P.H.P

MySQL

I n i t i a t i o n

Guillaume HÉNOT

V2.5

SOMMAIRE :

Qu'est-ce que PHP ?

Il a été créé, à l'origine, en 1995 par Rasmus Lerdorf, pour compter les passages sur son CV en ligne. PHP était alors pour « Personnal Home Page Tools ». Depuis, la petite bibliothèque de scripts d'origine a fortement évolué et est devenue un module du serveur Apache, capable de dialoguer avec de nombreuses bases de données. PHP signifie aujourd'hui « PHP : Hypertext Preprocessor ». Je vous renvoie vers la page « Histoire » du manuel PHP pour plus d'informations : http://fr.php.net/history/

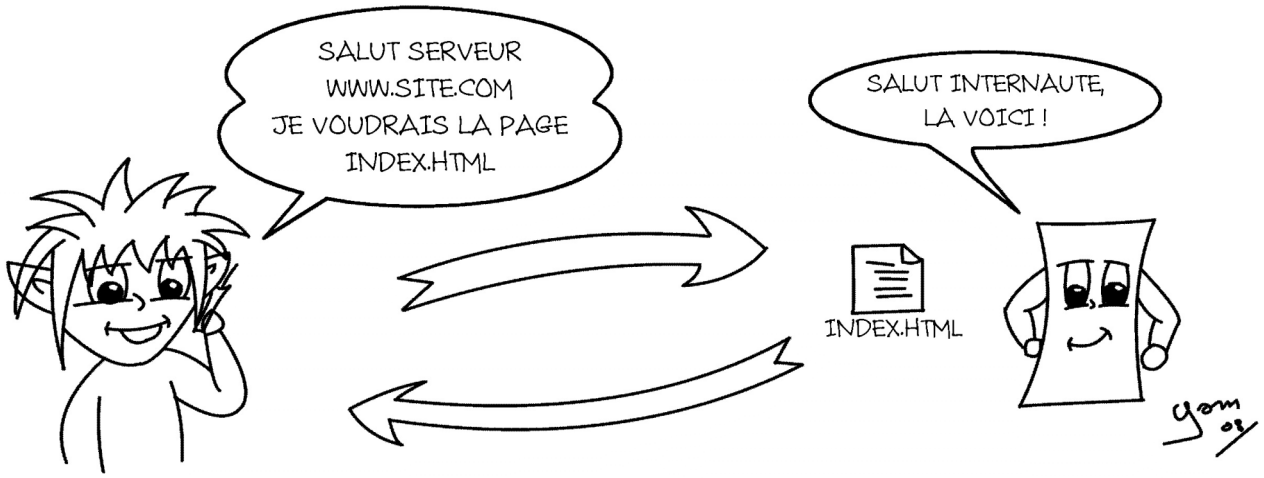
À l'heure actuelle, la dernière version de PHP disponible est la 5. La version 6 pointe le bout de son nez.

Un des grands amis de PHP est MySQL. C'est un serveur de bases de données. Tout comme PHP et Apache, il est sous licence GPL et est donc téléchargeable gratuitement sur Internet.
De par sa stabilité et son coût (gratuit !), nous retrouverons souvent ce trio.

PHP est un langage de scripts dit « **côté serveur** ». C'est-à-dire qu'il ne fonctionne que sur un serveur Web (ou local), et qu'il ne dépend pas de l'équipement informatique de l'internaute.

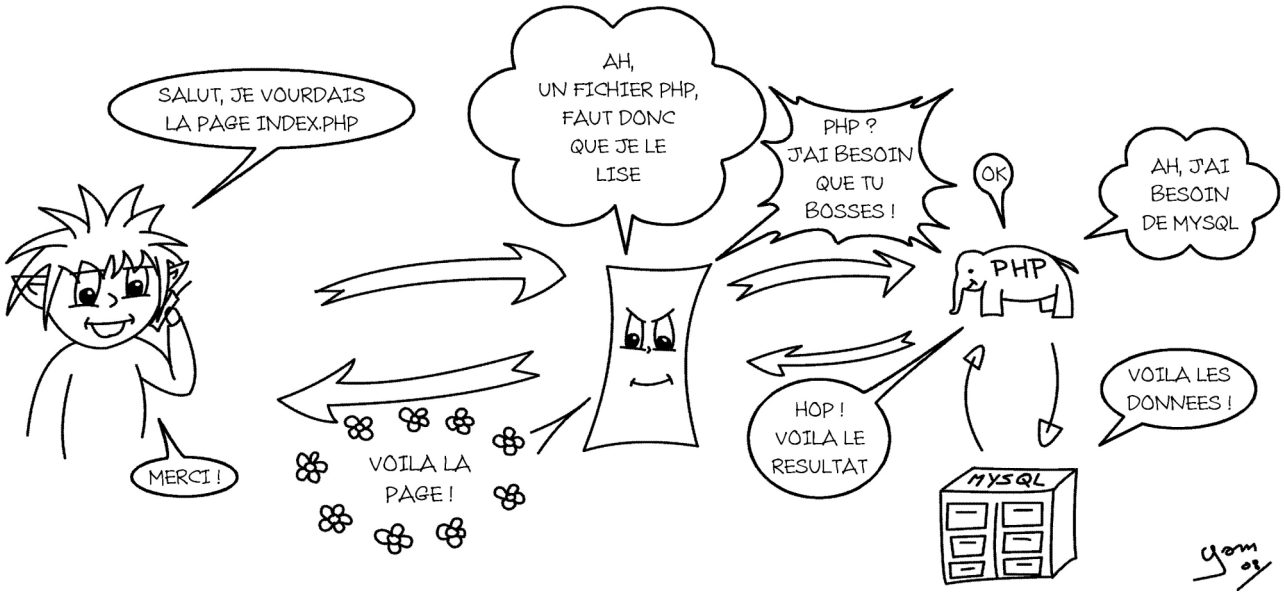
Quelle est là différence avec un langage dit « côté client » tel que HTML ou Javascript ?

• Voyons ce qu'il se passe en temps normal :



En effet, avec ce type de langage, le serveur ne fait qu'envoyer la page au navigateur et c'est ce dernier qui devra analyser le code et l'exécuter correctement.

• Dans un langage côté serveur, ça se passe plutôt comme ça :



Cette fois-ci, le serveur lit la page avant de la renvoyer, et fait travailler PHP et éventuellement MySQL pour ensuite construire une page standard HTML unique (parfois personnalisée) et la renvoyer au navigateur.

Installation et outils

SOUS WINDOWS :

- Installer EasyPHP (qui installe PHP, MySQL et Apache), et qui est téléchargeable là : http://www.easyphp.org/
- Il existe également WAMP (http://www.wampserver.com/) qui fonctionne plus ou moins de la même façon.
- Les dossiers à utiliser pour tester vos scripts sont à configurer via l'administration d'EasyPHP (ou de WAMP).

SOUS MACOS X :

Vous avez sur MacOS X Apache et PHP d'installés en natif, mais vous pouvez changer de package PHP pour un plus complet car celui fourni de base ne permet pas grand chose.
En outre, MacOS X étant un UNIX, il est particulièrement bien adapté pour faire serveur web.

- Installer les packages PHP et MySQL téléchargeables là (en anglais) : http://www.entropy.ch/software/macosx/
- Lancer le partage Web (c'est en fait le serveur apache) via les préférences système, vos scripts seront actifs en les plaçant dans le dossier « Sites » de votre dossier de départ. Et seront accessibles dans un navigateur, avec une adresse du type : http://localhost/~nomdutilisateurcourt/monscript.php
- Si les manipulations vous font peur, il existe aussi MAMP (http://www.mamp.info/) qui fonctionne comme WAMP ou EasyPHP ☺

LES BONS OUTILS

Pour développer en PHP, il faut peu de chose. Un simple éditeur de texte brut sera suffisant. Bref un truc comme le BlocNote (PC) ou TextEdit (Mac) suffiront. Toutefois les bons ouvriers travaillent avec de bons outils.
Eliminez d'office ceux qui ne proposent pas de coloration du code.
Côté logiciels gratuits il en existe un grand nombre :

- Notepad++ (PC), Context (PC), TextWrangler (Mac), Taco (Mac), etc …

Côté éditeurs payant, on en trouve à tous les prix :

- Dreamweaver (Mac & PC), TextMate (Mac), etc …

Prenez quelques heures pour tester différents logiciels et trouver celui qui vous convient le mieux.
Et enfin je ne le répèterai jamais assez, indentez votre code et commentez-le ! Ça facilite la relecture et le débuggage !

Syntaxe PHP

Faisons une analogie pour mieux nous représenter le fonctionnement de tout ça. Imaginez que vous êtes le patron d'une société. Vous avez sous vos ordres M. PHP ; M. PHP est une personne très serviable, toujours prête à vous obéir, quoi que vous lui demandiez. MAIS, pour que ça marche bien et qu'il vous donne entière satisfaction, il faut lui demander PRECISEMENT ce que vous voulez.
En gros, il ne réfléchit pas et ne prendra pas de décision.
Pour lui parler, il faudra donc employer une SYNTAXE précise.

À chaque fois que vous aurez besoin de ses services, il faudra le lui indiquer. On utilise pour ça, une balise d'ouverture ("<?php") et une balise de fermeture ("?>").

Exemple :

```
<?php
   … lignes de code …
?>
```

Entre ces 2 balises, il sait qu'il va devoir analyser et traiter les lignes de code.

Une ligne de code peut contenir des chaînes de caractères, des variables, des fonctions, des nombres, etc …

- Une **chaîne de caractères** sera TOUJOURS entourée de guillemets (simples ou doubles). Ex : `"ceci est une chaîne de caractères"`
- Une **Variable** sera TOUJOURS précédée du signe $. Ex : `$maVariable`
- Une **instruction**, c'est-à-dire en général une ligne de code, se termine avec un point-virgule. Ex : `echo 'toto';`

Si vous oubliez un ';', une parenthèse '()', une accolade '{}', PHP vous retournera une erreur de type « Parse error … » et vous indiquera la ligne.
Sachez que parfois, la ligne n'est pas la bonne, MAIS l'erreur ne peut se situer QU'AVANT l'endroit indiqué. Jamais après.

> **NOTE** : La différence entre les chaînes de caractères entre guillemets doubles " " et guillemets simples ' ' est que les chaînes entre guillemets doubles seront analysées mots par mots par PHP afin d'y trouver des variables et les remplacer par leur valeur, alors que les chaînes entre guillemets simples ne seront pas analysées et retourneront donc le texte tel quel.

Utilisation

Afin de dire au serveur que la page demandée doit être analysée par PHP, nous devons nommer la page avec une extension ".php".
Exemple : mapage.php

PHP va alors analyser la page et rechercher dedans les portions de code PHP. Soit la page est uniquement constituée de code PHP, soit elle est un mélange de PHP et de HTML.

Exemple de mélange :

```
<?php
   $mavar = "toto";
?>
<html>
   <head>
      <title>Ma page mélangée</title>
   </head>
   <body>
   <?php
      echo $mavar;
   ?>
   </body>
</html>
```

Les portions de code PHP ne seront pas visibles dans la page envoyée à l'internaute. Seul le résultat de l'analyse le sera.
Pour l'exemple ci-dessus, l'internaute ne verra que « toto » écrit dans la page, et le code source de la page HTML sera le suivant :

```
<html>
   <head>
      <title>Ma page mélangée</title>
   </head>
   <body>
   toto
   </body>
</html>
```

Commentaires

Lorsqu'on écrit un script PHP, nous savons très bien à quoi sert telle ou telle ligne de code … Mais que se passe-t-il si, 2 mois ou 3 mois après, vous reprenez votre script pour l'améliorer ? Ou pire, qu'une autre personne doive prendre le relais et améliorer votre script ?

Vous perdez beaucoup de temps à retrouver comment il fonctionne. Surtout si entre temps, vous avez travaillé sur d'autres projets.

C'est pourquoi les commentaires sont très utiles !

Ils vous permettent :

1) De noter dans le code ce à quoi sert telle ou telle ligne.
2) De prévoir éventuellement un « mode d'emploi » pour celui qui devra reprendre le script derrière vous.
3) De désactiver temporairement une ligne de code pour faire des tests , débugger, etc.

Il existe en PHP, 3 façons de faire des commentaires :

```
<?php

// avec 2 slashs '//', on peut commenter une seule ligne

#  avec le dièse '#', on peut commenter une seule ligne également

/* avec cette technique : '/* commentaires … */' (slash + étoile, commentaires, étoile + slash),
on peut commenter plusieurs lignes de code */

?>
```

Les Variables

Nous l'avons vu, les variables en PHP sont reconnaissables par le signe $ placé devant leur nom. On pourrait les comparer à de petites boîtes dans lesquelles on place des choses.

Cependant, pour qu'un nom de variable soit valide, il doit respecter certaines conditions.

- Il peut contenir des minuscules et/ou des majuscules (PHP est sensible à la casse, donc '$mavar' sera différent de '$maVar').
- Il peut contenir des chiffres MAIS ne peut pas commencer par eux. Ex : '$variable5' ou '$_50cents' sont acceptés, mais pas '$50_cents'.
- Seul le '_' (underscore) est accepté. Les autres caractères spéciaux sont interdits. Ex : '$ma_var' ou '$_mavar' sont acceptés, mais '$ma-var' ou '$ma.var' sont interdits.

