" id="search_field" placeholder="saisissez un critère de recherche" value="">
```

On peut d'ailleurs en profiter pour « blinder » le code du script « pagecible.php », histoire de pouvoir dormir sur nos deux oreilles :

```
<?php
// démarrage du gestionnaire de session
session_start();

echo '<br>Contenu de $_SESSION affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_SESSION as $postkey=>$postval) {
    $postkey = htmlentities($postkey, ENT_QUOTES, "UTF-8");
    $postval = htmlentities($postval, ENT_QUOTES, "UTF-8");
    echo "<li>{$postkey} => {$postval}</li>" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Si dans la vie de tous les jours, la paranoïa est plutôt un problème, en matière de développement logiciel, et en particulier de développement web, la paranoïa est une véritable qualité. Les fonctions `strip_tags()` et `htmlentities()` sont là pour nous aider, alors il ne faut surtout pas s'en priver.

Nous retrouverons d'ailleurs les fonctions `strip_tags()` et `htmlentities()` dans le chapitre suivant, dans lequel nous allons nous amuser à créer une fonction générique capable de générer pour nous un ou plusieurs champs de formulaire.

Il faut souligner que PHP s'est enrichi au fil du temps de fonctions dédiées au filtrage de données (avec possibilité de validation et de nettoyage). Ces techniques viennent compléter le panel de solutions présentées dans ce chapitre, elles sont décrites en annexe (cf. chapitre 5.5).

Ouf, pour une fois c'était un petit chapitre... mais un chapitre important !

## 4.4.5 Générateur de formulaire

### 4.4.5.1 Un générateur de champ « input »

Vous aurez sans doute remarqué, en créant des champs de formulaire dans les chapitres précédents, qu'il y a beaucoup de tâches répétitives dans la création d'un champ de formulaire. En effet, pour chaque champ que nous créons, nous devons mettre bout à bout des balises et des attributs HTML... à la longue cela devient vite barbant.

Avec toutes les connaissances que vous avez accumulées maintenant, vous devriez être en mesure de créer une fonction PHP capable de générer un champ de formulaire de ce type :

```
<input id="id_testchamp1" class="form-control"
name="testchamp1" value="toto@titi.fr" required=""
placeholder="saisissez une adresse email" type="email">
```

Prenez le temps de faire un petit inventaire des éléments qui composent ce champ de saisie :

- nous avons une balise « input »
- nous avons l'attribut « type », généralement fixé à « text » mais qui peut être différent si nous souhaitons générer des types différents comme le type « email » (dans notre exemple), le type « date », ou autre... Comme vous êtes des pros en HTML, vous savez aussi que, si on oublie de définir cet attribut, ou si on lui attribue une valeur erronée, c'est le type « text » qui sera retenu par le navigateur par défaut
- nous avons l'attribut « class », qui est facultatif, et qui est fixé ici à « form-control » parce que nous souhaitons bénéficier ici du rendu proposé par le framework CSS Bootstrap
- nous avons l'attribut « name », qui est facultatif (si notre formulaire est géré uniquement par du JS). Mais dans notre cas, nous voulons que notre formulaire fonctionne tout à fait normalement, et envoie ses données dans la variable PHP \$\_POST, donc en ce qui nous concerne l'attribut « name » est obligatoire
- Nous avons l'attribut « id » qui est facultatif. Il est surtout utile si on souhaite manipuler le DOM, encore qu'il soit possible de s'en passer et d'accéder à cet élément par d'autres moyens (il n'y a pas que la fonction getElementById() dans la vie). L'attribut « id » doit surtout être unique, et il est également utile dans notre cas, si nous voulons lier le champ de formulaire à une balise « label » (via l'attribut « for » de la balise « label »).
- L'attribut « required » est facultatif, je rappelle que s'il est défini il a pour effet de rendre notre champ obligatoire
- L'attribut « placeholder » est facultatif, je rappelle que s'il est défini il fournit une assistance à la saisie, si le champ de saisie est vide.

Bon, vous connaissiez tout ça, nous l'avions étudié dans les cours « HTML5 – Premiers pas », « CSS3 – Premiers pas », « Bootstrap – Premiers pas » et « Javascript – Premiers pas ». Mais je pense que c'était intéressant de rappeler ces éléments ici, et de constater que les connaissances accumulées au fil des semaines vont finalement nous être utiles pour créer un composant capable de faire le boulot à notre place.

Je dois dire que pour arriver à écrire un composant de ce type, quand c'est la première fois qu'on en écrit un, on tâtonne généralement beaucoup avant d'arriver à une version qui fonctionne, mais c'est un exercice très intéressant à réaliser. On apprend généralement beaucoup en faisant ce genre de travail. Si vous avez le temps, je vous encourage à tenter l'exercice, avant de lire la solution que je vous propose dans la suite de ce chapitre.

Il faut souligner aussi qu'il n'y a pas une seule manière de traiter ce sujet, on peut trouver différentes solutions, plus ou moins élégantes, plus ou moins efficaces.

La version que je vous propose ici est assez proche d'une version que j'avais réalisée avec des élèves, et elle n'est pas très éloignée d'un composant que j'ai utilisé sur certains projets.

```
/*
 * Fonction de génération d'une balise "input"
 * @param $type_champ
 * @param $nom_champ
 * @param string $valeur_champ
 * @param bool $required
 * @param string $placeholder
 * @return string
 */
function gen_form_input($type_champ, $nom_champ, $valeur_champ = '',
    $required = false, $placeholder = '') {
    if ($required === true) {
        $attribute_required = 'required';
    } else {
        $attribute_required = '';
    }
    $placeholder = trim(strip_tags($placeholder));
    if ($placeholder != '') {
        $placeholder = htmlentities($placeholder, ENT_QUOTES, 'UTF-8');
        $attribute_placeholder = 'placeholder="' . $placeholder . '"';
    } else {
        $attribute_placeholder = '';
    }
    if ($type_champ != 'hidden') {
        $attribute_class = 'class="form-control" ';
    } else {
        $attribute_class = '';
    }
    $nom_champ = trim(strip_tags($nom_champ));
    $valeur_champ = trim(strip_tags($valeur_champ));
    $valeur_champ = htmlentities($valeur_champ, ENT_QUOTES, 'UTF-8');
    return '<input type="'. $type_champ . '"'
        . 'name="'. $nom_champ . '"'
        . 'id="id_'. $nom_champ . '"'
        . 'value="'. $valeur_champ . '"'
        . $attribute_class. $attribute_required . ''
        . $attribute_placeholder . '/>' . PHP_EOL;
}
```

```
// Quelques tests :
echo gen_form_input('email', 'testchamp1', 'toto@titi.fr', true, 'saisissez une adresse email');
echo '<br>';
echo gen_form_input('text', 'testchamp2', '123', true, 'saisissez quelque chose');
echo '<br>';
echo gen_form_input('date', 'testchamp3', '', false, '');
echo '<br>';
echo gen_form_input('date', 'testchamp4');
echo '<br>';
```

Voici le code HTML généré par les 4 tests ci-dessus :

```
<input id="id_testchamp1" class="form-control"
name="testchamp1" value="toto@titi.fr" required=""
placeholder="saisissez une adresse email" type="email">
<br>
<input id="id_testchamp2" class="form-control"
name="testchamp2" value="123" required=""
placeholder="saisissez quelque chose" type="text">
<br>
<input id="id_testchamp3" class="form-control"
name="testchamp3" value="" type="date">
<br>
<input id="id_testchamp4" class="form-control"
name="testchamp4" value="" type="date">
.
```

Il est important de vérifier la qualité du code HTML produit, quand on crée un composant de ce type... et de multiplier les tests, bien au-delà du petit jeu d'essai ci-dessus.

J'ai mis peu de commentaires dans le code, à ce stade vous devez être en mesure de le comprendre, car je n'ai utilisé que des choses vues dans les précédents chapitres.

Quelques remarques s'imposent :

- J'ai usé et même abusé des fonctions `strip_tags()` et `htmlentities()`, en les appliquant au contenu des paramètres `$nom_champ`, `$valeur_champ` et `$placeholder`. C'est totalement volontaire, je vous rappelle que je suis parano 😊. Mon objectif ici, c'est de créer un composant le plus robuste possible.
  - o Comme je ne sais pas comment il va être utilisé, ni si les informations qui vont lui être transmises en paramètre sont correctement filtrées, je préfère leur appliquer la fonction `strip_tags()`.
  - o Comme je ne sais pas si les caractères tels que les apostrophes et surtout les guillemets – susceptibles de mettre « la grouille » dans mon code HTML – sont correctement échappés dans les paramètres `$valeur_champ` et `$placeholder`, je prends les devants en appliquant la fonction `htmlentities()`.

Une remarque plus générale par rapport au composant que nous venons de créer :

Quand on est un développeur débutant et qu'on s'attelle pour la première fois à un composant de ce type, on a souvent tendance à mettre des « echo » dans ce genre de composant. L'idée sous-jacente, c'est que le composant est dédié à la génération de code HTML, donc si on l'appelle, autant qu'il génère le code HTML par un « echo » plutôt qu'en renvoyant ce code HTML via le mot clé « return ».

Il s'agit d'une erreur courante que j'ai rencontrée à plusieurs reprises, il n'y a pas de honte à avoir ☺.

Mais il s'agit quand même d'une mauvaise idée, car dans la pratique on souhaite le plus souvent décorrérer la génération du code HTML (généré par le composant), de son envoi vers le buffer HTML.

Le fait de renvoyer le code HTML via le mot clé « return » nous permet de préparer ce code à l'avance, de le stocker, et de l'utiliser au moment opportun.

Il serait intéressant à ce stade de réfléchir à un composant, qui ferait fonctionnement la même chose, mais qui commencerait par générer les différents attributs dans un tableau PHP, pour ensuite exploiter ce tableau en fin de parcours, pour générer le code HTML final. Je vous laisse y réfléchir.

#### *4.4.5.2 Une première approche de la POO*

La POO, ou « Programmation Orientée Objet ».

Le composant que nous avons créé au chapitre précédent pourrait être le premier d'une série de composants dédiés à la génération de formulaires. En effet, il existe de nombreux types de champ de saisie, avec les listes déroulantes, les boutons-radios, les cases à cocher, les labels, les champs de type « textarea », etc...

Donc il semble logique de vouloir créer un panel de composants plus large que notre simple fonction `gen_form_input()`.

J'allais l'oublier, mais nous avons aussi des balises « input » de type « hidden » et « password », en plus des types « text » et des nouveaux types du HTML5 (« date », « search », « email », etc...).

En nous appuyant sur notre fonction `gen_form_input()`, nous pouvons créer un jeu de fonctions plus spécialisées, comme par exemple :

```
function gen_input_text($nom_champ, $valeur_champ = '',
                      $required = false, $placeholder = '') {
    return gen_form_input('text', $nom_champ, $valeur_champ,
                          $required, $placeholder);
}

function gen_input_hidden($nom_champ, $valeur_champ) {
    return gen_form_input('hidden', $nom_champ, $valeur_champ);
}

function gen_input_password($nom_champ, $valeur_champ = '',
                           $required = false, $placeholder = '') {
    return gen_form_input('password', $nom_champ, $valeur_champ,
                          $required, $placeholder);
}
```

Et ce n'est qu'un petit panel de ce que nous pouvons créer dans ce domaine.

Mais du coup, toutes ces petites fonctions autonomes... ça va vite devenir le bazar !

Il serait intéressant de les regrouper... par exemple dans une classe. En POO, une classe, c'est en quelque sorte le mode d'emploi – ou plus exactement le mode de fonctionnement - d'un objet. Une classe PHP se définit par le mot clé « `class` », suivi d'un nom, et d'accolades à l'intérieur desquelles nous pouvons placer nos différentes fonctions, qui pour le coup s'appellent des « méthodes » (en jargon POO).

Dans l'exemple ci-dessous, j'ai créé une classe que j'ai appelée `GenForm`, dans laquelle j'ai placé mes différentes fonctions (méthodes), j'ai retiré tout le gras, pour que l'on puisse se concentrer sur la structure de la classe :

```
class GenForm {
    public function gen_form_input($type_champ, $nom_champ, $valeur_champ = '',
                                   $required = false, $placeholder = '') {
        // code retiré pour alléger l'exemple
    }
    public function gen_input_text($nom_champ, $valeur_champ = '',
                                   $required = false, $placeholder = '') {
        return $this->gen_form_input('text', $nom_champ, $valeur_champ,
                                      $required, $placeholder);
    }
    public function gen_input_hidden($nom_champ, $valeur_champ) {
        return $this->gen_form_input('hidden', $nom_champ, $valeur_champ);
    }
    public function gen_input_password($nom_champ, $valeur_champ = '',
                                       $required = false, $placeholder = '') {
        return $this->gen_form_input('password', $nom_champ, $valeur_champ,
                                     $required, $placeholder);
    }
}
```

Vous voyez que l'on retrouve nos 4 fonctions, elles portent d'ailleurs toujours le nom de « function » malgré le fait qu'il s'agit de méthodes d'une classe. C'est une bizarrerie de PHP... no comment.

Quelques détails importants à noter :

- la présence du mot clé « public » devant le mot clé « function ». Ce mot clé indique que les fonctions sont visibles depuis l'extérieur de l'objet, on dit qu'elles sont « publiques ». Nous verrons dans un instant comment ça s'utilise
- la présence du mot clé « \$this » suivi d'une flèche, avant chaque appel de la méthode « gen\_form\_input ». Ce mot clé « \$this » indique que nous faisons appel à une méthode qui appartient à l'objet en cours. Par exemple, la méthode gen\_input\_text() appartient à un objet et utilise la méthode gen\_form\_input() qui appartient au même objet.

