

Tutoriels mirianet.com : PHP

1. [Introduction](#)
2. [Variables et constantes](#)
3. [Les types de variables](#)
4. [Structures de contrôle](#)
5. [Opérateurs](#)
6. [Fonctions](#)
7. [Classes et objets](#)
8. [PHP & Mysql](#)
9. [Trucs et astuces](#)
10. [Liens sur Php](#)

1. Introduction

PHP est un langage de script (non compilé), fréquemment utilisé pour le web car il permet de générer des pages html dynamiques (une même page PHP peut générer plusieurs pages html différentes, suivant son contexte d'appel).

Exemple1 : Le fameux "Hello World !"

Code PHP	Affichage HTML
<pre><?php //Message qui affiche "Hello Word !" echo "Hello World !"; /* Fin du programme c'est beau le php non !! */ ?></pre>	Hello World !

Vous noterez que les balises `<?php` (ou juste `<?`) et `?>` délimitent respectivement le début et la fin d'un script php, de plus le caractère ";" marque la fin d'une instruction. Quant aux commentaires, on en trouve de deux sortes :

// au début d'une ligne ou

/* en début de la première ligne

*\ à la fin de la dernière.

Exemple2 : "Hello World !" en rouge

Code PHP	Affichage HTML
<pre> <?php echo "Hello World !"; ? > </pre>	Hello World !

Ce dernier exemple est intéressant car il montre que l'on peut entremêler des balises html et des balises php, ce qui permet d'imaginer la puissance de cette technologie pour générer des pages html.

Il existe bien sûr d'autres technologies qui permettent de générer de façon similaire des pages html dynamiques, comme asp de Microsoft ou jsp de Sun (ce dernier utilisant le langage Java), mais à mon sens Php est la plus séduisante des méthodes, ne serait ce que par la multitude de forums, de cours et de scripts que l'on trouve sur le net ...

2. Variables et constantes

2.1 Les variables

On entend par variable toute entité pouvant stocker de l'information et permettant de la modifier. En php les variables sont toujours préfixées du signe \$ (dollar), exemple :

```
<?php
    $mavar=1;
?>
```

Initialise la variable "mavar" en lui affectant la valeur "1"

Il existe des variables **globales**, **statiques**, **d'environnement**, ainsi que des variables de **session**.

Pour comprendre les différences entre ces différents types de variables, on peut voir une page php comme un programme à part entière.

Ainsi une variable **globale** aura la même valeur **partout** dans la page (même à l'intérieur des [fonctions](#) que cette page peut contenir, alors qu'une *variable simple* ne sera visible que dans l'espace où elle a été définie (fonction ou corps de la page php). Les variables globales sont définie en utilisant le mot clé **global**, exemple :

```
<?php
    global $mavar;
    $mavar=1;
?>
```

Pour une variable **statique**, sa valeur sera mémorisée par la fonction où elle est définie mais elle ne sera pas visible en dehors de cette fonction. Les variables statiques sont définie en utilisant le mot clé **static**, exemple :

```
<?php
    static $mavar;
    $mavar=1;
?>
```

Les variables **d'environnement** sont des variables définies au niveau du serveur (par exemple la variable d'environnement \$_SERVER['SERVER_NAME'] correspond au nom du serveur qui traite les pages php) et sont communes à toutes les pages php, on parle aussi de variables **super globales**. Pour afficher la liste des variables d'environnement, on peut créer le script suivant :

```
<?php
```

```
phpinfo();  
?>
```

Voir un exemple de page générée avec la [fonction phpinfo\(\)](#). On peut aussi créer ses propres variables d'environnement avec l'instruction *putenv*, exemple :

```
<?php  
echo putenv( "MA_VARIABLE=mavaleur" );  
?>
```

Les variables de **session**, sont des variables également **super globales** sauf que leur durée de vie ne dure que le temps d'une session *.

Il y a plusieurs façons pour définir des variables de session, celle que je préfère (car elle marche toujours !) est `$_SESSION['nom variable']=valeur`, de toutes façons **il est impératif** de démarrer une session **dans toutes les pages** où l'on veut manipuler ces variables sous peine de ne pouvoir accéder à leur contenu. Une session se démarre à l'aide de la fonction : `session_start()`, exemple :

```
<?php  
session_start();  
$_SESSION[ 'LOGIN' ]= 'PAUL' ;  
?>
```

* Une session commence lorsque la fonction `session_start()` est rencontrée et se fini au bout d'un certain temps (paramètre propre au serveur). Par exemple, *un accès abonné* d'un site web est souvent régi par des variables de session : si on se connecte et que l'on ne fait rien pendant un certain temps, la session expire et il faut s'identifier de nouveau. Note : Le fait de fermer le navigateur permet aussi de fermer une session et en php la fonction **session_destroy()** met également un terme à la session en cours.

[En haut](#)

2.2 Les Constantes

Les constantes sont définies à l'aide de la fonction **define**, exemple :

Code PHP	Affichage HTML

```
<?php
define ('PI', 3.141593);
echo "PI=", PI, "<br />";
echo "PI sur 2=", PI / 2, "<br /
>";
?>
```

```
PI=3.141593
PI sur 2=1.5707965
```

Notez que pour les constantes le signe "\$" n'est pas utilisé de plus la durée de vie de la constante est celle de la page qui la contient.

3. Les types de variables

Tout langage de programmation peut définir une classification de ses variables en *type*, on parle alors de langage *typé*. Certains langages sont très typés (ex ADA95) et ne permettent pas de mélanger facilement les différents types, en temps que langage de script Php est beaucoup plus permissif à ce niveau, voyons néanmoins les différents types qu'il propose.

A noter qu'en Php il n'est pas besoin de déclarer explicitement un type associé à une variable, la nature du contenu même de celle-ci définissant implicitement son type.

- [Booléens](#)
- [Entiers](#)
- [Nombres à virgules](#)
- [Chaines de caractères](#)
- [Tableaux](#)
- [Objets](#)

• Booléens

C'est le type le plus simple. Un booléen ne peut prendre que deux valeurs **true** (vrai) ou **false** (faux).

Exemple : `$a=true;`

[En haut](#)

• Entiers

Ce sont les nombres sans virgules, positifs, négatifs ou nuls (ex : 1, 0, 25, -159, ...)

[En haut](#)

• Nombres à virgules

On peut dire que ce sont les nombres qui ne sont pas entiers, pour les écrire on écrit la "virgule" à l'aide d'un point

Exemple : `$a=1.23; b=145.123;`

[En haut](#)

• Chaines de caractères

Il s'agit de toute valeur non composée uniquement de chiffres, il faut toujours les entourer à l'aide des délimiteurs " ou '

Exemple : `"toto", 'rx1236'`

● Tableaux

Les tableaux sont des listes ordonnées de valeurs ou de variables mais qui ne sont pas nécessairement de même type. Un tableau est défini à l'aide du mot clé **array**.

En php (surtout en association avec une base de données), les tableaux sont des types très utilisés car ils permettent de structurer l'information.

Il existe deux types de tableaux : Les tableaux non-associatifs et associatifs, de plus chaque type de tableau peut avoir plusieurs dimensions.

● Tableaux non-associatifs

Dans ce type de tableau, les valeurs sont accessibles via l'index (ie leur rang) à l'intérieur du tableau. A noter que le premier indice est toujours **0**

Exemple :

Code PHP	Affichage HTML
<pre><?php // \$Fruits est une variable de type tableau \$Fruits=array ('pommes', 'oranges', 'poires', 'ananas'); /* On extrait l'élément d'indice 0 du tableau et on l'affiche */ \$fruit=\$Fruits[0]; echo "le fruit de rang 0 est ", \$fruit, "
"; /* On affiche l'élément d'indice 2 du tableau */ echo "le fruit de rang 2 est ", \$Fruits[2], "
"; /* On modifie l'élément d'indice 2 du tableau */ \$Fruits[2]='citron'; /* On affiche l'élément d'indice 2 du tableau */ echo "le fruit de rang 2 est</pre>	<p>le fruit de rang 0 est pommes le fruit de rang 2 est poires le fruit de rang 2 est maintenant citron</p>

```
maintenant <b>",$Fruits[2],"</b><br />";  
  
?>
```

Si on crée un tableau à l'aide de tableaux, on crée un tableau multi-dimensionnel : Dans ce cas là il faut utiliser autant d'indices que de dimensions pour accéder à un élément du tableau

Exemple :

Code PHP	Affichage HTML
<pre><?php /* \$Fruits est une variable de type tableau à une dimension */ \$Fruits=array ('pommes', 'oranges', 'poires', 'ananas'); /* \$Legumes est une variable de type tableau à une dimension */ \$Legumes=array ('carottes', 'courgettes', 'oignons'); /* \$Marche est une variable de type tableau à deux dimensions Les "Fruits" auront tous comme premier indice 0 Les "Legumes" auront tous comme premier indice 1 */ \$Marche=array(\$Fruits, \$Legumes); /* On affiche l'élément d'indice (0,0) de \$Marche */ echo "Marche[0][0]=",\$Marche[0] [0],"
"; /* On affiche l'élément d'indice (1,1) de \$Marche */ echo "Marche[1][1]=",\$Marche[1] [1],"
"; ?></pre>	<pre>Marche[0][0]=pommes Marche[1][1]=courgettes</pre>

[En haut](#)

● Tableaux associatifs

Dans ce type de tableau on utilise une **clé (key)** pour accéder à un élément.

