IAI-TOGO

COURS DE PROGRAMMATION WEB 2

PHP-MySQL

AMEVOR Kossi A. / AGBOKA / ATTIOGBE

Année académique 2021-2022

Sommaire

Généralités sur les langages informatiques

On appelle langage informatique un langage destiné à décrire l'ensemble des actions consécutives qu'un ordinateur doit exécuter. Les langages naturels (l'anglais, le français) représentent l'ensemble des façons qu'ont un groupe d'individu de communiquer. Les langages servant aux ordinateurs à communiquer n'ont rien à voir avec des langages informatiques, on parle dans ce cas de protocoles, ce sont deux notions totalement différentes. Un langage informatique est une façon pratique pour nous (humains) de donner des instructions à un ordinateur. Un langage informatique est rigoureux : à CHAQUE instruction correspond UNE action du processeur. Le langage utilisé par le processeur, c'est-à-dire les données telles qu'elles lui arrivent, est appelé langage machine. Il s'agit d'une suite de 0 et de 1 (du binaire) mais pour plus de « clarté » il peut être décrit en hexadécimal. Toutefois le langage machine n'est pas compréhensible facilement par l'humain moyen. Ainsi il est plus pratique de trouver un langage intermédiaire, compréhensible par l'homme, qui sera ensuite transformé en langage machine pour être exploitable par le processeur. L'assembleur est le premier langage informatique qui ait été utilisé. Celui-ci est encore très proche du langage machine mais il permet déjà d'être plus compréhensible. Toutefois un tel langage est tellement proche du langage machine qu'il dépend étroitement du type de processeur utilisé (chaque type de processeur peut avoir son propre langage machine). Ainsi un programme développé pour une machine ne pourra pas être porté sur un autre type de machine (on désigne par portable un programme qui peut être utilisé sur un grand nombre de machines).
Pour pouvoir l'utiliser sur une autre machine il faudra alors parfois réécrire entièrement le programme. Les langages informatiques

peuvent grossièrement se classer en deux catégories : les langages interprétés et les langages compilés.

1. Langage interprété

Un langage informatique est par définition différent du langage machine. Il faut donc le traduire pour le rendre intelligible du point de vue du processeur. Un programme écrit dans un langage interprété a besoin d'un programme auxiliaire (l'interpréteur) pour traduire au fur et à mesure les instructions du programme

2. Langage compilé

Un programme écrit dans un langage dit « compilé » va être traduit une fois pour toutes par un programme annexe (le compilateur) afin de générer un nouveau fichier qui sera autonome, c'est-à-dire qui n'aura plus besoin d'un programme autre que lui pour s'exécuter (on dit d'ailleurs que ce fichier est exécutable). Un programme écrit dans un langage compilé a comme avantage de ne plus avoir besoin, une fois compilé, de programme annexe pour s'exécuter. De plus, la traduction étant faite une fois pour toute, il est plus rapide à l'exécution. Toutefois il est moins souple qu'un programme écrit avec un langage interprété car à chaque modification du fichier source (fichier intelligible par l'homme : celui qui va être compilé) il faudra recompiler le programme pour que les modifications prennent effet.

D'autre part, un programme compilé a pour avantage de garantir la sécurité du code source. En effet, un langage interprété, étant directement intelligible (lisible), permet à n'importe qui de connaître les secrets de fabrication d'un programme et donc de copier le code voire de le modifier. Il y a donc risque de non-respect des droits d'auteur. D'autre part, certaines applications sécurisées nécessitent la

confidentialité du code pour éviter le piratage (transaction bancaire, paiement en ligne, communications sécurisées, etc.).

I. Qu'est-ce que PHP

I.1 Définition

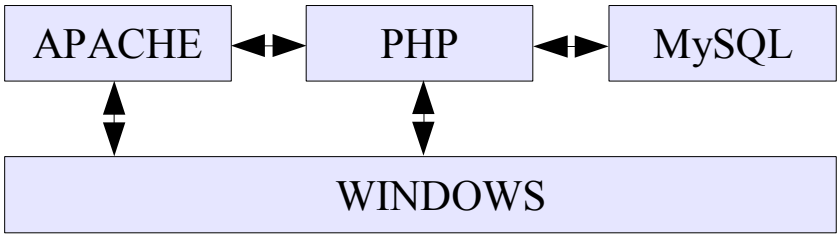
PHP est un langage de script HTML exécuté du côté du serveur. Il veut dire « PHP : Hypertext Preprocessor ». Sa syntaxe est largement inspirée du langage C, de Java et de Perl, avec des améliorations spécifiques. Le but du langage est d'écrire rapidement des pages HTML dynamiques.

I.2 Utilisation pratique

- Forums et Messageries
- Commerce électronique
- Banque / Comptes en ligne
- Publication en ligne
- Moteurs de recherche
- Tout ce que vous voulez, (sauf les jeux)

Résumons quelques sigles que nous allons utiliser par la suite.

- **HTML** : Hypertext Markup Language
- **PHP** : Hypertext PreProcessor
- **SQL** : Structured Query Language
- **MySQL** : serveur de base de données et les outils pour y accéder
- **WAMP** : Windows – Apache – MySQL – PHP, le quatuor gagnant des serveurs web.



I.3 Pages statiques vs pages dynamiques

Une page statique est une page écrite directement en HTML. Elle peut éventuellement incorporer du code Javascript lui donnant un semblant de « dynamisme » mais uniquement du côté du navigateur et des données locales.

Pour des traitements plus lourds nécessitant l'accès à une base de données, un formatage de tableau en fonction de résultats, une recherche poussée, du graphisme, il faut passer par des pages dynamiques et par un langage qui sera exécuté du côté du serveur : ASP sur les serveurs Microsoft/IIS, Perl, PHP…

I.4 Pages dynamiques et PHP

PHP est un langage Server Side ou côté serveur. Lors du chargement d'une page PHP, c'est le serveur qui va lire, interpréter et exécuter le code. Puis il renvoie le résultat, généralement sous la forme de code HTML au navigateur. Ainsi le navigateur et l'utilisateur ne voient jamais le véritable code PHP exécuté. De plus le résultat étant une page web classique en HTML, pas besoin d'installer sur le client des composants spécifiques (java, …). Il n'y a donc pas de notion de vitesse d'exécution du côté du client, mais celle du serveur est prépondérante.

I.5 Le nécessaire serveur

PHP ne sert pas seulement à faire des pages dynamiques. C'est un langage interprété qui au même titre que Perl, Python ou TCL est capable de lancer des scripts intéractifs ou non. Le minimum nécessaire et vital pour apprendre PHP est donc l'interpréteur PHP lui-même sur un environnement supporté (Unix, Windows, Mac, …).

I.6 Le nécessaire client

Pour développer il suffit d'un simple éditeur mais il vaut mieux préférer un éditeur plus évolué supportant la coloration syntaxique et quelques fonctions évoluées.

Pour les tests : un simple navigateur respectant les standards du web.

I.7 Le respect des standards

Le W3C est l'organisme international faisant loi dans le monde des standards du web. Il a défini une série de normes dont le HTML, le XML, le XHTML, les CSS, etc. Pourquoi respecter un standard ? C'est la garantie d'un bon fonctionnement et du bon affichage de manière identique de vos pages sur tous les navigateurs supportant ce standard

II. Les Bases : Présentation

PHP est un langage très souple prenant ses sources dans divers langages comme le C, le Perl, le C++. Il est donc possible d'avoir plusieurs styles de scripts (programmation classique dite procédurale ou programmation objet, ou programmation bordélique). Cette souplesse permet une très grande liberté, un peu comme en Perl. L'inconvénient est qu'on peut très vite obtenir du code illisible (bordélique), même si ça marche très bien. Prenez donc l'habitude de commenter votre code, de l'indenter et de ne placer qu'une instruction par ligne.

II.1 Intégration à HTML

Une page php porte l'extension « .php ». Une page PHP peut être entièrement programmée en PHP ou mélangée avec du code html. PHP est un langage « Embedded HTML », c'est à dire qu'il apparaît à n'importe quel endroit de la page HTML. Pour ça on le place dans des balises particulières : <?php et ?>. On peut aussi utiliser les balises <script language="php"> et </script>. La première forme est préférable pour plus de simplicité et une compatibilité XHTML. On écrit donc une page HTML dans laquelle on intègre du code PHP.

```
<html>
    <head>
        <title>Titre de la page</title>
    </head>
<body>
     <?php
        echo "Hello World !";
     ?>
</body>
</html>
```

Le code HTML généré sera le suivant

```
<html>
    <head>
        <title>Titre de la page</title>
    </head>
<body>
        Hello world!;
</body>
</html>
```

L'utilisation de balises pour l'intégration de code dans une page web est très souple et permet de jongler facilement avec du code PHP et du code HTML :

```
<? php if (expression) { ?>
  <strong>Ceci est vrai</strong>
<?php } else { ?>
<strong>Ceci est faux.</strong>
<?php } ?>
```

II.2 Séparateur d'instructions

Comme en C une instruction se termine par un point-virgule « ; ». Notez que la balise fermante ?> implique la fin d'une instruction.

```
<?php
    echo "Ceci est un test";
?>
<?php
    echo "Ceci est un test"
?>
```

II.3 Bloc d'instructions

Un bloc d'instructions se place entre accolades { et }. Un bloc d'instructions peut contenir du code de n'importe quelle longueur et

est considéré dans le reste du code comme une instruction unique. Si c'est une expression (qui a une valeur) on peut donc assigner le bloc, faire des calculs, ...

II.4 Commentaires

Les commentaires s'utilisent comme en C et en C++ avec /* .. */ et //. Notez qu'une balise fermante ferme le commentaire en cours.

```
<?php
/* echo "salut !" */
// Commentaire sur cette ligne
?>
```

III. Les variables

III.1 Déclarer une variable

Une variable commence par un dollar « $ » suivi d'un nom de variable. Les variables ne sont pas typées au moment de leur création. Attention PHP est sensible à la casse : var et Var ne sont pas les mêmes variables.

Voici les règles à respecter :

- Une variable peut commencer par une lettre
- Une variable peut commencer par un souligné (underscore) « _ »
- **Une variable ne doit pas commencer par un chiffre.**

```
// Déclaration et règles
$var=1; // $var est à 1
$Var=2; // $ Var est à 2
$_toto='Salut'; // Ok
$3petitscochons=5; // Invalide : commence par un chiffre
```

Leur type dépend de leur valeur et de leur contexte d'utilisation. Mais on peut forcer (cast) ponctuellement une variable à un type de données, ce qui s'appelle le transtypage. De même comme le type de variable peut changer en fonction de son utilisation ou du contexte, PHP effectue automatiquement un transtypage, ce qui peut parfois fournir des résultats surprenants. On affecte une valeur à une variable avec le signe égal « = » avec ou sans espace avant ou après.

```
// Déclaration et transtypage
$var = '2'; // Une chaîne 2
$var + = 1; // $var est maintenant un entier 3
$var = $var + 0.3; // $var est maintenant un réel de type double 3.3
$var = 5 + "3 petits cochons"; // $var est un entier qui vaut 8
```

Par défaut les variables sont assignées par valeur : la valeur assignée à la variable est recopiée dans la variable.