J'ai oublié de préciser, que pour que notre objet fonctionne, nous devons l'instancier de cette manière :

```
$objForm = new GenForm();
```

Eh oui, sans cette action d'instanciation, la classe GenForm est inutilisable. Je rappelle que ce n'est qu'une classe, pas un objet. Une classe c'est la notice descriptive du fonctionnement de l'objet, ce n'est pas l'objet lui-même. L'objet, c'est dans notre exemple la variable \$objForm. Voici quelques exemples d'utilisation de l'objet \$objForm, et surtout de ses méthodes publiques :

```
echo $objForm->gen_form_input('email', 'testchamp1', 'toto@titi.fr',
                                true, 'saisissez une adresse email');
echo '<br>';
echo $objForm->gen_input_text('testchamp2', '123', false,
                                'saisissez quelque chose');
echo '<br>';
echo $objForm->gen_input_hidden('testhidden', 'champ caché');
echo '<br>';
echo $objForm->gen_input_password('testpassword');
echo '<br>';
```

Le meilleur moyen de s'assurer du bon fonctionnement de ces méthodes, c'est d'observer le code HTML généré (via l'inspecteur d'élément du navigateur) :

```
<input type="email" name="testchamp1" id="id_testchamp1" value="toto@titi.fr"
class="form-control" required placeholder="saisissez une adresse email">
<br>
<input type="text" name="testchamp2" id="id_testchamp2" value="123" class="form-
control" required placeholder="saisissez quelque chose">
<br>
<input type="date" name="testchamp3" id="id_testchamp3" value class="form-control">
<br>
<input type="date" name="testchamp4" id="id_testchamp4" value class="form-control">
<br>
<input type="text" name="testchampx" id="id_testchampx" value class="form-control"
required placeholder="saisissez une \"adresse email">
<input type="text" name="testchampy" id="id_testchampy" value="toto@titi.fr" class=
"form-control" required>
```

Il y a encore plein de choses à voir concernant la POO, ce chapitre n'est qu'une courte introduction. Nous étudierons le sujet plus en détail dans un prochain chapitre, mais vous avez déjà acquis quelques connaissances essentielles sur...

- la notion de classe,
- la notion d'objet objet (et la manière dont on instancie un objet),
- la notion de méthode, et en particulier de méthode publique
- la manière dont une méthode peut utiliser une autre méthode du même objet (via le mot clé « \$this »)

Avec le recul que j'ai acquis au fur et à mesure des formations, je dirais que créer une classe contenant des méthodes dédiées à la génération de formulaire, c'est un excellent outil pédagogique pour appréhender la POO et commencer à s'approprier le sujet.

Mais il est temps de s'attaquer à un autre sujet, car je sais que vous avez hâte de découvrir de quelle manière PHP peut fonctionner avec une base SQL.

```

/**
 * Class GenForm
 * Regroupe des méthodes dédiées à la génération de champs de saisie :
 */
class GenForm
{

    /**
     * Fonction de génération d'une balise "input"
     * @param $type_champ
     * @param $nom_champ
     * @param string $valeur_champ
     * @param bool $required
     * @param string $placeholder
     * @return string
     */
    public function gen_form_input($type_champ, $nom_champ, $valeur_champ =
'', $required = false, $placeholder = '')
    {

        if ($required === true) {
            $attribute_required = 'required';
        } else {
            $attribute_required = '';
        }
        $placeholder = trim(strip_tags($placeholder));
        if ($placeholder != '') {
            $placeholder = htmlentities($placeholder, ENT_QUOTES, 'UTF-8');
            $attribute_placeholder = 'placeholder="' . $placeholder . '"';
        } else {
            $attribute_placeholder = '';
        }
        if ($type_champ != 'hidden') {
            $attribute_class = 'class="form-control" ';
        } else {
            $attribute_class = '';
        }
        $nom_champ = trim(strip_tags($nom_champ));
        $valeur_champ = trim(strip_tags($valeur_champ));
        $valeur_champ = htmlentities($valeur_champ, ENT_QUOTES, 'UTF-8');
        return '<input type="' . $type_champ . '"'
            . 'name="' . $nom_champ . '"'
            . 'id="id_' . $nom_champ . '"'
            . 'value="' . $valeur_champ . '"'
            . $attribute_class . $attribute_required . ''
            . $attribute_placeholder . '/>' . PHP_EOL;
    }

    public function gen_input_text($nom_champ, $valeur_champ = '',
                                    $required = false, $placeholder = '')
    {
        return $this->gen_form_input('text', $nom_champ, $valeur_champ,

```

```
$required, $placeholder);  
}  
  
    public function gen_input_hidden($nom_champ, $valeur_champ)  
    {  
        return $this->gen_form_input('hidden', $nom_champ, $valeur_champ);  
    }  
  
    public function gen_input_password($nom_champ, $valeur_champ = '',  
                                      $required = false, $placeholder = '')  
    {  
        return $this->gen_form_input('password', $nom_champ, $valeur_champ,  
$required, $placeholder);  
    }  
  
}  
  
$objForm = new GenForm();  
  
echo $objForm->gen_form_input('email', 'testchamp1', 'toto@titi.fr', true,  
'saisissez une adresse email');  
echo '<br>';  
echo $objForm->gen_input_text('testchamp2', '123', false, 'saisissez quelque  
chose');  
echo '<br>';  
echo $objForm->gen_input_hidden('testhidden', 'champ caché');  
echo '<br>';  
echo $objForm->gen_input_password('testpassword');  
echo '<br>';
```



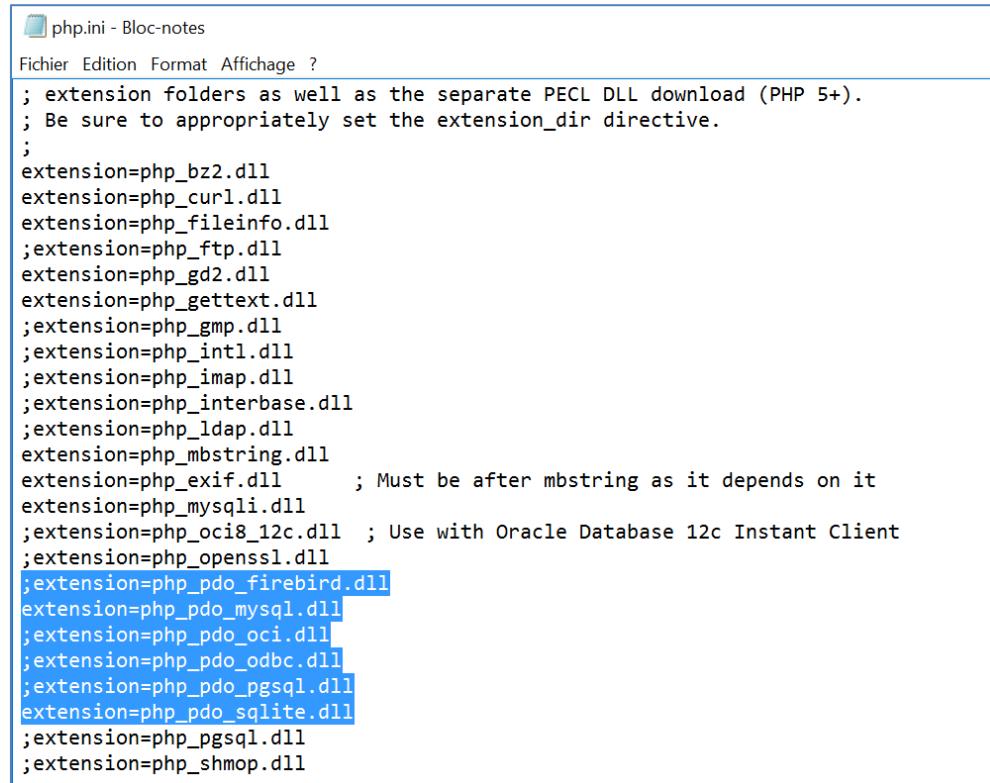
## 4.5 PHP et le SQL

### 4.5.1 Présentation de PDO

#### PDO = PHP Data Object

PDO est une extension PHP qui fournit une interface permettant de se connecter à différentes bases de données au travers d'un même jeu de fonctions PHP, là où auparavant chaque base de données nécessitait un driver et un jeu de fonctions PHP spécifiques.

Pour que PDO fonctionne avec une base de données particulière, il faut que l'extension correspondante soit activée dans le fichier de configuration php.ini de votre stack PHP :



```
php.ini - Bloc-notes
Fichier Édition Format Affichage ?
; extension folders as well as the separate PECL DLL download (PHP 5+).
; Be sure to appropriately set the extension_dir directive.
;
extension=php_bz2.dll
extension=php_curl.dll
extension=php_fileinfo.dll
;extension=php_ftp.dll
extension=php_gd2.dll
extension=php_gettext.dll
;extension=php_gmp.dll
;extension=php_intl.dll
;extension=php_imap.dll
;extension=php_interbase.dll
;extension=php_ldap.dll
extension=php_mbstring.dll
extension=php_exif.dll      ; Must be after mbstring as it depends on it
extension=php_mysqli.dll
;extension=php_occi8_12c.dll ; Use with Oracle Database 12c Instant Client
;extension=php_openssl.dll
;extension=php_pdo_firebird.dll
extension=php_pdo_mysql.dll
;extension=php_pdo OCI.dll
;extension=php_pdo_odbc.dll
;extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
;extension=php_pgsql.dll
;extension=php_shmop.dll
```

Dans l'exemple ci-dessus, les extensions actives sont celles qui n'ont pas de point virgule en première position, comme par exemple l'extension « `php_pdo_mysql` », qui est active par défaut.

Pour accéder à une base MySQL, on pourrait utiliser l'extension « `php_mysqli` » plutôt que PDO, mais PDO est beaucoup plus pratique, aussi je ne parlerai pas ici de l'extension « `php_mysqli` ».

Documentation officielle sur PDO :

[V](#)

## 4.5.2 Premiers exemples avec PDO

Deux syntaxes déclaratives permettant de faire la même chose :

- syntaxe n°1 :

```
<?php
// MySQL expects parameters in the string
$user = "root";
$password = "";
$db = new PDO('mysql:host=localhost;dbname=nomdelabase', $user, $password);

$stmt = $db->query('SELECT id, nom_categ_fr FROM produit_categorie');
foreach ($stmt->fetchAll() as $row) {
    print "id = {$row['id']} , nom = {$row['nom_categ_fr']} <br/>\n";
}
```

- syntaxe n°2 :

```
<?php
//paramètres de connexion
$host = 'localhost'; // le chemin vers le serveur
$db = 'nomdelabase'; // le nom de la base de donnée
$user = 'root'; // nom d'utilisateur pour se connecter
$pwd = ''; // mot de passe de l'utilisateur pour se connecter
$dsn = "mysql:dbname=$db;host=$host"; //creation du dsn pour mysql

//utilisation de la base
try {
    //Ouverture de l'accès à la base
    $pdo= new PDO($dsn, $user, $pwd);
    //echo "connectee a la base";
} catch(PDOException $e){
    echo 'Erreur : '.$e->getMessage().'  
';
    echo 'N° : '.$e->getCode();
    die('Echec de la connexion : '.$e->getMessage());
} ;

$stmt = $pdo->query('SELECT id, nom_categ_fr FROM produit_categorie');
foreach ($stmt->fetchAll() as $row) {
    print "id = {$row['id']} , nom = {$row['nom_categ_fr']} <br/>\n";
}
```

=> Résultat obtenu dans le navigateur (identique dans les 2 syntaxes) :

```
id = 1 , nom = Distributeurs Anti-limaces
id = 2 , nom = Spécial déchaumeur
id = 3 , nom = Saleuses électriques
id = 4 , nom = Microgranulateur électrique
id = 5 , nom = Semoir à engrais
id = 6 , nom = Agrainoir mobile
id = 0 , nom = _Sans Catégorie
id = 13 , nom = Doseur électrique
```

A noter : avec MySQL, pour récupérer l'identifiant d'une ligne que l'on vient d'insérer, on utilise la technique suivante :

```
$sql = "SELECT LAST_INSERT_ID() AS lastid ";
$stmt2 = $conn->query($sql);
$row = $stmt2->fetch(PDO::FETCH_LAZY);
echo 'ID of last insert: ', $row->lastid;
```

#### 4.5.3 Les différents modes Fetch de PDO

**La méthode FetchAll()** permet de renvoyer un jeu de données complet en une seule passe (cf. exemple en début de chapitre).