Exercice :

Indiquer si les variables suivantes sont valides ou non.

Nom de variable	Valide	Non-valide
$HacK3rS		
$Var_coul_255		
$10_pourcent		
$tel fixe		
$__QUESTION		
$tel-mobile		
SAdresse		

(Solution en annexe)

Une variable suivie de crochets est une variable 'tableau', c'est-à-dire contenant plusieurs **valeurs** dont chacune correspond à une **clé**. Entre les crochets, on pourra trouver un nom, un nombre ou une variable.

Ex : $varTab["nom"] ou $varTab[2] ou encore $varTab[$mavar].

On verra plus loin l'utilisation de telles variables.

On peut assigner une valeur à une variable (c'est d'ailleurs le principal intérêt …). Pour se faire, on écrit le nom de la variable, suivi du signe égal '=' puis de sa valeur. Bien sûr, on termine la ligne par un point-virgule ';'.

Exemple :

`$mavar1 = "il était une fois";`	La valeur est une chaîne de caractères.
`$mavar2 = 52;`	La valeur est un nombre entier
`$mavar3 = $mavar2 * 10;`	La valeur est le résultat du calcul de $mavar2 x 10, soit 52 x 10 soit 520.

> Noter que l'utilisation des guillemets (obligatoires car la valeur est une chaîne de caractères) dans le premier exemple : ($mavar1) peut être problématique si on veut afficher un mot entre guillemets !
> Il faudra « échapper » ceux-ci à l'aide d'un anti-slash ('\') pour qu'ils s'affichent tels quels et que ça ne génère pas d'erreur.

Exemple :

`echo "il \"était\" une fois";`	Affichera : il "était" une fois

La première fois qu'on donne une valeur à une variable dans le script, on dit qu'on « initialise » la variable. On peut l'initialiser en lui donnant une valeur comme ci-dessus, ou en lui donnant une valeur vide (ex : `$mavar = "";` ou `$mavar = 0;`)
Si pour une raison ou pour une autre la variable n'est pas initialisée, alors PHP retournera une erreur de type « undefined variable $var … ».

Les Variables Tableaux

Nous avons parlé très brièvement des variables tableaux. Elles sont plus communément appelées « tableaux » tout court.

Alors, qu'elle est la différence avec les autres variables ?

On a vu comment assigner une valeur à une variable :

$mavar = "ma valeur";

Un tableau va pouvoir contenir plusieurs valeurs. Comment ? Tout simplement en utilisant la fonction array() comme ceci :

Exemple :

```php
<?php

# ici un tableau ne contenant qu'une liste de valeurs. chaînes de caractères ou nombre.
$tableau1 = array("valeur1", "valeur2", 125);

# ici un tableau contenant une liste de valeurs associées chacune à une clé.
$tableau2 = array(
   "clé1" => "valeur 1",
   "clé2" => "valeur 2",
   "clé3" => 125
   );

?>
```

Dans le premier tableau ($tableau1), pour afficher une valeur particulière, il faudra la désigner par son numéro dans la liste. ATTENTION ! La numérotation dans un tableau commence à 0 et non à 1.

Pour afficher la première valeur (valeur1) de $tableau1, il faudra donc :

- Appeler notre variable $tableau1
- Lui dire le numéro de la valeur qu'on veut (entre crochets)

Donc, on a par exemple : `echo $tableau1[0];` qui affichera « valeur 1 » ou `echo $tableau1[2];` qui affichera « 125 ».

De même pour afficher la valeur (valeur2) de $tableau2, il faudra :

- Appeler notre variable $tableau2
- Lui dire quelle est la clé de la valeur qu'on souhaite (entre crochet)

Donc par exemple : `echo $tableau2["clé2"];` affichera « valeur 2 » ou `echo $tableau2["clé3"];` affichera « 125 ».

Les Super-Globales

Les variables que nous avons vues précédemment ne sont conservées en mémoire que durant l'exécution du script. En fin de page, elles sont automatiquement effacées. Elles ne peuvent donc pas passer d'une page à une autre.

Pour ce faire, il existe des variables spéciales. Des « super variables » dites « variables globales ». Globales, car elles peuvent être utilisées partout dans le script, et qu'elles permettent (dans une certaine mesure) de passer des données d'une page à une autre.

Il en existe plusieurs, ce sont des variables tableaux et on les reconnaît car leur nom est en majuscule et précédé d'un $ et d'un underscore « _ ». Les clés du tableau correspondent au nom de la variable passée, et les valeurs associées aux valeurs respectives :

- **$_GET** : contient les variables passées par l'URL de la page.

```php
<?php

# l'url de la page est : index.php?page=1&ligne=52
# alors :
echo $_GET["page"];  # affichera "1"
echo $_GET["ligne"]; # affichera "52"

?>
```

- **$_POST** : Contient les variables passées par un formulaire par la méthode POST.

```php
<?php

# le formulaire de la page précédente contenait 2 champs : « login » et « pass »
# alors :
echo $_POST["login"];  # affichera le login entré dans le formulaire
echo $_POST["pass"];   # affichera le mot de passe entré dans le formulaire

?>
```

- **$_SERVER** : Contient des variables concernant le serveur mais également des informations concernant le client (adresse IP, navigateur, etc …).

```php
<?php

echo $_SERVER["REMOTE_ADDR"];      # affichera l'IP de l'internaute
echo $_SERVER["HTTP_USER_AGENT"];  # affichera le navigateur de l'internaute

?>
```

- **$_SESSION** : Contient des variables mises en mémoire dans la session.
- **$_COOKIE** : Contient des variables mises en mémoire dans des cookies
- **$_FILES** : Celle-ci est un peu particulière, on reviendra dessus. Sachez simplement qu'elle permet de récupérer les informations lors d'un téléchargement de fichier sur le serveur (upload).

Les constantes

Une constante, c'est une variable … qui ne varie pas (d'où son nom) !
À la différence d'une variable normale, une constante s'écrit sans "$". Elle nécessite d'être clairement définie avant de pouvoir l'utiliser. Elle sera disponible de façon globale tout comme les variables Super-Globales.

Pour définir une Constante, on utilise la fonction "define()" en lui passant en 1er argument le nom de la constante à créer et en 2e argument la valeur à lui attribuer (qui peut très bien être une chaîne de caractères vide).

Exemple :

```php
<?php

# pour définir une constante :
define('SERVEUR', 'localhost');
# va créer une constante dont le nom sera 'SERVEUR' et dont la valeur sera "localhost".

# pour l'afficher :
echo SERVEUR;

?>
```

> Noter l'absence de "$" devant "SERVEUR" avec le echo.
> On peut très bien utiliser des constantes dans les calculs. Et les constantes sont utilisables aussi bien dans les pages que dans les fonctions, sans avoir à les passer comme paramètres.

Les constantes peuvent être aussi bien en minuscules qu'en majuscule. Par convention, il est commode de les mettre en majuscule car elles seront ainsi plus faciles à repérer dans le code. Il faut toutefois veiller à leur casse car rappelons-le, PHP est sensible à la casse et pour lui, la constante "SERVEUR" sera différente de la constante "Serveur".

La Concaténation

Ouhlà, voici un mot qui paraît bien compliqué pour une fonctionnalité pourtant bien souvent utile. « Concaténer » signifie « Associer » ; on va donc associer différents éléments avec la concaténation. Dans la pratique, en PHP, elle se fait grâce au point « . ».

Exemple 1 :

```php
<?php
echo "Il était une fois $mavar";
?>
```

Dans cet exemple, PHP va bien nous afficher une phrase dans laquelle $mavar aura été remplacé par sa valeur. La concaténation n'est donc pas obligatoire, car la variable est simple.

Mais qu'en est-il si la variable est une variable tableau, c'est à dire du type **$maVarTab[1]** ?

Exemple 2 :

```php
<?php
echo "Il était une fois $maVarTab[1]";  # peut retourner une erreur (selon la version de PHP).
echo "Il était une fois ".$maVarTab[1]; # affichera bien ce que l'on souhaite.
?>
```

Ici, dans le premier cas, PHP ne va pas comprendre s'il doit afficher la valeur de $maVarTab[1], ou la valeur de $maVarTab, suivie de [1]. Comme il ne sait pas, il renvoie une erreur.
Pour lever toute ambiguïté, il suffit de sortir la variable de la chaîne de caractères en utilisant la concaténation. On ferme donc la chaîne de caractères, on met un point et on ajoute la variable.

Dans l'exemple 2, la variable était placée en fin de chaîne. Si elle avait été au milieu, il aurait fallu reprendre et terminer la chaîne de caractères après.

Exemple 3 :

```php
<?php
echo "Il ".$maVarTab[1]." une fois"; # Noter les points avant ET après la variable
?>
```

Souvenez-vous de la note dans le chapitre "Syntaxe PHP", et amusez-vous à faire un test tel que celui-ci :

```php
<?php
$mot = 'Toto';
echo 'Bonjour' . $mot;     # affiche « Bonjour Toto »
echo "Bonjour $mot";    # affiche « Bonjour Toto »
echo 'Bonjour $mot';    # affiche « Bonjour $mot »
?>
```

Les Fonctions

On reconnaît une fonction au fait que c'est un nom, suivi de parenthèses ().
Ex : phpinfo();

Entre les parenthèses, on y glisse parfois des **arguments**.
Ex : rand(1,100);
1 et 100 sont les 2 arguments nécessaires à la fonction rand() pour fonctionner. Elle retournera alors un nombre entier choisi aléatoirement entre 1 et 100.

Certaines fonctions ont besoin d'un ou plusieurs arguments (parfois même 10 arguments !), d'autres n'ont besoin de rien. Ex : phpinfo(); n'a besoin d'aucun argument. Elle retourne une page HTML contenant les informations sur la version et la configuration de PHP installée.
Selon les fonctions, les arguments peuvent être des chaînes de caractères, des nombres ou des tableaux. Bien sûr, ils peuvent être des variables dont les valeurs sont de ces types.

PHP possède de très nombreuses fonctions. Je vous renvoie au manuel PHP pour en prendre connaissance. Il y en a plus de 700 pages et elles sont détaillées et expliquées. Voir en annexe pour les liens et références.

Nous verrons dans ce cours quelques-unes des principales fonctions, qui feront très certainement partie de celles que vous utiliserez le plus souvent.

FONCTIONS COURANTES

Parmi les fonctions que vous utiliserez le plus souvent vous trouverez très certainement les suivantes :

echo, print(), isset(), empty() ou encore include() et require()

echo ou print() ont toutes les deux la même fonction. Elles permettent de renvoyer vers l'affichage.

Par exemple, elles permettent d'afficher du code HTML et donc du texte dans une page.

À la différence de print(), echo n'a pas besoin de parenthèses pour fonctionner correctement. echo est d'ailleurs plus une structure du langage qu'une fonction en elle-même, mais elle se comporte de la même façon.

Il suffit simplement de placer derrière elle l'élément à afficher (chaîne de caractère, nombre, variable ou résultat d'une fonction).

```php
<?php
echo "il fait beau";
echo 123;
echo $toto;
echo maFonction(12,34,56);
?>
```

La fonction print() elle, nécessite bien ses parenthèses. Il suffira de passer l'élément à afficher en paramètre à la fonction print()

```php
<?php
print("il fait beau");
print(123);
print($toto);
print( maFonction(12,34,56) );
?>
```

isset() et empty() permettent toutes les deux de savoir si une variable existe. empty() vérifiera en plus si cette variable est vide. Ces 2 fonctions retourne TRUE (vrai) ou FALSE (faux)

```php
<?php

$note = 15;
if(isset($note))  echo 'La variable $note existe<br/>';
else        echo 'La variable $note n\'existe pas<br/>';

if(empty($note))  echo 'La variable $note existe et est vide (ou = 0) OU n\'existe pas du
tout<br/>';
else        echo 'La variable $note existe et n\'est pas vide (≠ de 0)<br/>';

?>
```

> En général on aura tendance à tester si une variable N'EST PAS vide grâce à !empty($var). Le "!" signifiant "l'inverse de" (donc ici, si $var N'EST PAS vide).

Include() et require() sont 2 fonctions qui permettent d'inclure des fichiers dans d'autres. Elles sont très pratiques et sont particulièrement souvent utilisées par exemple pour inclure dans une page un autre fichier contenant toutes les fonctions utiles au fonctionnement du site. Quand un fichier1 est inclus dans un fichier2, c'est comme si le fichier2 avait phagocyté le fichier1 et avait donc ajouté le code du fichier1 à son propre code.