C'est généralement beaucoup plus clair pour la lisibilité du code que de pouvoir utiliser des chaînes de caractères aux noms significatifs que des indices numériques.

Exemple :

Code PHP	Affichage HTML
<pre><?php /* \$Personne est une variable de type tableau */ \$Personne=array ('Nom'=>'Durand' , 'Prénom'=>'Jacques' , 'Age'=>25); /* On affiche l'élément ayant pour clé "nom" du tableau */ \$nom=\$Personne['Nom']; echo \$nom, "
"; /* On modifie l'élément ayant pour clé 'nom' */ \$Personne['Nom']='Dupont'; /* On affiche l'élément ayant pour clé "nom" du tableau */ \$nom=\$Personne['Nom']; echo \$nom, "
"; ?></pre>	Durand Dupont

Bien sûr on peut mélanger les deux types de tableaux (associatifs ou non)

Exemple :

Code PHP	Affichage HTML

```

<?php

/* $Personne est une variable
de type tableau */
$Personne=array();

/* On ajoute un élément
au tableau */
$Personne[0]=array(
    'Nom'=>'Durand',
    'Prénom'=>'Jacques',
    'Age'=>25
);

/* On ajoute un autre
élément au tableau */
$Personne[1]=array(
    'Nom'=>'Martin',
    'Prénom'=>'Isabelle',
    'Age'=>22
);

/* On affiche tous les
éléments du tableau avec
la commande "print_r" */

echo "<pre>";
print_r($Personne);
echo "</pre>";

?>

```

```

Array
(
    [0] => Array
        (
            [Nom] => Durand
            [Prénom] => Jacques
            [Age] => 25
        )
    [1] => Array
        (
            [Nom] => Martin
            [Prénom] => Isabelle
            [Age] => 22
        )
)

```

[En haut](#)

• Objets

Les objets sont des types structurés ayant leur propres méthodes (ie fonctions) et sont en fait des instances de **classes**

Nous les étudierons au chapitre sur les classes.

4. Structures de contrôle

Pour ceux qui sont familiers à la programmation et en particulier au langage C ou java, rien à signaler de spécial, on peut dire que la syntaxe du PHP est vraiment similaire.

Les principales structures :

- [Blocs](#)
- [Expressions conditionnelles \(tests\)](#)
- [Boucles et commandes break, continue et return](#)
- [Commandes include/include_once et require/require_once](#)

• Blocs

En php les blocs sont délimités par des accolades :

```
"{" début de bloc  
"}" fin de bloc
```

En soi un bloc n'a que peu d'intérêt (si ce n'est faire ressortir expressément une séquence d'instructions) par contre, son emploi est souvent essentiel dans les expressions conditionnelles et les boucles.

[En haut](#)

• Expressions conditionnelles (tests)

Ce sont les fameux *tests* qui font que sans eux, il faudrait écrire 150 programmes à la place d'un seul et lancer celui qui va bien au bon moment ...

Parmi ces structures on trouve bien sûr le **if**, le **if else** voire le **if elseif else** le **if else condensé** ainsi que l'expression **switch**

Prenons de suite un exemple, j'ai une variable *sexe* qui représente le sexe d'une personne masculin (h) ou féminin (f).

Je veux afficher la chaîne "bonjour monsieur !" en bleu si sexe correspond à un homme et "bonjour madame !" en rose si sexe correspond à une femme (l'exemple est un peu neuneu j'en conviens ...), on utilisera un **if else** :

```
<?php  
.....  
if ($sexe == "h")  
    echo "<span style=\"color: blue;\">Bonjour monsieur !</span >";  
else  
    echo "<span style=\"color: pink;\">Bonjour madame !</span >";  
?>
```

Remarque : Ecrire `\` signifie que l'on *échappe* le caractère `"` : En effet en PHP une chaîne de caractères étant délimitée par des `"` ou des `'` (mais il y a une petite nuance entre les deux), si elle doit contenir à son tour un délimiteur il faut le précéder de `\` (c-à-d l'échapper)

Le **if else condensé** revient au même que ci-dessus, sauf que l'écriture est beaucoup plus compacte, en fait cela marche comme ça :

```
$mvariable=(test expression) ? "valeur test vrai" : "valeur test faux";
```

On pourrait écrire le précédent exemple de la façon suivante (la couleur en moins par soucis de clarté du code) :

```
<?php
.....
$bonjour=($sexe == "h") ? "Bonjour monsieur !" : "Bonjour madame !";
echo $bonjour;
?>
```

Ci-dessus on sous entend que la seule alternative au sexe masculin est le sexe féminin, certes on est dans le vrai ! mais si on veut prévoir tous les cas on écrira alors :

```
<?php
.....
if ($sexe == "h")
    echo "<span style=\"color: blue;\">Bonjour monsieur !</span >";
elseif ($sexe == "f")
    echo "<span style=\"color: pink;\">Bonjour madame !</span >";
else
    echo "<span style=\"color: black;\">Bonjour sexe inconnu !</span >";
?>
```

Et les blocs dont je vous ai parlé dans tous ça ! J'y viens : Supposons maintenant, que l'on veuille afficher le message "bonjour madame !" à ces dames donc et que l'on veuille en plus incrémenter un compteur sur le nombre de femmes ayant visité la page, (les hommes n'auront rien !), on utilisera alors le **if elseif else** :

```
<?php
.....
//La variable "$cpt" vient par exemple d'une base de données
if ($sexe == "f") {
    echo "<span style=\"color: pink;\">Bonjour madame !</span >";
    $cpt++; //On incrémente le compteur de 1
}
//On a plus qu'à stocker de nouveau "$cpt" dans la base de
```

données

```
?>
```

En effet si on avait écrit :

```
<?php
.....
if ($sexe == "f")
    echo "<span style=\"color: pink;\">Bonjour madame !</span >";
    $cpt++; //On incrémente le compteur de 1
?>
```

Cela voudrait dire que certes uniquement les femmes auraient eu droit à un "Bonjour madame !", mais le compteur aurait comptabilisé toutes les connexions (hommes et femmes confondus). En fait il faut savoir qu'après un `if` ou un `else` (ou un `elseif`), on va exécuter un bloc délimité par des accolades et si le bloc n'a qu'une seule ligne, alors les accolades sont facultatives (mais il peut être bon de les mettre quand même !)

Il existe une structure de contrôle qui ressemble au **if elseif else** mais plus complexe, c'est le **switch** : Reprenons notre exemple du *Bonjour monsieur, bonjour madame*, avec l'instruction `switch` on écrira :

```
<?php
switch ($sexe) {
    case "h" :
        echo "bonjour monsieur<br>";
        break;
    case "f" :
        echo "bonjour madame<br>";
        break;
    default :
        echo "bonjour sexe inconnu<br>";
        break;
}
?>
```

Cette instruction se comporte en fait comme un aiguillage : On évalue l'expression entre parenthèses trouvée juste après le mot clé "switch" et on renvoi au "case" correspondant ou à l'étiquette "default" si non trouvé.

