PHP-MYSQL

I-Introduction

1- Qu'est-ce que PHP ?

PHP (officiellement "PHP: Hypertext Preprocessor") est un langage de script qui est principalement utilisé pour être exécuté par un serveur HTTP, mais il peut fonctionner comme n'importe quel langage interprété en utilisant les scripts et son interpréteur sur un ordinateur. On désigne parfois PHP comme une plate-forme plus qu'un simple langage.
Sa syntaxe et sa construction ressemblent à celles des langages C et Perl, à la différence que le PHP peut être directement intégré dans du code HTML.

Exemple

```
<html>
<head>
<title>Exemple</title>
</head>
<body>
<?
        echo "Bonjour, je suis un script PHP!";
?>
</body>
</html>
```

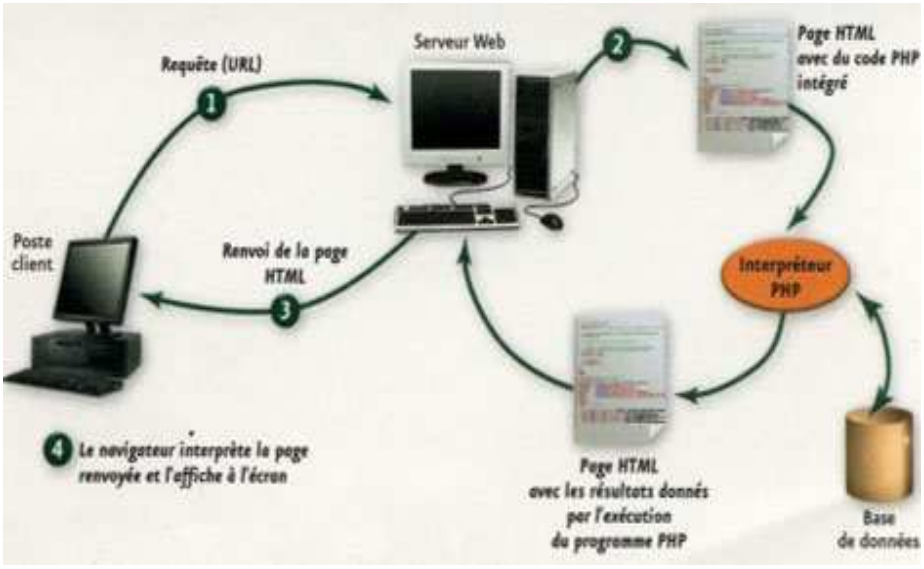
L'objet de ce langage est de permettre aux développeurs web d'écrire des pages dynamiques rapidement.

2- Fonctionnement :

PHP n'est pas un langage compilé, c'est un langage interprété par le serveur : le serveur lit le code PHP, le transforme et génère la page HTML. Pour fonctionner, il a donc besoin d'un serveur web. De ce fait une plateforme minimale de base pour l'exécution d'un site web développé en PHP comprend :

- l'interpréteur PHP (serveur PHP)
- un serveur web (Apache, IIS, …)

Lorsqu'un visiteur demande à consulter une page Web, son navigateur envoie une requête à un serveur HTTP. Si la page contient du code PHP, l'interpréteur PHP du serveur le traite et renvoie du code généré (HTML).



De ce fait le code PHP n'est jamais visible sur la page finale consultée par le client. Ainsi en éditant le source de la page on n'y trouvera que du code HTML.

3- Avantages, limitations.

Le langage PHP possède les même fonctionnalités que les autres langages permettant d'écrire des scripts CGI, comme collecter des données, générer dynamiquement des pages web ou bien envoyer et recevoir des cookies, … .

- La plus grande qualité et le plus important avantage du langage PHP est le support d'un grand nombre de bases de données et la simplicité d'interfaçage avec eux.
- PHP est utilisable sur la majorité des systèmes d'exploitation, comme Linux, de nombreuses variantes Unix (incluant HP-UX, Solaris et OpenBSD), Microsoft Windows, Mac OS X, RISC OS et d'autres encore.
- PHP supporte aussi la plupart des serveurs web actuels : Apache, Microsoft Internet Information Server, Personal Web Server, Netscape et iPlanet servers, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd et beaucoup d'autres encore.
- La gratuité et la disponibilité du code source (PHP est distribué sous licence GNU GPL)
- La simplicité d'écriture de scripts (apprentissage rapide);

II-Les fonctionnalités du langage

1- Premiers éléments du langage.