**La méthode Fetch()** permet d'adresser chaque enregistrement individuellement :

```
<?php

$rows = $db->query('SELECT symbol,planet FROM zodiac');

$firstRow = $rows->fetch();

print "The first results are that {$row['symbol']} goes with
{$row['planet']}";

?>
```

Fetch() et FetchAll() acceptent des paramètres complémentaires qui sont :

Constante	Format de ligne
PDO::FETCH_NUM	Tableau avec clés de type numérique (en fait les numéros de colonnes du Result Set)
PDO::FETCH_ASSOC	Tableau avec clés de type chaîne (en fait les noms de colonnes du Result Set)
PDO::FETCH_BOTH	Format par défaut si rien n'est précisé. Mixe les constantes PDO::FETCH_NUM et PDO::FETCH_ASSOC
PDO::FETCH_OBJ	Objet de classe <code>stdClass</code> avec noms de colonnes repris en noms de propriétés.
PDO::FETCH_LAZY	Object de classe <code>PDORow</code> avec noms de colonnes repris en noms de propriétés. Les propriétés ne sont générées que si elles font du Result Set, toute propriété qui est addressée alors qu'elle n'est pas encore créée déclenche une erreur. C'est donc un bon choix si la ligne du Result Set contient beaucoup de colonnes.
PDO::FETCH_BOUND	Le mode « fetch » permet de définir des variables dont les valeurs sont regénérées à chaque nouvel appel de <code>fetch()</code>
PDO::FETCH_INTO PDO::FETCH_CLASS	Permet de placer les lignes du Result Set dans des objets spécialisés de classes particulières. Pour employer ces modes, il faut au préalable créer une classe qui prolonge la classe intégrée de <code>PDOStatement</code>

Exemples d'utilisation :

- PDO::FETCH\_BOUND

```
$row = $db->query('SELECT symbol,planet FROM zodiac', PDO::FETCH_BOUND);
// Put the value of the 'symbol' column in $symbol
$row->bindColumn('symbol', $symbol);
// Put the value of the second column ('planet') in $planet
$row->bindColumn(2, $planet);
while ($row->fetch()) {
    print "$symbol goes with $planet. <br/>\n";
}
```

- PDO::FETCH\_INTO

```
class AvgStatement extends PDOStatement {
    public function avg() {
        $sum = 0;
        $vars = get_object_vars($this);
        // Remove PDOStatement's built-in 'queryString' variable
        unset($vars['queryString']);
        foreach ($vars as $var => $value) {
            $sum += strlen($value);
        }
        return $sum / count($vars);
    }
}
$row = new AvgStatement;
$results = $db->query('SELECT symbol,planet FROM zodiac', PDO::FETCH_INTO,
$row);
// Each time fetch() is called, $row is repopulated
while ($results->fetch()) {
    print "$row->symbol belongs to $row->planet (Average: {$row->avg()})<br/>\n";
}
```

**Utiliser PDO::exec() pour envoyer un INSERT, un DELETE, ou un UPDATE, comme dans les exemple suivants :**

```
$db->exec("INSERT INTO family (id,name) VALUES (1,'Vito')");
$db->exec("DELETE FROM family WHERE name LIKE 'Fredo'");
$db->exec("UPDATE family SET is_naive = 1 WHERE name LIKE 'Kay'");
```

**Utiliser `PDO::prepare()` et `PDOStatement::execute()` pour préparer et exécuter une requête en 2 temps :**

```
$st = $db->prepare('INSERT INTO family (id,name) VALUES (?,?)');
$st->execute(array(1, 'Vito'));

$st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));

$st = $db->prepare('UPDATE family SET is_naive = ? WHERE name LIKE ?');
$st->execute(array(1, 'Kay'));
```

La méthode `exec()` renvoie de la base de données ce que cette dernière lui transmet. Pour `INSERT`, `UPDATE` et `DELETE`, elle renvoie le nombre de lignes affectées par la requête.

Les méthodes `prepare()` et `execute()` sont utiles pour les requêtes qui doivent être exécutées plusieurs fois. On peut préparer une requête une fois, et l'exécuter plusieurs fois avec des paramètres différents :

```
$st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));
$st->execute(array('Sonny'));
$st->execute(array('Luca Brasi'));
```

Autre exemple d'utilisation :

```
// Prepare
$st = $db->prepare("SELECT sign FROM zodiac WHERE element LIKE ?");
// Execute once
$st->execute(array('fire'));
while ($row = $st->fetch()) {
    print $row[0] . "<br/>\n";
}
// Execute again
$st->execute(array('water'));
while ($row = $st->fetch()) {
    print $row[0] . "<br/>\n";
}
```

Exemple d'utilisation avec **plusieurs paramètres marqueurs** :

```
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element LIKE ? OR planet LIKE ?");

// SELECT sign FROM zodiac WHERE element LIKE 'earth' OR planet LIKE 'Mars'
$st->execute(array('earth', 'Mars'));
$row = $st->fetch();
```

Exemple d'utilisation avec **plusieurs paramètres nommés** :

```
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element LIKE :element OR planet LIKE
:planet");
// SELECT sign FROM zodiac WHERE element LIKE 'earth' OR planet LIKE 'Mars'
$st->execute(array('planet' => 'Mars', 'element' => 'earth'));
$row = $st->fetch();
```

Exemple d'utilisation avec **la méthode BindParam** :

```
$pairs = array('Mars' => 'water',
    'Moon' => 'water',
    'Sun' => 'fire');
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element LIKE :element AND planet LIKE
:planet");
$st->bindParam(':element', $element);
$st->bindParam(':planet', $planet);
foreach ($pairs as $planet => $element) {
    // No need to pass anything to execute() --
    // the values come from $element and $planet
    $st->execute();
    var_dump($st->fetch());
}
```

**Comptage de lignes avec la fonction RowCount()** :

- pour les INSERT, UPDATE, et DELETE exécutés avec la fonction exec(), la valeur renvoyée par cette fonction contient le nombre de lignes modifiées (cf. exemple de la page précédente).
- pour les INSERT, UPDATE, et DELETE exécutés avec PDO::prepare( ) et PDOStatement::execute( ), l'appel de la fonction PDOStatement::rowCount( ) permet de récupérer le nombre de lignes modifiées :

```
$st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));
print "Deleted rows: " . $st->rowCount();
$st->execute(array('Sonny'));
print "Deleted rows: " . $st->rowCount();
$st->execute(array('Luca Brasi'));
print "Deleted rows: " . $st->rowCount();
```

- comptage du nombre de lignes renvoyées par un SELECT (on peut préférer un COUNT(\*) à la méthode ci-dessous) :

```
$st = $db->query('SELECT symbol,planet FROM zodiac');
$all= $st->fetchAll(PDO::FETCH_COLUMN, 1);
print "Retrieved ". count($all) . " rows";
```

Articles sur les différents modes de consultation (fetch) de PDO :

<https://www.cloudconnected.fr/2007/10/10/les-fetch-modes-de-pdo/>

<https://www.cloudconnected.fr/2007/10/12/les-fetch-modes-de-pdo-2-les-modes-orientes-objet/>

#### 4.5.4 SQL et sécurité

La technique dite des « requêtes SQL paramétrées » constitue le socle que tout développeur SQL doit connaître pour produire un développement de qualité.

Les requêtes SQL paramétrées offrent tous les avantages :

- Sécurité : car elles protègent contre les attaques dites « par injection SQL »
- Souplesse d'écriture : nul besoin avec cette technique de concaténer laborieusement les paramètres à l'intérieur des requêtes SQL
- Performances : une requête paramétrée, si elle est exécutée plusieurs fois avec des valeurs différentes, va bénéficier du plan d'accès déterminé par le moteur SQL lors de la première exécution de la requête. Les exécutions suivantes d'une même requête seront dès lors beaucoup plus rapides (le chemin d'accès n'étant calculé qu'une fois).

La technique de l'attaque par injection SQL a été décrite dans de nombreux articles disponibles sur internet. Nous ne l'approfondirons pas ici, faute de temps, aussi je vous recommande la lecture des articles suivants :

- Wikipédia : [http://fr.wikipedia.org/wiki/Injection\\_SQL](http://fr.wikipedia.org/wiki/Injection_SQL)
- PHP Security Consortium :
  1. <http://phpsec.org/library/>
  2. <http://unixwiz.net/techtips/sql-injection.html>

Voici un exemple de script utilisant une requête SQL paramétrée :

```
<?php
//paramètres de connexion
$host  = 'localhost'; // le chemin vers le serveur
$db    = 'nomdelabase'; // le nom de la base de donnée
$user  = 'root'; // nom d'utilisateur pour se connecter
$pwd   = ''; // mot de passe de l'utilisateur pour se connecter
$dsn   = "mysql:dbname=$db;host=$host"; //creation du dsn pour mysql

//utilisation de la base
try {
    //Ouverture de l'accès à la base
    $db= new PDO($dsn, $user, $pwd);
    //echo"connectee a la base";
} catch(PDOException $e) {
    echo 'Erreur : '.$e->getMessage(). '<br />';
    echo 'N° : '.$e->getCode();
    die('Echec de la connexion : '.$e->getMessage());
} ;

// requête SQL paramétrée avec deux "jokers" matérialisés par les ?
$sql = 'SELECT id, nom_categ_fr FROM produit_categorie where id >= ? and id <= ?';
$params = array(1, 100);
try {
    $st = $db->prepare($sql); // préparation d'un "statement" à partir de la requête
    $ok = $st->execute($params); // exécution de la requête avec ses 2 paramètres
    if ($ok) {
        $row = $st->fetch(PDO::FETCH_ASSOC);
        while ($row != false) {
            print "id = {$row['id']} , nom = {$row['nom_categ_fr']} <br>".PHP_EOL;
            $row = $st->fetch(PDO::FETCH_ASSOC);
        }
    }
    unset($st);
} catch (PDOException $exc) {
    $stab_log = [];
    $stab_log ['ErrMsg'] = $exc->getMessage();
    $stab_log ['Trace'] = $exc->getTraceAsString();
    $stab_log ['Code'] = $exc->getCode();
    $stab_log ['File'] = $exc->getFile();
    $stab_log ['Line'] = $exc->getLine();
    $stab_log ['SQL_query'] = $sql;
    $stab_log ['SQL_args'] = var_export($params, true);
    foreach($stab_log as $key=>$value) {
        error_log($key . ' => ' . $value);
    }
}
}
```

## 4.5.5 Connexion à différents SGBD

PDO permet d'accéder à différentes bases de données :

```
<?php
// MySQL reçoit ses paramètres de connexion dans une chaîne appelée DSN (Data
Source Name)
// les paramètres sont séparés par des points virgules (;)
$mysql = new PDO('mysql:host=db.example.com;port=31075;dbname=food', $user,
$password);

// Connexion à un serveur MySQL local sur Linux
$mysql = new PDO('mysql:unix_socket=/tmp/mysql.sock', $user, $password)

// PostgreSQL reçoit aussi ses paramètres de connexion dans une chaîne mais
séparés par un blanc
$pgsql = new PDO('pgsql:host=db.example.com port=31075 dbname=food', $user,
$password);

// On peut placer "user" et "password" dans la DSN
$pgsql = new PDO("pgsql:host=db.example.com port=31075 dbname=food user=$user
password=$password");

// Oracle
// Si un nom de database est défini dans tnsnames.ora, on l'indique dans le
DSN
$oci = new PDO('oci:food', $user, $password);
// dans le cas contraire, on définit un "Instant Client URI"
$oci = new PDO('oci:dbname://db.example.com:1521/food', $user, $password);

// Sybase (si PDO utilise FreeTDS)
$sybase = new PDO('sybase:host=db.example.com;dbname=food', $user, $password);
// Microsoft SQL Server (si PDO utilise les librairies MS SQL Server)
$mssql = new PDO('mssql:host=db.example.com;dbname=food', $user, $password);

// ODBC avec une connexion prédéfinie
$odbc = new PDO('odbc:DSN=food');
// ODBC avec une connexion définie dans la DSN (nécessite la déclaration d'un
driver)
$odbc = new PDO('odbc:Driver={Microsoft Access Driver
(*.mdb)};DBQ=C:\\data\\food.mdb;Uid=Chef');

// SQLite attend un simple nom de fichier (pas de "user" ni de "password")
$sqlite = new PDO('sqlite:/usr/local/zodiac.db');
$sqlite = new PDO('sqlite:c:/data/zodiac.db');

// SQLite peut aussi manipuler en mémoire des base temporaires
$sqlite = new PDO('sqlite::memory:');
// SQLite DSN (v2 et v3)
$sqlite2 = new PDO('sqlite2:/usr/local/old-zodiac.db');
```

**Problème de socket lors de la connexion à MySQL avec PDO sous Linux :**

En environnement LAMP, il arrive – y compris avec Zend Server – de rencontrer l'erreur suivante lors de la connexion :

Fatal error: Uncaught exception 'PDOException' with message 'SQLSTATE[HY000] [2002]

[SQL Errcode 2002] SQLSTATE[HY000] [2002] Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)

Cela signifie que le connecteur a besoin qu'on lui précise au niveau du DSN, soit l'adresse TCP/IP (solution 1), soit l'emplacement du socket Linux pour MySQL (solution 2) :

1: "mysql:host=127.0.0.1" or "mysql:host=localhost;port=3306"

2: "mysql:unix\_socket=/var/run/mysqld/mysqld.sock"

A noter que la solution n°2 fonctionne bien sous Zend Server.

Une troisième solution a été évoquée dans les forums, qui n'a pas semblé fonctionner avec Zend Server, mais je l'ai quand même relevée ci-dessous, pour mémoire : elle consiste à modifier le fichier php.ini en spécifiant la ligne suivante (exemple fourni pour XAMPP) :

pdo\_mysql.default\_socket = /opt/lampp/var/mysql/mysql.sock

En environnement Zend Server, la ligne à insérer serait plutôt :

pdo\_mysql.default\_socket = /var/run/mysqld/mysqld.sock

Il ne faut pas oublier de redémarrer Apache après toute modification de php.ini.

Présentation exhaustive des paramètres de connexion ODBC, classés par bases de données :

<https://www.connectionstrings.com/>

## **4.6 Les objets en PHP**

TODO : chapitre en cours d'écriture

## **4.7 Principles d'architecture**

TODO : chapitre en cours d'écriture

Principe des contrôleurs à détailler.

Architecture MVC

Référence intéressante :

<http://bpesquet.developpez.com/tutoriels/introduction-genie-logiciel/>

## 4.8 Les fichiers

### 4.8.1 Les fichiers texte

Ouverture/Lecture/Mise à jour/Fermeture de fichiers

Exemples d'ouverture de fichier :

```
// ouvrir "filename" en lecture
$filehandle = fopen("filename", "r");
// ouvrir "filename" en écriture
$filehandle = fopen("filename", "w");
// ouvrir "filename" en écriture binaire
$filehandle = fopen("filename", "wb");
```

Fermeture d'un fichier :

```
fclose($filehandle);
```

Lire un fichier :

```
// Lit une ligne du fichier lié à $filehandle
$string = fgets($filehandle);
// Lit un caractère du fichier lié à $filehandle
$char = fgetc($filehandle);
// Lit un morceau d'octets du fichier lié à $filehandle
$text = fread($filehandle, $bytes );
// Lit l'intégralité du contenu du fichier "filename"
$text = file_get_contents("filename");
```

Ecrire dans un fichier :

```
// Ecrit $string dans le fichier lié à $filehandle
fwrite($filehandle, $string);
// Ecrit $string dans le fichier "filename"
file_put_contents("filename", $string);
```

On peut utiliser la fonction file\_get\_contents () avec la fonction explode(), pour charger un tableau à partir d'un fichier texte :

```
$settings = explode("\n", file_get_contents('scenario.settings.txt'));
```

On étudiera un exemple d'utilisation de certaines de ces fonctions, au chapitre suivant traitant des fichiers CSV.