Elles sont également utilisées pour inclure des morceaux de page comme un en-tête ou un pied de page par exemple. Ce qui permet de réduire de façon très importante le temps de maintenance sur le site (un seul fichier à modifier au lieu de l'ensemble des pages !).

```php
<?php

include('un_dossier/monFichier.html');
include('fichier.php');
include('unAutreFichier.txt');

# et donc de la même façon :
require('un_dossier/unFichier.php');

?>
```

Ces 2 fonctions possèdent chacune une cousine proche :

Include_once() et require_once()

Celles-ci permettent tout comme include() et require() d'inclure un fichier dans un autre. Mais à la différence qu'une vérification est faite avant et que si le fichier demandé à déjà été inclus dans la page une erreur sera retournée.

```php
<?php

include_once('un_dossier/monFichier.html');


# et donc de la même façon :
require_once('un_dossier/unFichier.php');

?>
```

LES FONCTIONS DE DATES

La fonction principale utilisée pour le calcul des dates en PHP est comme son nom l'indique si bien, la fonction "date()".
La fonction date() prend 2 arguments dont le 2e est facultatif.

Le premier argument est une chaîne de caractères contenant des lettres qui seront remplacées par certaines valeurs. Le tableau de correspondance avec l'ensemble des possibilités est consultable en annexe ou bien à l'adresse suivante :
http://php.net/manual/fr/function.date.php

```php
<?php

# affiche la date du jour :
echo date("d-m-Y").'<br/>'; # affiche par ex. "24-12-2008" ('Y' majuscule = année sur 4 chiffres)
echo date("d-m-y").'<br/>'; # affiche par ex. "24-12-08" ('y' minuscule = année sur 2 chiffres)

# affiche la date et l'heure :
echo date("d-m-Y H:i:s").'<br/>';

# Affiche la date suivie de l'heure et les minute, avec du texte
# Notez que le texte est échappé pour ne pas que les lettre soient remplacées par la fonction date()
echo date('d-m-Y H \h\e\u\r\e : i \m\i\n\u\t\e\s').'<br/>';
# l'affichage n'est cependant pas exactement ce que l'on attend …

# ma chaîne contient les caractères \n \r \t qui sont réservé en code
# \n et \r pour les retours à la ligne, et le \t pour une tabulation
# Il faut donc les échaté encore une fois avec un \
echo date("d-m-Y H \h\e\u\\r\e : i \m\i\\n\u\\t\e\s").'<br/>'; # affichera par ex. "24-12-2008 23
heures : 59 minutes"

?>
```

Comme on peut le constater ci-dessus, il est parfois fastidieux d'intercaler des dates et des chaînes de caractères.
La solution la plus simple consiste simplement à utiliser plusieurs fois la fonction date() et de la concaténer avec les chaînes de caractères.

```php
<?php
echo date("d-m-Y H").' heures : '.date("i").' minutes<br/>';
# affichera par ex. "24-12-2008 23 heures : 59 minutes"
?>
```

Un autre problème récurent sur les dates est le calcul de la différence entre 2 dates.
En effet savoir dire si une personne est connectée depuis tant de minutes s'avère indispensable par exemple dans les espaces sécurisés.

Le plus simple pour ce type d'utilisation est d'utiliser le "Timestamp UNIX" qui correspond au nombre de secondes écoulées depuis le 1er janvier 1970 0h GMT.

2 fonctions nous permettent d'avoir accès à ce timestamp : date("U") et time() :

```
<?php
# Afficher le nb de secondes écoulée depuis le 1er janvier 1970 (timestamp UNIX)
echo date("U").'<br/>';
echo time().'<br/>';
?>
```

Connaître ce timestamp, c'est bien, mais savoir à quelle date il correspond est aussi utile.

Justement, le 2e argument facultatif de la fonction date(), c'est notre timestamp.

Par défaut, si on ne le renseigne pas, il correspond au timestamp courant (maintenant quoi).

```
<?php
# Je souhaite afficher la date correspondant à un timestamp
echo date("d-m-Y H:i:s", time()).'<br/>'; # timestamp courant utilisé = valeur par défaut
echo date("d-m-Y H:i:s", 123456789).'<br/>'; # indique alors la date et l'heure correspondant à ce
timestamp précis.
?>
```

Autre problématique, nous avons une date et souhaitons la transformer en un timestamp unix.

On utilise pour ce faire la fonction mktime() qui prend au moins 6 arguments dans cet ordre précis : **HEURES, MINUTES, SECONDES, *MOIS, JOUR*, ANNEE**.

Les valeurs ne doivent pas comporter de "0" devant. Par exemple, inscrire "1" pour 1 heure ou "9" pour le 9e jour du mois. Et donc surtout pas 01 ou 09. Le résultat s'en trouverait faussé.

```
<?php
# mktime($heure, $minutes, $secondes, $mois, $jour, $annee);
# les heures, minutes, seconde, mois et jours ne doivent pas comporter de 0 devant (ex: mettre 1, 2,
 8 et pas 01, 02, 08)
$jourDeNaissance = mktime(10,0,0,12,25,1978);
echo 'Je suis né un '.date("l", $jourDeNaissance).'<br/>';
# Ci-dessus le nom du jour est en anglais.

# Si je veux traduire le nom du jour en français :
# On fait un tableau contenant les traductions
$T_jours = array(
   "Monday" => "Lundi",
   "Tuesday"   => "Mardi",
   "Wednesday" => "Mercredi",
   "Thursday"  => "Jeudi",
   "Friday" => "Vendredi",
   "Saturday"  => "Samedi",
   "Sunday" => "Dimanche",
   );

# On récupère le jour en anglais
$lejour = date("l", $jourDeNaissance);

# On affiche le tout, le nom du jour étant traduit
echo 'Je suis né un '.$T_jours[$lejour].'<br/>';
```

LES FONCTIONS DE TEXTES

Il existe un nombre très important de fonctions destinées à traiter les chaînes de caractères. Nous trouverons entre autres les suivantes, très utiles.

htmlspecialchars() et htmlspecialchar_decode()

htmlspecialchars() est une fonction qui permet automatiquement d'encoder en HTML les caractères spéciaux tels que "&", "<", ">", " ou '. C'est une première étape vers la sécurité, dans le cas où on laisserait la possibilité à un internaute de laisser un message sur un site. En effet, celui-ci pourrait être mal intentionné et en postant un message contenant du code HTML, il pourrait réussir à altérer la présentation du site.
Ces fonctions permettent donc d'éviter très simplement ce genre de désagréments.
Ainsi, le "&" devient alors "&", le "<" devient "<" , le ">" devient ">" et les guillemets doubles et simples deviennent respectivement """ et "'".

```php
<?php
# filter le code HTML
$chaine = '<div><p>Lorem ipsum <a href="chaîne_nettoyee.php">dolor</a> sit amet</p></div>';
echo $chaine.'<br/>'."\n";
# le "\n" permet de créer un retour à l a ligne, mais dans le code source de la page, à la
différence du <br/> qui fait un retour à la ligne dans la page HTML.

echo htmlspecialchars($chaine);
# le 2e argument est une constante qui indique comment la fonction doit traiter les guillemets.
echo htmlspecialchars($chaine, ENT_COMPAT); # filtre les " mais pas les ' (valeur par défaut)
echo htmlspecialchars($chaine, ENT_QUOTES); # filtre les " ET les '
echo htmlspecialchars($chaine, ENT_NOQUOTES); # ne filtre pas les guillemets du tout

# ajout de la gestion du charset :
echo htmlspecialchars($chaine, ENT_QUOTES, 'UTF-8');
?>
```

À l'inverse, htmlspecialchars_decode() permet de passer de l'entité HTML au caractère.
Ex: "<" deviendra "<".

```php
<?php
$chaine = '<div><p>Lorem ipsum <a href="chaîne_nettoyee.php">dolor</a> sit amet</p></div>';
$chaine_filtree = htmlspecialchars($chaine);

echo htmlspecialchars_decode($chaine_filtree);
echo htmlspecialchars_decode($chaine_filtree, ENT_COMPAT); # filtre les " seulement
echo htmlspecialchars_decode($chaine_filtree, ENT_QUOTES); # filtre les " ET les '
echo htmlspecialchars_decode($chaine_filtree, ENT_NOQUOTES); # ne filtre pas les guillemets du tout

# On peut évidemment aussi gérer le charset de la même manière qu'avec htmlspecialchars().
?>
```

htmlentities() et html_entity_decode()

htmlentities(), tout comme htmlspecialchars() vue plus haut, permet de convertir certains caractères en entité HTML. À la différence de htmlspecialchars(), htmlentities() traite l'ensemble des caractères possédant une écriture HTML. Comme par exemple (entre autre) tous les caractères accentués : "é" deviendra alors "é".
Il est également possible de gérer la façon dont les guillemets doivent être traités ainsi que l'encodage de caractères (charset).

```php
<?php
$chaine = "Oué c'est bientôt la fête !";
echo $chaine.'<br />'."\n";

# la fonction htmlentities transforme tous les caractères spéciaux en leur entité HTML
echo  htmlentities($chaine).'<br />'."\n";
echo htmlentities($chaine, ENT_COMPAT); # filtre les " mais pas les '
echo htmlentities($chaine, ENT_QUOTES); # filtre les " ET les '
echo htmlentities($chaine, ENT_NOQUOTES); # ne filtre pas les guillemets

echo htmlentities($chaine, ENT_QUOTES, 'UTF-8');
?>
```

html_entitiy_decode() permet de faire l'inverse et de passer d'une texte converti en un texte normal.

```php
<?php
# la fonction html_entity_decode fait l'inverse
$chaine = "Oué c'est bientôt la fête !";
$chaine_filtree = htmlentities($chaine).'<br />'."\n";
echo $chaine_filtree;

echo html_entity_decode($chaine_filtree).'<br />'."\n";
echo html_entity_decode($chaine_filtree, ENT_COMPAT); # filtre les " mais pas les '
echo html_entity_decode($chaine_filtree, ENT_QUOTES); # filtre les " ET les '
echo html_entity_decode($chaine_filtree, ENT_NOQUOTES); # ne filtre pas les guillemets

echo html_entity_decode($chaine_filtree, ENT_QUOTES, 'UTF-8');
?>
```

> NOTE : il est inutile d'additionner les 2 fonctions htmlspecialchars() et htmlentities() ! En effet, htmlentities() convertira aussi les caractères html comme "<", ">", etc … Le résultat obtenu ne serait pas celui attendu !

strip_tags()

strip_tags() est une fonction qui permet de purement et simplement éliminer toutes les balises HTML d'une texte.
Il est possible, en 2e argument, d'indiquer quelles sont les balises HTML acceptées (les autres seront éliminées). Il suffit juste d'indiquer les balise ouvrante les unes à la suite des autres, sans espace.
C'est une fonction très utile pour filtrer les contenus envoyés par les internautes.

```php
<?php

$chaine = '<div><p>Ma chaîne de caractères</p></div>';

echo $chaine.'<br />'."\n"; # chaîne normale

echo strip_tags($chaine).'<br />'."\n"; # chaîne filtrée

echo strip_tags($chaine, '<p><strong><u><em>'); # chaîne filtrée en conservant les balises <p> par exemple

?>
```

strlen()

strlen() fait partie de ces fonctions utiles pour les vérifications. En effet, elle permet de connaître le nombre de caractères d'une chaîne.
On peut ainsi signaler à l'internaute que tel ou tel texte est trop long par exemple.

```php
<?php

$chaine = "Le PHP c'est super bien !";

# Nombre de caractères dans la chaîne :
echo 'il y a '.strlen($chaine).' caract&egrave;res dans la cha&icirc;ne, espaces compris';

?>
```

addslashes() et stripslashes()

Ces deux fonctions permettent d'éviter certains problèmes dus aux guillemets présent dans les chaînes de caractères. Elles sont particulièrement utiles lors d'affichage ou d'entrée des données dans une base, pour éviter des bugs et autres plantages.

Leur utilisation est très simple :

```php
<?php

$chaine = "Il fait beau aujourd'hui<br/>";
echo addslashes($chaine);


$chaine = "Il fait beau aujourd\'hui<br/>";
echo stripslashes($chaine);

?>
```

Imaginons une requête SQL (ce que nous étudierons plus tard) :

```
"INSERT INTO tableSQL (texte) VALUES ('$chaine')"
```

cette chaîne contiendra donc :

```
"INSERT INTO tableSQL (texte) VALUES ('Il fait beau aujourd'hui')"
```

On s'aperçoit que la chaîne de caractères pose problème à cause de l'apostrophe. Il faut donc l'échapper avec un "\" pour éviter le plantage.
Pour le faire automatiquement on utile la fonction addslashes() :

```
"INSERT INTO tableSQL (texte) VALUES ('".addslashes($chaine)."')"
```

Comme le texte qui sera alors stocké dans la base sera rempli de "\", il faudra le nettoyer avant affichage dans la page via la fonction stripslashes().

utf8_encode() et utf8_decode()

Pour finir cette partie sur les chaînes de caractères, voyons comment gérer un mauvais affichage de certains caractères spéciaux dans une page.
Bien souvent les problèmes d'affichage de caractères sont dus à l'encodage de la page.
En effet, aujourd'hui, pour une plus grande compatibilité et pour faciliter l'accès, surtout sur les sites multilingues, il est recommandé d'utiliser le charset UTF-8.
Si celui-ci est très pratique, il nécessite tout de même une certaine rigueur.