On remarquera l'emploi de l'instruction **break** qui permet de sortir du switch dès que la bonne étiquette est trouvée, cependant on peut volontairement ne pas mettre d'instruction `break` ce qui donne un sens différent, exemple :

```
<?php
switch ($i) {
```

```

case 1 :
    echo "La valeur de i est 1<br />";
    break;
case 2:
case 3:
case 4:
case 5:
    echo "La valeur de i est 2, 3, 4 ou 5<br />";
    break;
default :
    echo "La valeur de i est différente de 1, 2, 3, 4 ou 5";
    break;
}
?>

```

[En haut](#)

- **Boucles et commandes break, continue et return**

En deux mots une boucle et une structure permettant de répéter un certain nombre de fois une même séquence d'instructions (même si les arguments d'appel peuvent varier), les commandes **break**, **continue** et **return** pouvant altérer l'enchaînement des ces instructions.

En vocabulaire clair on traduit cela sous la forme :

Tantque *condition à satisfaire* **Faire** *séquence d'instruction*

On trouve plusieurs sortes de boucles, chacune ayant ses spécificités, mais toutes respectant le principe général.

- La boucle **while**

Syntaxe :

while *condition*
exécuter bloc d'instruction

Exemple :

Code PHP	Affichage HTML

```

<?php
    $i = 0;
    while ($i <= 5) {
        echo "i=$i<br />";
        $i++;
    }
?>

```

```

i=0
i=1
i=2
i=3
i=4
i=5

```

Remarque : Dans l'exemple ci-dessus on voit bien la nécessité d'utiliser des accolades pour délimiter la séquence d'instructions devant être exécutée dans la boucle, en effet si on écrit :

```

<?php
    $i = 0;
    while ($i <= 5)
        echo "i=$i<br />";
        $i++;
?>

```

Le bloc d'instructions exécuté à l'intérieur de la boucle se résume alors à la seule instruction `echo "i=$i"`; qui s'exécutera une infinité de fois vu que la valeur de `i` ne change pas... Autrement dit, il est primordial de bien délimiter les blocs dans les structures de contrôles !

- La boucle **do...while**

Syntaxe :

do exécuter bloc d'instruction
while condition

Exemple :

Code PHP	Affichage HTML
<pre> <?php \$i = 1; do { echo "i=\$i
"; \$i++; } while (\$i==0); ?> </pre>	<pre> i=1 </pre>

Contrairement à la boucle **while** la boucle **do ... while** sera exécutée au moins une fois vu que la condition est évaluée à la fin. (Par exemple ici `$i` n'a jamais respecté la condition mais on est quand


```

<?php
$fruits=array(
    'pomme' ,
    'orange' ,
    'poire' ,
    'ananas'
);

$i=1;
foreach ($fruits as $fruit) {
    echo "Le fruit numéro ",$i,"
est ",$fruit,"<br>";
    $i++;
}
?>

```

Le fruit numéro 1 est pomme
Le fruit numéro 2 est orange
Le fruit numéro 3 est poire
Le fruit numéro 4 est ananas

- Les commandes **break continue** et **return**

Ces commandes peuvent altérer le déroulement du traitement des boucles. Pour résumer **break** ou **return** auront un effet de sortie de boucle, alors que **continue** permettra juste de sauter un traitement et de continuer à la séquence suivante.

Exemple avec **break** :

Code PHP	Affichage HTML
<pre> <?php \$i=0; while (true) { echo "i=\$i
"; if (\$i>=3) break; \$i++; } echo "Fin de la boucle
"; ?> </pre>	<pre> i=0 i=1 i=2 i=3 Fin de la boucle </pre>

Sans commande **break** la boucle ci-dessus ne se terminait jamais ...

Exemple avec **return** :

Code PHP	Affichage HTML
----------	----------------

```

<?php

    $i=0;
    while (true) {
        echo "i=$i<br />";
        if ($i>=3) return;
        $i++;
    }
    echo "Fin de la boucle<br />";

?>

```

```

i=0
i=1
i=2
i=3

```

Ici l'instruction **return** sort aussi de la boucle mais met également fin au script php

Exemple avec **continue** :

Code PHP

```

<?php

    for ($i=0;$i<=5;$i++) {
        if ($i==0 || $i == 2 || $i ==
4) continue;
        echo "i=$i<br />";
    }
    echo "Fin de la boucle<br />";

?>

```

Affichage HTML

```

i=1
i=3
i=5
Fin de la boucle

```

Ci-dessus la commande **continue** a pour but de passer à la valeur suivante de \$i lorsqu'elle vaut 0, 2 ou 4, ainsi les commandes suivantes (echo) ne sont pas exécutées pour ces valeurs de \$i.

[En haut](#)

- **Commandes include/include_once et require/require_once**

Ces fonctions permettent d'inclure dans le code php et à l'endroit où elles sont appelées, d'autres fichiers php mais également des fichiers d'autres formats (html, ascii ...). Le but étant de pouvoir réutiliser le contenu de ces fichiers sans le réécrire.

- **Commandes include et require**

Supposons par exemple, que l'on ait un certain nombre de constantes que l'on désire réutiliser dans toutes nos pages php. Au lieu de les définir dans chaque page, on peut les définir dans un fichier que l'on appellera *const.php* et par la suite on appellera ce fichier à l'aide de la fonction **include** ou **require** au

début de nos pages php.

On pourrait avoir le fichier const.php ayant pour contenu :

```
<?php
define ( 'PI' , 3.141593 );
define ( 'TVA' , 19.6 );
define ( 'bgcolor' , "#FFF000" );
.....
?>
```

Il ne resterait plus qu'à écrire nos pages PHP de la façon suivante, pour que ces constantes soient connues à l'intérieur de nos pages :

```
<?php
include ( 'const.php' );
.....
?>
```

ou encore :

```
<?php
require ( 'const.php' );
.....
?>
```

En supposons quand même que la page appelante et la page const.php se trouvent sous le même répertoire. (Dans le cas contraire il faudrait spécifier le chemin relatif pour accéder à la page const.php, par exemple : include("chemin/relatif/const.php");).

Ces deux fonctions bien que similaires dans l'ensemble, présentent quand même quelques différences, par exemple la commande include permet de retourner une valeur à l'aide de l'instruction **return**, alors que require ne le peut pas, de plus la commande require engendrera une *erreur fatale* si le fichier à inclure n'est pas trouvé alors que include n'affichera qu'un message d'erreur... D'une façon générale il est quand même conseillé d'**utiliser autant que possible la commande require à la place de include** et ceci pour des raisons aussi bien de sécurité que de performances.

● Commandes include_once et require_once

Ces commandes permettent d'inclure les fichiers spécifiés mais **une seule fois** à l'intérieur d'une même page même si ces fonctions sont appelées plusieurs fois. L'intérêt est alors de s'assurer justement que l'on ne pourra par appeler plusieurs fois ces fichiers pour les inclure.

Exemple : On souhaite définir une fonction (on verra plus loin comment on définit des fonctions), une fois pour toutes les pages php du site. Comme il a été vu pour les constantes, il est tentant d'écrire cette fonction dans un fichier, puis de la rendre visible dans n'importe quelle page au moyen de la fonction include ou require, ainsi il sera aisé de modifier cette fonction.

Supposons donc que l'on crée une fonction que l'on va appeler *Wellcome* et qui affichera le message *Bienvenue sur mon site !*, on peut créer le fichier fonctions.php avec pour contenu : `<?php`

```
function Wellcome() {  
    echo "Bienvenue sur mon site !<br />";  
}  
?>
```

Si on fait un include de cette page php à l'intérieur d'une autre page php, cette dernière se présentera de la façon suivante :

Code PHP	Affichage HTML
<pre><?php //On inclus le fichier php require('fonctions.php'); //on appelle la fonction "Wellcome" du fichier inclus ci- dessus Wellcome(); ?></pre>	Bienvenue sur mon site !

Jusque là tout va bien, mais supposons que l'on fasse une seconde inclusion de ce fichier (volontairement ou juste par l'intermédiaire d'autres fichiers inclus qui appellent à leur tour ce fichier), on va générer une erreur vu que l'on tente de redéfinir une seconde fois la fonction *Wellcome* qui a déjà été définie ... Voilà un peu le message que l'on aurait :

Code PHP	Affichage HTML

```

<?php
//On inclus le fichier php
require('fonctions.php');

/* On appelle la fonction
"Wellcome"
du fichier inclus ci-dessus */
Wellcome();

/*
Ici du code
encore et encore ...
*/

/*On inclus le fichier php une
seconde
fois (sans trop en être
conscient ) */
require('fonctions.php');

?>

```

Bienvenue sur mon site !