Exemple :

```
//Affectation par valeur de $mavar1 et $mavar2
$mavar1="Paris";
echo "\$mavar1= ",$mavar1,"<br />";
$mavar2="Lyon";
echo "\$mavar2= ",$mavar2,"<br />";
```

Le résultat de l'exécution de ce script donne :
$mavar1= Paris
$mavar2= Lyon

III.2 Portée des variables

La portée d'une variable dépend du contexte. Une variable déclarée dans un script et hors d'une fonction est globale mais par défaut sa portée est limitée au script courant, ainsi qu'au code éventuellement inclus (include, require) et n'est pas accessible dans les fonctions ou d'autres scripts.

```
$a = 1; // globale par défaut
function foo() {
 echo $a; // c'est une variable locale à la fonction : n'affiche rien
}
```

Pour accéder à une variable globale dans une fonction, il faut utiliser le mot-clé global.

```
$a=1; // globale par défaut
$b=2; // idem
function foo() {
 global $a,$b; // on récupère les variables globales
 $b=$a+$b;
}
```

echo $b;//affiche 3

PHP accepte les variables statiques. Comme en C une variable statique ne perd pas sa valeur quand on sort d'une fonction.

```
function test_static() {
 static $a=0;
 echo $a; // +1 à chaque passage dans la fonction
 $a++;
}
```

III.3 Variables prédéfinies et globales

III.3.1 Variables globales

PHP dispose d'un grand nombre de variables prédéfinies. Ces variables sont généralement de type scalaire ou des tableaux. Elles sont souvent de type superglobales, c'est à dire accessible depuis n'importe où sans notion de portée. Voici quelques tableaux prédéfinis (voir au point Tableaux pour comprendre leur utilisation).

- $_GLOBALS : tableau des variables globales. La clé est le nom de la variable.
- $_SERVER : variables fournies par le serveur web, par exemple 'SERVER_NAME'
- $_GET : variables fournies par HTTP par la méthode GET (formulaires)
- $_POST : idem mais pour la méthode POST
- $_COOKIE : les variables fournies par un cookie
- $_FILES : variables sur le téléchargement d'un fichier (upload)
- $_ENV : accès aux variables d'environnement du serveur
- $_SESSION : les variables de session (voir cours sur les sessions)

III.3.1 Variables prédéfinies

Variable	Description
$_SERVER['DOCUMENT_ROOT']	Racine du serveur
$_SERVER['HTTP_ACCEPT_LANGUAGE']	Langage accepté par le navigateur
$_SERVER['HTTP_HOST']	Nom de domaine du serveur
$_SERVER['HTTP_USER_AGENT']	Type de navigateur
$_SERVER['PATH_INFO']	Chemin WEB du script
$_SERVER['PATH_TRANSLATED']	Chemin complet du script
$_SERVER['REQUEST_URI']	Chemin du script
$_SERVER['REMOTE_ADDR']	Adresse IP du client
$_SERVER['REMOTE_PORT']	Port de la requête HTTP
$_SERVER['QUERY_STRING']	Liste des paramètres passés au script
$_SERVER['SERVER_ADDR']	Adresse IP du serveur
$_SERVER['SERVER_ADMIN']	Adresse de l'administrateur du serveur
$_SERVER['SERVER_NAME']	Nom local du serveur
$_SERVER['SERVER_SIGNATURE']	Type de serveur
$_SERVER['REQUEST_METHOD']	Méthode d'appel du script

III.4 Types de variables

III.4.1 booléens

Un booléen peut prendre deux valeurs TRUE ou FALSE. Les deux constantes TRUE et FALSE peuvent être utilisées sans aucune

distinction de casse (pas de différences entre les majuscules et les minuscules).

```
$var=FALSE; // FALSE, False, false, ...
$var2=True; // TRUE, True, true, ...
```

Comme en C, les tests de conditions dans les structures de contrôles effectuent une conversion booléenne de la condition.

```
if($var == true) echo "ok";
 if($var) echo "ok"; // Identique
```

Exemple :

```
<?php
    $a = 80;
    $b = ($a<95);
    echo $b; // Affiche 1
?>
```

III.4.2 Entiers

Un entier est l'ensemble des nombres naturels, c'est à dire sans virgule, positifs ou négatifs. Les entiers sont généralement codés sur 32 bits mais cela dépend de l'architecture. Si on affecte un nombre entier qui dépasse la capacité de la variable, celle-ci sera transformée en réel (float). Enfin il n'y a pas de notion d'entier non signé.

Lors de la conversion d'un booléen en entier, FALSE devient 0 et TRUE devient 1. Lors de la conversion d'un nombre à virgule flottante, le nombre sera arrondi à la valeur inférieure s'il est positif, ou supérieure s'il est négatif (conversion vers zéro).

```
Exemple : $varint = 1789;
          $varint = –758;
```

III.4.3 Virgule flottante

On parle ici des nombres réels, double ou float, c'est à dire les nombres à virgules. La virgule est spécifiée par le point « . ». La puissance de 10 s'exprime avec le « e » ou le « E ».

```
<?php
$vardbl = 1952.36;
$vardbl2= 1.95236E3;//Soit 1.95236 x 1000
echo $vardbl2,"<br />";//Affiche 1952.36
$vardbl3= 1.95236e3;
echo $vardbl3,"<br />";//Affiche 1952.36
echo $vardbl3*100000000000,"<br />";//Affiche 1.95236E14
?>
```

III.4.4 Chaînes de caractères

Une chaîne est une séquence de caractères. PHP travaille en ASCII soit 256 caractères, mais ne supporte pas encore le format Unicode, prévu dans la version 5. Il n'y a pas de limite théorique pour la taille de la chaîne.

On distingue les syntaxes suivantes pour utiliser une chaîne :

- Les guillemets simples '...' (apostrophes) : Comme en shell, tous les caractères inclus dans la chaîne sont sortis tels quels sans interprétation. Si vous devez afficher un guillemet simple, il faudra l'échapper : \'
- Les guillemets doubles "..." : Certaines séquences de caractères sont interprétées et les variables sont substituées (remplacées par leur valeur).

Voici le tableau issu de la documentation PHP des séquences pouvant être utilisés avec les guillemets doubles.

Séquence	Valeur
\n	Nouvelle ligne (linefeed, LF ou 0x0A (10) en ASCII)
\r	Retour à la ligne (carriage return, CR ou 0x0D (13) en ASCII)
\t	Tabulation horizontale (HT ou 0x09 (9) en ASCII)
\\	Antislash
\$	Caractère $
\"	Guillemets doubles
\[0-7]{1,3}	Une séquence de caractères qui permet de rechercher un nombre en notation octale.
\x[0-9A-Faf]{1,2}	Une séquence de caractères qui permet de rechercher un nombre en notation hexadécimale.

```
Echo 'Attention l\'abus d\'alcool est dangereux'
$var = 2345;
echo "la valeur de \$var est $var\n";
```

On peut ajouter du texte à une chaîne avec l'opérateur point égal «.=».

```
$str="Salut les Amis ! ";
$str.="Comment ça va ?"; // "Salut les amis  ! Comment ça va  ? "
$str2 = $str."ce soir"; // "Salut les Amis ! Comment ça va ce soir ? "
```

Si vous devez utiliser une chaîne de caractères comme valeur numérique (dans une addition par exemple, attention à son contenu. La chaîne sera de type double (réel) si elle contient un 'e' ou un 'E'. Sinon ce sera un entier. La valeur numérique est ensuite définie par le début de la chaîne. Si la chaîne commence par une valeur numérique, elle sera utilisée, sinon elle sera égale à 0 zéro. Si la première

expression est une chaîne, le type de variable dépend de la seconde expression.

$val = 10+"2.55"; // float, 12.55
$val = 1+"toto2"; // 1 + 0 = 1
$val = 2+"3 petits cochons"; // 2 + 3 = 5 (le 3 est en premier dans la chaîne)

III.5 Les tableaux

Un tableau PHP est une association ordonnée. Une association fait correspondre des valeurs à des clés.

Un tableau est créé avec la fonction array() qui prend comme arguments des paires « key => value » séparées par des virgules. La clé peut être soit un entier soit du texte. Attention, 8 est un entier, 08 une chaîne ! Si la clé est absente alors c'est la dernière clé entière plus 1 qui est choisie. Si c'est la première, c'est 0 zéro.

On accède aux éléments d'un tableau à l'aide des crochets « [et] ». On place entre ces crochets la clé entière ou la chaîne.

III.5.1 Les types de tableaux

On distingue deux types de tableau :

- Les tableaux numérotés ;

Ces tableaux sont très simples à imaginer. Regardez par exemple celui-ci, contenu de la variable $prenoms :

Clé	Valeur
0	François
1	Michel
2	Nicole
3	Véronique
4	Benoît
…	…

$prenoms est un array : c'est ce qu'on appelle une variable « tableau ». Elle n'a pas qu'une valeur, mais plusieurs (vous pouvez en mettre autant que vous voulez).

Dans un array, les valeurs sont rangées dans des « cases » différentes. Ici, nous travaillons sur un array numéroté, c'est-à-dire que chaque case est identifiée par un numéro. Ce numéro est appelé clé.

- **Construire un tableau numéroté**

Pour créer un tableau numéroté en PHP, on utilise généralement la fonction array.

Cet exemple vous montre comment créer l'array $prenoms :

```php
<?php
// La fonction array permet de créer un array
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique', 'Benoît');
?>
```

L'ordre a beaucoup d'importance. Le premier élément (« François ») aura le n°0, ensuite Michel le n°1, etc.
Vous pouvez aussi créer manuellement le tableau case par case :

```php
<?php
$prenoms[0] = 'François';
$prenoms[1] = 'Michel';
$prenoms[2] = 'Nicole';
?>
```

Si vous ne voulez pas avoir à écrire vous-mêmes le numéro de la case que vous créez, vous pouvez laisser PHP le sélectionner automatiquement en laissant les crochets vides :

```php
<?php
$prenoms[] = 'François'; // Créera $prenoms[0]
$prenoms[] = 'Michel'; // Créera $prenoms[1]
$prenoms[] = 'Nicole'; // Créera $prenoms[2]
?>
```

Pour afficher un élément, il faut donner sa position entre crochets après $prenoms. Cela revient à dire à PHP :
« Affiche-moi le contenu de la case n°1 de $prenoms. »
Pour faire cela en PHP, il faut écrire le nom de la variable, suivi du numéro entre crochets. Pour afficher « Michel », on doit donc écrire :

```
<?php
echo $prenoms[1];
?>
```

- Les tableaux associatifs

Les tableaux associatifs fonctionnent sur le même principe, sauf qu'au lieu de numéroter les cases, on va les étiqueter en leur donnant à chacune un nom différent.