- Les fichiers doivent êtres enregistrés au format UTF-8,
- Le code html doit contenir une balise meta indiquant l'utilisation de l'UTF-8,
- La base de données doit être en UTF-8,
- Les formulaires doivent accepter le charset UTF-8

Ça peut être fastidieux, mais ça apporte un nombre d'avantages importants.
Le problème, c'est que nous ne sommes pas toujours aussi rigoureux et que les navigateurs nous jouent aussi des tours.
Si rien n'est indiqué dans les meta de la page par exemple, c'est l'encodage par défaut du navigateur qui sera utilisé c'est à dire dans nos contrées, le ISO-LATIN-1 (ISO-8859-1).
Et du coup les caractères accentués, s'ils n'ont pas été converti en HTML ne seront pas affiché correctement.

Une solution existe heureusement en PHP !
utf8_encode() et utf8_decode() permettent de passer d'un encodage à l'autre, et vice-versa :

```php
<?php

# Solution pour les encodages de caractères
# Mon fichier est encodé en UTF8

$chaine = "Chaîne avec des accents : é è à ç";

# chaine d'origine, non codée
echo $chaine.'<br />'."\n";

# encode la chaîne depuis le ISO-8859-1 vers l'UTF8
echo utf8_encode($chaine).'<br />'."\n";

# décode la chaîne en UTF8 vers le ISO-8859-1
echo utf8_decode($chaine).'<br />'."\n";

?>
```

LES FONCTIONS TABLEAUX

Il est possible de travailler sur les variables tableaux, notamment et surtout au niveau des tris. Nous verrons également qu'il est possible de passer d'une chaîne de caractère à un tableau et inversement.

Les fonctions de tris

Tout d'abord, voyons la fonction sort() et ses copines : rsort(), asort(), arsort(), ksort() et krsort().

sort() permet, comme son nom l'indique, d'ordonner les éléments d'un tableau.

Elle exécute le tri directement sur le tableau original et ne retourne aucun résultat. Le tri se fait dans l'ordre alphabétique (ou croissant). Mais le couple clé => valeur ne sera pas conservé.

```php
<?php

$tableau = array('fraise', 'banane', 'pomme', 'abricot');
print_r($tableau); # pour voir le tableau original
sort($tableau); # la fonction sort() ne retournant rien, pas besoin de stocker le résultat dans une variable
print_r($tableau); # affiche le tableau trié


$tableau2 = array(
   'prenom' => 'Jack',
   'nom'    => 'BAUER',
   'ville'  => 'Los-Angeles',
   );
sort($tableau2);
print_r($tableau2); # ➔ bonjour le désastre, on perd les clés ! … Il faut utiliser dans ce cas asort() ou ksort() !

?>
```

Les autres fonctions fonctionnent sur le même modèle, mais permettent :

- rsort() : idem que sort(), mais tri le tableau en ordre inverse.
- asort() : tri le tableau en fonction des valeurs et en respectant le couple clé => valeur
- arsort() : idem que asort(), mais avec un tri en ordre inverse
- ksort() : tri le tableau en fonction des clés et en respectant le couple clé => valeur
- krsort() : idem que ksort(), mais avec un tri en ordre inverse

Compter les éléments d'un tableau

Pour savoir combien d'éléments un tableau contient, il suffit d'utiliser la fonction count() :

```php
<?php
$tableau = array('fraise', 'banane', 'pomme', 'abricot');
echo count($tableau); # affiche 4.
?>
```

Création de tableaux depuis une chaîne de caractères

Il est assez facile de passer d'une chaîne de caractères à un tableau et inversement. Si toutefois la chaîne de caractères est correctement formatée selon une nomenclature qui nous sera, elle, personnelle.

Il suffit pour créer un tableau depuis une chaîne de passer par la fonction **explode()**.
Cette fonction prend 2 arguments. Le premier étant une chaîne de 1 ou plusieurs caractères servant de "repère" pour découper le texte passé en 2e argument. Le 2e argument est donc le texte, la phrase, bref, la chaîne de caractères à découper.

```php
<?php

$chaine = "il ne fait pas beau";
$chaine2 = "banane,pamplemousse,fraise,framboise";
$chaine3 = "Carotte | tomate | choux fleur | navet";

# La fonction explode() permet de découper une chaîne de caractères en fonction d'une caractère part
iculier
$tableau  = explode(' ', $chaine);    # On utilise un espace comme séparateur
$tableau2 = explode(',', $chaine2);   # on utilise la virgule comme séparateur
$tableau3 = explode(' | ', $chaine3); # On peut utiliser une chaîne de caractères comme délimiteur
(espace-pipe-espace)

# affichons un des tableaux obtenus :
print_r($tableau);

?>
```

Une fois le tableau obtenu, on peut le compter, le trier, l'afficher comme on le souhaite (via une boucle foreach, voir ci-dessous), …

Maintenant qu'on sait découper une chaîne pour en faire un tableau, faisons l'inverse !
Il faudra pour cela utiliser la fonction **implode()**.
Celle-ci prend également 2 arguments, le premier étant le séparateur qu'on va intégrer à l a chaîne de caractères, le second étant bien sûr le tableau à traiter.

```php
<?php

$fruits = array("banane", "fraise", "kiwi", "mure", "groseille");

# implode() fait exactement l'inverse de la fonction explode()
echo implode(';', $fruits); # va afficher "banane;fraise;kiwi;mure;groseille"

?>
```

La Création de Fonctions

Enfin, sachez que vous pouvez vous-même créer des fonctions utilisables pour vos scripts. L'utilité est d'optimiser ou du moins de "capitaliser" le code de façon à ne pas devoir recopier inutilement des lignes de codes identique à plusieurs endroits et donc de simplifier la maintenance.

Pour définir une fonction personnelle, on utilise le terme **function**, suivi du nom de la fonction et de parenthèses (contenant éventuellement les arguments à passer à la fonction). Juste en dessous viennent les lignes de code, placées entre accolades comme dans l'exemple ci-dessous :

```
<?php
function nom_de_ma_fonction($debut, $fin)
{
   for($i = $debut ; $i <= $fin; $i++)
   {
      echo "Ligne ".$i."<br />";
   }
}
?>
```

La fonction « nom_de_ma_fonction() » définie ci-dessus demande 2 arguments ($debut et $fin). Elle les utilise ensuite pour afficher automatiquement des lignes avec un numéro dont la première aura le numéro $debut et la dernière le numéro $fin.

Pour appeler la fonction dans un script, il faudra simplement la nommer et lui passer les arguments nécessaires entre les parenthèses.
Exemple :

```
<?php
   nom_de_ma_fonction(12, 100);
?>
```

Ici, le résultat sera l'affichage de lignes dont la première sera « Ligne 12 » et la dernière sera « Ligne 100 ».
Si la fonction doit retourner une valeur, il faudra alors utiliser dedans le terme « **return** ».

```
<?php
function calcul($nb1, $nb2, $nb3)
{
   $nombre = $nb1 * $nb2 + $nb3;
   return $nombre; # Si on ne mets pas "return", la fonction ne retournera aucune valeur.
}

echo calcul(10,2,3); # Affichera 23
?>
```

Notez au passage que PHP respecte les règles de priorité mathématiques.

Notion de portée des variables

Une notion importante à connaître quand on commence à concevoir ses propres fonctions PHP est la portée des variables. En d'autres termes, quel est l'espace dans lequel elles "vivent" et dans lequel on peut les utiliser.
En effet, une variable "normale" n'existe et ne vit que dans la page courante (et les pages dans lesquelles celle-ci serait incluse).

Par contre, les variables définies dans les scripts (les pages donc) **NE SONT PAS UTILISABLES** directement dans les fonctions. Il faudra les y transmettre en passant par les arguments.

```php
<?php
$toto = 'youpi';

function maFonction($arg)
{
   $texte = strip_tags($arg);
   $texte = htmlentities($texte);

   # ligne sui ne sert à rien dans la fonction, mais qui est juste là pour montrer la portée des variables) :
   echo $toto; # affichera une erreur car $toto n'existe pas dans la fonction

   return $texte;
}

echo $toto; # affiche "youpi"
echo $texte; # n'affiche rien et retourne une erreur car $texte n'existe pas dans le script mais uniquement dans la fonction.
?>
```

> NOTE : les variables Super-Globales et les constantes sont accessibles aussi bien directement dans les fonctions que dans le script courant.

Il est possible de rendre temporairement globale une variable normale afin de la passer plus facilement à une fonction. Cela se fait dans une fonction via la structure de langage "global" suivi du nom de la variable.

```php
<?php
$toto = 'youpi';

function maFonction()
{
   global $toto;
   $texte = strip_tags($toto);
   $texte = htmlentities($texte);
   return $texte;
}
```

Cette fois, $toto est utilisable dans la fonction, sans avoir à le passer comme argument. $toto ne sera "*globale*" que pour cette fonction là.

Les Boucles

Lorsque des lignes de codes doivent êtres répétés de nombreuses fois, par exemple pour afficher le contenu d'une base de données, ou une liste, etc … il est très utile de faire appel aux « boucles ».
Il en existe 3 en PHP : La boucle « while », la boucle « for » et enfin la boucle « foreach ».

LA BOUCLE WHILE :

C'est la boucle « Tant que l'expression à tester est respectée, je fais l'action ».
Sa structure est simple :

```
<?php

$i = 0;
while($i < 100)
{
   echo 'Je ne dois pas coller mon chewing-gum sous la table<br />';
   $i++;
}

?>
```

Cet exemple va simplement écrire 100 fois la chaîne de caractères indiquée. Bref, elle va faire la punition à votre place, cool non ?
Notez toutefois que cette boucle peut tourner indéfiniment. En effet, si la condition passée en argument est toujours vérifiable (toujours exacte) alors la boucle ne s'arrêtera pas.

Si nous avions mis par exemple :

```
<?php

while(1 < 100)
{
   echo 'Je ne dois pas coller mon chewing-gum sous la table<br />';
}

?>
```

Ici, 1 est forcément toujours inférieur à 100, de ce fait, la condition est toujours respectée et la boucle se fait. Nous sommes alors en présence d'une boucle infinie.
C'est pourquoi au premier exemple, nous avons initialisé une variable $i juste avant la boucle, que nous testons sa valeur et que nous l'incrémentons à chaque boucle.
Lorsque $i sera égale à 100, alors la condition ne sera plus respectée et nous sortirons de la boucle.

LA BOUCLE FOR :

La boucle for (« pour » en anglais) nécessite elle aussi une expression à tester. Mais elle nécessite 3 arguments.

```
<?php
for($i = 0 ; $i < 100 ; $i++)
{
   echo 'Je ne dois pas coller mon chewing-gum sous la table<br />';
}
?>
```

On remarque que le résultat de l'exemple sera le même que l'exemple avec while. Toutefois, on remarque aussi que l'initialisation de la variable $i son test et son incrémentation, sont les 3 arguments nécessaires à la boucle for.

LA BOUCLE FOREACH :

C'est une boucle particulièrement dédiée aux tableaux. En effet, on pourrait la traduire de cette façon : « pour chaque ligne du tableau, je fais … »

```
<?php
$T_var = array('a','b','c','d');

foreach($T_var as $valeur)
{
   echo $valeur . '<br/>';
}
?>
```

ici, nous afficherons donc chacune des valeurs du tableau $T_var avec un retour à la ligne à chaque fois.

Mais nous pouvons aussi récupérer les index qui correspondent aux valeurs du tableau :

```
<?php
$T_var = array(
   'nom'    => 'TERIEUR',
   'prenom' => 'Alain',
   'Adresse'   => '12 rue du courant d\'air'
   );

foreach($T_var as $key => $valeur)
{
   echo $key . ' : ' . $valeur . '<br/>';
}
?>
```

Le code ci-dessus affichera :

```
nom : TERIEUR
prenom : Alain
adresse : 12 rue du courant d'air
```

Structures de contrôle (conditions)

Il y a en PHP, 2 structures de contrôle permettant de tester des variables et de réagir en conséquence. La structure « If … else » et la structure « switch ».

IF … ELSEIF … ELSE

On peut la traduire par Si … Sinon si … Sinon. Chaque test est effectué l'un après l'autre. S'il est vérifié alors on exécute le code correspondant et on passe à la suite du code. Si rien n'est vérifié alors c'est le code présent dans le else qui sera exécuté.

```php
<?php

$note = 12;
if($note < 10)
{
   echo 'Tu n'as pas la moyenne !';
}
else if($note == 10)
{
   echo 'Tu as tout juste la moyenne';
}
else
{
   echo 'Bravo, tu as plus que la moyenne !';
}

?>
```

Toute sorte de test est possible entre les parenthèses. Les opérateurs possibles sont '**==**' (pareil que / égal) ; '**!=**' (différent de) ; '**<=**' (inférieur ou égal) ; '**>=**' (supérieur ou égal).
Il en existe d'autres, mais on ne les verra pas ici, ils permettent en plus de comparer le type de donnée (chaîne de texte, entier, etc …)

Si le « if » est obligatoire, les « else if » et « else » sont eux, facultatifs. On peut aussi mettre autant de « else if » que l'on souhaite alors que pour une même structure de contrôle, il n'y aura qu'un seul « if » et (éventuellement) un seul « else ».