Fatal error: Cannot redeclare Wellcome() (previously declared in C:\program files\htmlfiles\divers\mirianet\fonctions.php:3) in C:\program files\htmlfiles\divers\mirianet\fonctions.php on line 2

Pour contourner ce problème on va utiliser la commande **require_once** qui nous garantira l'unicité de l'inclusion quoi qu'il arrive. Voilà alors ce que l'on obtient :

Code PHP

```

<?php
//On inclus le fichier php
require_once('fonctions.php');

/* On appelle la fonction
"Wellcome"
du fichier inclus ci-dessus */
Wellcome();

/*
Ici du code
encore et encore ...
*/

/*On inclus le fichier php une
seconde
fois (sans trop en être

```

Affichage HTML

Bienvenue sur mon site !

```
conscient ) */
require_once('fonctions.php');

?>
```

Attention : Bien que très utiles ces commandes présentent un danger si elles sont appelées à tort et à travers ...

Supposons le script php suivant écrit dans la page *mauvaisepage.php*

```
<?php
//On récupère le nom du fichier à inclure par la méthode "GET"
$file=$_GET['file'];

//On inclus le fichier
include($file);

?>
```

Ce code est dangereux car il donne la main à l'utilisateur, en effet si ce dernier tape l'adresse suivante dans son navigateur : <http://www.monsite.com/mauvaisepage.php?file=fichiersecret> il aura en clair le contenu du fameux *fichiersecret* dans son navigateur ...

5. Opérateurs

- [Opérateurs arithmétiques](#)
- [Opérateurs d'affectation](#)
- [Opérateurs binaires](#)
- [Opérateurs logiques](#)
- [Opérateurs de comparaison](#)

• Opérateurs arithmétiques

Comme leurs nom l'indique, ils servent à faire des calculs arithmétiques entre variables :

Nom	Action	Exemple	Commentaire
Addition	Somme de deux nombres	$\$c = \$a + \$b$	$\$c$ contient la somme des valeurs des variables $\$a$ et $\$b$
Soustraction	Différence de deux nombres	$\$c = \$a - \$b$	$\$c$ contient la différence entre la valeur de $\$a$ et celle de $\$b$
Multiplication	Multiplie deux nombres	$\$c = \$a * \$b$	$\$c$ contient le produit des variables $\$a$ et $\$b$
Division *	Divise un nombre par un autre	$\$c = \$a / \$b$	$\$c$ contient le quotient de $\$a$ divisé par $\$b$
Modulo **	Retourne la reste de la division entière d'un nombre par un autre	$\$c = \$a \% \$b$	$\$c$ contient le reste de $\$a$ divisé par $\$b$

* : Le résultat sera la division décimale, par exemple :

Si $\$a=5$ et $\$b=1.2$ $\$c$ vaudra : 4.16666666667

** : Exemple si $\$a=5$ et $\$b=2$ $\$c$ vaudra 1

[En haut](#)

• Opérateurs d'affectation

Ces opérateurs permettent d'associer une valeur à une variable :

Nom	Action	Exemple	Commentaire
=	Affecte la valeur située à droite de l'opérateur, à la variable située à gauche	$\$a=3$ $\$b="coucou"$	La variable $\$a$ a pour valeur 3 La variable $\$b$ contient la chaîne de caractères "coucou"

+=	Affecte à la variable située à gauche sa valeur + la valeur située à droite de l'opérateur	$\$a+=2$	Cette expression est équivalente à $\$a = \$a + 2$
-=	Affecte à la variable située à gauche sa valeur - la valeur située à droite de l'opérateur	$\$a-=1.2$	Cette expression est équivalente à $\$a = \$a - 1.2$
++	Incrémente la variable de 1	$\$a++$ ou $++\$a$ *	Cette expression est équivalente à $\$a = \$a + 1$
--	Décrémente la variable de 1	$\$a--$ ou $--\$a$ *	Cette expression est équivalente à $\$a = \$a - 1$
.=	Ajout de la chaîne de caractères à droite de l'opérateur à celle à gauche de l'opérateur	$\$chaîne="Salut"$ $\$chaîne.=" à tous"$	Cette expression est équivalente à $\$chaîne= \$chaîne . " à tous"$

* : Il y a une différence entre $\$a++$ (ou $\$a--$) et $++\$a$ (ou $--\$a$), en effet si on fait une action sur $\$a$ en même temps, dans le premier cas ($\$a++$) l'action sera réalisée sur $\$a$ puis $\$a$ incrémentée de 1, dans le second cas ($++\$a$) on incrémente d'abord $\$a$ de 1 puis on fait l'action requise sur $\$a$, exemple :

Code PHP	Affichage HTML
<pre><?php \$a=1; echo \$a++, "
"; echo \$a, "
"; ?></pre>	<pre>1 2</pre>

Code PHP	Affichage HTML
<pre><?php \$a=1; echo ++\$a, "
"; echo \$a, "
"; ?></pre>	<pre>2 2</pre>

[En haut](#)

● Opérateurs binaires

Ils permettent de manipuler des *bits* dans un entier (Normalement vous n'aurez pas trop besoin de les

utiliser)

Nom	Exemple	Commentaire
ET (AND) logique	$\$a \& \b	Les bits positionnés à 1 dans $\$a$ ET dans $\$b$ sont positionnés à 1
OU (OR) logique	$\$a \b	Les bits positionnés à 1 dans $\$a$ OU $\$b$ sont positionnés à 1
Xor	$\$a \wedge \b	Les bits positionnés à 1 dans $\$a$ OU dans $\$b$ sont positionnés à 1
NON (NOT)	$\sim \$a$	Les bits qui sont positionnés à 1 dans $\$a$ sont positionnés à 0, et vice versa
Décalage à gauche	$\$a \ll \b	Décale les bits de $\$a$ $\$b$ fois sur la gauche (chaque décalage équivaut à une multiplication par 2)
Décalage à droite	$\$a \gg \b	Décalage des bits de $\$a$ $\$b$ fois par la droite (chaque décalage équivaut à une division par 2)

[En haut](#)

• Opérateurs logiques

Ce sont les opérateurs qui servent à construire des expressions booléennes (dont le résultat est vrai ou faux).

Nom	Exemple	Commentaire
ET (And) *	$\$a \text{ and } \b	Vrai si $\$a$ ET $\$b$ sont vrais
OU (OR) *	$\$a \text{ or } \b	Vrai si $\$a$ OU $\$b$ est vrais
XOR (Xor)	$\$a \text{ xor } \b	Vrai si $\$a$ OU $\$b$ est vrai, mais pas les deux en même temps (on peut dire aussi "ou bien")
NON (Not)	$! \$a$	Vrai si $\$a$ est faux
ET (And) *	$\$a \&\& \b	Vrai si $\$a$ ET $\$b$ sont vrais
OU (Or) *	$\$a \b	Vrai si $\$a$ OU $\$b$ est vrai

* : La différence entre **and** et **&&** est simplement que l'opérateur **&&** a plus de priorité que **and** (idem pour **or** et **||**).

[En haut](#)

• Opérateurs de comparaison

Ces opérateurs permettent de comparer des variables entre elles et servent donc aussi à construire des expressions booléennes.

Nom	Exemple	Commentaire
Egal	$\$a == \b	Vrai si \$a est égal à \$b
Identique	$\$a === \b	Vrai si \$a est égal à \$b et qu'ils sont de même type (PHP 4 seulement)
Différent	$\$a != \b ou $\$a <> \b	Vrai si \$a est différent de \$b
Inférieur	$\$a < \b	Vrai si \$a est strictement inférieur à \$b
Supérieur	$\$a > \b	Vrai si \$a est strictement supérieur à \$b
Inférieur ou égal	$\$a <= \b	Vrai si \$a est inférieur ou égal à \$b
Supérieur ou égal	$\$a >= \b	Vrai si \$a est supérieur ou égal à \$b

6. Fonctions

Comme dans tout langage de programmation, on est souvent amené à réutiliser plusieurs fois à l'intérieur d'un même programme, une séquence d'instructions (avec peut être des paramètres différents), plutôt que d'écrire à chaque fois ces mêmes lignes de code, on crée un petit programme à l'intérieur du programme principal, c'est la notion de **fonction**.

- [Les arguments de fonction](#)
- [Les valeurs de retour](#)
- [Fonctions natives Php](#)

Exemple :

Code PHP	Affichage HTML
<pre><?php /* On défini la fonction "Welcome" */ function Wellcome(\$user) { \$message="Bienvenue sur mon site "; echo \$message, "", \$user, "<br / >"; } /* Appel de cette fonction avec "user" qui vaut "Dupont" */ Wellcome("Dupont"); /* Appel de cette fonction avec "user" qui vaut "Martin" */ Wellcome("Martin"); ?></pre>	<p>Bienvenue sur mon site Dupont Bienvenue sur mon site Martin</p>

La fonction *Wellcome* définie ci-dessus, à pour but d'afficher le même message pour des personnes ayant des noms différents.