Par exemple, supposons que je veuille, dans un seul array, enregistrer les coordonnées de quelqu'un (nom, prénom, adresse, ville, etc.). Si l'array est numéroté, comment savoir que le n°0 est le nom, le n°1 le prénom, le n°2 l'adresse... ? C'est là que les tableaux associatifs deviennent utiles.

- **Construire un tableau associatif**

Pour en créer un, on utilisera la fonction ***array*** comme tout à l'heure, mais on va mettre « l'étiquette » devant chaque information :

```
<?php
// On crée notre array $coordonnees
$coordonnees = array (
'prenom' => 'François',
'nom' => 'Dupont',
'adresse' => '3 Rue du Paradis',
'ville' => 'Marseille');
?>
```

Vous remarquez qu'on écrit une flèche (=>) pour dire « associé à ». Par exemple, on dit « ville » associée à « Marseille ».

Nous avons créé un tableau qui ressemble à la structure suivante :

Clé	Valeur
prenom	François
nom	Dupont
adresse	3 Rue du Paradis
ville	Marseille

Il est aussi possible de créer le tableau case par case, comme ceci :

```
<?php
$coordonnees['prenom'] = 'François';
$coordonnees['nom'] = 'Dupont';
$coordonnees['adresse'] = '3 Rue du Paradis';
$coordonnees['ville'] = 'Marseille';
?>
```

- **Afficher un tableau associatif**

Pour afficher un élément, il suffit d'indiquer le nom de cet élément entre crochets, ainsi qu'entre guillemets ou apostrophes puisque l'étiquette du tableau associatif est un texte.

Par exemple, pour extraire la ville, on devra taper :

```
<?php
echo $coordonnees['ville'];
?>
```

Comme vous l'avez vu dans mes exemples, ils ne servent pas à stocker la même chose…

- **Les arrays numérotés** permettent de stocker une série d'éléments du même type, comme des prénoms. Chaque élément du tableau contiendra alors un prénom.
- **Les arrays associatifs** permettent de découper une donnée en plusieurs sous-éléments. Par exemple, une adresse peut être découpée en nom, prénom, nom de rue, ville…

III.5.2 Parcourir un tableau

Lorsqu'un tableau a été créé, on a souvent besoin de le parcourir pour savoir ce qu'il contient. Nous allons voir trois moyens d'explorer un array :

- la boucle for ;
- la boucle foreach ;
- la fonction print_r (utilisée principalement pour le débogage).

➢ La boucle for

Il est très simple de parcourir un tableau numéroté avec une boucle for. En effet, puisqu'il est numéroté à partir de 0, on peut faire une boucle for qui incrémente un compteur à partir de 0 :

```
<?php
// On crée notre array $prenoms
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique', 'Benoît');
// Puis on fait une boucle pour tout afficher :
for ($numero = 0; $numero < 5; $numero++)
{
echo $prenoms[$numero] . '<br />'; // affichera $prenoms[0],
$prenoms[1] etc.
}
?>
```

- **La boucle foreach**

La boucle foreach est une sorte de boucle for spécialisée dans les tableaux.

foreach va passer en revue chaque ligne du tableau, et lors de chaque passage, elle va mettre la valeur de cette ligne dans une variable temporaire (par exemple $element).

```
<?php
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique','Benoît');
foreach($prenoms as $element)
{
echo $element . '<br />'; // affichera $prenoms[0], $prenoms[1]
etc.
}?
>
```

L'avantage de foreach est qu'il permet aussi de parcourir les tableaux associatifs.

Exemple :

```
<?php
$coordonnees = array (
'prenom' => 'François',
'nom' => 'Dupont',
'adresse' => '3 Rue du Paradis',
'ville' => 'Marseille');
foreach($coordonnees as $element)
{
echo $element . '<br />';
}?
>
```

Toutefois, avec cet exemple, on ne récupère que la valeur. Or, on peut aussi récupérer la clé de l'élément. On doit dans ce cas écrire foreach comme ceci :

```
<?php
$coordonnees = array (
'prenom' => 'François',
'nom' => 'Dupont',
'adresse' => '3 Rue du Paradis',
'ville' => 'Marseille');
foreach($coordonnees as $cle => $element)
{
echo '[' . $cle . '] vaut ' . $element . '<br />';
}?
>
```

- **Afficher rapidement un array avec print_r**

Parfois, en codant votre site en PHP, vous aurez sous les bras un array et vous voudrez savoir ce qu'il contient, juste pour votre information. Vous pourriez utiliser une boucle for ou, mieux, une boucle foreach. Mais si vous n'avez pas besoin d'une mise en forme spéciale et que vous voulez juste savoir ce que contient l'array,

vous pouvez faire appel à la fonction print_r. C'est une sorte de echo spécialisé dans les arrays.

Cette commande a toutefois un défaut : elle ne renvoie pas de code HTML comme
 pour les retours à la ligne. Pour bien les voir, il faut donc utiliser la balise HTML <pre> qui nous permet d'avoir un affichage plus correct.

```
<?php
$coordonnees = array (
'prenom' => 'François',
'nom' => 'Dupont',
'adresse' => '3 Rue du Paradis',
'ville' => 'Marseille');
echo '<pre>';
print_r($coordonnees);
echo '</pre>';
?>
```

III.6 Les constantes

Les constantes sont des noms qui prennent une valeur ne pouvant pas être modifiée une fois fixée. Une constante n'est accessible qu'en lecture seule. Elles sont sensibles à la casse et doivent par convention être écrites en majuscules.

On définit une constante avec la fonction define() et doit respecter certaines règles :

- une constante ne commence pas par un $
- une constante est accessible depuis n'importe quel endroit du code
- une constante ne peut pas être redéfinie
- une constante ne peut contenir un scalaire (entier, booléen, chaîne, double).

Define (CONSTANTE,"valeur");
echo CONSTANTE; // affiche "valeur"
Exemple : define("PI", 3.1415926535) ;
echo PI ;

Quelques constantes prédéfinies :

Constantes	Significations
PHP_VERSION	Version de PHP installée sur le serveur
PHP_OS	Nom du système d'exploitation du serveur
DEFAULT_INCLUDE_PATH	Chemin d'accès aux fichiers par défaut
__FILE__	Nom du fichier en cours d'exécution
__LINE__	Numéro de la ligne en cours d'exécution

III.7 Conversion de type

Malgré la grande souplesse, ou le grand laxisme, selon les opinions, de PHP à l'égard des types des variables, il peut être indispensable de convertir explicitement une variable d'un type dans un autre. C'est particulièrement vrai pour les variables issues d'un formulaire, ce dernier étant l'outil essentiel de communication du poste client au serveur. Ces variables sont toujours de type string.
Pour convertir une variable d'un type dans un autre, utilisez la syntaxe suivante :

$result = (type_désiré) $mavar;

Dans l'exemple suivant, vous transformez une chaîne de caractères successivement en nombre décimal puis en entier et enfin en booléen :

```
<?php
$var="3.52 kilomètres";
$var2 = (double) $var;
echo "\$var2= ",$var2,"<br />";//affiche "$var2=3.52"
$var3 = (integer) $var2;
echo "\$var3= ",$var3,"<br />";//affiche "$var3=3"
$var4 = (boolean) $var3;
echo "\$var4= ",$var4,"<br />";//affiche "$var4=1" soit la valeur true
?>
```

Vous avez également la possibilité de modifier le type de la variable elle-même au moyen de la fonction **settype()**, dont la syntaxe est la suivante :

boolean settype($var,"type _désiré")

Elle retourne la valeur TRUE si l'opération est réalisée et FALSE dans le cas contraire. Avec cette fonction, le code précédent devient :

```
<?php
$var="3.52 kilomètres";
settype($var,"double");
echo "\$var= ",$var,"<br />";//affiche "$var=3.52"
settype($var,"integer");
echo "\$var= ",$var,"<br />";//affiche "$var=3"
settype($var,"boolean");
echo "\$var= ",$var,"<br />";//affiche "$var=1" soit la valeur true
?>
```

IV. Les opérateurs

IV.1 Opérateurs arithmétiques

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
!	Non logique

Exemple :

$nombre = 2 + 4 ; // $nombre prend la valeur 6

$nombre = 3*5+1 ; // $nombre prend la valeur 16

$nombre = 10%3 ; // $nombre prend la valeur 1 car il reste 1

IV.2 Opérateurs d'assignation

Le principal est le = mais on a aussi comme en C des opérateurs combinés +=, -=, *=, /=, %=, .= ...

IV.3 Opérateurs de comparaison

Opérateur	Signification	Exemple
==	Est égal à (valeur)	$a = "1000"; $b = "+100.0"; Echo ($a == $b) ;
===	Est égal à (type et valeur)	$a = "1000"; $b = "+100.0"; Echo ($a === $b) ;
>	Est supérieur à	
<	Est inférieur à	
>=	Est supérieur ou égale à	
<=	Est inférieur ou égal à	
!=	Est différent de (valeur)	$a = "1000"; $b = "+100.0"; Echo ($a != $b) ;
!==	Est différent de (type et valeur)	$a = "1000"; $b = "+100.0"; Echo ($a !== $b) ;

Il y a aussi l'opérateur ternaire « ?: » expr1?expr2:expr3 Si expr1 est vrai alors expr2 sinon expr3.

IV.4 Opérateur d'erreur

On dispose d'un opérateur spécial @ qui appliqué à une expression empêche la sortie d'un message d'erreur en cas de problème. On peut toujours récupérer le message d'erreur éventuel à l'aide de la variable $php_errormsg mais uniquement si l'option « track_errors » est à « On » dans le php.ini.

$retour = @$tab['toto']; // ne retourne pas d'erreurs si l'index toto n'existe pas

IV.5 Opérateurs d'incrémentation/décrémentation

On dispose comme en C des opérateurs ++ et --, à utiliser avant ou après le nom de variable.