SWITCH

Le switch permet de tester et comparer certaines valeurs, mais pas de conditions. Chaque test est un « cas » (« case » en anglais). Lorsque la valeur testée est identique au cas, le code contenu entre ce cas et l'instruction « break » est exécuté puis on sort du switch. Si l'instruction « break » est oublié, tout le reste du code sera traité.

Enfin, si aucun cas ne correspond à la valeur testée, c'est le code qui suit l'instruction « default » qui sera exécuté. Cette instruction est facultative.

```php
<?php

$valeur_a_tester = 'banane';
switch($valeur_a_tester)
{
   case 'carotte': // notez ici les « : » et non un « ; »
      echo 'La carotte est un légume';
   break;

   case 'banane':
      echo 'La banane est un fruit';
   break;

   default;
      echo 'La valeur n\'est ni une carotte, ni une banane !';
}

?>
```

En-têtes de page

Un en-tête de pages est une information envoyée par le serveur au navigateur pour que celui-ci sache ce qu'il reçoit. Par défaut, une page HTML enverra un en-tête "text/html". Il est possible en PHP de définir quel en-tête envoyer.

Ceci se fait via la fonction header(). On peut donc envoyer un en-tête "Content-type: text/html" mais aussi des en-têtes image comme "Content-type: image/jpeg" par exemple ou des en-tête type "Content-type: application/pdf" lorsqu'on génère des fichiers PDF.

Toutefois, un des en-têtes très pratiques en PHP est l'en-tête de redirection. Celui-ci permet de rediriger automatiquement vers une autre page.

```php
<?php
# redirection vers une autre page :
header("location: autrePage.php");
# la redirection est instantanée et l'internaute ne la verra pas forcément.
?>
```

> **ATTENTION** : Rien ne doit avoir été envoyé au navigateur avant l'appel de la fonction header(). Aucun echo, aucun print(), aucun code HTML ni même le moindre espace avant <?php !!!
> Sans quoi header() renverra une erreur.

Les sessions

Un des problèmes majeurs du protocole http utilisé sur Internet est qu'il est "amnésique". D'une page à l'autre, il ne se souvient plus de ce qu'on a fait sur la page précédente.

Évidemment, il suffit de transférer les variables dans l'url de la page ou bien via un formulaire pour qu'elles soient utilisables sur la page suivante. Mais cela peut être fastidieux à mettre en place, surtout vu le nombre important de liens que peut comporter une page web.

Prenons un exemple tout simple. On demande sur la première page du site le nom de la personne, de façon à pouvoir l'afficher sur les pages suivantes.
La première page du site se compose donc d'un simple formulaire composé d'un champ "nom" et d'un bouton d'envoi bien sûr.
Considérant que le formulaire à été envoyé avec la méthode POST, nous obtiendrons sur la page suivante le code ci-dessous :

```php
<?php
$nomDuVisiteur = $_POST['nom'];
echo $nomDuVisiteur;
?>
```

Si cette page contient des liens vers d'autres pages, celles-ci ignoreront le nom du visiteur étant donné qu'il ne leur sera pas transmis.
Le rajouter automatiquement à chaque lien sera évidemment une solution … à oublier car peu pratique.
Afin de pouvoir utiliser le nom du visiteur sur chacune des pages du site sans avoir à lui redemander ni à le passer avec chaque lien, nous allons utiliser les **SESSIONS**.

Une session, c'est un petit espace mémoire, temporaire, sur le serveur. Nous pouvons y stocker des informations qui nous seront utiles sur plusieurs pages de notre site (panier, login de connexion, langue du site, etc …)
La mise en place et l'utilisation est très très simple, et la procédure toujours la même :

- On démarre la session grâce à la fonction session_start(). Celle-ci doit être le plus haut possible dans la page et aucun caractère HTML ne doit être avant elle, même pas un espace !
- La session utilise la variable Super-Globale $_SESSION. C'est donc un tableau.
- Pour enregistrer une valeur en session, il suffit de l'ajouter au tableau $_SESSION en créant un index.
- Pour utiliser une des valeurs en session, il suffit d'appeler $_SESSION['monIndex']
- Pour détruire une session (pour détruire un panier, ou déconnecter la personne de son espace membre par ex., il faudra utiliser session_destroy()

```php
<?php

# Pour accéder aux données contenues dans la session
# ou pour en insérer dedans, il faut TOUJOURS "démarrer la session"
session_start();

# je récupère ce qui est envoyé par le formulaire
$nomDuVisiteur = $_POST['nom'];


# Pour stocker des données, il suffit de les déclarer
# dans le tableau $_SESSION[]
$_SESSION['visiteur'] = $nomDuVisiteur;

# Si je veux afficher (accéder) à une valeur particulière
# il suffit d'aller chercher l'index voulu dans le tableau
echo $_SESSION['visiteur']; # doit afficher le nom du visiteur entré dans le formulaire

# Pour détruire une session, il suffit de la détruire
session_destroy();

?>
```

Sur les pages suivantes du site, il suffira de démarrer la session et d'appeler $_SESSION['visiteur'] pour afficher le nom du visiteur.

> **ATTENTION** : session_destroy() n'est à utiliser que quand on souhaite vraiment supprimer une session complète. C'est le tableau $_SESSION complet qui sera vidé.

Pour n'effacer qu'une seule donnée dans le tableau, on pourra plutôt utiliser la fonction unset()

```php
<?php
unset($_SESSION['nom']);
?>
```

Les sessions ont en général une durée de vie de l'ordre de 20 minutes, mais ça peut-être variable en fonction des réglages des serveurs.

Les Cookies

Contrairement à ce qu'on pourrait penser, les cookies ne sont pas des gâteaux !
En fait, tout comme les sessions, ils permettent de stocker des informations de façon à pouvoir les utiliser sur plusieurs pages.
Mais à la différence des sessions, les cookies sont stockés sur l'ordinateur de l'internaute. Ce qui à des avantages … et des inconvénients.
Côté avantages, leur durée de vie pourra être plus importante (jusqu'à un an !), et le stockage se faisant sur l'ordinateur du client (l'internaute), nous n'aurons pas à nous inquiéter de la place que ça prend. Pour le client, ça ne représentera qu'un tout petit fichier texte d'un Ko.
Côté inconvénients, et bien c'est justement stocké chez le client, ce qui implique qu'il peut éventuellement l'éditer et donc fausser les données qui sont dedans. Il peut également le supprimer.
Pire, l'internaute peut tout à fait ne pas accepter les cookies sur son ordinateur (par crainte des virus, des pirates, etc …) vu tout le battage médiatique et la psychose créé il y a quelques années.

En ce qui nous concerne, l'utilisation des cookies est très pratique pour des données "optionnelles" sur le site, c'est-à-dire rien de vital ni de sensible.
On évitera d'y stocker évidemment des mots de passe, mais aussi le prix des articles du panier par exemple.

CREATION D'UN COOKIE

Pour créer un cookie, il faut utiliser la fonction setcookie().

```php
<?php
# Création d'un cookie simple, dit de "session" (qui disparaît à la fermeture du navigateur)
setcookie('nomDuCookie', 'valeur');

# On peut indiquer en plus une date de validité, exprimée en secondes (ici : maintenant + 1h donc)
setcookie('nom', 'Guillaume', time()+3600);
?>
```

> **ATTENTION** : le nombre de seconde renvoyé par la fonction time() dépendra du réglage serveur. Or il est possible que celui-ci diffère de l'heure de l'ordinateur de l'internaute. Or ce paramètre est comparé avec l'heure du client.

D'autres réglages existent, je vous invite à consulter le manuel si besoin :
http://fr2.php.net/manual/fr/function.setcookie.php

SUPPRESSION D'UN COOKIE

Pour supprimer un cookie, il suffit d'appeler la fonction setcookie() de la façon suivante :

```php
<?php
# suppression du cookie "nom" :
setcookie('nom');
?>
```

UTILISATION DES DONNEES D'UN COOKIE

Pour utiliser les données stockées dans un cookie, c'est très très simple.

En effet, il suffit d'appeler la Super-Globale $_COOKIE.

```php
<?php
# Pour afficher le contenu du cookie "nom" :
echo $_COOKIE['nom'];
?>
```

REGLES GENERALES ET LIMITATIONS

Il existe quelques limitations concernant l'utilisation des cookies :

- Seul le site qui a créé le cookie peut l'utiliser
- La taille d'un cookie doit être inférieur à 4Ko
- Vous ne pouvez créer que 20 cookies maximum par domaine (par site quoi)

> Rien ne doit être envoyé avant setcookie() (comme pour session_start() par exemple), donc pas de HTML avant ! Pas même un espace !

> La fonction setcookie() peut très bien retourner true (donc qu'elle a bien fonctionnée) nous ne serons néanmoins pas sûr du tout que le cookie ait été accepté par le client. Pour le savoir, il faudra faire un test.

MySQL

Voici notre 2e employé, M. MySQL. Il est l'archiviste dans notre société, il stocke les données et s'occupe de nous les fournir. C'est notre **serveur de Bases de Données**.

Afin d'obtenir des données provenant de la base, il faudra effectuer des requêtes. Celles-ci seront en langage SQL.

Nous allons voir dans un premier temps SQL, ce qu'est un serveur de base de données et le langage SQL en lui-même. Pour nous simplifier la tâche on utilisera pour nos travaux un outil nous permettant de travailler directement avec MySQL, il s'agit de phpMyAdmin, un ensemble de scripts PHP open source permettant une gestion assez complète de MySQL.

Nous verrons ensuite les requêtes principales que nous utiliserons.

Et enfin, parce que notre interlocuteur principal est M. PHP, il nous faudra voir comment celui-ci communique avec M. MySQL. Quelles sont les fonctions principales à utiliser et comment traiter les résultats.

Qu'est-ce que SQL?

SQL est un langage (Structured Query Language = Langage structuré de requêtes) ou plutôt une "norme" mise au point afin d'uniformiser les interactions entre les langages de programmation et les bases de données (BDD). En soit ce n'est donc pas lui qui vous permettra, seul, de créer des site internet dynamiques. Par contre, utilisé correctement avec d'autres langages tels que PHP ou ASP (entre autre) par exemple, il vous comblera de bonheur.

ÇA RESSEMBLE A QUOI ?

À des lettres ☺. En fait comme nous l'avons vu plus haut, SQL n'est qu'un langage ou même titre que l'Anglais ou le Français. Il servira à interagir avec un serveur de bases de données. Et là, il en existe plusieurs. Le plus connu étant certainement MySQL qui est très souvent retrouvé en compagnie de PHP et Apache (c'est le trio infernal du Web). Mais il existe d'autres serveurs de bases de données, par exemple : mSQL, SQLite, PostgreSQL, SQL Server, Oracle, DB2 …

Chacun d'eux nécessitera parfois des adaptations du code SQL utilisé pour les requêtes (au niveau des fonctions utilisées ou des types de champs surtout), mais grosso modo on retrouvera ses petits.

BON ALORS UN SERVEUR DE BASES DE DONNEES, ÇA RESSEMBLE A QUOI ?

Imaginons une personne chargée de s'occuper d'une salle d'archive par exemple. Il travaillera dans une grande pièce ➔ nous appellerons cette pièce le SERVEUR (MySQL au hasard par exemple).

Dans cette pièce, on va trouver plusieurs grosses armoires correspondant par exemple chacune à une société ➔ nous appellerons ces armoires des BASES DE DONNÉES.