• Les arguments de fonction

Les arguments de fonction sont les paramètres que l'on va passer à cette séquence d'instructions, on les place entre les **parenthèses** de la fonction, séparés par des **virgules**, et ils peuvent être de n'importe quel type de variable.

Il existe deux modes de passage de ces paramètres : **par valeur** (mode par défaut) et **par référence**. Dans le mode **par valeur**, on récupère la valeur de l'argument et à la sortie de la fonction l'argument gardera sa valeur initiale, quelles que soient les opérations qu'il a subit dans celle ci, alors que dans le mode **par référence** la valeur de l'argument en sortie de la fonction peut avoir été modifié suivant les opérations qu'il a subit.

Exemple de passage par valeur :

Code PHP	Affichage HTML
<pre><?php function ParValeur(\$a, \$b) { \$a=\$a+\$b; \$b=2*\$a+\$b; echo "Corps de la fonction :<br / >"; echo "a=\$a b=\$b
"; } \$a=2; \$b=3; echo "Avant la fonction :
"; echo "a=\$a b=\$b
"; ParValeur(\$a, \$b); echo "Sortie de la fonction :
"; echo "a=\$a b=\$b
"; ?></pre>	<p>Avant la fonction : a=2 b=3 Corps de la fonction : a=5 b=13 Sortie de la fonction : a=2 b=3</p>

Exemple de passage par référence :

Code PHP	Affichage HTML

```

<?php

function ParReference(&$a, &$b) {
    $a=$a+$b;
    $b=2*$a+$b;
    echo "Corps de la fonction :<br /
>";
    echo "a=$a b=$b<br />";
}

$a=2;
$b=3;

echo "Avant la fonction :<br />";
echo "a=$a b=$b<br />";

ParReference(&$a, &$b);

echo "Sortie de la fonction :<br />";
echo "a=$a b=$b<br />";

?>

```

Avant la fonction
a=2 b=3
Corps de la fonction
a=5 b=13
Sortie de la fonction
a=5 b=13

On notera d'après l'exemple ci-dessus que pour passer les variables par référence il faut les précéder du signe **&**

Remarque : On peut rendre un **argument optionnel** en lui affectant une valeur par défaut, exemple :

Code PHP	Affichage HTML
<pre> <?php function ArgOptionnel(\$texte="rien") { echo "texte vaut : ", \$texte, "<br / >"; } ArgOptionnel("toto"); ArgOptionnel(); ?> </pre>	<pre> texte vaut : toto texte vaut : rien </pre>

[En haut](#)

● Les valeurs de retour

Une fonction peut *retourner* une valeur grâce à l'instruction **return**, dans ce cas on peut récupérer la valeur retournée dans une variable avec la syntaxe :

```
<?php
.....
$mavar=mafonction(<liste d'arguments>);
?>
```

Remarques :

- Dans les premiers exemples les fonctions *ParValeur* et *ParReference* ne retournaient aucune valeur.
- Une fonction peut retourner n'importe quel type de variable, mais ne peut en retourner qu'un seul.

Exemple de valeur de retour : Fonction qui retourne la valeur maximum du tableau passé en argument

Code PHP	Affichage HTML
<pre><?php function MaxArray(\$Tableau) { \$max=" "; for (\$i=0;\$i<=count(\$Tableau);\$i++) if (\$max==" " \$Tableau[\$i]>=\$max) \$max=\$Tableau[\$i]; return \$max; } \$Exemple=array(-1,1.2,5,36.8); \$maxExemple=MaxArray(\$Exemple); echo "Le max est \$maxExemple
"; ?></pre>	Le max est 36.8

[En haut](#)

● Fonctions natives Php

On appelle *fonction native* toute fonction prédéfinie au niveau du langage Php lui même (même si elles

sont appelées de la même façon que les fonctions définies par l'utilisateur). Ce sont des fonctions qui sont *compilées* sur le serveur ce qui donne des performances bien meilleures que pour les fonctions définies par l'utilisateur (surtout pour des fonctions faisant de très nombreux calculs).

En php il existe un très grand nombre de fonctions que l'on peut classer par catégories :

- Les fonctions sur les chaînes de caractères
- Les fonctions mathématiques
- Les fonctions sur les tableaux
- Les fonctions d'accès aux bases de données
- Les fonctions de manipulation de fichiers
-
-

Parmi ces fonctions, on peut noter deux types :

- Les fonctions du noyau php
- Les fonctions des bibliothèques ou modules

En fait ces deux catégories sont toutes des fonctions *compilées* sur le serveur sauf que les fonctions du noyau * seront systématiquement présentes alors que les fonctions bibliothèques le seront uniquement si le module en question a été installé.

Exemple : Les fonctions de la bibliothèque **gd** qui sont des fonctions pour manipuler des images (création, conversion de format ...) ne seront disponibles que si la dite bibliothèque a été installée (cf. `phpinfo()`).

Je vous recommande d'aller faire un tour sur le site suivant, qui vous donnera un aperçu des fonctions disponibles en PHP <http://fr3.php.net/quickref.php>

* : Bien sûr en fonction de la version même du noyau, on peut trouver des fonctions présentes dans une version et qui n'existaient pas dans une version plus ancienne.

7. Classes et objets

La notion de classe en PHP est apparue à la version 3. Depuis la version 5.0.0 (13/07/2004) on trouve de nombreuses fonctionnalités (méthodes et attributs privés (*private*), publics (*public*), protégés (*protected*) ...) qui font que PHP se rapproche de plus en plus d'un langage de programmation orientée objet (POO) à part entière, comme C++ ou Java.

Ici c'est une **approche php version 4.3.x** qui vous sera donnée, de nombreux hébergeurs n'intégrant pas encore le php 5. Pour plus d'informations sur PHP 5 et les classes je vous renverrais au site officiel php, chapitre sur les classes et objets en php 5 : <http://fr3.php.net/php5>

Contrairement à la programmation séquentielle classique, l'arrivée du C++ et de Java ont apportés un concept nouveau avec des entités qui communiquent entre elles au moyen de méthodes (fonctions), on appelle ces entités des **objets**. En fait on définit (instancie) les objets à partir de classes ce qui permet de différencier ces deux notions très proches l'une de l'autre. Un des grands avantages de ce type de programmation est de créer du code totalement réutilisable mais également beaucoup plus clair et évolutif.

- [Instanciation](#)
- [Attributs](#)
- [Constructeurs](#)
- [Méthodes](#)
- [Héritage](#)

• Instanciation

On dit qu'on **instancie** une classe pour créer un objet, par la suite l'entité qui sera manipulée dans le code sera donc l'objet et non la classe (Il faut voir l'objet comme une définition réelle d'une classe qui elle reste un peu théorique tant qu'elle n'a pas été instanciée).

Le mot clé permettant de réaliser l'instanciation est l'opérateur **new**

Exemple :

```
<?php

/* Une classe qui ne sert
pas à grand chose mais qui
permet de comprendre */

class exemple {

    function exemple() {}
}
```

```
/* L'objet "ex1" est une
instance de la classe exemple */
```

```
$ex1=new exemple();
```

```
?>
```

Dans l'exemple ci-dessus on a défini une classe : la *classe exemple*. Cette classe ne sert pas à grand chose sous cette forme en effet, mais on peut voir comment on définit et instancie un objet de la classe avec l'opérateur **new**.

Note : La fonction qui a le même nom que la classe et qui (en php) doit être unique, s'appelle le [constructeur](#)

[En haut](#)

• Attributs

Les attributs sont des variables internes à la classe, ils peuvent être de tout type (même de type objet), ils sont déclarés à l'intérieur de la classe à l'aide du mot clé **var**.

Exemple :

Code PHP	Affichage HTML
<pre><?php class exemple { var \$nom="classe exemple"; function exemple() {} } /* on instancie la classe avec l'objet "ex1" */ \$ex1=new exemple(); /* On affiche l'attribut "nom" de la classe */ echo \$ex1->nom; ?></pre>	<pre>classe exemple</pre>

[En haut](#)

• Constructeurs

Comme on l'a vu dans le premier exemple, le **constructeur** est une fonction qui a le même nom que la classe. Un constructeur peut avoir (comme toute fonction) des arguments qui peuvent être utilisés à l'intérieur de la classe.