Lorsque vous créez un code, il doit être placé entre balises php pour que celui- ci soit interprété, comme ceci:

```
<?
echo 'bonjour';
?>
ou encore
<?PHP
echo 'bonjour';
?>
Ce qui affichera à l'écran ' bonjour '.
```

Une syntaxe PHP se termine TOUJOURS par un point-virgule, si vous l'oubliez vous verrez apparaître une PARSE ERROR lors de l'exécution de votre fichier. Les commentaires peuvent se présenter sous deux formes :

```
<? //ceci est
// un commentaire
/* ceci est un
 commentaire */
?>
```

2- Intégration de PHP dans une page HTML.

a- Généralités

L'un des avantages du PHP est qu'il s'intègre facilement dans du code HTML classique. Chacun peut à sa guise inclure quelques parties en PHP dans des parties de code HTML. Remarque importante : Du moment que des parties de code PHP ont été intégrées dans une page HTML cette dernière doit impérativement être renommée en extension **.php**

Exemple

```
<html>
<body>
<font size="2" face="Arial">Le texte en HTML</font> <br>
<?
echo "<font size=\"2\" face=\"Arial\"> Le texte est en PHP.</font>";
?>
<br>
<font size="2" face="Arial"> < ? echo 'Encore du texte en PHP' ?></font>
</body>
</html>
```

Le résultat obtenu sera :

Le texte est en HTML
Le texte est en PHP
Encore du texte en PHP

b- La fonction include()

La fonction include() de PHP permet d'inclure un fichier dans un autre lors de son exécution.
Un élément HTML répétitif qui apparaît dans toutes les pages de votre site (menu par exemple) pourra être isolé dans un fichier PHP. Un appel de ce fichier grâce à la fonction include() apparaîtra dans toutes les pages de votre site. Ainsi si le menu doit être par exemple modifié il suffira uniquement de changer le fichier contenant le menu.

Syntaxe

```
<?
Include ("nom_du_fichier_a_inclure") ;
?>
```

Dans l'exemple suivant un fichier menu.php contiendra le code HTML nécessaire à la génération d'un menu en HTML.
Un second fichier page.php inclura ce fichier menu.php.

Exemple fichier **menu.php**

```
<a href= "menu1.php" > Menu1 </a><br>
<a href="menu2.php" > Menu2 </a><br>
<a href="menu3.php" > Menu3 </a><br>
```

Fichier **page.php**

```
<html>
<body>
<? Include ("menu.php") ; / /inclusion du fichier contenant le menu ?>
</body>
</html>
```

Le résultat obtenu sera :

Menu1
Menu2
Menu3

Remarque : Il ne faut exécuter que le fichier page.php.
Ainsi toutes les pages du site sensées contenir ce menu intègreront grâce à la fonction include() le fichier menu.php. Il suffira de changer le fichier menu.php afin que toutes les pages qui l'intègrent se trouvent automatiquement changées.

3- Variables, chaînes et concaténation

Une variable est définie sous la forme $variable_nom.
L'affectation d'une variable se fait de la manière suivante : $variable_nom = valeur.
L'appel de la valeur affectée à une variable se fait par son nom.

Syntaxe

```
<?
$variable1 = 'Bonjour 1'; //Affectation d'une chaîne avec quote simple
$variable2 = "Bonjour 2"; // Affectation d'une chaîne avec quote double
$variable3 = 5; // Affectation d'un entier
$variable4 = 2 + (3 * 5); // Affectation d'un résultat d'opération
$variable5 = true; // Affectation Booléenne
?>
```

L'affichage des variables combinées à des chaînes de caractères peut se faire de plusieurs manières en utilisant les cotes simples (') ou les doubles cotes (").

Exemple 1

```
<?
$nom = 'visiteur';
echo "bonjour $nom";
?>
```

Le résultat obtenu sera :

bonjour visiteur

Exemple 2

```
<?
$nom = 'visiteur';
echo 'bonjour '.$nom;
?>
```

Le résultat obtenu sera :

bonjour visiteur

Exemple 3

```
<?
$nom = 'visiteur';
echo 'bonjour $nom'; // La variable $nom ne sera pas interprétée
?> j
```

Le résultat obtenu sera :

bonjour $nom

Remarques : Si vous utilisez les ' au lieu des " , la variable n'est pas interprétée comme une variable mais comme une chaîne de caractère.