## 4.8.2 Les fichiers CSV

Exemples de manipulation de fichiers CSV :

```
/*
 * copie d'un CSV dans un autre
*/
// forçage d'une durée limite d'exécution supérieure à la limite standard
set_time_limit ( 360 );

$repert = dirname ( __FILE__ );

$fichier_input = $repert . '/csvpays.csv';
$fichier_output = $repert . '/csvpays2.csv';

$file_in = @fopen ( $fichier_input, "r" );
$file_out = @fopen ( $fichier_output, "w" );

if (is_resource ( $file_in ) && is_resource ( $file_out )) {
    while ( ! feof ( $file_in ) ) {
        $fields = fgetcsv ( $file_in, 0, ',', '"' );
        if (is_array ( $fields )) {
            fputcsv ( $file_out, $fields, ';', '"' );
        }
    }
}
if (is_resource ( $file_in )) {
    fclose ( $file_in );
}
if (is_resource ( $file_out )) {
    fclose ( $file_out );
}
```

## 4.8.3 Les fichiers XML

### 4.8.3.1 Générer du XML

Exemple de création d'un menu en XML à partir d'un tableau PHP :

```

header('Content-Type: text/xml');
print '<?xml version="1.0"?>' . "\n";
print "<menus>\n";

$menus = array(
    "home" => array("id"=>"home", "parent"=>"home", "title"=>"Home",
        "url"=>"index.php" ),
    "contact" => array("id"=>"contact", "parent"=>"home", "title"=>"Contact",
        "url"=>"contact.php" ),
    "services" => array("id"=>"services", "parent"=>"home", "title"=>"Services",
        "url"=>"services.php" ),
    "colpicker" => array("id"=>"colpicker", "parent"=>"services",
    "title"=>"ColorPicker",
        "url"=>"colorpicker.php" ),
    "calendrier" => array("id"=>"calendrier", "parent"=>"services",
    "title"=>"Calendrier",
        "url"=>"calendrier.php" ),
    "dragdroplist" => array("id"=>"dragdroplist", "parent"=>"services",
    "title"=>"DragDropList",
        "url"=>"./dragdroplist/index.html" ),
    "ddstickies" => array("id"=>"ddstickies", "parent"=>"services",
    "title"=>"DropDownStickies",
        "url"=>"./dropdiv/index.php" ),
    "linkgraph" => array("id"=>"linkgraph", "parent"=>"services", "title"=>"LinkGraph",
        "url"=>"./linkgraph/linkgraph.php" ),
    "vectorgraph" => array("id"=>"vectorgraph", "parent"=>"services",
    "title"=>"VectorGraph",
        "url"=>"./vector/index.php" ),
    "slideshow" => array("id"=>"slideshow", "parent"=>"services", "title"=>"Slideshow",
        "url"=>"./slideshow/index.php" )
) ;

foreach ($menus as $menu) {
    $wi = 1;

    foreach($menu as $tag => $data) {
        if ($wi == 1) {
            print '      <menu id="' . htmlspecialchars($data) . '">' . "\n";
            $wi = 2;
        }
        print "          <$tag>" . htmlspecialchars($data) . "</{$tag}>\n";
    }
    print "      </menu>\n";
}

print "</menus>\n";

```

Dans l'exemple ci-dessus, le XML a été créé "manuellement" par concaténation de différentes chaînes de caractères. Cette solution nécessite d'être très vigilant dans l'écriture du code, car il est facile de produire un code XML non conforme. Pour éviter cela, on peut recourir à la classe PHP DomDocument, comme dans l'exemple ci-dessous :

```
$dom = new DomDocument('1.0', 'UTF-8');
$dom->formatOutput = true;

$root = $dom->createElement( "menus" );
$dom->appendChild( $root );

foreach( $menus as $menu )
{
    $bn = $dom->createElement( "menu" );
    $bn->setAttribute( 'id', $menu['id'] );

    $parent = $dom->createElement( "id" );
    $parent->appendChild( $dom->createTextNode( $menu['id'] ) );
    $bn->appendChild( $parent );

    $parent = $dom->createElement( "parent" );
    $parent->appendChild( $dom->createTextNode( $menu['parent'] ) );
    $bn->appendChild( $parent );

    $title = $dom->createElement( "title" );
    $title->appendChild( $dom->createTextNode( $menu['title'] ) );
    $bn->appendChild( $title );

    $title = $dom->createElement( "url" );
    $title->appendChild( $dom->createTextNode( $menu['url'] ) );
    $bn->appendChild( $title );

    $root->appendChild( $bn );
}

header( "Content-type: text/xml" );
echo $dom->saveXML();
```

#### 4.8.3.2 Lire du XML

Pour lire un petit fichier XML, la technique la plus simple consiste à utiliser la fonction PHP `simplexml_load_file()`, comme dans l'exemple suivant :

```
$url = 'menus.xml';
$menus = simplexml_load_file($url);
print '<ul>';
foreach ($menus->menu as $item) {
    print '<li><a href="' . htmlentities($item->id) . '">' .
        htmlentities($item->title) . '</a></li>';
}
print '</ul>';
```

On peut aussi recourir à la classe `DomDocument` :

```
$url = 'menus.xml';

$dom = new DOMDocument;
$dom->load($url);

foreach ($dom->getElementsByTagName('menu') as $menu) {
    // childNodes holds the author values
    $text_nodes = $menu->childNodes;

    $button = array();
    foreach ($text_nodes as $text) {
        $button []= array $text->nodeName => $text->nodeValue) ;
    }
    print_r($button); echo "<br/>" ;
}
```

Pour aider à mieux comprendre le code ci-dessus, voici un échantillon du fichier XML utilisé dans l'exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<menus>
    <menu id="home">
        <id>home</id>
        <parent>home</parent>
        <title>Accueil</title>
        <url>index.php</url>
    </menu>
    <menu id="applications">
        <id>applications</id>
        <parent>home</parent>
```

```

<title>Applications</title>
<url>mnu_applications.php</url>
</menu>
...
</menus>
```

La fonction `simplexml_load_file()` est bien adaptée à la lecture de fichier XML de petite taille, mais la lecture de fichiers XML de grande taille nécessite l'emploi d'un parser SAX, comme dans l'exemple suivant :

Echantillon du fichier XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
<book>
<author>Jack Herrington</author>
<title>PHP Hacks</title>
<publisher>O'Reilly</publisher>
</book>
<book>
<author>Jack Herrington</author>
<title>Podcasting Hacks</title>
<publisher>O'Reilly</publisher>
</book>
</books>
```

Code source PHP utilisant un parser SAX pour la lecture du fichier XML ci-dessus :

```

/**
 * Exemple de Jack Herrington pris sur le site :
 * http://www.ibm.com/developerworksopensource/library/os-xmldomphp/
 */
$g_books = array ();
$g_elem = null;

function startElement($parser, $name, $attrs) {
    global $g_books, $g_elem;
    if ($name == 'BOOK')
        $g_books [] = array ();
    $g_elem = $name;
}

function endElement($parser, $name) {
    global $g_elem;
    $g_elem = null;
}

function textData($parser, $text) {
```

```
global $g_books, $g_elem;
if ($g_elem == 'AUTHOR' || $g_elem == 'PUBLISHER' || $g_elem == 'TITLE') {
    $g_books [count ( $g_books ) - 1] [$g_elem] = $text;
}
}

$parser = xml_parser_create ();

xml_set_element_handler ( $parser, "startElement", "endElement" );
xml_set_character_data_handler ( $parser, "textData" );

$f = fopen ( 'books.xml', 'r' );

while ( $data = fread ( $f, 4096 ) ) {
    xml_parse ( $parser, $data );
}

xml_parser_free ( $parser );

foreach ( $g_books as $book ) {
    echo $book ['TITLE'] . " - " . $book ['AUTHOR'] . " - ";
    echo $book ['PUBLISHER'] . "\n";
}
```

On recommandera la lecture de l'excellent dossier de Jack Herrington sur la manipulation de fichier XML en PHP :

<http://www.ibm.com/developerworksopensource/library/os-xmldomphp/>

## 4.9 Le CRUD

CRUD est l'acronyme de « Create Retrieve Update Delete », soit les 4 opérations de base que l'on applique – en mode administration – à une entité de base de données.

TODO : chapitre en cours d'écriture

## 5 Annexe

### 5.1 Opérateurs

PHP fournit un grand nombre d'opérateurs, qui sont décrits sur la page suivante :  
<http://fr.php.net/manual/fr/language.operators.php>

Extrait de la page ci-dessus :

#### Sommaire

- [La priorité des opérateurs](#)
- [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](#)

Un opérateur est quelque chose qui prend une ou plusieurs valeurs (ou expressions, dans le jargon de la programmation) et qui retourne une autre valeur (donc la construction elle-même devient une expression).

Vous trouverez dans la suite de ce chapitre des tableaux présentant quelques séries d'opérateurs qu'il est indispensable de connaître.

Description	Symbole	Exemple
Renvoie "true" si les valeurs \$a et \$b sont égales	Egaux (=)	\$a == \$b
Renvoie "true" si les valeurs \$a et \$b sont égales et de même type	Identiques (==>)	\$a === \$b
Renvoie "true" si les valeurs \$a et \$b sont différentes	Différents (<>)	\$a != \$b or \$a >> \$b
Renvoie "true" si les valeurs \$a et \$b sont différentes, ou de type différent	Pas identiques (!==)	\$a !== \$b
Renvoie "true" si la première valeur est inférieure à la seconde	Inférieur à (<)	\$a < \$b
Renvoie "true" si la première valeur est supérieure à la seconde	Supérieur à (>)	\$a > \$b
Renvoie "true" si la première valeur est inférieure ou égale à la seconde	Inférieur ou égal à (<=)	\$a <= \$b
Renvoie "true" si la première valeur est supérieure ou égale à la seconde	Supérieur ou égal à (>=)	\$a >= \$b

Cette distinction entre « Egaux » et « Identiques » peut sembler étrange. Pour la comprendre, un exemple s'impose :

```
$val1 = 123;
$val2 = '123';

echo 'test1'.PHP_EOL ;
if ($val1 == $val2) {
    echo "les contenus sont similaires mais pas de même type,";
    echo "le double égal renvoie 'true' dans ce cas".PHP_EOL;
} else {
    echo "ce message ne s'affichera pas";
}

echo 'test2'.PHP_EOL ;
if ($val1 === $val2) {
    echo "ce message ne s'affichera pas";
} else {
    echo "les contenus sont similaires mais pas de même type,";
    echo "le triple égal renvoie 'false' dans ce cas".PHP_EOL;
}
```

Ce qu'il faut comprendre, c'est que dans le cas du double égal, PHP convertit la valeur alphanumérique de la variable \$val2 dans le même type que celui de la variable \$val1. Si après cette conversion, le contenu des variables \$val1 et \$val2 est bien identique, alors la condition renvoie « true ».

On peut combiner plusieurs conditions dans un même test, ce petit tableau présente les opérateurs permettant de combiner les conditions :

Description	Symbol	Exemple
Renvoie "true" si les 2 évaluations renvoient "true"	and	\$a and \$b
Retourner vrai si l'une ou l'autre évaluation est vraie ou si les deux sont vraies.	or	\$a or \$b
Retournez vrai si l'une ou l'autre évaluation est vraie, mais pas les deux.	xor	\$a xor \$b
Renvoie "true" si l'évaluation renvoie "false" (lire "non vrai")	!	!\$a
identique à "and" (et plus utilisé que « and »)	&&	\$a && \$b
identique à "or" (et plus utilisé que « or »)		\$a    \$b

Quelquefois, on peut rencontrer des conditions complexes, nécessitant de définir des priorités dans les conditions, dès lors l'ajout de parenthèses peut se révéler nécessaire, voire indispensable. Amusez-vous à tester et modifier les tests ci-dessous :

```
$val1 = 123;
$val2 = '123';
if (($val1 == $val2 || $val1 == true) && $val1 === 124) {
    echo 'blablabla';
}
if ($val1 == $val2 || $val1 == true && $val1 === 124) {
    echo 'blublublublu';
}
```

Les développeurs anglophones, utilisent des abréviations qu'il est intéressant de connaître, particulièrement quand on a besoin de lire de la documentation en anglais :

Français	Anglais	Abréviation anglaise
Egal (=)	Equal	EQ
Different (<>)	Not equal	NE
Inférieur à (<)	Less Than	LT
Supérieur à (>)	Greater Than	GT
Inférieur ou égal à (<=)	Less or Equal	LE
Supérieur ou égal à (>=)	Greater or Equal	GE

## 5.2 Préparation d'un jeu de données SQL avec Excel

Ce chapitre présente une astuce simple pour créer rapidement une table SQL et son contenu à partir d'une liste extraite d'une page HTML.

Il s'agit en l'occurrence de créer une table contenant une liste de pays.

Cette technique simple mais peu connue permet de constituer rapidement des jeux de données pouvant être utilisés dans le cadre de tests ou de petits projets.