$a++; // retourne $a puis l'incrémente de 1
++$a; // incrémente $a de 1 puis retourne $a
$a--; // retourne $a puis décrémente de 1
--$a; // décrémente $a de 1 puis retourne $a

Exemple :

Décrémentation	Incrémentation

$var=56;
echo $var—; //affiche 56 puis décrémente $var.
echo $var; //affiche 55.
echo —$var; //décrémente $var puis affiche 54.
	$var=56;
	echo $var++; //affiche 56 puis incrémente $var.
	echo $var; //affiche 57.
	echo ++$var; //incrémente $var puis affiche 58.

IV.6 Opérateurs logiques

Les opérateurs logiques sont :

Opérateur	Description

OR	Teste si l'un au moins des opérandes a la valeur TRUE :
	$a = true;
	$b = false;
	$c = false;
	$d = ($a OR $b);//$d vaut TRUE.
	$e = ($b OR $c); //$e vaut FALSE.

||	Équivaut à l'opérateur OR mais n'a pas la même priorité.

Opérateur	Description

XOR	Teste si un et un seul des opérandes a la valeur TRUE :
	$a = true;
	$b = true;
	$c = false;
	$d = ($a XOR $b); //$d vaut FALSE.
	$e = ($b XOR $c); //$e vaut TRUE.

AND	Teste si les deux opérandes valent TRUE en même temps :
	$a = true;
	$b = true;
	$c = false;
	$d = ($a AND $b); //$d vaut TRUE.
	$e = ($b AND $c); //$e vaut FALSE.

&&	Équivaut à l'opérateur AND mais n'a pas la même priorité.

!	Opérateur unaire de négation, qui inverse la valeur de l'opérande :
	$a = TRUE;
	$b = FALSE;
	$d = !$a; //$d vaut FALSE.
	$e = !$b; //$e vaut TRUE.

IV.7 Opérateurs de chaînes

Il y a deux opérateurs de chaînes : le « . » qui concatène deux chaînes entre elles et le « .= » déjà vu qui est l'opérateur d'assignation.

```
$a="Bonjour";
$b=$a." les amis"; // $b contient Bonjour les amis
$b.="! Salut."; // $b contient Bonjour les amis! Salut.
```

V. Les instructions de contrôle

V.1 L'instruction if

If (expression)

{

// bloc de code

}

Exemple :

```
<?php
$a=6;
if(is_integer($a) && ($a<10 && $a>5) && ($a%2==0) ) {
echo "Conditions satisfaites";
}
?>
```

L'expression composée : **(is_integer($a) && ($a<10 && $a>5) && ($a%2==0))** est évaluée à TRUE si $a répond simultanément aux trois conditions suivantes : être un entier, être compris entre 5 et 10 et être divisible par 2, soit pour $a les valeurs possibles de 6 et 8 uniquement. Le message ne s'affiche donc que dans ces cas.

V.1.1 L'instruction if...else

L'instruction if...else permet de traiter le cas où l'expression conditionnelle est vraie et en même temps d'écrire un traitement de rechange quand elle est évaluée à FALSE, ce que ne permet pas une instruction if seule. L'instruction ou le bloc qui suit else est alors le seul à être exécuté. L'exécution continue ensuite normalement après le bloc else.

Exemple :

```
<?php
$prix=55;
if($prix>100)
{
echo "<b>Pour un montant d'achat de $prix F, la remise est de 10 %
</b><br />";
echo "Le prix net est de ",$prix*0.90;
}
else
{
echo "<b>Pour un montant d'achat de $prix F, la remise est de 5
%</b><br />";
echo "<h3>Le prix net est de ",$prix*0.95,"</h3>";
}
?>
```

Le bloc qui suit les instructions if ou else peut contenir toutes sortes d'instructions, y compris d'autres instructions if...else. Nous obtenons dans ce cas une syntaxe plus complexe, de la forme :

```
if(expression1)
{//Bloc 1}
elseif(expression2)
{//Bloc 2}
else
{//Bloc 3}
```

Exemple :

```
<?php
$nombre = -4;
if($nombre == 0) {
echo "le nombre est égal à zéro";
}
else {
if($nombre > 0) {
echo "le nombre est positif";
} else{
echo "le nombre est négatif";
}
}
?>
```

V.1.2 L'opérateur ?

L'opérateur ? permet de remplacer avantageusement une instruction if...else en évaluant une expression et en attribuant à une variable une première valeur si la condition est vraie ou une autre valeur si elle est fausse.

Sa syntaxe est la suivante :

$var = expression ? valeur1 : valeur2

Elle est équivalente à :

```
if (expression) {
        $var=valeur1;
}
else {
        $var=valeur2;
}
```

Exemple :

```
$var = ($prix>100)? "la remise est de 10 %":"la remise est de 5 %";
echo "<b>Pour un montant d'achat de $prix €: $var </b><br />";
```

V.1.3 L'instruction switch...case

Supposez que vous vouliez associer un code de département avec son nom réel. Avec une suite d'instructions if, vous écririez le script suivant :

```
<?php
$dept=75;
if($dept==75) { echo "Paris"; }
```

```
if($dept==78) { echo "Hauts de Seine"; }
if($dept==91) { echo "Yvelines"; }
if($dept==93) { echo "Seine Saint Denis"; }
?>
```

Ce code peut être simplifié sans multiplier les instructions if grâce à l'instruction **switch...case**. Cette dernière permet de comparer la valeur d'une expression avec une liste de valeurs prédéterminées par le programmeur et d'orienter le script en fonction de la valeur de cette expression. La syntaxe de cette instruction est la suivante :

```
switch(expression)
{
case valeur1:
//bloc d'instructions 1;
break;
case valeur2:
//bloc d'instructions 2;
break;
.....
case valeurN:
//bloc d'instructions N;
break;
default:
//bloc d'instructions par défaut;
break;
}
```

Si l'expression qui suit le mot-clé switch vaut valeur1, les instructions qui suivent la première instruction case sont exécutées, après quoi l'exécution passe à la fin du bloc switch. Il en va de même pour les valeurs suivantes. Si aucune concordance n'est trouvée, ce sont les instructions qui suivent l'instruction default qui sont exécutées. La présence de cette instruction n'est pas obligatoire, mais elle est conseillée pour faire face à toutes les éventualités, telles les erreurs de saisie, par exemple. Chaque groupe case doit se terminer par une instruction break, faute de quoi les autres blocs case sont aussi exécutés.

Exemple :

```
<?php
$dept=75;
switch($dept)
{
//Premier cas
case 75:
case "Capitale":
echo "Paris";
break;
//Deuxième cas
case 78:
echo "Hauts de Seine";
break;
//Troisième cas
case 93:
case "Stade de france":
echo "Seine Saint Denis";
break;
//la suite des départements...
//Cas par défaut
default:
echo "Département inconnu en Ile de France";
break;
}
?>
```

V.2 Les instructions de boucles

Les boucles permettent de répéter des opérations élémentaires un grand nombre de fois sans avoir à réécrire le même code. Selon l'instruction de boucle utilisée, le nombre d'itérations peut être défini à l'avance ou être déterminé par une condition particulière.

V.2.1 La boucle for

Présente dans de nombreux langages, la boucle for permet d'exécuter plusieurs fois la même instruction ou le même bloc sans avoir à réécrire les mêmes instructions. Sa syntaxe est la suivante :

~~For (expression1; expression2; expression3)~~

For (expr_initial ; expr_final ; expr_pas){

    //instruction ou bloc;

}

Exemple :

```
<?php
for($i=1;$i<7;$i++)
{
echo "<h$i> $i :Titre de niveau $i </h$i>";
}
?>
```

Les trois expressions utilisées dans la boucle for peuvent contenir plusieurs parties séparées par des virgules. La boucle peut en ce cas être réalisée sur plusieurs variables, comme illustré à l'exemple ci-après :

```
<?php
for ($i=1,$j=9; $i<10,$j>0; $i++,$j--)
//$i varie de 1 à 9 et $j de 9 à 1
{
echo "<span style=\"border-style:double;border-width:3;\"> $i + $j=10</span>";
}
?>
```

V.2.2 La boucle while

C'est le moyen le plus simple pour faire une boucle. On traduit la boucle while par '**tant que**'. Tant que la/les conditions est/sont vérifiée/s, on traite les instructions situées dans la boucle. La particularité de cette instruction est que la condition est testée à chaque début de boucle.

- **Syntaxe de while** :

```
<?php
Initialisation;
while(condition) {
instruction 1;
incrementation;
…
} ?>
```

Exemple

```
<?php
$n=1;
while($n%7!=0 )
{
$n = rand(1,100);
echo $n,"   /";
}
?>
```

V.2.3 La boucle do … while

L'instruction do{ … } while() traduite par: 'répéter / faire … tant que' est une alternative à l'instruction while(). Elle permet de tester la condition après la première itération et exécution du premier bloc d'instructions. **Dans le cas de la boucle while, la condition est examinée avant la boucle** tandis que pour la boucle do-while elle est examinée à la fin.

Ainsi, même si cette condition n'est pas vérifiée, la boucle s'exécutera au moins une fois.

```
<?php
do {
// bloc d'instructions;
```

```
}
while(condition);
?>
```

Exemple :

```
<?php
do
{
$n = rand(1,100);
echo $n,"  / ";
}
while($n%7!=0);
?>
```

V.2.4 foreach

La boucle « foreach » est peut-être l'une des plus intéressantes pour la manipulation de tableaux ou de résultats de requêtes SQL. Elle permet de lister les tableaux. Elle dispose de deux syntaxes :

Foreach (array_expression as $value) commandes
Foreach (array_expression as $key => $value) commandes

La première syntaxe récupère les éléments du tableau un par un, séquentiellement. La valeur de l'élément courant du tableau est placée dans $value.

La seconde syntaxe est presque identique, sauf qu'en plus la clé (l'index) de l'élément actuel est placée dans $key.

```
<?php
//Création du tableau de 9 éléments
for($i=0;$i<=8;$i++) {
        $tab[$i] = pow(2,$i);
}
$val ="Une valeur";
echo $val,"<br />";
//Lecture des valeurs du tableau
echo"Les puissances de 2 sont :";
foreach($tab as $val) {
        echo $val." : ";
}
?>
```

V.3 L'instruction break

Il est possible d'arrêter complètement une boucle for, foreach ou while avant son terme normal si une condition particulière est vérifiée, à l'aide de l'instruction break. Le script n'est pas arrêté, comme avec l'instruction exit, et seule la boucle en cours se termine.

```
<?php
//Création d'un tableau de noms
$tab[1]="Basile";
$tab[2]="Conan";
$tab[3]="Albert";
$tab[4]="Vincent";
//Boucle de lecture du tableau
for($i=1;$i<count($tab);$i++)
{
if ($tab[$i][0]=="A")
{
echo "Le premier nom commençant par A est le n° $i: ",$tab[$i];
break;
}
}
?>
```

V.4 L'instruction continue

À la différence de l'instruction break, l'instruction continue n'arrête pas la boucle en cours mais seulement l'itération en cours. La variable compteur est incrémentée immédiatement, et toutes les instructions qui suivent le mot-clé continue ne sont pas exécutées lors de l'itération en cours.