Remarque sur l'usage des cotes simples ou doubles

Lors de l'usage des ' pour l'affichage d'une chaîne de caractère contenant des apostrophes, vous devez impérativement faire précéder ces apostrophes d'antislash. De la sorte l'apostrophe ne sera pas confondue avec le caractère de fin de la chaîne à afficher. Dans le cas contraire un message d'erreur sera affiché.

Exemple 4

```
<?
echo 'vous n\'êtes pas inscrit';
?>
```

Le résultat obtenu sera :

Vous n'êtes pas inscrit

Il en va de même lors de l'usage de " pour l'affichage d'une chaîne de caractère contenant au préalable des doubles cotes.

```
<?
echo"<a href=\"http://www-etu\">lien.php</a>";
?>
```

Le résultat obtenu sera :

Lien.php

4- Conditions et boucles.

a- Conditions if

La syntaxe de base d'une instruction conditionnelle est la suivante :

```
<?
 if($var == 'condition')
{
 // 'condition vérifiée';
}
else
{
//'condition non vérifiée' ;
}
?>
```

Les opérateurs de contrôle sont les suivants :

==	strictement égal
!=	différent
>	plus grand que
<	inférieur à
>=	supérieur à
<=	inférieur à
&&	et
||	ou
AND	et
OR	ou
TRUE	1 ou oui
FALSE	0 ou non

Exemple

```
<?
$montant='100';
```

```
if ($montant <1000) { echo 'Montant inférieur à 1000'; }
else { echo 'Montant supérieur à 1000'; }
?>
```

Le résultat obtenu sera :

Montant inférieur à 1000

b- Conditions elseif

```
<?
If ($var == 'condition1')
{
        // 'condition1 vérifiée';
}
elseif ($var == 'condition2')
{
        // 'condition2 vérifiée';
}
...
elseif ($var == 'conditionN)
{
        // 'conditionN vérifiée';
}
...
else
{
        echo 'Aucune condition n'est vérifiée';
}
?>
```

Les structures **elseif** pouvant se répéter autant de fois que des conditions prévues l'exigeront.

Exemple 1

```
<?
$variable = 3;
// serie de tests ,
if ($variable == 1)
{
echo 'La variable a pour valeur 1';
}
elseif ($variable == 2)
{
echo 'La variable a pour valeur 2';
}
elseif ($variable == 3)
{
echo 'La variable a pour valeur 3';
}
elseif ($variable == 4)
{
echo 'La variable a pour valeur 4';
}
elseif ($variable == 5)
{
echo 'La variable a pour valeur 5';
}
else
{
echo 'La variable n'appartient pas à l'intervalle [15]';
}
?>
```

Le résultat obtenu sera :

La variable a pour valeur 3

c- Conditions SWITCH

Le switch est une structure qui permet d'éviter la lourdeur de l'écriture de l'exemple 2. Elle permet en outre une meilleure lisibilité.

Syntaxe

```
<?
switch ($variable)
{
        case condition1:
                //Traitement de la condition 1
                break;
        case condition2:
                //Traitement de la condition 2
                break;
                ….
        case conditionN:
                //Traitement de la condition N
                break;
        default:
                //Traitement par défaut
}
?>
```

Exemple

```
<?
$variable = 3;
switch ($variable)
{
        case 1:
                echo 'La variable a pour valeur 1';
                break;
        case 2:
                echo 'La variable a pour valeur 2';
                break;
        case 3:
                echo 'La variable a pour valeur 3';
                break;
        case 4:
                echo 'La variable a pour valeur 4';
                break;
        case 5:
                echo 'La variable a pour valeur 5';
                break;
        default:
                echo 'La variable n'appartient pas à l'intervalle [1,5]';
}
?>
```

Le résultat obtenu sera :

La variable a pour valeur 3

d- Itération avec WHILE

Syntaxe

```
<?
While (condition)
        {
        //Traitements
        }
?>
```

Exemple

```
<?
$nbre_maximum = 6;
$i = 0; //initialisation de l'indice d'incrémentation
while ($i < $nbre_maximum) //condition
        {
```

```
                    echo $i.' est inférieur à '. $nbre_maximum.'<br>';
                    $i++; // $i++ est équivalent à ($i+1)
                }
        echo $i.' est égal à '. $nbre_maximum;
        ?>
```