On peut récupérer cette liste de pays sur Wikipedia, sous la rubrique ISO\_3166-1 :

[http://fr.wikipedia.org/wiki/ISO\\_3166-1](http://fr.wikipedia.org/wiki/ISO_3166-1)

Mais pour gagner du temps, on pourra se référer au code SQL de la table LSTPAYS fourni en annexe du présent document. Ce source permettra de créer rapidement la table des pays, puis de l'exporter sous Excel (par exemple via le logiciel System i Navigator), pour pouvoir tester ce qui va suivre.

Après avoir copié-collé le tableau des pays dans Excel, et avoir éliminé les données qui ne nous intéressaient pas, on obtient le tableau suivant :

	A	B	C	D
1	Nom français	Nom ISO	Norme	Désignation
2	AFG	AF	(ISO 3166-2)	AFGHANISTAN
3	ZAF	ZA	(ISO 3166-2)	AFRIQUE DU SUD
4	ALA	AX	(ISO 3166-2)	ALAND, ILES
5	ALB	AL	(ISO 3166-2)	ALBANIE
6	DZA	DZ	(ISO 3166-2)	ALGERIE
7	DEU	DE	(ISO 3166-2)	ALLEMAGNE
8	AND	AD	(ISO 3166-2)	ANDORRE
9	AGO	AO	(ISO 3166-2)	ANGOLA
10	AIA	AI	(ISO 3166-2)	ANGUILLA
11	ATA	AQ	(ISO 3166-2)	ANTARCTIQUE
12	ATG	AG	(ISO 3166-2)	ANTIGUA-ET-BARBUDA
13	ANT	AN	(ISO 3166-2)	ANTILLES NEERLANDAISES

Nous souhaitons récupérer les colonnes A, B et D. Pour ce faire, nous allons créer une colonne supplémentaire dans laquelle nous allons utiliser une formule de concaténation, de manière à créer des requêtes INSERT pour chaque ligne du tableau Excel. Concrètement, la requête de concaténation se présente de la façon suivante :

```
= "INSERT INTO MABIB/LSTPAYS VALUES (" & A2 & ", " & B2 & ", " & D2 & ");"
```

Dès que la requête fonctionne pour une ligne, vous pouvez la dupliquer sur toutes les lignes en-dessous.

Il me semble que c'est plus parlant quand on le voit en action sur Excel :

The screenshot shows an Excel spreadsheet titled "Microsoft Excel - liste\_pays.xls". The data is organized into two columns: "Désignation" (Column D) and "Requête" (Column E). Column D contains country names, and Column E contains the corresponding SQL INSERT statements. The formula in cell E2 is copied down to row 7. The formula in E2 is: = "INSERT INTO MABIB/LSTPAYS VALUES (" & A2 & ", " & B2 & ", " & D2 & ");".

	D	E
1	Désignation	
2	AFGHANISTAN	= "INSERT INTO MABIB/LSTPAYS VALUES ('AFG', 'AF', 'AFGHANISTAN');
3	AFRIQUE DU SUD	= "INSERT INTO MABIB/LSTPAYS VALUES ('ZAF', 'ZA', 'AFRIQUE DU SUD');
4	ALAND, ILES	= "INSERT INTO MABIB/LSTPAYS VALUES ('ALA', 'AX', 'ALAND, ILES');
5	ALBANIE	= "INSERT INTO MABIB/LSTPAYS VALUES ('ALB', 'AL', 'ALBANIE');
6	ALGERIE	= "INSERT INTO MABIB/LSTPAYS VALUES ('DZA', 'DZ', 'ALGERIE');
7	ALLEMAGNE	= "INSERT INTO MABIB/LSTPAYS VALUES ('DEU', 'DE', 'ALLEMAGNE');

Il faut ensuite copier le contenu de la colonne E dans le presse-papier, puis effectuer un « collage spécial » vers une colonne vierge d'Excel en utilisant l'option « par valeurs ». Vous obtenez ainsi un script SQL d'insertion de toutes les lignes du tableau Excel.

J'allais oublier un détail important : le problème des quotes, ou apostrophes. Je vous invite à regarder la colonne E se situant en face de la ligne de la « COTE D'IVOIRE ». Normalement, vous devriez avoir ceci :

```
INSERT INTO MABIB.LSTPAYS VALUES ('CIV', 'CI', 'COTE D'IVOIRE');
```

Si vous y regardez de près, vous constaterez que vous avez un nombre d'apostrophes impair entre les parenthèses du mot-clé SQL VALUES. Donc cette requête SQL ne pourra pas fonctionner, pas plus sous DB2 que sous MySQL. Et on retrouve le même problème sur tous les libellés contenant une ou plusieurs apostrophes.

Vous pouvez corriger facilement le problème en utilisant la fonction Excel suivante :

```
=SUBSTITUE(D2;"'";"''")
```

... ce qui nous donne la formule Excel suivante :

```
= "INSERT INTO MABIB/LSTPAYS VALUES ('" & A2 & "", "' & B2 & "", "' & SUBSTITUE(D2;"'";"''") & "'");"
```

Après correction, la requête d'insertion pour la Côte d'Ivoire ressemblera à ceci :

```
INSERT INTO MABIB.LSTPAYS VALUES ('CIV', 'CI', 'COTE D'IVOIRE');
```

La correction étant effectuée pour l'ensemble des lignes du tableau, vous pouvez recopier la colonne E dans le presse-papier Windows, puis effectuer un « collage spécial par valeur » dans une colonne vierge d'une autre feuille Excel.

Vous disposez maintenant d'un script SQL d'insertion prêt à l'emploi vous permettant d'alimenter la table SQL des pays, que vous pouvez sauvegarder au format texte.

Pour gagner du temps, vous trouverez au chapitre suivant une table des pays déjà préparée, qui pourra servir pour le développement de tests.

### **5.3 Tables des pays au format SQL**

La table SQL ci-dessous pourra nous servir de table exemple pour tester différentes techniques en PHP.

```
CREATE TABLE `countries` (
  `id` int(6) NOT NULL auto_increment,
  `codinter` char(3) NOT NULL default '',
  `codfra` char(2) NOT NULL default '',
  `countryname` varchar(250) NOT NULL default '',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

INSERT INTO countries (codinter, codfra, countryname)
VALUES
  ('AFG', 'AF', 'AFGHANISTAN'),
  ('ZAF', 'ZA', 'AFRIQUE DU SUD'),
  ('ALA', 'AX', 'ALAND, ILES'),
  ('ALB', 'AL', 'ALBANIE'),
  ('DZA', 'DZ', 'ALGERIE'),
  ('DEU', 'DE', 'ALLEMAGNE'),
  ('AND', 'AD', 'ANDORRE'),
  ('AGO', 'AO', 'ANGOLA'),
  ('AIA', 'AI', 'ANGUILLE'),
  ('ATA', 'AQ', 'ANTARCTIQUE'),
  ('ATG', 'AG', 'ANTIGUA-ET-BARBUDA'),
  ('ANT', 'AN', 'ANTILLES NEERLANDAISES'),
  ('SAU', 'SA', 'ARABIE SAOUDITE'),
  ('ARG', 'AR', 'ARGENTINE'),
  ('ARM', 'AM', 'ARMENIE'),
  ('ABW', 'AW', 'ARUBA'),
  ('AUS', 'AU', 'AUSTRALIE'),
  ('AUT', 'AT', 'AUTRICHE'),
  ('AZE', 'AZ', 'AZERBAIDJAN'),
  ('BHS', 'BS', 'BAHAMAS'),
  ('BHR', 'BH', 'BAHREIN'),
  ('BGD', 'BD', 'BANGLADESH'),
  ('BRB', 'BB', 'BARBADE'),
  ('BLR', 'BY', 'BELARUS'),
  ('BEL', 'BE', 'BELGIQUE'),
  ('BLZ', 'BZ', 'BELIZE'),
  ('BEN', 'BJ', 'BENIN'),
```

```
('BMU', 'BM', 'BERMUDES'),
('BTN', 'BT', 'BHOUTAN'),
('BOL', 'BO', 'BOLIVIE'),
('BIH', 'BA', 'BOSNIE-HERZEGOVINE'),
('BWA', 'BW', 'BOTSWANA'),
('BVT', 'BV', 'BOUVET, ILE'),
('BRA', 'BR', 'BRESIL'),
('BRN', 'BN', 'BRUNEI DARUSSALAM'),
('BGR', 'BG', 'BULGARIE'),
('BFA', 'BF', 'BURKINA FASO'),
('BDI', 'BI', 'BURUNDI'),
('CYM', 'KY', 'CAIMANES, ILES'),
('KHM', 'KH', 'CAMBODGE'),
('CMR', 'CM', 'CAMEROUN'),
('CAN', 'CA', 'CANADA'),
('CPV', 'CV', 'CAP-VERT'),
('CAF', 'CF', 'CENTRAFRICAINE, REPUBLIQUE'),
('CHL', 'CL', 'CHILI'),
('CHN', 'CN', 'CHINE'),
('CXR', 'CX', 'CHRISTMAS, ILE'),
('CYP', 'CY', 'CHYPRE'),
('CCK', 'CC', 'COCOS (KEELING), ILES'),
('COL', 'CO', 'COLOMBIE'),
('COM', 'KM', 'COMORES'),
('COG', 'CG', 'CONGO'),
('COD', 'CD',
    'CONGO, LA REPUBLIQUE DEMOCRATIQUE DU'),
('COK', 'CK', 'COOK, ILES'),
('KOR', 'KR', 'COREE, REPUBLIQUE DE'),
('PRK', 'KP',
    'COREE, REPUBLIQUE POPULAIRE DEMOCRATIQUE DE'),
('CRI', 'CR', 'COSTA RICA'),
('CIV', 'CI', 'COTE D'IVOIRE'),
('HRV', 'HR', 'CROATIE'),
('CUB', 'CU', 'CUBA'),
('DNK', 'DK', 'DANEMARK'),
('DJI', 'DJ', 'DJIBOUTI'),
('DOM', 'DO', 'DOMINICAINE, REPUBLIQUE'),
('DMA', 'DM', 'DOMINIQUE'),
('EGY', 'EG', 'EGYPTE'),
('SLV', 'SV', 'EL SALVADOR'),
('ARE', 'AE', 'EMIRATS ARABES UNIS'),
('ECU', 'EC', 'EQUATEUR'),
('ERI', 'ER', 'ERYTHREE'),
('ESP', 'ES', 'ESPAGNE'),
('EST', 'EE', 'ESTONIE'),
('USA', 'US', 'Etats-Unis'),
('ETH', 'ET', 'ETHIOPIE'),
('FLK', 'FK', 'FALKLAND, ILES (MALVINAS)'),
('FRO', 'FO', 'FEROE, ILES'),
('FJI', 'FJ', 'FIDJI'),
('FIN', 'FI', 'FINLANDE'),
('FRA', 'FR', 'FRANCE'),
('GAB', 'GA', 'GABON'),
```

```

('GMB ', 'GM ', 'GAMBIE'),
('GEO ', 'GE ', 'GEORGIE'),
('SGS ', 'GS ',
    'GEORGIE DU SUD ET LES ILES SANDWICH DU SUD'),
('GHA ', 'GH ', 'GHANA'),
('GIB ', 'GI ', 'GIBRALTAR'),
('GRC ', 'GR ', 'GRECE '),
('GRD ', 'GD ', 'GRENADE'),
('GRL ', 'GL ', 'GROENLAND'),
('GLP ', 'GP ', 'GUADELOUPE'),
('GUM ', 'GU ', 'GUAM'),
('GTM ', 'GT ', 'GUATEMALA'),
('GGY ', 'GG ', 'GUERNESEY'),
('GIN ', 'GN ', 'GUINEE '),
('GNB ', 'GW ', 'GUINEE BISSAU '),
('GNQ ', 'GQ ', 'GUINEE EQUATORIALE '),
('GUY ', 'GY ', 'GUYANA'),
('GUF ', 'GF ', 'GUYANE FRANCAISE '),
('HTI ', 'HT ', 'HAITI '),
('HMD ', 'HM ',
    'HEARD, ILE ET MCDONALD, ILES'),
('HND ', 'HN ', 'HONDURAS'),
('HKG ', 'HK ', 'HONG KONG'),
('HUN ', 'HU ', 'HONGRIE'),
('IMN ', 'IM ', 'ILE DE MAN '),
('UMI ', 'UM ',
    'ILES MINEURES ELOIGNEES DES ETATS-UNIS '),
('VGB ', 'VG ',
    'ILES VIERGES BRITANNIQUES'),
('VIR ', 'VI ',
    'ILES VIERGES DES ETATS-UNIS '),
('IND ', 'IN ', 'INDE'),
('IDN ', 'ID ', 'INDONESIE '),
('IRN ', 'IR ',
    'IRAN, REPUBLIQUE ISLAMIQUE D'' '),
('IRQ ', 'IQ ', 'IRAQ'),
('IRL ', 'IE ', 'IRLANDE '),
('ISL ', 'IS ', 'ISLANDE '),
('ISR ', 'IL ', 'ISRAEL '),
('ITA ', 'IT ', 'ITALIE '),
('JAM ', 'JM ', 'JAMAIQUE '),
('JPN ', 'JP ', 'JAPON '),
('JEY ', 'JE ', 'JERSEY '),
('JOR ', 'JO ', 'JORDANIE '),
('KAZ ', 'KZ ', 'KAZAKHSTAN '),
('KEN ', 'KE ', 'KENYA '),
('KGZ ', 'KG ', 'KIRGHIZISTAN '),
('KIR ', 'KI ', 'KIRIBATI '),
('KWT ', 'KW ', 'KOWEIT '),
('LAO ', 'LA ',
    'LAOS, REPUBLIQUE DEMOCRATIQUE POPULAIRE '),
('LSO ', 'LS ', 'LESOTHO '),
('LVA ', 'LV ', 'LETTONIE '),
('LBN ', 'LB ', 'LIBAN ')