Exemple :

```php
<?php
//Interruption d'une boucle for
for($i=0;$i<20;$i++)
{
if($i%5==0) { continue;} ←
echo $i,"<br />";
}
//********************
//Interruption d'une boucle foreach
$tab[1]="Ain";
$tab[2]="Allier";
$tab[27]="Eure";
$tab[28]="Eure et Loir";
$tab[29]="Finistère";
$tab[33]="Gironde";
foreach($tab as $cle=>$valeur)
{
if($tab[$cle][0]!="E") { continue;} ←
echo "code $cle : département ",$tab[$cle],"<br />";
}
?>
```

V.5 require et include (_once)

« require » et « include » incluent à l'endroit actuel et exécutent le fichier PHP. Ils sont identiques dans leur fonctionnement à une exception : le traitement des erreurs. Un include produit un « warning » (le code continue en principe à s'exécuter) tandis qu'un require produit une « erreur fatale » (l'exécution s'arrête).

Comme require et include sont des éléments du langage et pas des fonctions il n'y a pas besoin d'utiliser les parenthèses.

« require_once » et « include_once » ressemblent à leurs homologues avec cependant une différence. Quoi qu'il arrive, le fichier est inclus une seule fois. Si un second « require_once » apparaît avec le même fichier, l'instruction ne sera pas exécutée.

VI. Les fonctions

Une fonction est un bloc de code qui n'est pas exécuté de manière linéaire dans un script. Ce code ne le sera que lors de l'appel explicite de la fonction. Écrit une seule fois, ce code peut être exécuté aussi souvent que nécessaire. Cela allège d'autant l'ensemble du code.

➢ Quelques exemples de fonctions PHP

PHP propose des centaines et des centaines de fonctions prêtes à l'emploi. Sur le site officiel, la documentation PHP les répertorie toutes, classées par catégories.

Ces fonctions sont très pratiques et très nombreuses. En fait, c'est en partie là que réside la force de PHP : ses fonctions sont vraiment excellentes car elles couvrent la quasi-totalité de nos besoins. Voici un petit aperçu des fonctions qui existent :

- Une fonction qui permet de rechercher et de remplacer des mots dans une variable.
- Une fonction qui envoie un fichier sur un serveur.
- Une fonction qui permet de créer des images miniatures (aussi appelées thumbnails).
- Une fonction qui envoie un mail avec PHP (très pratique pour faire une newsletter !).
- Une fonction qui permet de modifier des images, y écrire du texte, tracer des lignes, des rectangles, etc.
- Une fonction qui crypte des mots de passe.
- Une fonction qui renvoie l'heure, la date…
- Etc…

➢ Traitement des chaînes de caractères

De nombreuses fonctions permettent de manipuler le texte. En voici quelques-unes qui vont vous montrer leur intérêt.

- strlen : longueur d'une chaîne

```
<?php
$phrase = 'Bonjour les Zéros ! Je suis une phrase !';
$longueur = strlen($phrase);
echo 'La phrase ci-dessous comporte ' . $longueur . ' caractères
:<br />' . $phrase;
?>
```

- str_replace : rechercher et remplacer

str_replace remplace une chaîne de caractères par une autre.
Exemple :

```
<?php
$ma_variable = str_replace('b', 'p', 'bim bam boum');
echo $ma_variable;
?>
```

- str_shuffle : mélanger les lettres

Pour vous amuser à mélanger aléatoirement les caractères de votre chaîne.
Exemple :

```
<?php
$chaine = 'Cette chaîne va être mélangée !';
$chaine = str_shuffle($chaine);
echo $chaine;
?>
```

- strtolower : écrire en minuscules

strtolower met tous les caractères d'une chaîne en minuscules.

Exemple :

```
<?php
$chaine = 'COMMENT CA JE CRIE TROP FORT ???';
$chaine = strtolower($chaine);
echo $chaine;
?>
```

NB : Il existe *strtoupper* qui fait la même chose en sens inverse : minuscules → majuscules.

- **Fonction de récupération de la date**

Nous allons découvrir la fonction qui renvoie l'heure et la date. Il s'agit de *date* (un nom facile à retenir, avouez !). Cette fonction peut donner beaucoup d'informations. Voici les principaux paramètres à connaître :

Paramètre	Description
H	Heure
i	Minute
d	Jour
m	Mois
Y	Année

Si vous voulez afficher l'année, il faut donc envoyer le paramètre Y à la fonction :

```
<?php
$annee = date('Y');
echo $annee;
?>
```

On peut bien entendu faire mieux, voici la date complète et l'heure :

```
<?php
// Enregistrons les informations de date dans des variables
$jour = date('d');
$mois = date('m');
$annee = date('Y');
$heure = date('H');
$minute = date('i');
// Maintenant on peut afficher ce qu'on a recueilli
echo 'Bonjour ! Nous sommes le ' . $jour . '/' . $mois . '/' .
$annee . 'et il est ' . $heure. ' h ' . $minute;
?>
```

VI.1 Créons nos propres fonctions

- Syntaxe et portée

Voici la syntaxe d'une fonction.

```
function fonc($arg1, $arg2, $arg3, ..., $argn)  {
   //bloc de commandes
return $valeur
}
```

Une fonction n'a pas besoin d'être déclarée avant d'être utilisée (sauf si vous voulez rester compatible avec PHP3), du moment qu'elle apparaît dans le script.

VI.2 Arguments

On peut passer autant d'arguments que souhaité à une fonction PHP, de tous types. Les arguments sont séparés par des virgules. Par défaut ils sont passés par copie.

```
Function affiche($msg) {
echo $msg;
}
```

On peut aussi définir des valeurs par défaut mais ATTENTION, dans la liste des paramètres les valeurs par défaut doivent être en dernier.

```
function affiche($nom, $prenom="toto") {
 echo "$nom. " ".$prenom" ;
 }
```

VI.3 Valeur de retour

On retourne une valeur avec l'instruction « return ». On peut retourner tout type de valeurs, mais pas plusieurs en même temps. Cependant rien n'empêche dans ce cas de retourner un tableau.

```
function carre ($num) {
return $num * $num;
}
echo carre (4);  // affiche '16'
```

VI.4 Fonction retournant plusieurs valeurs

PHP n'offre pas la possibilité de retourner explicitement plusieurs variables à l'aide d'une syntaxe du type :

return $a,$b,$c,...

Pour pallier cet inconvénient, il suffit de retourner une variable de type array contenant autant de valeurs que désiré.

Exemple : Ecrire une fonction qui calcule le module et l'argument d'un nombre complexe donné.

VI.5 Fonction à paramètre de type array

En passant un tableau comme paramètre à une fonction, cette dernière n'a en apparence qu'un seul paramètre. Ce sont en fait les éléments du

tableau qui sont utilisés et traités par la fonction, chacun devenant comme un paramètre particulier. C'est donc dans le corps de la fonction que vous pourrez déterminer le nombre d'éléments du tableau et utiliser cet ensemble de valeurs, qui seront lues à l'aide d'une boucle.

Exemple : Créer une fonction qui réalise le produit de N nombres qui lui sont passés en tant qu'éléments du tableau, affiche le nombre de paramètre et retourne leur produit.

VI.6 Fonctions de lecture de tableau

Exemple :

```
//Définition de la fonction
function lectab($tab,$bord,$lib1,$lib2)
{
echo "<table border=\"$bord\" width=\"100%\"><tbody><tr><th>$lib1</th>
<th>$lib2 </th></tr>";
foreach($tab as $cle=>$valeur)
{
echo "<tr><td>$cle</td> <td>$valeur </td></tr>";
}
echo "</tbody> </table><br />";
}
//Définition des tableaux
$tab1 = array("France"=>"Paris","Allemagne"=>"Berlin","Espagne"=>"Madrid");
$tab2 = array("Poisson"=>"Requin","Cétacé"=>"Dauphin","Oiseau"=>"Aigle");
//Appels de la fonction
lectab ($tab1,1,"Pays","Capitale");
lectab ($tab2,6,"Genre","Espèce");
```

VI.7 Les variables statiques

Lors de l'appel d'une fonction, les variables locales utilisées dans le corps de la fonction ne conservent pas la valeur qui leur est affectée par la fonction. La variable redevient en quelque sorte vierge après chaque appel.

Pour conserver la valeur précédemment affectée entre deux appels d'une même fonction, il faut déclarer la variable comme statique en la

faisant précéder du mot-clé static, et ce avant de l'utiliser dans le corps de la fonction. Le deuxième appel de la fonction peut réutiliser la valeur qu'avait la variable après le premier appel de la fonction, et ainsi de suite à chaque nouvel appel.

L'utilisation typique des variables statiques concerne les fonctions qui effectuent des opérations de cumul. Une variable déclarée comme static ne conserve toutefois une valeur que pendant la durée du script. Lors d'une nouvelle exécution de la page, elle reprend sa valeur initiale. Il ne faut donc pas compter sur cette méthode pour transmettre des valeurs d'une page à une autre, même si ces dernières appellent la même fonction contenant des variables statiques.

Exemple :

```
function acquis($capital,$taux)
{
static $acquis=1;
//corps de la fonction
}
```

VII. Gestion des formulaires

Les formulaires sont les éléments les plus importants dans la réalisation d'un site web dynamique. C'est un moyen très adapté pour réaliser les échanges de données avec les utilisateurs.
De la saisie jusqu'à la récupération des données, nous allons voir toutes les notions de base qu'il faut connaitre lorsque l'on manipule les formulaires.

En effet, les deux valeurs possibles de l'attribut méthode nous permettent d'avoir deux méthodes possibles pour envoyer les données dans un formulaire : soit par la méthode GET soit par la méthode POST.
Les deux méthodes sont à peu près les mêmes puisque toutes les deux permettent d'envoyer des données. Toutefois, chacune a sa façon de gérer les choses et le choix des méthodes utilisées nécessite la vigilance du développeur.

VII.1 Transmission des données d page en page

- **Transmettre des données avec l'URL**

Savez-vous ce qu'est une URL ? Cela signifie Uniform Resource Locator, et cela sert à représenter une adresse sur le web.
Toutes les adresses que vous voyez en haut de votre navigateur, comme par exemple http://www.courwebiai.com, sont des URL.
Par exemple, après avoir fait une recherche sur Google, la barre d'adresse contient une URL longue qui ressemble à ceci : http://www.google.fr/search?rlz=1C1GFR343&q=siteduzero
Dans cet exemple, les informations après le point d'interrogation sont des données que l'on fait transiter d'une page à une autre. Nous allons découvrir dans cette partie comment cela fonctionne.