Le résultat obtenu sera :

0 est inférieur à 6
1 est inférieur à 6
2 est inférieur à 6
3 est inférieur à 6
4 est inférieur à 6
5 est inférieur à 6
6 est égal à 6

e- Itération avec FOR

Syntaxe

```
        <?
        for($i=0; $i != condtion ; $i++)
                {
                //Traitements réalisés
                }
        ?>
```

Exemple

```
        <?
        $nbre_maximum = 6;
        //----------DEBUT BOUCLE----------
        for ($i=0; $i != $nbre_maximum ; $i++)
                {
                echo $i.'est inférieur à '. $nbre_maximum.'<br>';
                }
        //----------FIN BOUCLE----------
        echo $i.' est égal à '. $nbre_maximum;
        ?>
```

Le résultat obtenu sera :

0 est inférieur à 6
1 est inférieur à 6
2 est inférieur à 6
3 est inférieur à 6
4 est inférieur à 6
5 est inférieur à 6
6 est égal à 6

5- Fonctions

PHP propose une palette approximative de 2000 fonctions prédéfinies. Toutefois il vous est possible de créer vos propres bibliothèques de fonctions pour vos usages spécifiques, une meilleure lisibilité du code et une réutilisation.

a- Définition des fonctions

Les fonctions peuvent se distinguer en deux sous groupes :

- les fonctions qui effectuent un traitement (affichage par exemple)
- les fonctions qui effectuent un traitement et retournent un résultat

Syntaxe

```
        function nom_de_la_fonction ($paramètres)
                {
                //traitement sur les paramètres effectué
                }
```

L'appel de la fonction se fait de la manière suivante :

```
        nom_de_la_fonction ($paramètres) ;
```

Exemple

```
<?
function afficher_nom_prenom ($nom,$prenom)
        {
        echo 'Bonjour '.$nom. ' '.$prenom ; .
        }
afficher_nom_prenom ('Tarak','Joulak') ;
echo ' <br>' ;
$nom1='Mourad' ;
$prenom1='Zouari' ;
afficher_nom_prenom ($nom1,$prenom1) ;
?>
```

Le résultat obtenu sera :

Bonjour Tarak Joulak
Bonjour Mourad Zouari

Dans le cas suivant un traitement est effectué à l'intérieur de la fonction puis retourné par cette dernière. De ce fait la fonction doit donc être affectée à une variable.

Syntaxe

```
function nom_de_la_fonction ($paramètres)
{
//traitement sur les paramètres effectué
return ($resultat) ;
}
```

L'appel de la fonction se fait de la manière suivante :

```
$variable = nom_de_la_fonction ($paramètres) ;
```

Exemple

```
<?
        function additionner ($variable1,$variable2)
                {
                $total = $variable1 + $variable2;
                return ($total) ;
                }
        $resultat= additionner (1,2) ;
        echo $resultat.' <br>' ;
        $var1=6 ;
        $var2=7 ;
        $resultat= additionner ($var1,$var2) ;
        echo $resultat.' <br>' ;
?>
```

Le résultat obtenu sera :

3
13

b- Librairie de fonctions

Idéalement toutes les fonctions créées devraient être regroupées dans un même fichier créant ainsi une bibliothèque de fonctions. Ce fichier sera appelé à l'intérieur des autres fichiers par le biais de la fonction include.

Exemple fichier fonction.inc.php

```
<?
//Ce fichier contiendra l'ensemble des fonctions que vous développerez
function additionner ($variable1,$variable2)
{
$total = $variable1 + $variable2;
 return ($total) ;
}
function afficher_nom_prenom ($nom,$prenom)
```

```
{
echo 'Bonjour '.$nom. ' '.$prenom ;
 }
?>
```

Exemple fichier page.php

```
<?
Include (”fonction.inc.php”) ;
$resultat= additionner (1,2) ;
echo $resultat.' <br>' ;
afficher_nom_prenom ('Tarak','Joulak') ;
?>
```

Le résultat obtenu sera :

```
3
Bonjour Tarak Joulak
```

6- Tableaux

a- Tableaux numérotés et tableaux associatifs

Il existe deux types de tableaux de variables sous PHP :