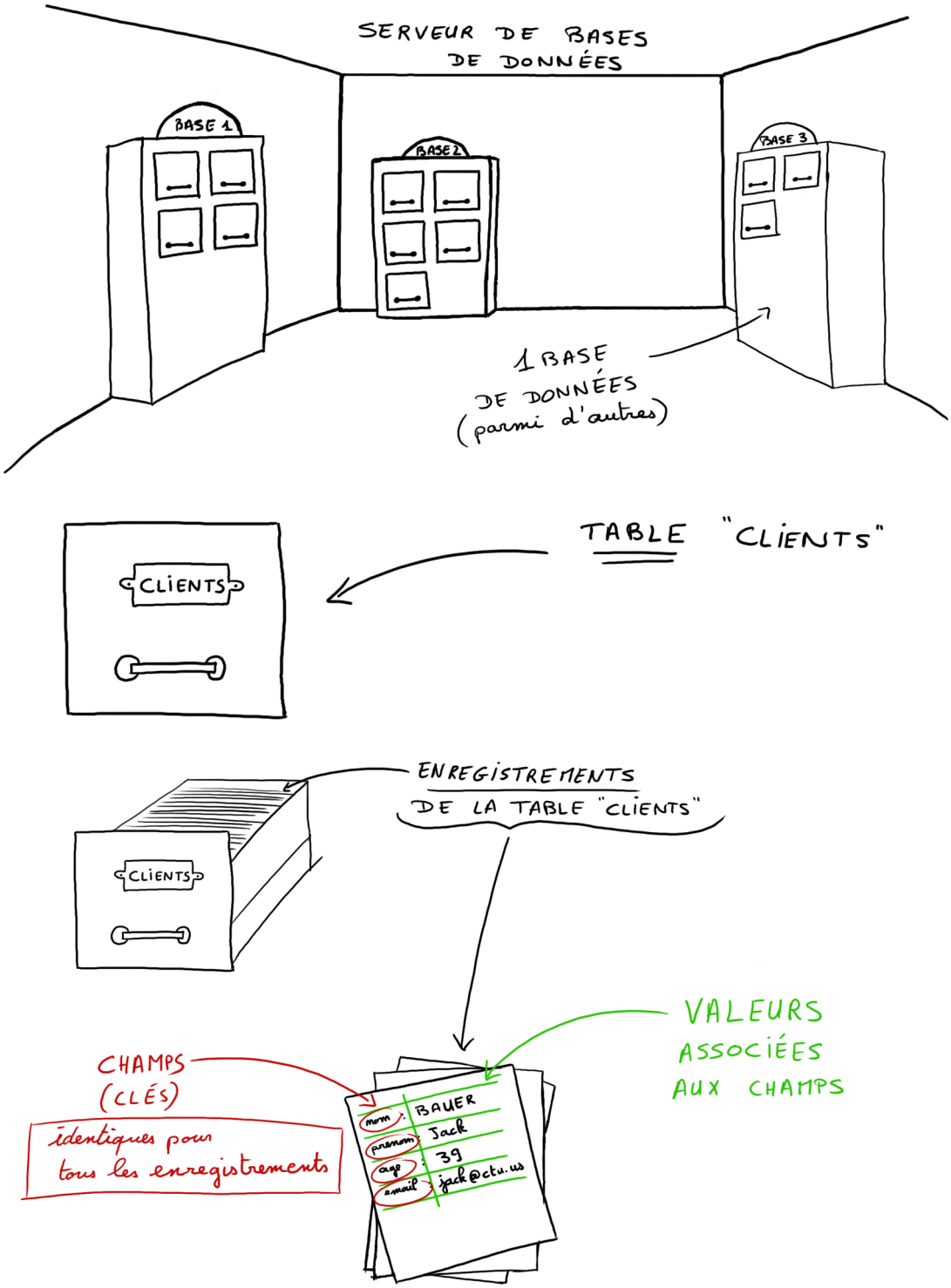
Chacune de ces armoires comporte un ou plusieurs tiroirs ➔ nous appellerons ceux-ci des TABLES.

Dans chacun de ces tiroirs on va trouver des fiches ➔ nous les appellerons les ENREGISTREMENTS. Toutes les fiches d'un même tiroir seront formatées de la même façon. On retrouvera donc le même type de données sur chacune d'elles.

Ces données seront formatées de la façon suivante : « nom » (par exemple nom, prénom…) et « valeur associée au nom » (par exemple Bauer Jack…) ➔ Nous appellerons ça des CHAMPS et des VALEURS

RECAPITULONS

Nous avons donc un SERVEUR contenant des BASES DE DONNÉES dans lesquelles nous trouverons des TABLES avec des ENREGISTREMENTS contenant des CHAMPS et leur VALEUR associée.



Installation

SOUS WINDOWS :

MySQL est installé par défaut avec EasyPHP ou WAMP.

SOUS MACOS X :

Soit vous avez installé MAMP pour travailler votre PHP, auquel cas MySQL est installé par défaut et vous n'avez rien à faire.
Soit vous avez opté pour l'installation Warrior version Geek, et il vous faudra alors l'installer vous-même car il n'est pas installé par défaut sur OS X.
Rassurez-vous, ça reste assez simple tout de même ☺.

Installer le package MySQL compatible avec votre système, téléchargeable là (en anglais) :
http://dev.mysql.com/downloads/mysql/
(version "x86" pour un Mac Intel, "Power_PC" pour un G4-G5 ; 64bits si votre ordi prend en compte cette cadence, c'est la cas pour les machines les plus récentes).
Stoppez et relancez le partage Web (c'est en fait le serveur apache) via les préférences système, vos scripts seront actifs en les plaçant dans le dossier « Sites » de votre dossier de départ. Et seront accessibles dans un navigateur avec une adresse du type :
http://localhost/~nomdutilisateurcourt/monscript.php
Pensez aussi à installer phpMyAdmin :
http://www.phpmyadmin.net/home_page/downloads.php

LES BONS OUTILS

Pour apprendre SQL, il faut peu de chose. Un outil est particulièrement bien fait : phpMyAdmin. C'est un ensemble de scripts PHP conçu pour permettre une gestion très complète des bases de données MySQL.
C'est lui que nous utiliserons et que vous retrouverez à peu près chez tous les hébergeurs. Il en existe d'autres qui font le même travail, mais aucun n'a réussi à s'imposer comme lui.
phpMyAdmin est gratuit, il est téléchargeable à l'adresse suivante :
http://www.phpmyadmin.net/home_page/downloads.php
Il est également déjà pré installé dans les logiciels comme EasyPHP, WAMP ou MAMP.
Si vous pouvez utiliser un éditeur de texte (Dreamweaver, TextMate, Notepad++, …) pour préparer vos requêtes SQL, les commandes que nous passerons à MySQL (en l'occurrence ici), c'est à travers phpMyAdmin que nous les feront exécuter. Evidement, en production, vous ferez exécuter ces requêtes via des scripts PHP (par exemple).

Syntaxe SQL

(My)SQL utilise un certain nombre de mots-clés réservés pour ses commandes. Des mots comme SELECT, DROP, FROM, etc…

Bien qu'il n'y ai aucune obligation ni vraiment de norme, il est de bonne augure que ceux-ci soient toujours écrits en MAJUSCULE. Ça aide à la lecture et à un éventuel débugage.

Nous aurons donc des lignes de commandes qui devraient ressembler à des choses comme ça :

```
SELECT nom, prenom FROM clients WHERE ville='paris'
```

Une mauvaise utilisation de ces mots-clés retournera une erreur. En effet, imaginez donc ceci :

```
SELECT select FROM select WHERE where='select'
```

Le serveur ne fera pas la différence entres les majuscules et les minuscules, le serveur ne comprend donc pas ce qui est demandé (nous non plus d'ailleurs), du coup ça plante.

Mais nous verrons tout ça par la suite ☺

Et vous trouverez la liste des mots réservés (et donc à éviter) en annexe.

Commentaires

Lorsqu'on écrit de longues requêtes SQL, on peut être amené à devoir débugger ou tester certaines parties de la requête en en masquant d'autre. Pour ce faire il suffira d'utiliser des commentaires.

Les commentaires vous permettent :

D'ajouter une information pour nous dans la requête sans l'altérer.

De désactiver temporairement une partie du code pour faire des tests , débugger, etc.

Il existe en SQL, 2 façons de faire des commentaires :

```
-- avec 2 tirets suivi d'un espace '-- ', on peut commenter une seule ligne. On retrouvera ce type de commentaire lors de la sauvegarde de nos bases par exemple.


/* avec cette technique : '/* commentaires … */' (slash + étoile, commentaires, étoile + slash), on peut commenter plusieurs lignes de code ou bien une portion de la requête */

Par exemple :

SELECT prenom, nom FROM clients /* WHERE ville = 'New-York' */ ORDER BY nom

Ci-dessus, la portion suivante : WHERE ville = 'New-York' , est commentée et ne sera donc pas interprétée.

La requête exécuté est donc finalement celle-ci :

SELECT prenom, nom FROM clients ORDER BY nom
```

Passons aux choses sérieuses

Pour simplifier l'apprentissage, nous allons utiliser phpMyAdmin, mais en utilisant une seule fonction de cet outil très pratique : la fenêtre d'exécution de code SQL. Vous y aurez accès via l'une de ces 2 icônes (l'une des deux sera toujours présente) :

Dans la page qui s'ouvre, vous trouverez un grand champ de saisie de texte (pour y copier vos requêtes) et le bouton "Exécuter" (pour exécuter la requête).

Nous allons donc commencer par créer une BASE DE DONNÉES nommée " courssql ".
Pour ce faire, il suffit de taper la commande SQL suivante :

```
CREATE DATABASE courssql
```

Dans la colonne de gauche apparaît maintenant votre base " courssql ".
Pour être utilisable, il faut que la base de donnée possède AU MOINS une table.
L'étape suivante est donc tout logiquement la création d'une table. Celle-ci devra posséder au moins 1 champ et bien sûr dans l'idéal, plusieurs champs. Commençons avec une petite table, que nous améliorerons après.

Pour créer une table dans une base, la syntaxe est la suivante :

```
CREATE TABLE [nom_de_la_base].[nom_de_la_table] (
   [nom_du_champ] [type_du_champ]([taille_du_champs]) [options],
   [nom_du_2e_champ] [type_du_2e_champ]([taille_du_2e_champs]) [options],
   … etc …
);
```

Dans notre cas ici, la table sera la suivante :

```
CREATE TABLE courssql.clients (
   nom VARCHAR(50) NOT NULL,        -- chaîne de 50 caractères maximum
   prenom VARCHAR(50) NOT NULL,     -- chaîne de 50 caractères maximum
   age TINYINT UNSIGNED NOT NULL    -- pas besoin des valeurs négatives ici
);
```

Pour simplifier le code par la suite, il vous suffit dans phpMyAdmin de cliquer sur le nom de votre base de données afin de la sélectionner et nous travaillerons dès lors uniquement dans cette base de données, sans être obligé de le spécifier à chaque fois dans les requêtes
Nous allons maintenant nous amuser un peu avec notre table.

En tant que développeur, vous serez très souvent amenés à faire de nombreuses requêtes SQL. 4 d'entres-elles vont rapidement sortir du lot car vous les utiliserez tous les jours : ce sont les requêtes permettant d'insérer, de sélectionner, de modifier et d'effacer des enregistrements.

> Remarquons aussi plusieurs choses :
> - Il n'y a ni espaces, ni accents dans les noms de mes champs ou de ma table. SQL les accepterait, à condition de protéger les noms avec `nom du champ`. Mais par souci d'éviter des bugs par la suite, la bonne habitude, c'est d'éviter d'en mettre.
> - Les noms de la table et des champs sont en minuscules. Là également, pas d'obligation, mais c'est une habitude. Et c'est du coup plus pratique pour la lecture.
> - Il y un point-virgule à la fin de la ligne. Il n'est pas obligatoire non plus car il n'y a qu'une seule requête, Il le sera par contre dès l'instant où vous exécuterez plusieurs requêtes d'un coup. C'est donc encore une fois une bonne habitude de le mettre.

Décryptons maintenant la requête que nous venons d'effectuer.

```
CREATE TABLE clients
```

Cette partie, comme elle le dit si bien, permet de créer une table ayant pour nom " clients ".
Entre parenthèses juste derrière, nous trouverons les noms des champs et leur type.

```
nom VARCHAR(50) NOT NULL, prenom VARCHAR(50) NOT NULL, age TINYINT UNSIGNED NOT NULL
```

Nous créons ici 3 champs. Le premier a pour nom "prénom". Il est de type "VARCHAR" et d'une longueur de 50 caractères maximum.

VARCHAR signifie "Variable Character" (nombre de caractères variables). Le "(50)" qui le suit aussitôt lui indique qu'il devra faire entre 0 et 50 caractères.
Un VARCHAR peut accueillir un maximum de 255 caractères. Une taille doit toujours lui être indiqué.
Le 2e champs, le champs "nom", est identique au premier.
Le 3e champs aura pour nom "age" et sera de type "TINYINT" et "UNSIGNED".
TINYINT signifie "Tiny Integer" (petit entier). Il accepte des valeurs comprises entre -128 et +127.
UNSIGNED signifie que l'on ne souhaite pas de valeurs négatives. L'intervalle du TINYINT est alors décalé vers les positifs et il devient donc 0 à 255.
VARCHAR et **TINYINT** ne sont que deux types de données parmi d'autre.
Sur les pages suivantes, vous trouverez des tableaux de ceux que vous retrouverez dans phpMyAdmin par exemple ainsi que la place qu'ils vont utiliser en mémoire.

Les types de données dans les champs

POUR LES TYPES NUMERIQUES :

TINYINT	Entier très petit, compris entre -128 et +127. si l'option UNSIGNED est utilisée, ce nombre sera compris entre 0 et 255.	1 octet
SMALLINT	Entier petit compris entre -32 768 et 32 767, si l'option UNSIGNED est utilisée, ce nombre sera compris entre 0 et 65 535.	2 octets
MEDIUMINT	Entier moyen compris entre -8 388 608 et 8 388 607, si l'option UNSIGNED est utilisée, ce nombre sera compris entre 0 et 16777215	3 octets
INT	Entier standard compris entre –2 147 483 648 et 2 147 483 647. Si l'option UNSIGNED est utilisée, ce nombre sera compris entre 0 et 4 294 967 295	4 octets
BIGINT	Entier grand	8 octets
FLOAT	Décimal de simple précision	4 octets
DOUBLE, REAL	Décimal de double précision	8 octets
DECIMAL (entier,décimal)	Réel, définissez la longueur de chacune des deux parties.	variable

Le nombre d'octets détermine la place que prend chaque champ et influe, donc, sur la grandeur des nombres que l'on peut y stocker.
Par exemple, une colonne de type **TINYINT** occupe 1 octet (=8bits). Vous pouvez y stocker 2^8 (256) valeurs différentes. C'est à dire des nombres allant de -128 à +127 si on prend en compte le signe, ou de 0 à 255 si le nombre est "non signé".
Une colonne de type **MEDIUMINT** occupe 3 octets (3x8 bits = 24 bits) et permet donc de stocker 2^{24}, soit 16 777 216, valeurs différentes.
Si le champ **INT** est couramment utilisé, le champ **BIGINT**, lui, l'est assez rarement.
Les types entiers ont un paramètre optionnel permettant de définir le nombre de caractères utilisés pour représenter les valeurs de la colonne, c'est la case "taille/valeur" que l'on trouve dans phpMyAdmin. Cette spécification de taille est utilisée pour remplir à gauche, avec le caractère de remplissage par défaut (l'espace en l'occurrence), les nombres dont la taille est inférieure à celle spécifiée mais uniquement à l'affichage : cela ne réduit pas l'intervalle de validité des valeurs qui peuvent être stockées dans la colonne. Lorsqu'elle est utilisée avec l'attribut de colonne optionnel ZEROFILL, le caractère de remplissage par défaut est remplacé par le caractère zéro. Par exemple, pour une colonne dont le type est INT(5) ZEROFILL, la valeur 4 sera lue 00004.