```

```

('LBR', 'LR', 'LIBERIA'),
('LBY', 'LY',
    'LIBYENNE, JAMAHIRIYA ARABE'),
('LIE', 'LI', 'LIECHTENSTEIN'),
('LTU', 'LT', 'LITUANIE'),
('LUX', 'LU', 'LUXEMBOURG'),
('MAC', 'MO', 'MACAO'),
('MKD', 'MK',
    'MACEDOINE, L''EX-REPUBLIQUE YOUGOSLAVE DE '),
('MDG', 'MG', 'MADAGASCAR'),
('MYS', 'MY', 'MALAISIE'),
('MWI', 'MW', 'MALAWI'),
('MDV', 'MV', 'MALDIVES'),
('MLI', 'ML', 'MALI'),
('MLT', 'MT', 'MALTE'),
('MNP', 'MP',
    'MARIANNES DU NORD, ILES'),
('MAR', 'MA', 'MAROC'),
('MHL', 'MH', 'MARSHALL, ILES'),
('MTQ', 'MQ', 'MARTINIQUE'),
('MUS', 'MU', 'MAURICE'),
('MRT', 'MR', 'MAURITANIE'),
('MYT', 'YT', 'MAYOTTE'),
('MEX', 'MX', 'MEXIQUE'),
('FSM', 'FM',
    'MICRONESIE, ETATS FEDERES DE '),
('MDA', 'MD', 'MOLDOVA'),
('MCO', 'MC', 'MONACO'),
('MNG', 'MN', 'MONGOLIE'),
('MNE', 'ME', 'MONTENEGRO'),
('MSR', 'MS', 'MONTSERRAT'),
('MOZ', 'MZ', 'MOZAMBIQUE'),
('MMR', 'MM', 'MYANMAR'),
('NAM', 'NA', 'NAMIBIE'),
('NRU', 'NR', 'NAURU'),
('NPL', 'NP', 'NEPAL '),
('NIC', 'NI', 'NICARAGUA'),
('NER', 'NE', 'NIGER'),
('NGA', 'NG', 'NIGERIA '),
('NIU', 'NU', 'NIUE '),
('NFK', 'NF', 'NORFOLK, ILE '),
('NOR', 'NO', 'NORVEGE '),
('NCL', 'NC', 'NOUVELLE-CALEDONIE'),
('NZL', 'NZ', 'NOUVELLE-ZELANDE '),
('IOT', 'IO',
    'OCEAN INDIEN, TERRITOIRE BRITANNIQUE DE L'' '),
('OMN', 'OM', 'OMAN'),
('UGA', 'UG', 'OUGANDA'),
('UZB', 'UZ', 'OUZBEKISTAN '),
('PAK', 'PK', 'PAKISTAN'),
('PLW', 'PW', 'PALAOS'),
('PSE', 'PS',
    'PALESTINIEN OCCUPE, TERRITOIRE'),
('PAN', 'PA', 'PANAMA'),

```

```
('PNG', 'PG',
    'PAPOUASIE-NOUVELLE-GUINEE'),
('PRY', 'PY',
    'PARAGUAY'),
('NLD', 'NL',
    'PAYS-BAS'),
('PER', 'PE',
    'PEROU'),
('PHL', 'PH',
    'PHILIPPINES'),
('PCN', 'PN',
    'PITCAIRN'),
('POL', 'PL',
    'POLOGNE'),
('PYF', 'PF',
    'POLYNESIE FRANCAISE'),
('PRI', 'PR',
    'PORTO RICO'),
('PRT', 'PT',
    'PORTUGAL'),
('QAT', 'QA',
    'QATAR'),
('REU', 'RE',
    'REUNION'),
('ROU', 'RO',
    'ROUMANIE'),
('GBR', 'GB',
    'ROYAUME-UNI'),
('RUS', 'RU',
    'RUSSIE, FEDERATION DE'),
('RWA', 'RW',
    'RWANDA'),
('ESH', 'EH',
    'SAHARA OCCIDENTAL'),
('BLM', 'BL',
    'SAINT-BARTHELEMY'),
('KNA', 'KN',
    'SAINT-KITTS-ET-NEVIS'),
('SMR', 'SM',
    'SAINT-MARIN'),
('MAF', 'MF',
    'SAINT-MARTIN (PARTIE FRANCAISE)'),
('SPM', 'PM',
    'SAINT-PIERRE-ET-MIQUELON'),
('VAT', 'VA',
    'SAINT-SIEGE (ETAT DE LA CITE DU VATICAN)'),
('VCT', 'VC',
    'SAINT-VINCENT-ET-LES GRENADINES'),
('SHN', 'SH',
    'SAINTE-HELENE'),
('LCA', 'LC',
    'SAINTE-LUCIE'),
('SLB', 'SB',
    'SALOMON, ILES'),
('WSM', 'WS',
    'SAMOA'),
('ASM', 'AS',
    'SAMOA AMERICAINES'),
('STP', 'ST',
    'SAO TOME-ET-PRINCIPE'),
('SEN', 'SN',
    'SENEGAL'),
('SRB', 'RS',
    'SERBIE'),
('SYC', 'SC',
    'SEYCHELLES'),
('SLE', 'SL',
    'SIERRA LEONE'),
('SGP', 'SG',
    'SINGAPOUR'),
('SVK', 'SK',
    'SLOVAQUIE'),
('SVN', 'SI',
    'SLOVENIE'),
('SOM', 'SO',
    'SOMALIE'),
('SDN', 'SD',
    'SOUDAN'),
('LKA', 'LK',
    'SRI LANKA'),
('SWE', 'SE',
    'SUEDE'),
('CHE', 'CH',
    'SUISSE'),
('SUR', 'SR',
    'SURINAME'),
('SJM', 'SJ',
    'SVALBARD ET ILE JAN MAYEN'),
('SWZ', 'SZ',
    'SWAZILAND'),
('SYR', 'SY',
    'SYRIENNE, REPUBLIQUE ARABE'),
('TJK', 'TJ',
    'TADJIKISTAN'),
```

```
('TWN', 'TW', 'TAIWAN, PROVINCE DE CHINE'),  
('TZA', 'TZ', 'TANZANIE, REPUBLIQUE UNIE DE'),  
('TCD', 'TD', 'TCHAD'),  
('CZE', 'CZ', 'TCHEQUE, REPUBLIQUE'),  
('ATF', 'TF', 'TERRES AUSTRALES FRANCAISES'),  
('THA', 'TH', 'THAILANDE'),  
('TLS', 'TL', 'TIMOR-LESTE'),  
('TGO', 'TG', 'TOGO'),  
('TKL', 'TK', 'TOKELAU'),  
('TON', 'TO', 'TONGA'),  
('TTO', 'TT', 'TRINITE-ET-TOBAGO'),  
('TUN', 'TN', 'TUNISIE'),  
('TKM', 'TM', 'TURKMENISTAN'),  
('TCA', 'TC', 'TURKS ET CAIQUES, ILES'),  
('TUR', 'TR', 'TURQUIE'),  
('TUV', 'TV', 'TUVALU'),  
('UKR', 'UA', 'UKRAINE'),  
('URY', 'UY', 'URUGUAY'),  
('VUT', 'VU', 'VANUATU'),  
('VEN', 'VE', 'VENEZUELA'),  
('VNM', 'VN', 'VIET NAM'),  
('WLF', 'WF', 'WALLIS-ET-FUTUNA'),  
('YEM', 'YE', 'YEMEN'),  
('ZMB', 'ZM', 'ZAMBIE'),  
('ZWE', 'ZW', 'ZIMBABWE');
```

## 5.4 Mesure de performances

Il est quelquefois utile de pouvoir mesurer les performances d'un script PHP.

On peut pour ce faire s'appuyer sur la fonction PHP microtime() que l'on encapsulera dans une fonction getmicrotime(), de la façon suivante :

```
function getmicrotime() {  
    list($usec, $sec) = explode(" ", microtime());  
    return ((float)$usec + (float)$sec);  
  
}  
  
// premier appel de la fonction getmicrotime() avant l'exécution du code à "mesurer"  
  
$time_start = getmicrotime();  
  
// placer ici le code dont vous souhaitez mesurer les performances  
  
// second et dernier appel de la fonction getmicrotime()  
  
$time_stop = getmicrotime();  
  
$time_dif = $time_stop - $time_start;  
  
echo "<p><h2>Temps d'exécution de la reprise :</h2> {$time_dif} secondes </p>"  
;
```

## 5.5 Les filtres

L'extension "Filter" apporte à PHP la possibilité de filtrer les données soit en les validant, soit en les nettoyant.

Cette extension est particulièrement utile pour contrôler et filtrer les données provenant de l'extérieur comme des données de formulaires par exemple.

Il y a donc deux possibilités de filtrage : la validation et le nettoyage.

La Validation sert à vérifier si une donnée répond à certains critères (comme par exemple le filtre de validation d'email : FILTER\_VALIDATE\_EMAIL). En cas d'anomalie, un message d'anomalie est renvoyé par la fonction filter\_var(), mais la donnée contrôlée n'est pas modifiée.

Le nettoyage va nettoyer les données, par exemple en retirant des caractères indésirables. Par exemple, la fonction filter\_var() associée au mot clé FILTER\_SANITIZE\_EMAIL va permettre d'éliminer les caractères inappropriés pour une adresse email.

Lien vers la documentation officielle relative à l'extension "Filter" :

<http://www.php.net/manual/fr/intro.filter.php>

A voir aussi dans la documentation officielle :

- la page consacrée aux autres filtres :

<http://www.php.net/manual/fr/filter.filters.misc.php>

- la page consacrée aux "drapeaux" des filtres :

<http://www.php.net/manual/fr/filter.filters.flags.php>

Quelques exemples ci-dessous pris sur :

<http://www.php.net/manual/fr/filter.examples.validation.php>

```
/*
 * Exemple #1 Validation d'adresses email avec filter_var()
 */
$email_a = 'joe@example.com';
$email_b = 'bogus';

if (filter_var($email_a, FILTER_VALIDATE_EMAIL)) {
```

```

        echo "Cette ($email_a) adresse email est considérée comme valide.";
    }
    if (filter_var($email_b, FILTER_VALIDATE_EMAIL)) {
        echo "Cette ($email_b) adresse email est considérée comme valide.";
    }

/*
 * Exemple #2 Validation d'adresses IP avec filter_var()
 */
$ip_a = '127.0.0.1';
$ip_b = '42.42';

if (filter_var($ip_a, FILTER_VALIDATE_IP)) {
    echo "Cette ($ip_a) adresse IP est considérée comme valide.";
}
if (filter_var($ip_b, FILTER_VALIDATE_IP)) {
    echo "Cette ($ip_b) adresse IP est considérée comme valide.";
}

/*
 * Exemple #3 Passage d'options à la fonction filter_var()
 */
$int_a = '1';
$int_b = '-1';
$int_c = '4';
$options = array(
    'options' => array(
        'min_range' => 0,
        'max_range' => 3,
    )
);
if (filter_var($int_a, FILTER_VALIDATE_INT, $options) !== FALSE) {
    echo "Cet entier ($int_a) est considéré comme valide (entre 0 et 3).\n";
}
if (filter_var($int_b, FILTER_VALIDATE_INT, $options) !== FALSE) {
    echo "Cet entier ($int_b) est considéré comme valide (entre 0 et 3).\n";
}
if (filter_var($int_c, FILTER_VALIDATE_INT, $options) !== FALSE) {
    echo "Cet entier ($int_c) est considéré comme valide (entre 0 et 3).\n";
}

$options['options']['default'] = 1;
if (($int_c = filter_var($int_c, FILTER_VALIDATE_INT, $options)) !== FALSE) {
    echo "Cet entier ($int_c) est considéré comme valide (entre 0 et 3) et vaut
$int_c.";
}

/*
 * Exemple : Nettoyage et validation d'adresses email
 * Exemples pris sur
 * http://www.php.net/manual/fr/filter.examples.sanitization.php
 */

```

```
$a = 'joe@example.org';
$b = 'bogus - at - example dot org';
$c = '(bogus@example.org)';

$sanitized_a = filter_var($a, FILTER_SANITIZE_EMAIL);
if (filter_var($sanitized_a, FILTER_VALIDATE_EMAIL)) {
    echo "Cette (a) adresse email nettoyée est considérée comme valide.";
}

$sanitized_b = filter_var($b, FILTER_SANITIZE_EMAIL);
if (filter_var($sanitized_b, FILTER_VALIDATE_EMAIL)) {
    echo "Cette (b) adresse email nettoyée est considérée comme valide.";
} else {
    echo "Cette (b) adresse email nettoyée est considérée comme invalide.";
}

$sanitized_c = filter_var($c, FILTER_SANITIZE_EMAIL);
if (filter_var($sanitized_c, FILTER_VALIDATE_EMAIL)) {
    echo "Cette (c) adresse email nettoyée est considérée comme valide.";
    echo "Avant : $c\n";
    echo "Après : $sanitized_c\n";
}
```

## 5.6 Les pièges du « select multiple »

Ce chapitre reprend l'exemple du formulaire tel qu'il a été traité au chapitre 4.4.3 (dans sa version finale).

L'exemple traité dans ce chapitre annexe est très proche de l'exemple initial, si ce n'est que le champ de type « select » du formulaire initial s'est vu doter de l'attribut « multiple ».

Le fait que le champ « select » fonctionne avec l'attribut « multiple » a un impact important sur le code PHP, car le tableau `$_POST['myselect']` va recevoir un tableau PHP (contenant la liste des options sélectionnées par l'utilisateur). Cela a un impact sur la manière dont les options du champ « select » sont « selected » ou pas, et sur la manière de filtrer et de transmettre le contenu du champ « myselect » au script « pageable.php ».