- Les tableaux à index numériques (tableaux numérotés) dans lesquels l'accès à la valeur de la variable passe par un index numérique
  ex : $tableau[0], $tableau[1], $tableau[2], …
- Les tableaux à index associatifs (ou tableaux associatifs) dans lesquels l'accès à la valeur de la variable passe par un index nominatif
  ex : $tableau[nom], $tableau[prénom], $tableau[adresse], …

Syntaxe

```
//Tableau à index numéroté
$tableau = array (valeur0,valeur1,valeur2, …) ;
//Accès à chacune des valeurs
$tableau[0] donnera valeur0
$tableau[1] donnera valeur1
…
//Tableau à index associatif
$tableau = array (variable1 => valeur1, variable2 => valeur2, …) ;
//Accès à chacune des valeurs
$tableau[variable1] donnera valeur1
$tableau[variable2] donnera valeur2
…
```

Exemple

```
<?
//Tableau à index numéroté
$tableau1 = array ('Château','Maison','Bateau') ;
echo "Contenu du tableau 1 :<br>";
echo $tableau1[0]."<br>";
echo $tableau1[1]."<br>";
echo $tableau1[2]."<br>";

//Tableau à index associatif . .
$tableau2 = array ('prenom' =>'Tarak','nom' =>'Joulak','ville' =>'Marrakech') ;
echo "Contenu du tableau 2 :<br>";
echo $tableau2['prenom']"<br>";
echo $tableau2['nom']."<br>";
echo $tableau2['ville']"<br>";
?>
```

Le résultat obtenu sera :

Contenu du tableau 1 :
Château
Maison
Bateau
Contenu du tableau 2 :
Tarak
Joulak
Marrakech

b- Parcours des tableaux

PHP intègre une structure de langage qui permet de parcourir un à un les élément d'un tableau : **foreach().**

foreach ($tableau **as** $valeur)

Syntaxe

```
{
//Appeller ici la valeur courante par $valeur
…
}
```

Exemple

```
<?
//Cas du tableau numéroté
$tableau1 = array ('chateau','maison','bateau') ;
foreach ( $tableau1 as $valeur )
{
echo $valeur."<br>";
}
?>
```

Le résultat obtenu sera :

château
maison
bateau

Exemple

```
<?
//Cas du tableau associatif
$tableau2 = array ('prenom' =>'Tarik','nom' =>'Joulak','ville' =>Marrakech) ;
foreach ( $tableau2 as $valeur )
{
echo $valeur."<br>";
}
?>
```

Le résultat obtenu sera :

Tarik
Joulak
Marrakech

Dans l'exemple suivant et pour le cas des tableaux associatifs nous pouvons grâce à la boucle foreach() aussi bien énumérer le nom des variable que leur valeur.

Exemple

```
<?
$tableau2 = array ('Prenom' =>'Tarak','Nom' =>'Joulak','Ville' =>Marrakech) ;
foreach ( $tableau2 as $variable=>$valeur )
{
echo " $variable a pour valeur  $valeur."<br>";
}
?>
```

Le résultat obtenu sera :

Prenom a pour valeur Tarak
Nom a pour valeur Joulak
Ville a pour valeur Marrakech

c- Recherche dans un tableau

array_key_exists() permet de vérifier si dans un tableau associatif une variable associative (clef) existe ou non. Elle retourne donc une valeur booléenne (True ou False).

Syntaxe

```
$resultat= array_key_exists('nom_de_la variable', tableau_associatif) ;
```

Exemple

```
<?
$tableau2 = array ('prenom' =>'Tarak','nom' =>'Joulak','ville' =>Marrakech) ;
if (array_key_exists("nom", $tableau2))
{
```

```
echo 'La variable "Nom" se trouve dans le tableau et a pour valeur: '.$tableau2['nom'];
}
else
{
echo 'La variable "Nom" ne se trouve pas dans le tableau';
}
?>
```

Le résultat obtenu sera :
La variable "Nom" se trouve dans le tableau et a pour valeur: Joulak

in_array() permet de déterminer si une valeur existe dans le tableau. Elle retourne donc une valeur booléenne (True ou False).