Exemple :

Code PHP	Affichage HTML
<pre data-bbox="34 562 808 1921"><?php class exemple { var \$nom="classe exemple"; var \$message; function exemple(\$message=" ") { \$this->message=\$message; } } /* on instancie la classe avec l'objet "ex1" en appelant son constructeur avec l'argument "message" qui vaut "bonjour" */ \$ex1=new exemple("bonjour"); /* On affiche l'attribut "nom" de la classe */ echo \$ex1->nom,"
"; /* On affiche l'attribut "message" de la classe */ echo \$ex1->message,"
"; ?></pre>	<pre data-bbox="815 562 1588 1921">classe exemple bonjour</pre>

Cet exemple est un peu plus intéressant, car on voit bien que l'on peut créer des objets ayant des attributs différents à partir d'une même classe (si je change "bonjour" en "hello" je change aussi l'argument "message" de la classe).

Remarque : L'opérateur **this** sert à faire référence à l'objet lui-même, ainsi quand on écrit : `$this->message=$message` cela signifie que l'**attribut message** de l'objet reçoit la valeur de la **variable message** passée en argument au constructeur.

[En haut](#)

● Méthodes

Les méthodes sont des fonctions propres à la classe. D'une certaine façon on peut voir le constructeur comme une méthode (indispensable) de la classe, en tous cas au niveau syntaxe c'est la même chose. Voyons maintenant un exemple plus complet et par conséquent plus intéressant.

On veut créer une classe pour personnaliser l'affichage d'un tableau html. Nous appellerons cette classe *table* et son constructeur aura 3 arguments : `$nbLig` (le nombre de lignes du tableau), `$nbCol` (le nombre de colonnes du tableau) et `$Data` (les valeurs à placer dans les cellules du tableau). D'autre part, nous allons définir cette classe dans un fichier séparé, ce qui facilitera sa réutilisation :

La classe *table* sera défini dans le fichier *class.table.php* :

```
<?php

/* Ceci est le fichier
class.table.php */

class table {

var $NL="\r\n";
var $NULVAL="&nbsp;";
var $TableStart("<table border=\"1\">");
var $nbLig;
var $nbCol;
var $Data=array();

function table($nbLig, $nbCol, $Data) {
    $this->nbLig=$nbLig;
    $this->nbCol=$nbCol;
    $this->Data=$Data;
}

function display() {
    $nbLig=$this->nbLig;
    $nbCol=$this->nbCol;
```

```

$Data=$this->Data;
$NL=$this->NL;
$NULVAL=$this->NULVAL;

$len=max(count($Data),$nbLig*$nbCol);

echo $this->TableStart."$NL";

for ($i=0; $i<=$len-1;$i++) {
    $j=$i+1;
        if ($i % $nbCol ==0) echo "<tr>$NL";
        if ($i<=count($Data)-1)
            echo "<td>",$Data[$i],"</td>$NL";
        else
            echo "<td>",$NULVAL,"</td>$NL";
        if ($j % $nbCol ==0) echo "</tr>$NL";
    }

echo "</table>$NL";

}

}

?>

```

La classe *table* permet de créer un tableau html grâce à la méthode *display*.
 Ci-dessous un exemple d'utilisation de cette classe :

Code PHP	Affichage HTML															
<pre> <?php /* on inclus le fichier contenant la classe */ require_once('class.table.php'); \$Data=array(1,2,3,4,5,6); \$table1=new table(2,3,\$Data); \$table2=new table(3,3,\$Data); ?> <html> </pre>	<p style="text-align: center;">table 1</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>2</td><td>3</td></tr> <tr><td>4</td><td>5</td><td>6</td></tr> </table> <p style="text-align: center;">table 2</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>2</td><td>3</td></tr> <tr><td>4</td><td>5</td><td>6</td></tr> <tr><td> </td><td> </td><td> </td></tr> </table>	1	2	3	4	5	6	1	2	3	4	5	6			
1	2	3														
4	5	6														
1	2	3														
4	5	6														

```

<head></head>
<body>

<center>

<?php

echo "<br />table 1<br />";
$table1->display();

echo "<br />table 2<br />";
$table2->display();

?>

</center>
</body></html>

?>

```

[En haut](#)

● Héritage

La notion d'héritage permet de créer des classes qui *héritent* des propriétés (attributs, méthodes, constructeurs) de classes déjà définies appelées *classes parentes*. L'avantage est que l'on peut définir des classes parentes (ou mères) et des classes enfants qui auront bien sûr un air de famille, mais également des besoins sensiblement différents (comme dans une vraie famille !!).

En clair on peut **surcharger** (c-à-d redéfinir) les méthodes des classes parentes dans les classes enfants, pour palier à un besoin particulier tout en évitant de redéfinir (et donc de réécrire) les attributs et méthodes communs à tout le monde.

Dans les classes, l'héritage est défini à l'aide du mot clé **extends** :

```
class classEnfant extends classParente
```

Exemple : Nous allons créer une classe *tablecss* qui hérite de la classe *table* définie ci-dessus, la seule différence est que celle-ci pourra donner un style (au sens css) au tableau html résultant.

```

<?php

/* fichier class.tablecss.php */

/* Vu que 'tablecss' hérite de
'table' il faut inclure le fichier

```

```

où est définie 'table' */

require_once('class.table.php');

class tablecss extends table {

function table($nbLig, $nbCol, $Data) {
    $this->nbLig=$nbLig;
    $this->nbCol=$nbCol;
    $this->Data=$Data;
}

/* On surcharge la méthode 'display'
de la classe mère, elle sera donc
remplacée par celle-ci */

function display($style="") {
    if ($style != "")
        $this->TableStart("<table style=\"\$style\">");
    parent::display();
}

}
?>

```

Mise en application :

Code PHP	Affichage HTML
<pre> <?php require_once('class.table.php'); require_once('class.tablecss. php'); \$Data=array(1,2,3,4,5,6); \$table1=new table(2,3,\$Data); \$table2=new tablecss(3,3,\$Data); \$style="text-align: center; background-color: Yellow; border-style: solid; border- color: Blue; border-width: thin; </pre>	<p style="text-align: center;">table 1</p>

```
width: 200px; padding: 5px;";
```

```
?>
```

```
<html>
```

```
<head></head>
```

```
<body>
```

```
<center>
```

```
<?php
```

```
echo "<br />table 1 <br />";
```

```
$table1->display();
```

```
echo "<br />table 2 avec le style
```

```
'style'<br />";
```

```
$table2->display($style);
```

```
?>
```

```
</center>
```

```
</body></html>
```

1	2	3
4	5	6

table 2 avec le style 'style'

1	2	3
4	5	6

Explication :

-On notera d'abord que les attributs ne sont pas redéclarés dans la classe fille : c'est normal puisque celle-ci hérite des attributs de sa mère.

-En ce qui concerne la surcharge, nous avons quand même compliqué les choses : La méthode *display* est bel et bien surchargée (car redéfinie dans la classe fille), par contre on appelle la méthode de la classe mère à l'intérieur de la classe fille avec l'opérateur **parent** (et non plus **this** *).

Ce qui fait que la méthode *display* commence par modifier l'attribut *TableStart* (c'est là qu'on définit un style css à la table) puis se contente d'appeler la méthode *display* de la classe parente.

* : Ici l'appel de la méthode *display* avec l'opérateur **this** aurait été catastrophique : on appelle une méthode à l'aide d'elle-même sans jamais avoir un point d'arrêt (c'est comme une fonction récursive qui ne s'arrêterait jamais).

8. PHP & Mysql

Php fourni un support pour de nombreux **SGBD** (Système de Gestion de Base de Données). Parmi eux on trouve (liste non exhaustive), Mysql, Oracle, Ingres, Postgres, SQL Server ainsi qu'un accès générique via les pilotes ODBC (ODBC : **O**pen **D**ata **B**ase **C**onnectivy qui est un moyen standard d'accès aux bases de données sous windows).

Chaque SGBD contient un nombre de fonctions plus ou moins important mais parmi eux **Mysql** se révèle être le système pour lequel php offre le support le plus étendu en terme de fonctions permettant d'accéder et d'interroger ce type de base de données.

Nous verrons ici un aperçu de ce que l'on peut faire avec une base de données de type Mysql, pour les autres types veuillez donc vous référer à la documentation officielle le mécanisme étant cependant le même.