Mais l'attribut « multiple » pose aussi quelques difficultés qui sont dûes au fait que le champ « select » doit impérativement contenir au moins une option « selected » pour que le champ fasse partie de la requête HTTP transmise au script PHP. Sans cela, le champ de saisie n'apparaît tout simplement pas dans `$_POST`, ce qui est un comportement pour le moins étrange. Il a donc été nécessaire de prendre en compte ce comportement imprévu, en particulier dans la fonction `get_form_select_selected()`.

Le script « pageable.php » est très légèrement impacté par la modification, son code source est donné en premier :

```
<?php
// démarrage du gestionnaire de session
session_start();

echo '<br>Contenu de $_SESSION affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_SESSION as $postkey=>$postval) {
    $postkey = htmlentities($postkey, ENT_QUOTES, "UTF-8");
    if (is_array($postval)) {
        $tmp = [];
        foreach($postval as $postval2) {
            $tmp[] = htmlentities($postval2, ENT_QUOTES, "UTF-8");
        }
        $postval = implode(' | ', $tmp);
    } else {
        $postval = htmlentities($postval, ENT_QUOTES, "UTF-8");
    }
    echo "<li>{$postkey} => {$postval} </li>" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Le code source du script « pageform.php » démarre sur la page suivante.

```

<?php
// démarrage du gestionnaire de session
session_start();
?><!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title>Exemple de formulaire (2)</title>
    <link rel="stylesheet" href="pageform.css">
</head>
<body>
<?php
    // Initialisation du tableau des erreurs
    $erreurs = [];
    $gooddata = [];
    // Boucle pour analyser si des erreurs sont présentes
    foreach($_POST as $keypost=>$valpost) {
        if ($keypost != 'valid') {
            if ($keypost == 'myselect') {
                // myselect est un tableau, il doit être traité à part
                if (count($valpost) == 0) {
                    $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
                } else {
                    $gooddata[$keypost] = [];
                    foreach($valpost as $valpost2) {
                        $tmp = trim(strip_tags($valpost2));
                        if ($tmp != '') {
                            $gooddata[$keypost][] = $tmp;
                        }
                    }
                    if (count($gooddata[$keypost])==0) {
                        $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
                    }
                }
            } else {
                $valpost = strip_tags($valpost);
                $valpost = trim($valpost);
                if ($valpost == '') {
                    $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
                } else {
                    $gooddata[$keypost] = $valpost;
                }
            }
        }
    }
    // si le tableau $gooddata contient des données et qu'il n'y a pas d'erreurs
    if (count($gooddata)>0 && count($erreurs)==0) {
        // copie des bonnes données dans $_SESSION
        $_SESSION = $gooddata;
        // redirection vers "pagecible.php"
        header('location: pagecible.php');
        exit; // on stoppe l'exécution du script courant
    }
    // Si des erreurs sont présentes, alors on en affiche la liste
    if (count($erreurs)>0) {
        echo '<fieldset style="background-color: bisque; border-radius: 5px;">' . PHP_EOL;
        echo '<legend>Liste des erreurs</legend>' . PHP_EOL;
        echo '<ul>' . PHP_EOL;
        foreach($erreurs as $valerr) {
            echo "<li>$valerr</li>" . PHP_EOL;
        }
    }

```

```

        echo '</ul>'.PHP_EOL;
        echo '</fieldset>'.PHP_EOL;
    }

    /**
     * Renvoie en sortie la valeur d'un champ de saisie
     * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
     *   'post' (valeur par défaut)
     *   'get'
     * @param $field
     * @param string $method
     * @return string
     */
    function get_form_input_value($field, $method='post') {
        // suppression des blancs parasites et forçage des valeurs en minuscule
        $field = strtolower(trim($field));
        $method = strtolower(trim($method));

        if ($method == 'get') {
            $value = isset($_GET[$field]) ? $_GET[$field] : '';
        } else {
            $value = isset($_POST[$field]) ? $_POST[$field] : '';
        }
        $value = trim(strip_tags($value));
        $value = htmlentities($value, ENT_QUOTES, "UTF-8");
        return $value;
    }

    /**
     * Renvoie en sortie la valeur "selected" pour les champs de type SELECT
     * si la valeur du champ considéré est égale au second paramètre
     * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
     *   'post' (valeur par défaut)
     *   'get'
     * Le paramètre $multiple est facultatif (booléen à false par défaut)
     * @param $field
     * @param $option_key
     * @param string $method
     * @param string $multiple
     * @return string
     */
    function get_form_select_selected($field, $option_key, $method='post',
        $multiple=false) {
        // suppression des blancs parasites et forçage des valeurs en minuscule
        $field = strtolower(trim($field));
        $method = strtolower(trim($method));
        if ($method == 'get') {
            $value = isset($_GET[$field]) ? $_GET[$field] : '';
        } else {
            $value = isset($_POST[$field]) ? $_POST[$field] : '';
        }
        if ($multiple == false) {
            $value = trim($value);
            if ($value == $option_key) {
                return 'selected';
            }
        } else {
            if (!is_array($value) || count($value)==0) {
                // si le tableau est vide et que $option_key est à blanc,

```

```

        // cela signifie qu'on est sur la première option et il est
        // préférable de mettre cette option à "selected" pour bloquer
        // un comportement étrange constaté sur Firefox et Chrome, avec
        // les champs "select" dotés de l'attribut "multiple"
        if ($option_key == '') {
            return 'selected';
        }
        // dans tous les autres cas, on sort sans le "selected"
        return '';
    } else {
        if (in_array($option_key, $value)) {
            // si l'option se trouve dans le tableau alors sortie avec
            "selected"
                return 'selected';
        }
    }
    return '';
}
?>
<form method="POST">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                   placeholder="saisissez un critère de recherche"
                   value=<?php echo get_form_input_value('search'); ?>>
        />
    </p>
    <p>
        <label for="number_field">Exemple de champ "number"</label>
        <input type="number" name="number" id="number_field"
               value=<?php echo get_form_input_value('number'); ?>>
    />
    </p>
    <p>
        <label for="select_field">Exemple de champ "select"</label>
        <select name="myselect[]" id="select_field" multiple size="4">
            <?php
                // la première option (avec code à blanc) est intégrée dans la boucle
                // afin de pouvoir être sélectionnée par défaut si aucune option
                // n'est sélectionnée
                foreach(array('', 'A', 'B', 'C') as $valselect) {
                    if ($valselect == '') {
                        echo "<option value=\"{$valselect}\\" "
                            . get_form_select_selected('myselect', $valselect, 'post',
true)
                            . ">Choisissez une valeur</option>" . PHP_EOL;
                    } else {
                        echo "<option value=\"{$valselect}\\" "
                            . get_form_select_selected('myselect', $valselect, 'post',
true)
                            . ">valeur {$valselect}</option>" . PHP_EOL;
                    }
                };
            ?>
        </select>
    </p>
</fieldset>

```