Syntaxe

```
$resultat=in_array ( nom_de_la_valeur, tableau) ;
```

Exemple

```
<?
$tableau2 = array ('prenom' =>'Tarak','nom' =>'Joulak','ville' =>Marrakech) ;
if (in_array("Tarak", $tableau2))
{
echo 'La valeur "Tarak" existe bien dans le tableau';
}
else
{
echo 'La valeur "Tarak" ne se trouve pas dans le tableau';
}
?>
```

Le résultat obtenu sera :
La valeur "Tarak" existe bien dans le tableau

array_search fonctionne comme in_array : il travaille sur les valeurs d'un tableau.
Si il a trouvé la valeur, array_search renvoit la clé correspondante (c'est-à-dire le numéro si c'est un tableau numéroté, ou le nom de la variable si c'est un tableau associatif). Si il n'a pas trouvé la valeur, array_search renvoit false (comme in_array).

Syntaxe

```
$resultat=array_search ( nom_de_la_valeur, tableau) ;
```

Exemple

```
<?
$tableau2 = array ('prenom' =>'Tarak','nom' =>'Joulak','ville' =>Marrakech) ;
if ($position = array_search("Marrakech ", $tableau2))
{
echo '"Marrakech" se trouve en position ' . $position . '<br />';
}
else
{
echo '" Marrakech " ne se trouve pas dans le tableau.';
}
?>
```

Le résultat obtenu sera :
"Marrakech" se trouve en position ville

IV- Accès fichiers

1- Ouverture / fermeture de fichier

Avant toute opération de lecture ou écriture sur un fichier il y a nécessité de l'ouvrir.
Et à la fin de tout traitement d'un fichier il y a nécessité de le fermer.

Syntaxe

```
<? /
/ Ouverture de fichier
$monfichier = fopen("nom_du_fichier", "r+"); /
/ Traitements sur le fichier /
/ ….. /
/ Fermeture du fichier
fclose($monfichier);
?>
```

Fopen() prend en entrée :
-le nom du fichier (ou meme une url)
-le mode d'ouverture du fichier
Il retourne un handle.
Fclose() prend en entrée le handle envoyé par le fopen.
Les modes d'ouverture d'un fichier sont les suivants :

- '**r**' Ouvre en lecture seule, et place le pointeur de fichier au début du fichier.
- '**r+**' Ouvre en lecture et écriture, et place le pointeur de fichier au début du fichier.
- '**w**' Ouvre en écriture seule ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
- '**w+**' Ouvre en lecture et écriture ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
- '**a**' Ouvre en écriture seule ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.
- '**a+**' Ouvre en lecture et écriture ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.

2- Lecture de fichier

a- fgets() string **fgets** (resource handle ,int length)
fgets retourne la chaîne lue jusqu'à la longueur length - 1 octet depuis le pointeur de fichier handle , ou bien la fin du fichier, ou une nouvelle ligne (qui inclue la valeur retournée), ou encore un EOF (celui qui arrive en premier). Si aucune longueur n'est fournie, la longueur par défaut est de 1 ko ou 1024 octets.
La fonction prend en paramètre :
- le handle retourné par la fonction fopen()
- la taille en octets de lecture (optionnel)

Elle retourne en sortie une chaîne de caractères.

Exemple :

L'exemple suivant va permettre de lire dans un fichier texte (fichier.txt) ligne par ligne puis d'afficher le résultat à l'écran en mettant un numéro à chaque ligne.

Fichier fichier.txt

```
Cours php
A Marrakech
```

Fichier page.php

```
<?
$i=0;
$fichier = 'fichier.txt';
$fp = fopen($fichier,'r') ; //ouverture du fichier en lecture seulement
while ( !feo($fp)) // tant qu'on n'est pas a la fin du fichier
{
$ligne = fgets($fp); //Lecture ligne par ligne
echo 'Ligne '.$i++.'--->'.$ligne.'<br>';
}
```

```
fclose($fp);
?>
```

Le résultat obtenu sera :

Ligne 2 ---> Cours php
Ligne 3 ---> A Marrakech

b -fread()

string fread (resource handle , int length)
Une autre manière de faire de la lecture dans un fichier est d'utiliser fread()
fread lit jusqu'à length octets dans le fichier référencé par handle . La lecture s'arrête lorsque length octets ont été lus, ou que l'on a atteint la fin du fichier, ou qu'une erreur survient (le premier des trois).