Nous ne nous attarderons pas sur les types décimaux. Il faut simplement savoir que **FLOAT** a deux paramètres optionnels : le premier indique la taille de stockage à utiliser pour la valeur (par exemple 4octets), le second indique le nombre de chiffres à stocker et afficher derrière la virgule. Le type **DECIMAL** est par contre différent : il est stocké sous forme de chaîne et possède deux paramètres obligatoires, son utilisation dépend néanmoins de votre version de MySQL. Vous pouvez consulter le manuel pour en savoir plus. Un champs de type **DECIMAL(5,2)** accueillera des nombre formatés de la façon suivante : "00000.00".
Concernant le signe, nous avons vu que les champs pouvaient être signés ou non. Les champs sont signés par défaut. Pour demander à MySQL de ne pas tenir compte du signe, il faut ajouter l'argument **UNSIGNED** à la suite de la définition de la colonne (ex : **TINYINT UNSIGNED**).
Enfin, vous pouvez aussi utiliser l'attribut **AUTO_INCREMENT** pour incrémenter automatiquement un champ. On l'utilise généralement avec un champs "id".

LES TYPES CHAINES DE CARACTERES :

CHAR(n)	Chaîne de n caractère, taille fixe	
VARCHAR(M)	Chaîne de caractères variables. M peut être compris entre 1 et 255.	255 caractères max
TINYBLOB, TINYTEXT	Petite zone de texte. Objet d'une longueur maximale de 255 caractères, TINYTEXT aura un contenu de type ASCII (casse insensible) et TINYBLOB aura un contenu de type binaire (casse sensible).	255 caractères Max
BLOB, TEXT	Zone de texte standard Objet d'une longueur maximale de 65535 caractères, TEXT aura un contenu de type ASCII (casse insensible) et BLOB aura un contenu de type binaire (casse sensible).	65 535 caractères max
MEDIUMBLOB, MEDIUMTEXT	Zone de texte moyenne. Objet d'une longueur maximale de 16 777 216 caractères, MEDIUMTEXT aura un contenu de type ASCII (casse insensible) et MEDIUMBLOB aura un contenu de type binaire (casse sensible).	16 millions de caractères max
LONGBLOB, LONGTEXT	Grande zone de texte. Objet d'une longueur maximale de 4 294 967 295 caractères, LONGTEXT aura un contenu de type ASCII (casse insensible) et LONGBLOB aura un contenu de type binaire (casse sensible).	4 milliards de caractères max
ENUM('valeur','valeur2',...)	Une valeur parmi plusieurs Objet texte qui ne peut avoir qu'une des valeurs 'valeur','valeur2',...	65 535 valeurs max.
SET('valeur','valeur2',...)	Une ou plusieurs valeurs parmi plusieurs. Objet texte qui peut avoir une ou plusieurs des valeurs 'valeur','valeur2',...	64 valeurs max.

La table ci dessous illustre les différences de taille entre les deux types CHAR et VARCHAR :

Valeur	CHAR(4)	Zone stockage	VARCHAR(4)	Zone stockage
''	' '	4 octets	''	1 octet (0+1)
'ab'	'ab '	4 octets	'ab'	3 octets (2+1)
'abcd'	'abcd'	4 octets	'abcd'	5 octets (4+1)
'abcdefgh'	'abcd'	4 octets	'abcd'	5 octets (4+1)

Les valeurs retournées seront les mêmes, car dans tous les cas, les espaces seront ignorés.
Il faut noter que les champs de type CHAR et VARCHAR ne sont pas sensibles à la casse. Autrement dit, lors d'une recherche, "texte" sera identique à "TexTe" . En effet, pour rendre ces types de colonnes sensibles à la différence entre majuscule et minuscule, il faut ajouter l'argument BINARY dans la définition du champ (ex : VARCHAR(25) BINARY).
Les quatre types de champs suivants (BLOB et TEXT) n'ont, quant à eux, aucun argument. Ils sont utilisés pour stocker tout type de données (texte, images, etc). Il faut noter qu'une colonne de type BLOB est sensible à la casse tandis qu'une colonne de type TEXT ne l'est pas.

> **NOTE** : Lorsque l'on parle d'être sensible à la casse ou non, cela concerne uniquement les recherches et les tris. Ainsi, un champ de type TEXT peut contenir des majuscules (et les affichera telles quelles) mais ne fera pas la différence entre majuscule et minuscule lors des tris et des recherches.

Enfin, les types ENUM et SET permettent de stocker une ou plusieurs (pour SET) valeurs parmi un ensemble défini de valeurs. On peut donc associer ENUM aux champs de type radio, et SET aux champs de type « case à cocher » (checkbox). A noter que ces deux types peuvent contenir des valeurs NULL.

LES CHAMPS DE TYPES DATE ET HEURE :

DATE	Date (ex: 2000-08-24)	3 octets
TIME	Heure (ex: 23:44:05)	3 octets
DATETIME	Date et heure (ex: 2000-08-24 23:44:05)	8 octets
YEAR	Année (ex: 2000)	1 octet

Le type **DATE** est utilisé pour manipuler simplement une date, sans l'heure. MySQL retourne et affiche les valeurs de type DATE au format 'YYYY-MM-DD'
Le type **TIME** est utilisé pour stocker une heure avec les minutes et les secondes selon le format 'HH:MM:SS'.
Le type **DATETIME** est utile pour manipuler en même temps une date et une heure. . MySQL retourne et affiche les valeurs de type DATETIME au format 'YYYY-MM-DD HH:MM:SS'.
Le type **TIMESTAMP** est équivalent à DATETIME.
Je vous invite à regarder plus en détail le manuel de MySQL concernant les dates pour plus d'information :
http://dev.mysql.com/doc/refman/5.0/fr/date-and-time-types.html

Nous verrons que bien que la date soit stockée au format Américain, il est très simple dans la requête elle-même de la reformater comme nous le souhaiterons. De même nous pourrons récupérer tout ou partie seulement de ce champ.

Insérer des données dans la table

Nous allons donc enregistrer des données dans notre table clients. Nous allons utiliser pour cela la requête **INSERT**. Sa syntaxe est la suivante :

```
INSERT INTO [table] ([champ1], [champ2]) VALUES ([valeur 1], [valeur 2]);
```

Dans notre cas donc :

```
INSERT INTO clients (nom, prenom, age) VALUES ('BAUER', 'Kim', 18);
INSERT INTO clients (nom, prenom, age) VALUES ('BAUER', 'Jack', 36);
INSERT INTO clients (nom, prenom, age) VALUES ('MOSBY', 'Ted', 30);
INSERT INTO clients (nom, prenom, age) VALUES ('HOUSE', 'Gregory', 45);
```

Il existe aussi une version allégée équivalente aux 4 lignes ci-dessus :

```
INSERT INTO clients (nom, prenom, age)
            VALUES   ('BAUER', 'Kim', 18),
                     ('BAUER', 'Jack', 36),
                     ('MOSBY', 'Ted', 30),
                     ('HOUSE', 'Gregory', 45);
```

Comme on peut le constater, les différentes entrées sont simplement séparées par une virgule.

> **NOTE** : Nous ne sommes pas obligés de respecter l'ordre des champs dans la table lorsque nous les indiquons dans la 1e partie de la requête, de même, nous ne sommes pas obligés de tous les mettre, mais en revanche il faut ABSOLUMENT que les valeurs de la 2e partie de la requête correspondent bien aux champs indiqués dans la 1e partie.

Notez enfin que du côté des valeurs, les chaînes de caractères doivent être entourées de guillemets (OBLIGATOIRE !).

Récupérer des données

Pour afficher des données provenant d'une table, il faut aller les **sélectionner**. On va donc utiliser pour cela la requête **SELECT**.

Sa syntaxe est la suivante :

```
SELECT [nom_des_champs] FROM [nom_de_la_table] [AUTRES_CLAUSES];
```

Commençons avec une requête simple (car la requête SELECT peut rapidement se complexifier).

```
SELECT * FROM clients;
```

Cette requête va aller chercher tous les enregistrements contenus dans la table. La petite * signifiant "tous les champs", nous allons récupérer toutes les informations de chacun de nos clients.

Elle est équivalente à l'écriture suivante qui est à préconiser :

```
SELECT nom, prenom, age FROM clients;
```

Il est bien sûr possible de ne sélectionner qu'un ou plusieurs champs en particuliers :

```
SELECT prenom, age FROM clients;
```

Actuellement, nos requêtes nous ont renvoyé l'ensemble des enregistrements de la table. Il est possible de préciser la requête afin de limiter le nombre de résultats. Ça se fait grâce à la clause **WHERE** qu'on pourrait traduire par "Là où".

Si je veux par exemple lister tous les clients de plus de 35 ans :

```
SELECT prenom, age FROM clients WHERE age > 35;
```

> **NOTE** : Les opérateurs :
>
> < (inférieur à),
>
> > (supérieur à),
>
> <= (inférieur ou égal à),
>
> >= (supérieur ou égal à),
>
> <> (différent de),
>
> = (égal) sont utilisables.

Il est possible d'avoir plusieurs conditions dans la clause WHERE. Elles seront alors liées par le mot clé **AND** :

```
SELECT prenom, age FROM clients WHERE age > 30 AND age < 40;
```

Imaginons maintenant que nous voulions trier les résultats par ordre alphabétique de nom par exemple, ce qui sera fort utile dans le cas de longues liste. Il faut pour cela ajouter la clause ORDER BY suivi du nom du champs à trier.

```
SELECT prenom, age FROM clients ORDER BY nom;
```

Par défaut, l'ordre est croissant (alphabétique). Il est possible de demander un ordre décroissant en ajoutant "DESC" après le nom du champ :

```
SELECT prenom, age FROM clients ORDER BY nom DESC;
```

Pour forcer l'ordre alphabétique, vous pouvez aussi ajouter "ASC" après le nom du champ :

```
SELECT prenom, age FROM clients ORDER BY nom ASC;
```

On peut naturellement ordonner en fonction de plusieurs champs :

```
SELECT prenom, age FROM clients ORDER BY nom ASC, age DESC;
```

La requête ci-dessus permettant de classer par nom en ordre alphabétique, puis, s'il y a des noms identiques, de classer ceux-ci par age décroissant.

Il est possible de combiner plusieurs clauses à la suite, **la clause WHERE viendra alors en premier** :

```
SELECT prenom, nom, age FROM clients WHERE age > 15 AND age < 40 ORDER BY nom ASC, age DESC;
```

Nous voudrions maintenant afficher les 2 clients les plus âgés, du plus vieux au plus jeune.
Nous allons pour cela utiliser la clause **LIMIT** qui permet de limiter le nombre de résultats :

```
SELECT prenom, nom, age FROM clients ORDER BY age DESC LIMIT 0,2;
```

> **NOTE** : La clause LIMIT se place TOUJOURS EN DERNIER.

La clause LIMIT prend 2 arguments, dont le premier est facultatif (par défaut, il vaut 0).
En fait, la clause LIMIT X,Y retourne Y enregistrements, à partir de l'enregistrement X des résultats de la requête.
Par exemple :

```
SELECT prenom, nom, age FROM clients ORDER BY age LIMIT 1; -- idem que LIMIT 0,1
```

Retournera un seul enregistrement, le premier qui correspondra au résultat de :

```
SELECT prenom, nom, age FROM clients ORDER BY age;
```

Par contre :

```
SELECT prenom, nom, age FROM clients ORDER BY age LIMIT 1,1;
```

Retournera 1 enregistrement, en l'occurrence le 2e. Eh oui, souvenez-vous, les enregistrements sont comptés à partir de 0 et non de 1. Tout comme c'est le cas en PHP, ASP, JavaScript, et autres langages de programmation.