- [Connection et déconnection](#)
- [Requête sélection](#)
- [Requête ajout](#)
- [Requête suppression](#)
- [Requête mise à jour](#)

• **Connection et déconnection**

L'accès à une base de données mysql en php nécessite 4 paramètres :

1-Le nom d'hôte

C'est l'adresse de la machine contenant la base, en phase de développement et sur un poste local c'est souvent *localhost*, dans les autres cas cet identifiant est fourni par l'hébergeur.

2-Le nom de la base

Le nom de la base elle même, sachant qu'un SGBD peut bien sûr contenir plusieurs bases (chaque base contenant à son tour une ou plusieurs tables).

3-Le login

Le login d'accès (on parle aussi d'utilisateur de la base ou *user*), il est également fourni par l'hébergeur.

4-Le mot de passe

Le mot de passe (fourni aussi par l'hébergeur).

Exemple de connection et déconnection à une base mysql :

```
<?php
/* script de connection à une base mysql */

$db_hote="localhost"; //Nom d'hôte (ici on est en local)
```

```
$db_name="mirianet";           //Nom de la base de données
$db_user="root";              //Login ou user
$db_pass="";                  //Mot de passe

//Ouverture de la connection
$dblink=mysql_connect($db_hote, $db_user, $db_pass);

//Sélection de la base
mysql_select_db($db_name);

//Fermeture de la connection
mysql_close($dblink);

?>
```

Explication :

La fonction *mysql_connect* ouvre une connection sur l'hôte *\$db_hote*, avec les identifiants *\$db_user* et *\$db_pass* et retourne un identifiant de connection ou *false* si la connection échoue. La fonction *mysql_select_db* sélectionne sur l'hôte la base de données de nom *\$db_name*, enfin *mysql_close* ferme la connection dont l'identifiant a été retourné par *mysql_connect* (c-à-d *\$dblink*).

[En haut](#)

● Requête sélection

Comme son nom l'indique il s'agit d'une requête permettant de sélectionner des données d'une table dans une base (en vue peut être de les afficher). Comme dans tout type de requête, il faut bien sûr au préalable ouvrir une connection et à la fin la fermer. Au niveau sql ces requêtes sont caractérisées par le mot clé **SELECT**.

Exemple :

```
<?php

$db_hote="localhost";
$db_name="mirianet";
$db_user="root";
$db_pass="";

//Ouverture de la connection
$dblink=mysql_connect($db_hote, $db_user, $db_pass);

//Sélection de la base
mysql_select_db($db_name);
```

```

/*On veut ici sélectionner tous les
noms de clients de la table client */

$query="select client_nom from clients";

/* On exécute la requête en précisant la chaîne sql
et l'identifiant de la connexion */
$result=mysql_query ($query, $dblink);

// On affiche le résultat (un nom par ligne)
while($row = mysql_fetch_array($result)) {
    echo "Nom = ",$row['client_nom'], "<br />";
}

//On libère la ressource $result
mysql_free_result($result);

//Fermeture de la connexion
mysql_close($dblink);

?>

```

Ici la fonction *mysql_query* a pour but de récupérer les données définies à l'aide de la chaîne sql *\$query* pour l'identifiant de connexion *\$dblink*. (*mysql_query* retourne *false* en cas d'échec ou un identifiant de ressource).

La fonction *mysql_fetch_array* retourne dans un tableau une ligne de la ressource définie par la requête sql et passe à la ligne suivante, enfin *mysql_free_result* libère la ressource *\$result* (ce qui fait que l'on pourrait tout de suite après exécuter une autre requête sélection et utiliser la même variable *\$result* pour stocker les résultats).

Note :

Il existe une méthode condensée (*mysql_db_query*) qui exécute à la fois la sélection de la base (*mysql_select_db*) et l'exécution de la requête (*mysql_query*), mais cette méthode est déclarée **dépréciée**, ne l'utilisez pas ou plus car elle disparaîtra sûrement un jour !!

[En haut](#)

● Requête ajout

Ce type de requête permet d'ajouter des données dans une table de la base. Au niveau sql ces requêtes sont caractérisées par le mot clé **INSERT**.

Exemple :

```
<?php
```

```

$db_hote="localhost";
$db_name="mirianet";
$db_user="root";
$db_pass="";

//Ouverture de la connection
$dblink=mysql_connect($db_hote, $db_user, $db_pass);

//Sélection de la base
mysql_select_db($db_name);

//On veut ici ajouter un client

$query="insert into clients (client_id, client_nom, client_prenom,
age)";
$query.=" values (null, 'd\'albert', 'pierre', 30)";

/* On exécute la requête en précisant la chaîne sql
et l'indentifiant de la connection */
$succes=mysql_query ($query, $dblink);

if (!$succes)
    echo "Echec de la requête<br />";
else
    echo "Requête réalisée avec succès<br />";

//Fermeture de la connection
mysql_close($dblink);

?>

```

Remarques :

- La valeur retournée par *mysql_query* n'est exploitable que par un critère *vrai/faux* et il n'est pas besoin de libérer la ressource.
- Ici on suppose que *client_id* est la *clé primaire* de la table client et est de type *numéro auto*, ainsi le fait de mettre *null* pour cette valeur permet à mysql d'insérer un numéro automatiquement.
- Les chaînes de caractères doivent être encadrées de simples quotes (') et si une chaîne contient elle même une simple quote ce caractère doit être *échappé* (précédé du caractère \).

[En haut](#)

● Requête suppression

Ce type de requête permet de supprimer des enregistrements dans une table, ces requêtes sont

caractérisées par le mot clé **DELETE**.

Exemple :

On veut supprimer de la table *commandes* toutes les commandes où *commande_id* est plus petit que 100 (le champs *commande_id* étant de type entier)

```
<?php

$db_hote="localhost";
$db_name="mirianet";
$db_user="root";
$db_pass="";

//Ouverture de la connection
$dblink=mysql_connect($db_hote, $db_user, $db_pass);

//Sélection de la base
mysql_select_db($db_name);

//On supprime les commandes où commande_id plus petit que 100
$query="delete from commandes where commande_id < 100";

/* On exécute la requête en précisant la chaine sql
et l'indentifiant de la connection */
$succes=mysql_query ($query, $dblink);

if (!$success)
    echo "Echec de la requête<br />";
else
    echo "Requête réalisée avec succès<br />";

//Fermeture de la connection
mysql_close($dblink);

?>
```

[En haut](#)

- **Requête mise à jour**

Ce type de requête permet de modifier un enregistrement présent dans une table, ces requêtes sont caractérisées par le mot clé **UPDATE**.

Exemple :

On veut modifier dans la table clients le prénom du client que l'on a inséré, on suppose que ce client a pour client_id 23

```
<?php

$db_hote="localhost";
$db_name="mirianet";
$db_user="root";
$db_pass="";

//Ouverture de la connection
$dblink=mysql_connect($db_hote, $db_user, $db_pass);

//Sélection de la base
mysql_select_db($db_name);

//On change le prénom du client

$query="update clients set client_prenom='paul' where client_id=23";

/* On exécute la requête en précisant la chaine sql
et l'indentifiant de la connection */
$succes=mysql_query ($query, $dblink);

if (!$succes)
    echo "Echec de la requête<br />";
else
    echo "Requête réalisée avec succès<br />";

//Fermeture de la connection
mysql_close($dblink);

?>
```


<pre><?php \$pos = strpos("toto", "oto"); echo "1 position=", \$pos, "
"; if (\$pos >= 0) echo "1 position >=0<br / >"; \$pos = strpos("toto", "tot"); echo "2 position=", \$pos, "
"; if (\$pos >= 0) echo "2 position >=0<br / >"; \$pos = strpos("toto", "asdfs"); echo "3 position=", \$pos, "
"; if (\$pos >= 0) echo "3 position >=0<br / >"; ?></pre>	<pre>1 position=1 1 position >=0 2 position=0 2 position >=0 3 position= 3 position >=0</pre>
--	--

Pour palier à ce problème, php version >= 4 lève l'ambiguïté à l'aide de l'opérateur ===

Code PHP	Affichage HTML
<pre><?php \$pos = strpos("toto", "oto"); if (\$pos !== false) echo "1 chaine trouvée
"; \$pos = strpos("toto", "tot"); if (\$pos !== false) echo "2 chaine trouvée
"; \$pos = strpos("toto", "asdfs"); if (\$pos !== false) echo "3 chaine trouvée
"; ?></pre>	<pre>1 chaine trouvée 2 chaine trouvée</pre>

Autrement dit si on veut tester si une sous chaîne est présente en position 0 à l'intérieur d'une autre, il faut faire deux tests :

1. S'assurer que strpos retourne 0
2. Tester avec l'opérateur !== (ou ===)

Pour des tests simples on peut contourner le problème de la façon suivante.