- **Envoyer des paramètres dans l'URL**

Exemple :

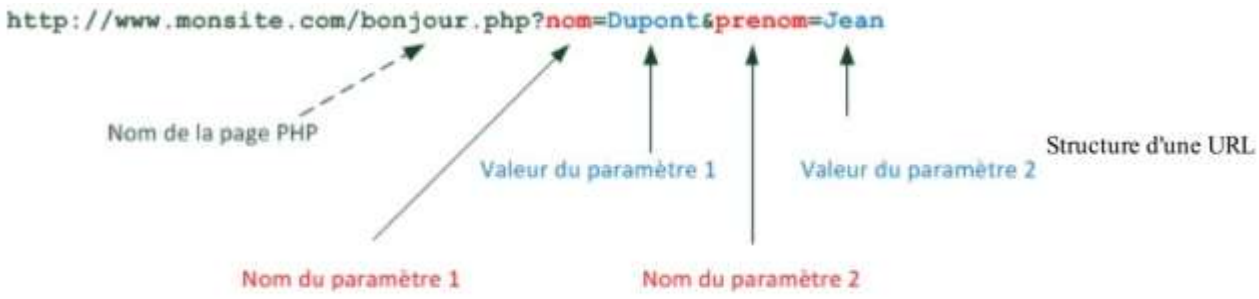
Imaginons que votre site s'appelle coursiai.com et que vous avez une page PHP intitulée bonjour.php. Pour accéder à cette page, vous devez aller à l'URL suivante :

http://www. coursiai.com/bonjour.php

Pour envoyer des informations à la page bonjour.php. Vous allez ajouter des informations à la fin de l'URL, comme ceci :

http://www.coursiai.com/bonjour.php?nom=Dupont&prenom=Jean

Ce que vous voyez après le point d'interrogation, ce sont des paramètres que l'on envoie à la page PHP. Celle-ci peut récupérer ces informations dans des variables. Voyez sur la figure suivante comment on peut découper cette URL.



Le point d'interrogation sépare le nom de la page PHP des paramètres. Ensuite, ces derniers s'enchaînent selon la forme nom=valeur et sont séparés les uns des autres par le symbole &.

- **Créer un lien avec des paramètres**

```
<a href="bonjour.php?nom=Dupont&am;prenom=Jean">
    Dis-moi bonjour
!</a>
```

- **Récupérer les paramètres en PHP**

Nous savons maintenant comment former des liens pour envoyer des paramètres vers une autre page. Nous avons pour cela ajouté des paramètres à la fin de l'URL.

Intéressons-nous maintenant à la page qui réceptionne les paramètres. Dans notre exemple, il s'agit de la page bonjour.php.

Celle-ci va automatiquement créer un array au nom un peu spécial : **$_GET**. Il s'agit d'un array associatif dont les clés correspondent aux noms des paramètres envoyés en URL.

Reprenons notre exemple pour mieux voir comment cela fonctionne. Nous avons fait un lien vers bonjour.php?nom=Dupont&prenom=Jean, cela signifie que nous aurons accès aux variables suivantes :

Nom	Valeur
$_GET['nom']	Dupont
$_GET['prenom']	Jean

Exemple :

```
<p>
Bonjour <?php echo $_GET['prenom'] . ' ' . $_GET['nom']; ?> !
</p>
```

- **Tester la présence d'un paramètre**

Pour tester la présence d'un parmètre, on peut faire appel à une fonction un peu spéciale : **isset()**. Cette fonction teste si une variable existe. Nous allons nous en servir pour afficher un message spécifique si le nom ou le prénom sont absents.

```
<?php
if (isset($_GET['prenom']) AND isset($_GET['nom'])) // On a le nom et le prénom
{
echo 'Bonjour ' . $_GET['prenom'] . ' ' . $_GET['nom'] . ' !';
}
else // Il manque des paramètres, on avertit le visiteur
{
echo 'Il faut renseigner un nom et un prénom !';
}?
>
```

VII.2 Formulaire avec la méthode GET

La méthode GET consiste à transmettre les données via l'URL. Il est intéressant de l'utiliser dans le cas où on veut faire transiter des données de pages en pages. Cependant, il faut avoir conscience que les données seront transmises par http et donc visibles dans la barre d'adresse. **Cet aspect mérite une attention particulière parce que de la sécurité des données est menacée**. Dans le cas où les données sont confidentielles, mieux vaut chercher à ne pas utiliser cette méthode.

VII.3 Formulaire avec la méthode POST

Une autre méthode pour l'envoi des formulaires est la méthode POST. Il est obligatoire de l'utiliser dans certains cas comme l'envoi de fichiers, envoi de données lourdes, envoi de données confidentielles. **La méthode POST est également recommandée si le formulaire déclenche une action spécifique qui doit être renouvelée à chaque fois**. Avec la méthode POST, il est possible de revenir en arrière et de resoumettre le formulaire.

VII.3 Les éléments d'un tableau

➢ Zone de texte

Zone de Texte	<input name="Nom">	
Zone de texte	<input name="prenom" value="Maman">	Par défaut
Mot de passe	<input type="password" name="pass">	

➢ L'élément TEXTAREA

```
<Textarea name= "commentaire " rows=10 cols=40 >
        //Taper vos commentaires ici
</Textarea>
```

➢ Bouton option

```
<input type="radio" name="sexe" value="F" checked>Feminin <BR>
<input type="radio" name="sexe" value="M">Masculin
```

➢ Case à cocher

```
<input type="checkbox" name="check" value="Word" checked>Word
<BR>
<input type=" checkbox" name="check" value="Excel">Excel <BR>
<input type=" checkbox" name="check" value="Access">Access
```

➢ L'élément Select—option

Cet élément sert à définir des listes (menus déroulant ou ascenseurs). Elle s'utilise avec l'élément OPTION

```
<select name="liste">
<option> Taro
<option> Patate
```

<option> Igname
<option selected> Pomme de Terre
<option> Oignon
<option> Ail
</select >

- **Boutons**

<input type="submit" value="Envoyer">
<input type="reset" value="Annuler">

VII.3 Récupération des données

Dans l'attribut action d'un formulaire, on spécifie l'adresse du fichier dans lequel les données sont renvoyées et traitées.

- **Utilisation des super globales**

Les données transmises se trouvent souvent sous forme de tableau dans des variables qu'on appelle des **superglobales** :
Le tableau **$_GET** permet de récupérer toutes les données envoyées par la méthode GET. Il contient toutes les données transmises via l'URL.

Le tableau **$_POST** contient les données envoyées par la méthode POST.
Le tableau **$_FILES** est spécifique à l'envoi de fichiers. Il contient les informations du fichier quand l'utilisateur télécharge un fichier via un formulaire.
Pour récupérer une donnée, on utilise généralement le nom du champ comme index du tableau $_POST. La fonction isset() permet de savoir

si une variable a été affectée ou non. Si vous ne cochez pas la case musique, la variable $_POST['musique'] ne contient pas de valeur. **Et puisqu'on en parle, profitons pour voir ce qu'est la fonction empty().**

La fonction 'empty()' a le même rôle que la fonction isset() sauf qu'elle renvoie true lorsque la variable vaut 0 ou non définie.

VII.4 Les sessions

Les sessions constituent un moyen de conserver des variables sur toutes les pages de votre site. Jusqu'ici, nous étions parvenus à passer des variables de page en page via la méthode GET (en modifiant l'URL : page.php?variable=valeur) et via la méthode POST (à l'aide d'un formulaire).

Mais imaginez maintenant que vous souhaitez transmettre des variables sur toutes les pages de votre site pendant la durée de la présence d'un visiteur. Ce ne serait pas facile avec GET et POST car ils sont plutôt faits pour transmettre les informations une seule fois, d'une page à une autre. On sait ainsi envoyer d'une page à une autre le nom et le prénom du visiteur, mais dès qu'on charge une autre page ces informations sont « oubliées ». C'est pour cela qu'on a inventé les sessions.

- Fonctionnement des sessions

Comment sont gérées les sessions en PHP ? Voici les trois étapes à connaître :

- Un visiteur arrive sur votre site. On demande à créer une session pour lui. PHP génère alors un numéro unique. Ce numéro est souvent très gros et écrit en hexadécimal, par exemple : a02bbffc6198e6e0cc2715047bc3766f.
  Ce numéro sert d'identifiant et est appelé « ID de session » (ou PHPSESSID). PHP transmet automatiquement cet ID de page en page en utilisant généralement un cookie.
- Une fois la session générée, on peut créer une infinité de variables de session pour nos besoins. Par exemple, on peut créer une variable $_SESSION['nom'] qui contient le nom du visiteur, $_SESSION['prenom'] qui contient le prénom, etc. Le serveur conserve ces variables même lorsque la page PHP a fini d'être générée. Cela veut dire que, quelle que soit la page de votre site, vous pourrez récupérer par exemple le nom et le prénom du visiteur via la superglobale $_SESSION.
- Lorsque le visiteur se déconnecte de votre site, la session est fermée et PHP « oublie » alors toutes les variables de session que vous avez créées. Il est en fait difficile de savoir précisément quand un visiteur quitte votre site. En effet, lorsqu'il ferme son navigateur ou va sur un autre site, le vôtre n'en est pas informé. Soit le visiteur clique sur un bouton « Déconnexion » (que vous aurez créé) avant de s'en aller, soit on attend quelques minutes d'inactivité pour le déconnecter automatiquement : on parle alors de timeout. Le plus souvent, le visiteur est déconnecté par un **timeout**.

Tout ceci peut vous sembler un peu compliqué, mais c'est en fait très simple à utiliser. Vous devez connaître deux fonctions :

- **session_start()** : démarre le système de sessions. Si le visiteur vient d'arriver sur le site, alors un numéro de session est généré pour lui. Vous devez appeler cette fonction au tout début de chacune des pages où vous avez besoin
  des variables de session.
- **session_destroy()** : ferme la session du visiteur. Cette fonction est automatiquement appelée lorsque le visiteur ne charge plus de page de votre site pendant plusieurs minutes (c'est le timeout), mais vous pouvez aussi créer une page « Déconnexion » si le visiteur souhaite se déconnecter manuellement.

Exercice : créer deux variables de session. Afficher ces variables sur une autre page.

VII.5 Les cookies

Travailler avec des cookies revient à peu près à la même chose qu'avec des sessions, à quelques petites différences près que nous allons aborder dans cette partie.

Qu'est-ce qu'un cookie ?