Mettre à jour des données

Vous serez couramment amenés à devoir modifier des données déjà entrées dans la base de données. Il vous faudra pour cela utiliser la requête UPDATE.
Sa syntaxe est légèrement différente des précédentes requêtes.

```
UPDATE [nom_de_la_table] SET [champ1]=[valeur_champ1], [champ2]=[valeur_champ2] WHERE [condition];
```

La clause WHERE est ici fortement recommandée ! En effet, sans elle, tous les enregistrements de la table seront mis à jour !

```
UPDATE clients SET age = 32 WHERE prenom = 'Jack';
```

Ci-dessus, seuls les clients s'appelant "Jack" (mais TOUS ceux-là) auront leur âge mis à jour à la valeur 32.

Une des erreurs classique est donc l'oubli de la clause WHERE :

```
UPDATE clients SET prenom = 'Ted';
```

Dans ce cas ci, tous nos clients vont désormais s'appeler "Ted". Vous voyez le problème ?

> **NOTE** : Une fois les données mises à jour, n'espérez pas de retour en arrière, ni Pomme Z, ni Crtl Z ne seront possibles.

L'astuce pour sélectionner une ligne en particulier et pas une autre consiste à rajouter un champs ID dans nos tables, qui aura la particularité d'être un index unique qui s'auto incrémente. Cet ID sera très pratique si ce n'est indispensable dans la conception de nos tables et afin de pouvoir mettre à jour une ligne en particulier sans danger (et sans modifier le voisin). La requête ressemblera alors à ça :

```
UPDATE clients SET prenom = 'Ted' WHERE id = 2;
```

Nous verrons un peu plus tard comment modifier la structure d'une table.

Supprimer des données

Par "supprimer une donnée" on doit entendre "supprimer un enregistrement". Ainsi, se seront les nom, prénom et âge de nos clients que nous supprimerons ici (leur fiche complète donc).
Il n'est pas possible de ne supprimer que le prénom par exemple. Il faudra pour cela faire une mise à jour des données en les mettant à "vide".

La suppression d'enregistrement donc se fait grâce à la requête DELETE. Tout comme la requête UPDATE, il est fortement recommandé de l'utiliser avec la clause WHERE.

Sa syntaxe est très simple :

```
DELETE FROM [nom_de_la_table] [autres_clauses];
```

Ainsi la requête suivante :

```
DELETE FROM clients;
```

Effacera … tout le contenu de la table, c'est à dire tous nos clients !

```
DELETE FROM clients WHERE nom = 'BAUER';
```

Celle-ci effacera tous les clients dont le nom est "BAUER".

> **NOTE** : Sauf dans le cas de champs en Binaire, la base de données ne tiendra pas compte de la casse des caractères.

Evidemment, avec l'utilisation d'un ID, nous saurons exactement quel enregistrement effacer.

> **NOTE** : Tout comme avec la requête UPDATE, point de salut si vous vous trompez. Aucun retour en arrière ne sera possible. Si phpMyAdmin reconnaît ce type de requête (potentiellement dangereuse donc) et vous demande une confirmation, il vous faudra la prévoir dans les scripts que vous aurez à créer vous même.

Comment utiliser le SQL avec PHP ?

Maintenant que nous connaissons le code SQL à utiliser pour effectuer des requêtes, la seule chose à savoir, c'est comment demander à PHP de faire la requête pour nous.

Souvenez-vous, on demande toujours à M. PHP de demander à M. MySQL.
Bref, dans la pratique, le code à utiliser est très simple. C'est toujours la même procédure :

1) Je me connecte au serveur de BDD (MySQL donc)
2) Je sélectionne la base de données à utiliser
3) J'exécute la requête
4) Je traite les résultats
5) Je me déconnecte du serveur.

Ce qui nous donne le code suivant :

```php
<?php
#Connexion à MySQL :
mysql_connect($host, $user, $pass);

# Choix de la BDD :
mysql_select_db($bdd);

# Exécution de la requête (je récupère tous les champs de tous les enregistrements de la table $table) :
$sql = mysql_query("SELECT * FROM $table");

# Je traite les résultats :
while($data = mysql_fetch_array($sql))
{
   echo $data['champ1'].' '. $data['champ2'].'<br />';
}

# Je ferme la connexion à la BDD :
mysql_close();
?>
```

> **NOTE** : Il est tout à fait possible, dans le cas ci-dessus, de se déconnecter de la BDD avant d'effectuer le traitement des données. En effet, les données sont stockées dans $sql, rester connecté est désormais inutile.
> De même, si on oublie de fermer la connexion ce n'est pas grave, elle sera automatiquement coupée en fin de script.

Comme on peut le constater, les fonctions entrant en interaction avec MySQL commencent toutes par "mysql_".

mysql_connect() attend 3 arguments : le serveur de connexion, le nom d'utilisateur et le mot de passe de connexion.
Ces 3 informations vous seront fournies par l'hébergeur. En local, on retrouvera en général les données suivantes :

- Serveur : "localhost"
- Utilisateur : "root"
- Mot de passe : rien ou "root"

Le nom de la base de données vous sera également fourni par l'hébergeur, mais vous aurez parfois le choix du nom comme en local.

mysql_query() sera la fonction que vous utiliserez le plus car c'est elle qui exécutera les requêtes. Elle attend une chaîne de caractères en argument (la requête SQL donc).
Elle retourne une ressource (imaginons une pile de fiches) non visualisable directement. Il va falloir la traiter ensuite grâce à une boucle par exemple (regarder chaque fiche séparément).

Pour traiter cette ressource, il existe 4 fonctions :

- mysql_fetch_array()
- mysql_fetch_assoc()
- mysql_fetch_row()
- mysql_fetch_object()

Les 4 fonctionnent a peu près sur le même principe (il y a des différences dans l'affichage des données par la suite) nous utiliseront mysql_fetch_array().
mysql_fetch_array() lit l'enregistrement renvoyé par la BDD et créé un tableau dont les clés sont les champs de la BDD et les valeurs associées sont les valeurs contenue dans la BDD pour cet enregistrement.

Par exemple :

```php
$sql = mysql_query("SELECT * FROM clients WHERE prenom = 'Ted');
$data = mysql_fetch_array($sql);

# le tableau $data pourrait s'écrire de la façon suivante :
$data = array(
   'prenom' => 'Ted',
   'nom'    => 'MOSBY',
   'age'    => 30,
   );
# Heureusement, mysql_fetch_array() permet de faire ça automatiquement !
# Il reste donc à afficher les données comme avec n'importe quelle variable tableau.
```

La boucle while dont on se sert plus haut pourrait être "traduite" en français par "Tant qu'il y a des fiches dans le paquet de fiches $sql, alors je lis une fiche, je créé un tableau $data contenant les données de cette fiche, et je les affiche. Puis je passe à la fiche suivante et je recommence". Lorsqu'il n'y aura plus de fiches (plus d'enregistrements non traités), on passera à la suite du script.

```php
<?php
#Connexion à MySQL :
mysql_connect('localhost', 'root', 'root');

# Choix de la BDD :
mysql_select_db('courssql');

# Exécution de la requête (je récupère tous les champs de tous les enregistrements de la table clients) :
$sql = mysql_query("SELECT * FROM clients");

# Je traite les résultats :
while($data = mysql_fetch_array($sql))
{
   echo $data['prenom'].' '. $data['nom'].' '. $data['age'].'<br />';
}

# Je ferme la connexion à la BDD :
mysql_close();
?>
```

DEBUGGAGE AVEC MYSQL

Il arrive souvent que les requêtes ne fonctionnent pas. En général de notre faute (faute de frappe, mauvaise syntaxe, …).

MySQL ne retourne pas toujours d'erreur mais rien ne s'affiche. Ou bien il y a parfois de nombreuses erreurs (surtout s'il y a une boucle).

Afin de nous aider à trouver une solution, il existe 2 fonctions utiles : die() et mysql_error().

die() permet de faire mourir le script à la moindre erreur, ce qui permet d'éviter de donner trop d'informations à un éventuel internaute malintentionné.

mysql_error() nous permet de récupérer l'erreur générée et donc de trouver plus facilement pourquoi ça bug.

L'utilisation est très simple :

```php
$sql = mysql_query("SELECT * FROM clients") or die(mysql_error());
```

En d'autres termes, on lui dit : "*Exécute la requête et si tu n'y arrive pas, tu meurs et dans ton dernier souffle, tu nous dis pourquoi*". On est sympa avec MySQL non ? ☺

ANNEXES

CORRECTION DE L'EXERCICE SUR LES VARIABLES :

Nom de variable	Valide	Non-valide
$HacK3rS	X	
$Var_coul_255	X	
$10_pourcent		X
$tel fixe		X
$__QUESTION	X	
$tel-mobile		X
SAdresse		X

PARAMETRES DE LA FONCTION DATE()

Caractères	Description	Exemple de valeurs retournées
Jour	---	---
d	Jour du mois, sur deux chiffres (avec un zéro initial)	01 à 31
D	Jour de la semaine, en trois lettres (et en anglais)	Mon à Sun
j	Jour du mois sans les zéros initiaux	1 à 31
l ('L' minuscule)	Jour de la semaine, textuel, version longue, en anglais	Sunday à Saturday
N	Représentation numérique ISO-8601 du jour de la semaine (ajouté en PHP 5.1.0)	1 (pour Lundi) à 7 (pour Dimanche)
S	Suffixe ordinal d'un nombre pour le jour du mois, en anglais, sur deux lettres	st, nd, rd ou th. Fonctionne bien avec j
w	Jour de la semaine au format numérique	0 (pour dimanche) à 6 (pour samedi)
z	Jour de l'année	0 à 366
Semaine	---	---
W	Numéro de semaine dans l'année ISO-8601, les semaines commencent le lundi (ajouté en PHP 4.1.0)	Exemple : 42 (la 42ème semaine de l'année)
Mois	---	---
F	Mois, textuel, version longue; en anglais, comme January ou December	January à December
m	Mois au format numérique, avec zéros initiaux	01 à 12
M	Mois, en trois lettres, en anglais	Jan à Dec
n	Mois sans les zéros initiaux	1 à 12
t	Nombre de jours dans le mois	28 à 31

Année	---	---
L	Est ce que l'année est bissextile	1 si bissextile, 0 sinon.
o	L'année ISO-8601. C'est la même valeur que Y, excepté que si le numéro de la semaine ISO (W) appartient à l'année précédente ou suivante, cette année sera utilisé à la place. (ajouté en PHP 5.1.0)	Exemples : 1999 ou 2003
Y	Année sur 4 chiffres	Exemples : 1999 ou 2003
y	Année sur 2 chiffres	Exemples : 99 ou 03
Heure	---	---
a	Ante meridiem et Post meridiem en minuscules	am ou pm
A	Ante meridiem et Post meridiem en majuscules	AM ou PM
B	Heure Internet Swatch	000 à 999
g	Heure, au format 12h, sans les zéros initiaux	1 à 12
G	Heure, au format 24h, sans les zéros initiaux	0 à 23
h	Heure, au format 12h, avec les zéros initiaux	01 à 12
H	Heure, au format 24h, avec les zéros initiaux	00 à 23
i	Minutes avec les zéros initiaux	00 à 59
s	Secondes, avec zéros initiaux	00 à 59
u	Microsecondes (ajouté en PHP 5.2.2)	Exemple : 54321
Fuseau horaire	---	---
e	L'identifiant du fuseau horaire (ajouté en PHP 5.1.0)	Exemples : UTC, GMT, Atlantic/Azores
I (i majuscule)	L'heure d'été est activée ou pas	1 si oui, 0 sinon.
O	Différence d'heures avec l'heure de Greenwich (GMT), exprimée en heures	Exemple : +0200
P	Différence avec l'heure Greenwich (GMT) avec un deux-points entre les heures et les minutes (ajouté dans PHP 5.1.3)	Exemple : +02:00
T	Abréviation du fuseau horaire	Exemples : EST, MDT ...
Z	Décalage horaire en secondes. Le décalage des zones à l'ouest de la zone UTC est négative, et à l'est, il est positif.	-43200 à 50400
Date et Heure complète	---	---
c	Date au format ISO 8601 (ajouté en PHP 5)	2004-02-12T15:19:21+00:00
r	Format de date » RFC 2822	Exemple : Thu, 21 Dec 2000 16:01:07 +0200
U	Secondes depuis l'époque Unix (1er Janvier 1970, 0h00 00s GMT)	Voir aussi time()

Liens et livres utiles

LIENS :

Codes source et ressources du cours :

http://cours.guillaumehenot.com/php/

Initiation PHP / Cours :

http://www.phpdebutant.org

http://www.lesiteduzero.com

http://www.apprendre-php.com/

Manuel PHP téléchargeable ou en ligne :

http://www.php.net/fr/manual/

http://www.nexen.net

Ressources diverses & aide :

http://www.developpez.com

http://www.mozilla.org/developer/

http://www.alsacreations.com/

LOGICIELS :

WAMP : http://www.wampserver.com/

MAMP : http://www.mamp.info/en/index.html

MySQL (pour une installation à part) : http://dev.mysql.com/downloads/mysql/

phpMyAdmin (si besoin aussi) : http://www.phpmyadmin.net/home_page/downloads.php

LIVRES

La Bible PHP5 (Micro Application)

PHP5 – Le guide complet (Micro Application)

PHP5 & SQL "Mémento" (Eyrolles)

MySQL "Mémento" (Eyrolles)

SQL pour les nuls (First Interactive)