```
<input type="submit" name="valid" value="Valider" />
</form>
</body>
</html>
```

## **5.7 Composer**

TODO : chapitre en cours d'écriture



## 5.8 Bibliographie

TODO : chapitre à compléter

### Ressources bibliographiques pour PHP

- Un ouvrage au contenu très riche, édité chez Sitepoint :  
PHP Master, Write Cutting-Edge Code
  - par Lorna Mitchell, Davey Shafik & Matthew Turland , Ed. Sitepoint
  - <http://www.sitepoint.com/books/phppro1/>
- Packt Publishing propose un riche catalogue d'ouvrages consacrés à PHP (je vous recommande notamment « PHP Jquery Cookbook ») :
  - <http://www.packtpub.com/books/php>
- PHP Architect propose également un fond très intéressant d'ouvrages dédiés à PHP (plusieurs ouvrages sont édités avec le soutien de Zend) :
  - <http://www.phparch.com/books/> (je recommande notamment « PHP Playbook »)
- L'éditeur américain Oreilly propose un impressionnant catalogue d'ouvrages consacrés à PHP et à d'autres sujets :
  - <http://oreilly.com/> (je recommande notamment le "PHP Cookbook" de David Sklar et Adam Trachtenberg)

Pour une étude approfondie de PHP, et de la POO en particulier, je vous recommande les ouvrages suivants :

- *PHP and MySQL® Web Development, Fourth Edition*

par Luke Welling; Laura Thomson, (Addison-Wesley Professional)

- *PHP 5 in Practice*

par Elliot White III; Jonathan Eisenhamer (Publisher: Sams)

- *PHP Cookbook, 2nd Edition,*

par David Sklar and Adam Trachtenberg (O'Reilly)

- *PHP Hacks*

par Jack D. Herrington (O'Reilly)

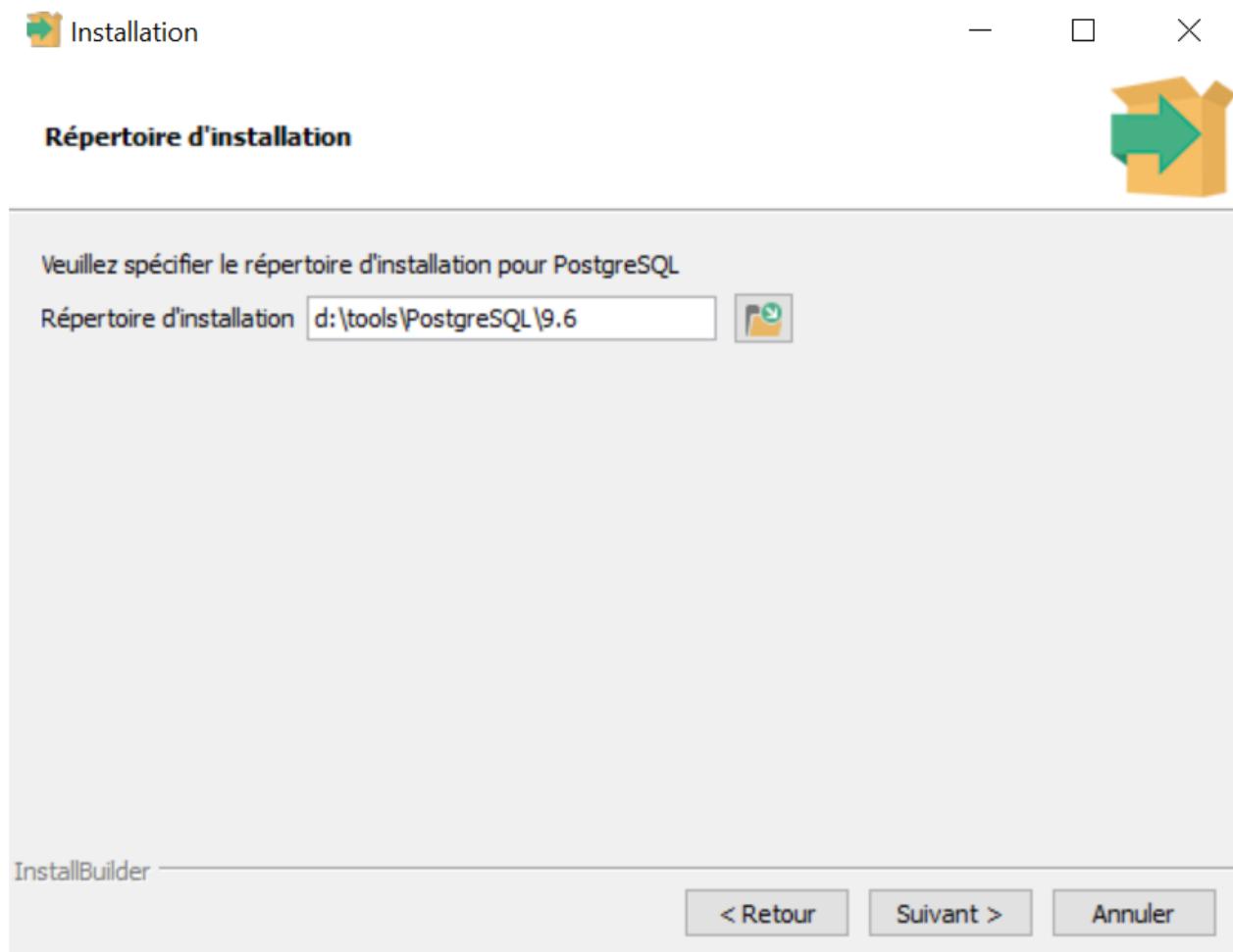
- *Object-Oriented PHP*

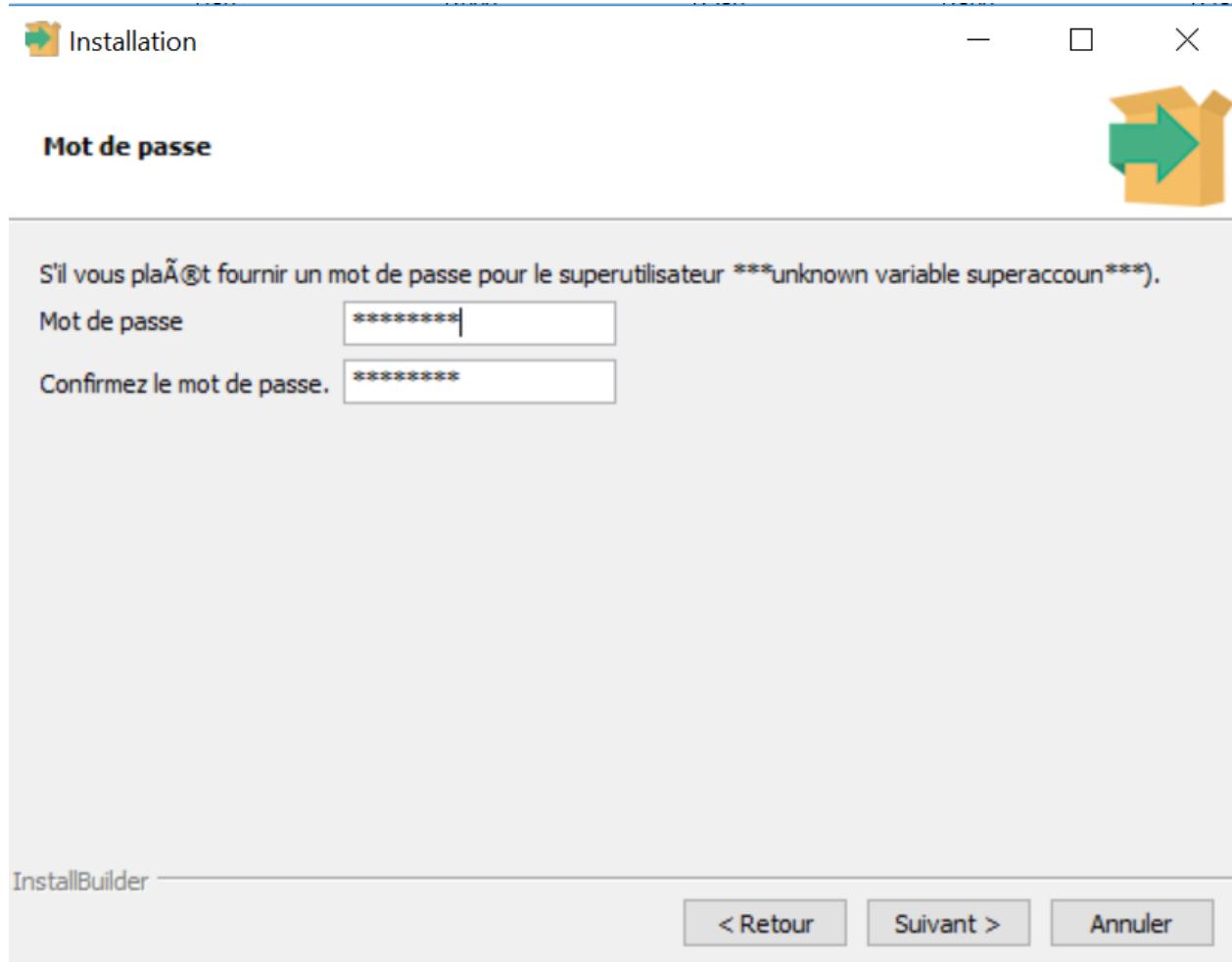
par Peter Lavin (No Starch Press)

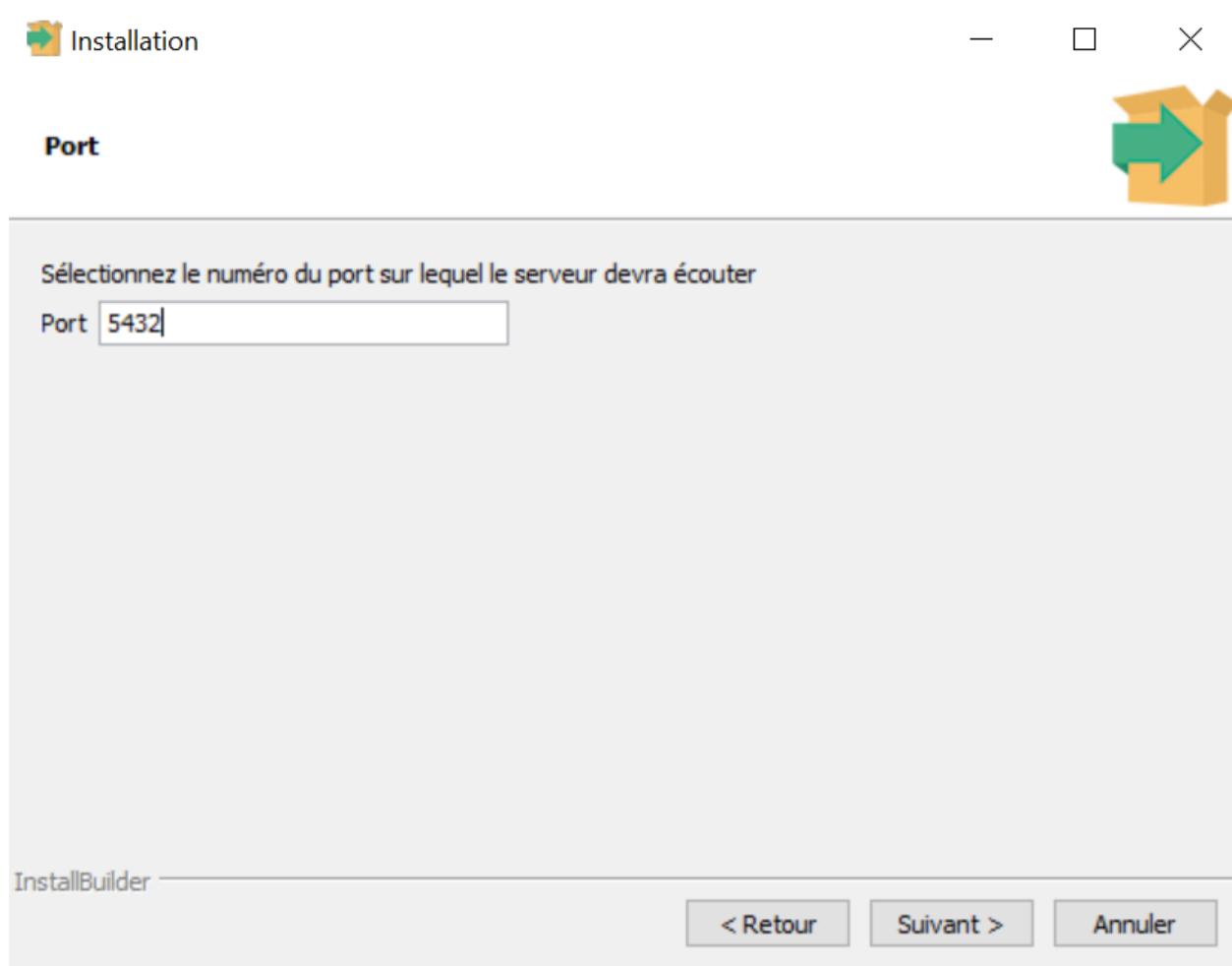
## 5.x PHP et PostgreSQL

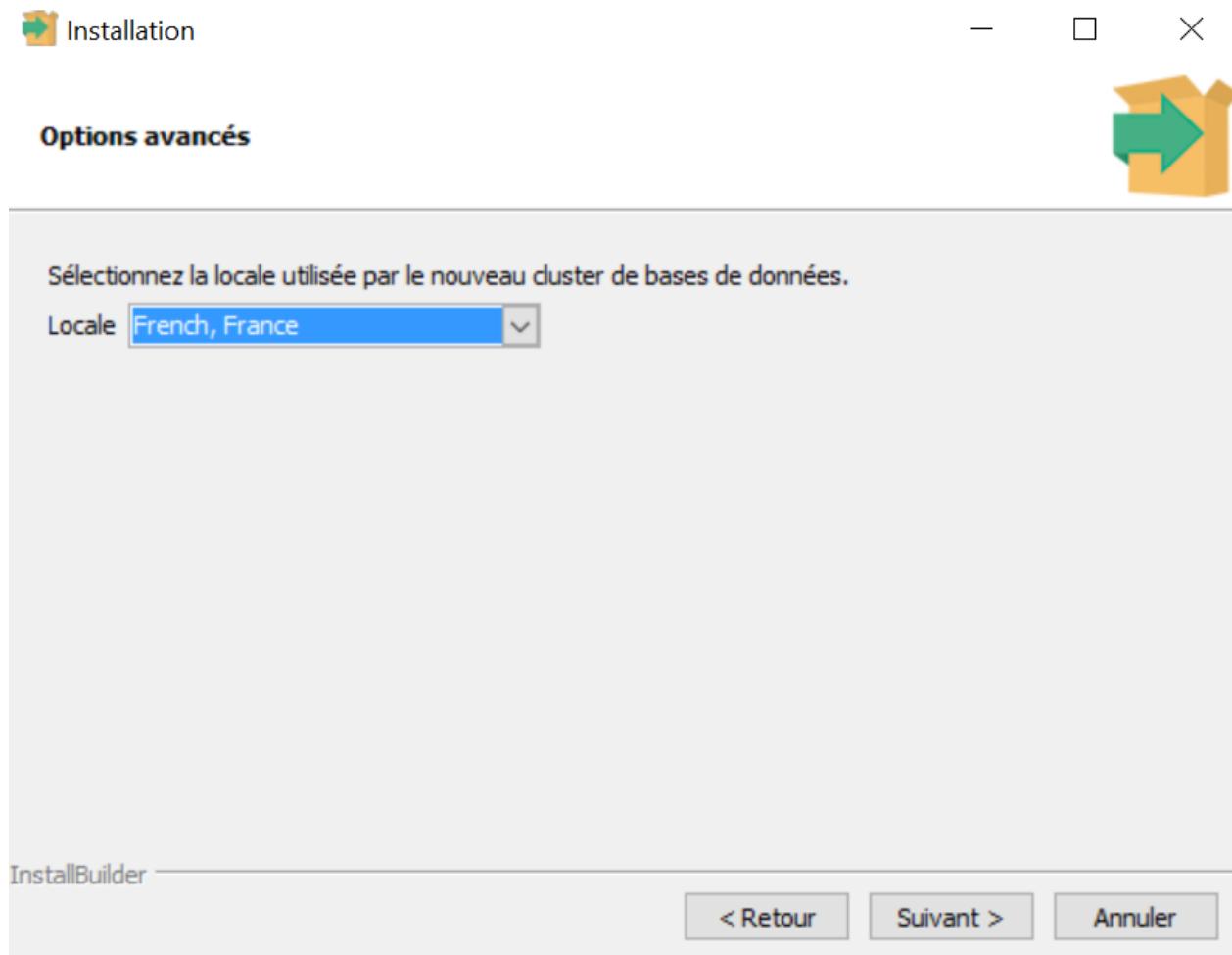
TODO : chapitre à compléter

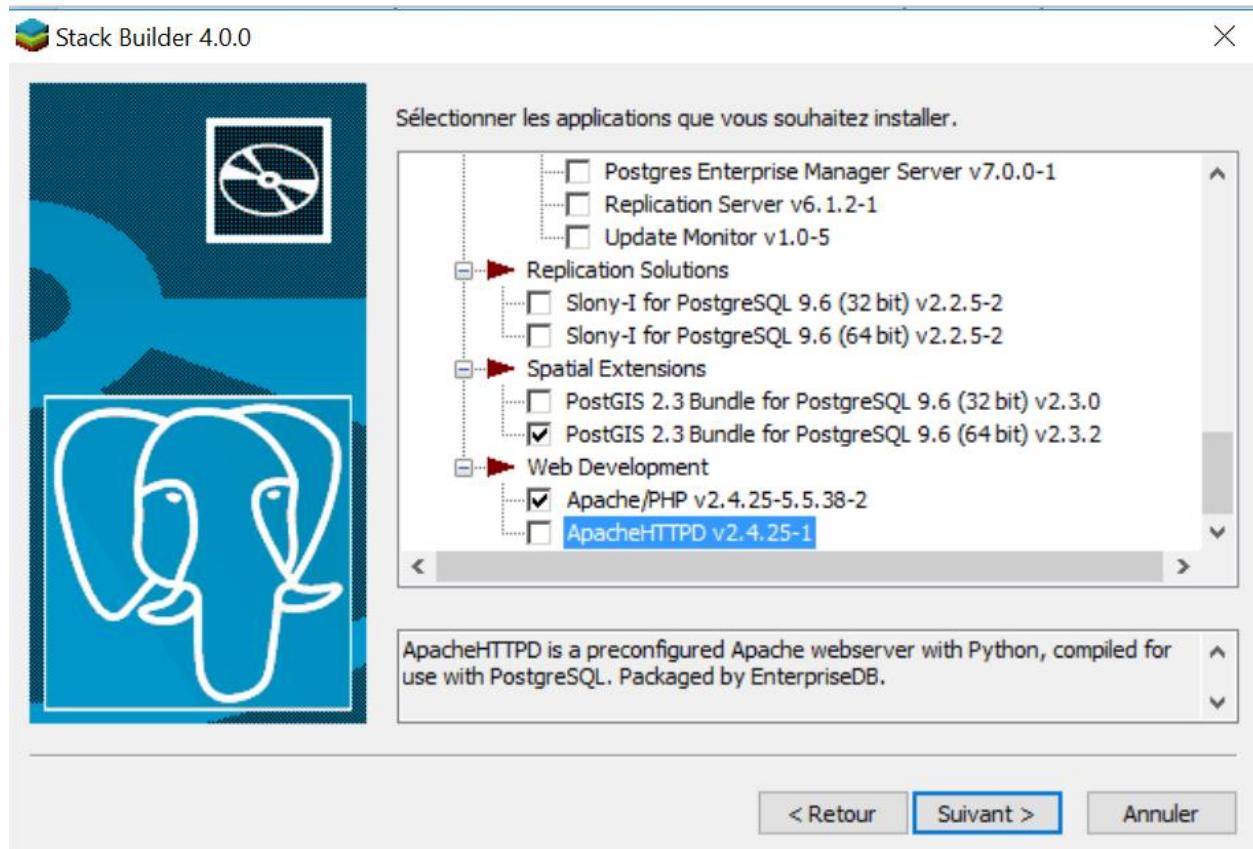












## **5.x PHP et MongoDB**

TODO : chapitre à créer

## 6 Changelog

Version 1.0 publiée le 18/07/2017 :

- Publication initiale (provisoire)

Version 1.1 publiée le 19/07/2017 :

- Correction de quelques coquilles
- Ajout du chapitre 5.6 (pièges du « select multiple »)
- Rédaction du chapitre 4.4.5 (générateur de formulaire)