Un cookie, c'est un petit fichier que l'on enregistre sur l'ordinateur du visiteur.
Ce fichier contient du texte et permet de « retenir » des informations sur le visiteur. Par exemple, vous inscrivez dans un cookie le pseudo du visiteur, comme ça la prochaine fois qu'il viendra sur votre site, vous pourrez lire son pseudo en allant regarder ce que son cookie contient.

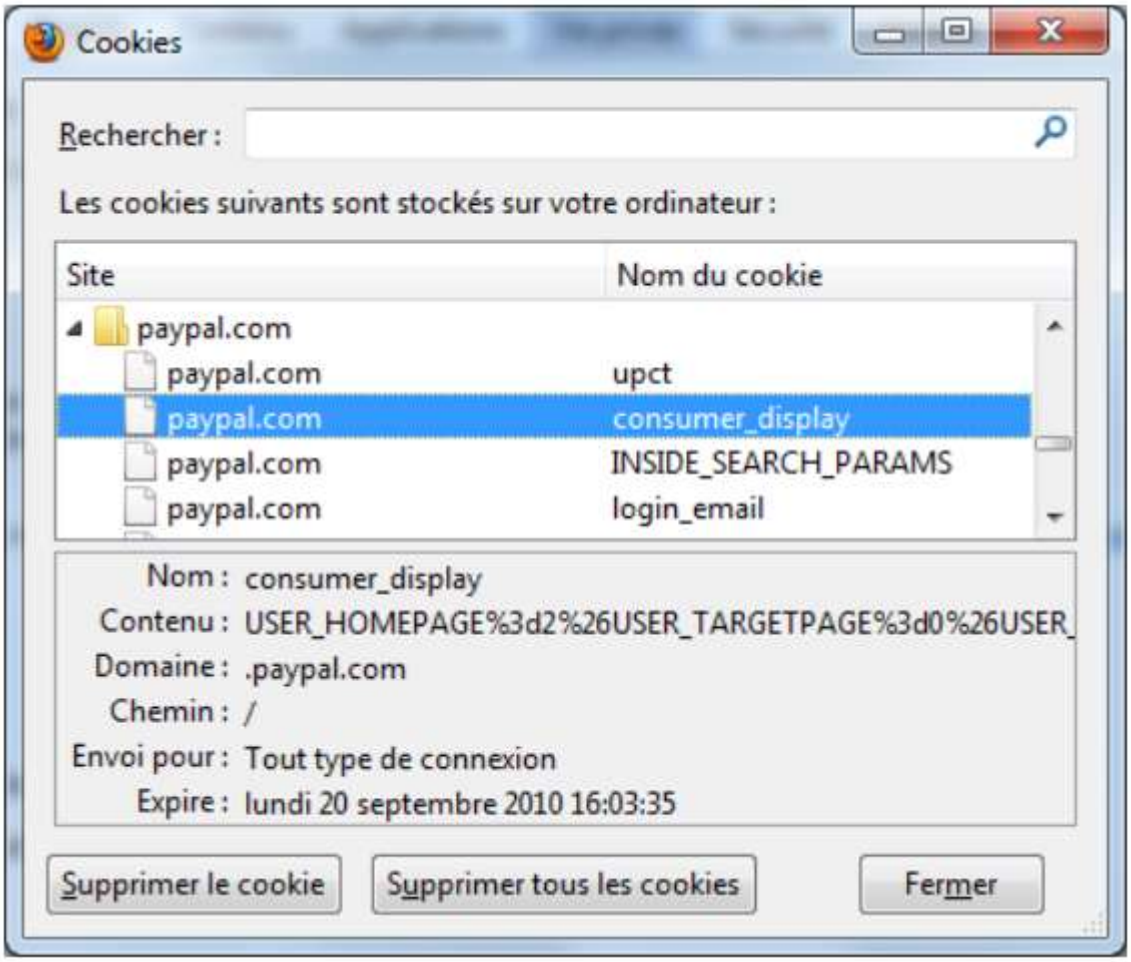
Parfois les cookies ont une mauvaise image. On fait souvent l'erreur de penser que les cookies sont « dangereux ». Non, ce ne sont pas des virus, juste de petits fichiers texte qui permettent de retenir des informations. Au pire, un site marchand peut retenir que vous aimez les appareils photos numériques et vous afficher uniquement des pubs pour des appareils photos.

Chaque cookie stocke généralement une information à la fois. Si vous voulez stocker le pseudonyme du visiteur et sa date de naissance, il est donc recommandé de créer deux cookies.

Où sont stockés les cookies sur mon disque dur ?

Cela dépend de votre navigateur web. Généralement on ne touche pas directement à ces fichiers, mais on peut afficher à l'intérieur du navigateur la liste des cookies qui sont stockés. On peut choisir de les supprimer à tout moment.

Par exemple Si vous travaillez avec Mozilla Firefox, vous pouvez aller dans le menu Outils / Options / Vie privée et cliquer sur Supprimer des cookies spécifiques. Vous obtenez la liste et la valeur de tous les cookies stockés, comme sur la figure suivante :



Les cookies sont classés par site web. Chaque site web peut écrire, comme vous le voyez, plusieurs cookies. Chacun d'eux a un nom et une valeur. Vous noterez que comme tout cookie qui se respecte, chacun a une date d'expiration. Après cette date, ils sont automatiquement supprimés par le navigateur.

Les cookies sont donc des informations temporaires que l'on stocke sur l'ordinateur des visiteurs. La taille est limitée à quelques kilo-octets : vous ne pouvez pas stocker beaucoup d'informations à la fois, mais c'est en général suffisant.

- Écrire un cookie

Comme une variable, un cookie a un nom et une valeur. Pour écrire un cookie, on utilise la fonction PHP ***setcookie*** (qui signifie « Placer un cookie » en anglais). On lui donne en général trois paramètres, dans l'ordre suivant :

- le nom du cookie (ex. : pseudo) ;
- la valeur du cookie (ex. : M@teo21) ;
- la date d'expiration du cookie, sous forme de timestamp (ex. : 1090521508).

Le paramètre correspondant à la date d'expiration du cookie représente le nombre de secondes écoulées depuis le 1er janvier 1970. Le timestamp est une valeur qui augmente de 1 toutes les secondes. Pour obtenir le timestamp actuel, on fait appel à la fonction time(). Pour définir une date d'expiration du cookie, il faut ajouter au « moment actuel » le nombre de secondes au bout duquel il doit expirer.

Si vous voulez supprimer le cookie dans un an, il vous faudra donc écrire : time() + 365*24*3600. Cela veut dire :

Voici donc comment on peut créer un cookie :

```
<?php
      setcookie('pseudo', 'M@teo21', time() + 365*24*3600);
 ?>
```

- Sécuriser son cookie avec le mode httpOnly

httpOnly rend votre cookie inaccessible en JavaScript sur tous les

navigateurs qui supportent cette option (c'est le cas de tous les

```
<? php
setcookie('pseudo', 'M@teo21', time() + 365*24*3600, null, null, false, true);
?>
```

Le dernier paramètre true permet d'activer le mode httpOnly sur le cookie, et donc de le rendre en quelque sorte plus sécurisé.

- Créer le cookie avant d'écrire du HTML

Il y a un petit problème avec setcookie… Comme pour session_start, cette fonction ne marche que si vous l'appelez avant tout code HTML (donc avant la balise <!DOCTYPE>).

Exercice :

Ecrire deux cookies, un qui retient le pseudo d'un utilisateur pendant un an, et un autre qui retient le nom de son pays :

```
<?php
setcookie('pseudo', 'M@teo21', time() + 365*24*3600, null, null,
false, true); // On écrit un cookie
setcookie('pays', 'France', time() + 365*24*3600, null, null, false,
true); // On écrit un autre cookie...
// Et SEULEMENT MAINTENANT, on peut commencer à écrire du code html
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Ma super page PHP</title>
</head>
<body>
etc.
```

- Afficher un cookie

C'est la partie la plus simple. Avant de commencer à travailler sur une page, PHP lit les cookies du client pour récupérer toutes les informations qu'ils contiennent. Ces informations sont placées dans la superglobale $_COOKIE, sous forme d'array, comme d'habitude.
De ce fait, si je veux ressortir le pseudo du visiteur que j'avais inscrit dans un cookie, il suffit d'écrire :
$_COOKIE['pseudo'].

```
<p>
Hé ! Je me souviens de toi !<br />
Tu t'appelles <?php echo $_COOKIE['pseudo']; ?> et tu viens de
<?php echo $_COOKIE['pays']; ?> c'est bien ça ?
</p>
```

- Modifier un cookie existant

Pour modifier un cookie déjà existant, il faut refaire appel à setcookie en gardant le même nom de cookie, ce qui « écrasera » l'ancien. Par exemple, si maintenant le client habite en Chine, on fera :

```
setcookie('pays', 'Chine', time() + 365*24*3600, null, null, false,
true);
```

VIII. Les bases de données MySQL

VIII.1 Présentation

MySQL est un SGBDR : « Système de Gestion de base de Données Relationnel » qui se définit luimême comme étant « La base de données Open Source la plus populaire au monde ». Rien que ça. Et c'est vrai ! Edité par la société MySQL AB, MySQL est un produit Open Source libre d'utilisation sous licence GPL pour les projets libres. Cependant une licence commerciale est nécessaire dans les autres cas, notamment si on souhaite redistribuer MySQL avec un produit non libre ou si on souhaite un support technique.

La version de production actuelle de MySQL est la version 4 (4.0.17 à l'écriture de ce support), mais la grande majorité des serveurs des hébergeurs sont encore dans les dernières versions de MySQL 3.23 (3.23.58). La future version actuellement en développement est la 5.0.0 et n'est surtout pas à utiliser en production.

Les principales qualités de MySQL sont sa simplicité et sa rapidité. Son principal défaut est le manque de fonctionnalités dites avancées (dans les versions précédentes) : clé étrangères, procédures stockées, triggers et selects imbriqués notamment. Mais cela ne doit pas occulter sa puissance avec l'ajout de fonctionnalités avancées comme une syntaxe SQL étendue (replace, limit, delete), les index de recherche « fulltext » permettant de créer des moteurs de recherche, ...

La prochaine version stable (5.0) comblera les lacunes des précédentes versions avec le support complet de la syntaxe SQL ANSI-99.

VIII.2 Outils

- **PhpMyAdmin**

S'il reste bien entendu possible d'utiliser MySQL en ligne de commande, un peu comme « sqlplus » de Oracle, on est bien plus à l'aise avec un environnement plus intuitif. Ainsi, l'outil phpMyAdmin est une interface web à MySQL permettant d'effectuer la plupart des tâches de maintenance et d'utilisation. Cette solution fonctionne depuis n'importe quel navigateur et est indépendante de la machine.