Exemple : Je veux tester si la variable `$_SERVER['HTTP_REFERER']` (qui est une variable serveur contenant l'url rencontrée avant la présente url) commence par `http://www.mirianet.com` je peux procéder comme ceci :

```
<?php

if (strpos('X' . $_SERVER['HTTP_REFERER'], "http://www.mirianet.com") ==
1)
    echo "On vient de mirianet.com<br />";
else
    echo "On vient ne vient pas de mirianet.com<br />";

?>
```

[En haut](#)

- **Passer un tableau dans une url**

Il est évident qu'un tableau (qui est un type de variable évolué), ne peut être passé comme une variable de base (nombre ou chaîne de caractères) dans une url pour être récupéré par la méthode GET. En effet si on écrit le code suivant :

```
<?php

$tab=array('pierre', 'paul', 'jacques');

Header ("Location: test.php?tab=" . $tab);
exit(1);

?>
```

Voilà un peu ce qu'on va retrouver dans la barre d'adresse du navigateur :

http://mondomaine/test.php?tab=Array

Autrement dit inexploitable ...

Pour palier à ce problème et à condition que le tableau soit de petite taille (ne pas oublier que la longueur des arguments d'une url ne doit pas excéder 255 caractères) on peut utiliser les fonctions **serialize** et **unserialize**

Code PHP	Affichage HTML
----------	----------------

```

<?php
// Premier fichier
$tab=array('pierre', 'paul',
'jacques');

Header ("Location: test.php?
tab=".urlencode(serialize
($tab));

exit(1);

?>

<?php

// Fichier test.php
$tab=unserialize($_GET
['tab']);

echo "<pre>";
print_r($tab);
echo "</pre>";

?>

```

```

Array
(
[0] => pierre
[1] => paul
[2] => jacques
)

```

On voit que par cette méthode le tableau a bien été transmis entre les deux pages, bien sûr dans le cas de tableaux plus grands on peut utiliser des variables de session, le problème c'est que quand on utilise des variables de sessions pour passer des données d'une page à une autre, il faut parfois les supprimer de la session (dans la page d'arrivée) pour ne pas avoir de mauvaises surprises éventuelles (par exemple dans une autre page).

[En haut](#)

● Grouper des variables de sessions

On peut être amené à grouper des vraiables de session afin d'en supprimer (rapidement) certaines et pas d'autres. Supposons que l'on crée automatiquement certaines variables de session (comme par exemple la date), et lorsqu'un usager s'identifie par l'intermédiaire d'une login / mot de passe, on créer également un certain nombre de variable de session (comme son nom, son prénom ...).

On veut pouvoir supprimer uniquement les variables de sessions spécifiques à l'utilisateur sans détruire les autres :

Solution 1

On détruit à *la main* les variables une par une et si on rajoute une variable il faut aussi penser à la détruire dans le script qui les détruit.

Solution 2

On groupe les variables de session en tableau associatif, il suffit donc de détruire une *clé* du tableau pour détruire le groupe complet.

Code PHP	Affichage HTML
<pre><?php session_start(); \$_SESSION['val1'] = 'toto'; \$_SESSION['date'] = date("d/m/Y H:i"); \$_SESSION['USER']['NOM'] = 'Durand'; \$_SESSION['USER']['PRENOM'] = 'Pierre'; \$_SESSION['USER']['AGE'] = 26; if (isset(\$_SESSION['USER'])) echo "\\$_SESSION['USER'] existe
"; else echo "\\$_SESSION['USER'] n'existe pas
"; echo "<pre>"; print_r(\$_SESSION); echo "</pre>"; echo "On détruit Session['USER']
"; unset(\$_SESSION['USER']); if (isset(\$_SESSION['USER'])) echo "\\$_SESSION['USER'] existe
"; else echo "\\$_SESSION['USER'] n'existe pas
"; echo "<pre>"; print_r(\$_SESSION); echo "</pre>"; ?></pre>	<pre>\$_SESSION['USER'] existe Array ([val1] => toto [date] => 23/03/2005 17:28 [USER] => Array ([NOM] => Durand [PRENOM] => Pierre [AGE] => 26)) On détruit Session['USER'] \$_SESSION['USER'] n'existe pas Array ([val1] => toto [date] => 23/03/2005 17:28)</pre>

[En haut](#)

- **Tester l'existence d'une url**

La fonction **file_exists** teste l'existence d'un fichier local au serveur. Il peut être utile de tester l'existence d'une url (c-à-d d'une page web pas forcément sur le serveur). La fonction `url_exists` implémentée ci-dessous répond à ce problème.

```
<?php
function url_exists($url) {
    $fp=@fopen($url,"r");
    // la fonction renvoi 1 si l'url existe 0 sinon.
    return ($fp)? 1 : 0;
}
?>
```

[En haut](#)

- **Supprimer les accents d'une chaîne**

Supprimer les accents d'une chaîne de caractères peut être utile et est facile à implémenter grâce à la fonction PHP **strstr**.

Code PHP	Affichage HTML
<pre><?php function strip_accents(\$texte) { \$Accents="ÀÁÂÃÄÅàáâãäåÏÎÏîíîÛÜÛüúûüÿÑñ"; \$NoAccents="aaaaaaaaaaaaoooooooooooo eeeeeeeecciiiiiiiuuuuuuuuynn"; return strstr(\$texte, \$Accents, \$NoAccents); } \$texte="Chaîne avec différents accents ?";</pre>	<p>texte : Chaîne avec différents accents ?</p> <p>texte sans accents : Chaîne avec différents accents ?</p>

```
echo "texte : <br />";
echo $texte, "<br /><br />";

echo "texte sans accents : <br />"
echo strip_accents($texte), "<br />";

?>
```

[En haut](#)

• Mini crypteur/décrypteur

Un petit script permettant de crypter puis de décrypter un texte (Celui ou celle qui utilisera ce script pour cacher des données sensibles n'engage que sa propre responsabilité !!)

Code PHP

```
<?php
function crypte($texte) {
    $retour=" ";
    for ($i=0;$i<=strlen($texte)-1;$i++)
        $retour.=chr(ord(substr($texte,$i,1))+1);
    return $retour;
}

function decrypte($texte) {
    $retour=" ";
    for ($i=0;$i<=strlen($texte)-1;$i++)
        $retour.=chr(ord(substr($texte,$i,1))-1);
    return $retour;
}

?>

<html>
<head>
<title></title>
</head>
<body>

<?php
$texte="Les sanglots longs des violons de
```

```

l'automne bercent mon cœur d'une langueur monotone";
echo "texte d'origine=<br />",$texte,"<br /><br />";

$textecode=crypte($texte);
echo "texte crypté=<br />",$textecode,"<br /><br />";

$textedecode=decrypte($textecode);
echo "texte décrypté=<br />",$textedecode,"<br /><br />";

?>

</body>
</html>

```

Affichage HTML

```

texte d'origine=
Les sanglots longs des violons de l'automne bercent mon cœur d'une langueur monotone

texte crypté=
Mft!tbohmpu!mpoht!eft!wjpmpot!ef!m"bvupnof!cfsdfou!npo!dvs!e"vof!mbohvfvs!
npopupof

texte décrypté=
Les sanglots longs des violons de l'automne bercent mon cœur d'une langueur monotone

```

10. Liens sur Php

<http://fr3.php.net/manual/fr/>

Le site officiel (version française)

<http://www.nexen.net/>

Un site en Français avec un très bon cours, de nombreux exemples, un forum ...

<http://www.phpdebutant.org/>

Comme son nom l'indique, orienté débutants ...

<http://www.phpclasses.org/>

Site spécialisé sur les classes en PHP. De très nombreuses classes créées par les utilisateurs du site classées par catégories dans des domaines très variés. Ce site en anglais nécessite une inscription (gratuite et rapide) pour pouvoir visualiser les classes et les télécharger.

http://www.technosphere.tm.fr/chaine_serverscript/mysql/02_tutoriel_p1.cfm

Un mini tutoriel d'introduction aux bases de données relationnelles et au langage sql.