VIII.3 Créer une base

A partir de l'écran d'accueil de phpMyAdmin, on saisit le nom de la base dans « Créer une base de données ». Il faut de préférence choisir un nom simple et intuitif. Puis on clique sur « Créer ». Après la création une nouvelle page s'affiche : c'est la page principale d'administration de la base. En haut seront toujours présents après l'exécution d'une commande les résultats de celle-ci. Cet écran permet notamment l'exécution de commandes SQL, et le travail sur les tables.

VIII.4 MySQL et PHP

VIII.4.1 Connexion à une base de données

Pour que la base de données MySQL soit accessible à partir des pages d'un site, il faut pouvoir l'utiliser par l'intermédiaire d'un script. MySQL est utilisable par d'autres langages que PHP, Java par exemple. Le couple PHP-MySQL est cependant le plus répandu sur le Web. L'accès à une base MySQL et son utilisation, qu'il s'agisse d'insérer, de modifier ou de lire des données, suit les étapes ci-dessous :

1. Connexion au serveur MySQL.
2. Envoi de diverses requêtes SQL au serveur (insertion, lecture, suppression ou mise à jour des données).
3. Récupération du résultat d'une requête.
4. Fermeture de la connexion au serveur.

VIII.4.1.1 Connexion au serveur MySQL

- **Se connecter à la base de données en PHP**

Pour pouvoir travailler avec la base de données en PHP, il faut d'abord s'y connecter.

Nous allons apprendre dans cette partie à lire des données dans une base de données. Or, PHP doit faire l'intermédiaire entre vous et MySQL. Problème : PHP ne peut pas dire à MySQL dès le début « Récupère-moi ces valeurs ».

En effet, MySQL demande d'abord un nom d'utilisateur et un mot de passe. S'il ne le faisait pas, tout le monde pourrait accéder à votre BDD et lire les informations (parfois confidentielles !) qu'elle contient.
Il va donc falloir que PHP s'authentifie : on dit qu'il établit une connexion avec MySQL. Une fois que la connexion sera établie, vous pourrez faire toutes les opérations que vous voudrez sur votre base de données !

Comment se connecte-t-on à la base de données en PHP ? PHP propose plusieurs moyens de se connecter à une base de données MySQL.

- **L'extension mysql_** : ce sont des fonctions qui permettent d'accéder à une base de données MySQL et donc de

communiquer avec MySQL. Leur nom commence toujours par mysql_. Toutefois, ces fonctions sont vieilles et on recommande de ne plus les utiliser aujourd'hui.

- **L'extension mysqli_** : ce sont des fonctions améliorées d'accès à MySQL. Elles proposent plus de fonctionnalités et sont plus à jour.
- **L'extension PDO** : c'est un outil complet qui permet d'accéder à n'importe quel type de base de données. On peut donc l'utiliser pour se connecter aussi bien à MySQL que PostgreSQL ou Oracle.

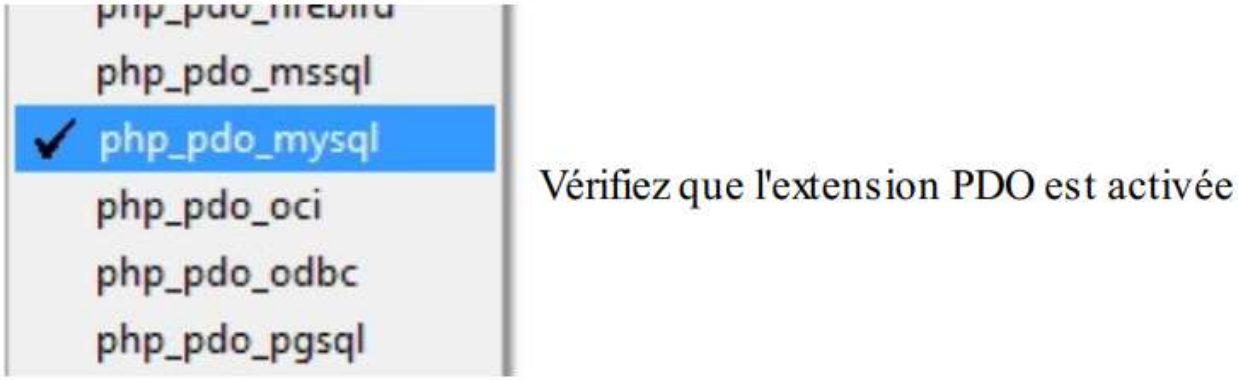
  Ce sont toutes des extensions car PHP est très modulaire. On peut très facilement ajouter ou supprimer des éléments à PHP, car tout le monde n'a pas forcément besoin de toutes les fonctionnalités.

  Quel moyen choisir parmi tous ceux-là ?

Vous l'aurez compris, les fonctions mysql_ ne sont plus à utiliser (on dit qu'elles sont « obsolètes »). Il reste à choisir entre mysqli_ et PDO. Nous allons ici utiliser PDO car c'est cette méthode d'accès aux bases de données qui va devenir la plus utilisée dans les prochaines versions de PHP. D'autre part, le gros avantage de PDO est que vous pouvez l'utiliser de la même manière pour vous connecter à n'importe quel autre type de base de données (PostgreSQL, Oracle…) (figure suivante).

- **Activer PDO**

Normalement, PDO est activé par défaut. Pour le vérifier (voir la figure suivante), faites un clic gauche sur l'icône de WAMP dans la barre des tâches, puis allez dans le menu PHP / Extensions PHP et vérifiez que php_pdo_mysql est bien coché.



- **Se connecter à MySQL avec PDO**

Maintenant que nous sommes certains que PDO est activé, nous pouvons nous connecter à MySQL. Nous allons avoir besoin de quatre renseignements :

- **le nom de l'hôte** : c'est l'adresse de l'ordinateur où MySQL est installé (comme une adresse IP). Le plus souvent, MySQL

est installé sur le même ordinateur que PHP : dans ce cas, mettez la valeur localhost (cela signifie « sur le même ordinateur »). Néanmoins, il est possible que votre hébergeur web vous indique une autre valeur à renseigner (qui ressemblerait à ceci : sql.hebergeur.com). Dans ce cas, il faudra modifier cette valeur lorsque vous enverrez votre site sur le Web ;

- **la base** : c'est le nom de la base de données à laquelle vous voulez vous connecter. Dans notre cas, la base s'appelle test. Nous l'avons créée avec phpMyAdmin dans le chapitre précédent ;
- **le login** : il permet de vous identifier. Renseignez-vous auprès de votre hébergeur pour le connaître. Le plus souvent (chez un hébergeur gratuit), c'est le même login que vous utilisez pour le FTP ;
- **le mot de passe** : il y a des chances pour que le mot de passe soit le même que celui que vous utilisez pour accéder au FTP. Renseignez-vous auprès de votre hébergeur.

```
<?php
$bdd = new PDO('mysql:host=localhost;dbname=test', 'root', ' ');
?>
```

➢ Tester la présence d'erreurs

Si vous avez renseigné les bonnes informations (nom de l'hôte, de la base, le login et le mot de passe), rien ne devrait s'afficher à l'écran. Toutefois, s'il y a une erreur (vous vous êtes trompés de mot de passe ou de nom de base de données, par exemple), PHP risque d'afficher toute la ligne qui pose l'erreur, ce qui inclut le mot de passe.

Vous ne voudrez pas que vos visiteurs puissent voir le mot de passe si une erreur survient lorsque votre site est en ligne. Il est préférable de traiter l'erreur. En cas d'erreur, PDO renvoie ce qu'on appelle une exception qui permet de « capturer » l'erreur.

```
<?php
try
{
$bdd = new PDO('mysql:host=localhost;dbname=test', 'root', ' ');
}catch (Exception $e)
{
die('Erreur : ' . $e->getMessage());
}?
>
```

VIII.4.2 Les requêtes

VIII.4.2.1 Exécuter une requête

Pour récupérer des informations de la base de données, nous avons besoin de notre objet représentant la connexion à la base : il s'agit de $bdd. Nous allons effectuer la requête comme ceci :

```
$reponse = $bdd->query('Tapez votre requête SQL ici');
```

NB : $reponse contient maintenant la réponse de MySQL

VIII.4.2.4 Récupérer les résultats

Pour récupérer les entrées, on prend la réponse de MySQL et on y exécute fetchAll().

```
<?php
  $donnees = $reponse->fetchAll();
?>
```

- **Les requêtes préparées**.

Le système de requêtes préparées a l'avantage d'être beaucoup plus sûr mais aussi plus rapide pour la base de données si la requête est exécutée plusieurs fois. C'est ce qu'on préconise si vous voulez adapter une requête en fonction d'une ou plusieurs variables.

Dans un premier temps, on va « préparer » la requête sans sa partie variable, que l'on représentera avec un marqueur sous forme de point d'interrogation :

Exemple :

```
<?php
$req = $bdd->prepare('SELECT * FROM client WHERE nom = ?');
?>
```

Au lieu d'exécuter la requête avec query() comme la dernière fois, on appelle ici prepare().

La requête est alors prête, sans sa partie **variable**. Maintenant, nous allons exécuter la requête en appelant execute et en lui transmettant la liste des paramètres :

```
<?php
$req = $bdd->prepare('SELECT * FROM client WHERE nom = ?');
$req->execute(array($_GET[nom]));
?>
```

La requête est alors exécutée à l'aide des paramètres que l'on a indiqués sous forme d'array.

S'il y a plusieurs marqueurs, il faut indiquer les paramètres dans le bon ordre :

```
<?php
$req = $bdd->prepare('SELECT * FROM client WHERE nom = ? AND
age <  ?');
$req->execute(array($_GET[nom], ($_GET[age]));
?>
```

Le premier point d'interrogation de la requête sera remplacé par le contenu de la variable $_GET[nom], et le second par le contenu de $_GET['age']. Le contenu de ces variables aura été automatiquement sécurisé pour prévenir les risques d'injection SQL.

- **Activation des erreurs**

Pour afficher des détails sur les erreurs lors des requêtes, il faut activer les erreurs lors de la connexion à la base de données via PDO.

```
<?php
$bdd = new PDO('mysql:host=localhost;dbname=test', 'root', '',
array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
?>
```

Désormais, toutes vos requêtes SQL qui comportent des erreurs les afficheront avec un message beaucoup plus clair.

VIII.4.2.5 Insertion des données

Sans les requêtes préparées :

```
<?php
  $bdd->exec(Votre requête d'insertion)  ;
?>
```

Avec les requêtes préparées :

```
<?php
  $bdd->prepare(Votre requête d'insertion)  ;
?>
```

NB : Les modifications et les suppressions des données seront abordées au TP.

Table des matières