Exemple :

```
<?
// Lit un fichier, et le place dans une chaîne
$filename = "fichier.txt";
$handle = fopen ($filename, "r");
$contents = fread ($handle,filesize ($filename));
echo "-->".$contents; fclose ($handle);
?>
```

Le résultat obtenu sera :

--> Cours php A Marrakech

c- file()

array **file** (string filename)
Encore une autre manière de lire dans un fichier est d'utiliser la fonction **file()** Identique à readfile, hormis le fait que file retourne le fichier dans un tableau. Chaque élément du tableau correspondant à une ligne du fichier, et les retour−chariots sont placés en fin de ligne.

Exemple :

```
<?php
$filename = "test/fichier.txt";
$fcontents = file($filename);
echo $fcontents[0].”<br>";
echo $fcontents[1]."<br>";
?>
```

Le résultat obtenu sera :

Cours php
A Marrakech

3- Ecriture dans fichier

La fonction pour l'écriture dans un fichier est fputs()
int fputs (int handle , string string , int length)
fputs écrit le contenu de la chaîne string dans le fichier pointé par handle . Si la longueur length est fournie, l'écriture s'arrêtera après length octets, ou à la fin de la chaîne (le premier des deux).
fwrite retourne le nombre d'octets écrits ou FALSE en cas d'erreur.

Syntaxe

```
<?
$monfichier = fopen("nom_du_fichier", "r+");
fputs ($monfichier, "Texte à écrire");
fclose ($monfichier);
?>
```

Exemple

```
<?
$filename = "fichier.txt";
$monfichier = fopen ($filename, "a+");
$contenu="ceci est le texte a ecrire \r\n";
fputs($monfichier, $contenu);
fclose ($monfichier);
?>
```

Le résultat obtenu sera :
Le fichier fichier.txt aura une ligne supplémentaire contenant « ceci est le texte à écrire »

4- Fonctions diverses de traitement de fichier :

- **feof** ($handle) fonction qui retourne un booleen pour indiquer la fin du fichier parcouru. Elle prend en entrée le handle retourné par la fonction fopen().
- **filesize** ($nom_du_fichier) fonction qui retourne la taille du fichier en octets.
- **file_exists** ($nom_du_fichier) fonction qui retourne un booleen indiquant si le fichier existe ou non.

IV- Passage et transmission de variables

1-Passage et transmission de variables par formulaire

Quand dans un site web un formulaire est rempli et envoyé, le contenu des champs saisis est transféré à la page destination sous forme de variables. Ce passage de variables ou de paramètres peut se faire de deux manières : en GET ou en POST.

Syntaxe

```
<html>
<body>
<!—Envoi d'un formulaire en POST -->
-
<form method="post" action="destination.php">
        <input type="text" ville="nom" size="12"><br>
        <input type="submit" value="OK">
</form>
<!—Envoi d'un formulaire en GET ->
<form method="get" action=" destination.php">
        <input type="text" name="nom" size="12"><br>
        <input type="submit" value="OK">
</form>
</body>
</html>
```

En GET les paramètres apparaissent associés à l'url sous formes de variables séparées par des & (http://localhost/destination.php?ville=Marrakech).
En POST le passage de paramètre se fait de manière invisible.
Selon que la méthode d'envoi a été du GET ou du POST la récupération du contenu des variables est faite selon une syntaxe différente :

Syntaxe

```
<?
//Dans le cas d'un envoi des paramètres en POST
$variable1=$_POST['nom_du_champ'] ;
//Dans le cas d'un envoi des paramètres en GET
$variable1=$_GET['nom_du_champ'] ;
?>
```

Remarque :
Si les paramètres sont envoyées à la page elle-même, (action="formulaire.php") aurait pu ne pas être nominative mais exploiter une variable serveur PHP_SELF . Ainsi $_SERVER['PHP_SELF'] retournera naturellement formulaire.php.
Cela donnera donc :

```
<form method="post" action="<? echo $_SERVER['PHP_SELF']; ?>" >
```

à la place de

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

2-Passage et transmission de variables par hyperlien

Des paramètres ou variables peuvent passer d'une page source vers une page destination sans transiter par un formulaire pour leur envoi. Les hyperliens peuvent être des vecteurs de passage de paramètre.

Syntaxe

```
<!—Syntaxe d'envoi -->
<a href=destination.php ?variable1=contenu1&variable2=contenu2&...> Lien </a>
```

La récupération des paramètres dans la page destination se fait par le tableau **$_GET**

Exemple :

Dans l'exemple suivant nous allons créer un fichier menu.php contenant un menu fait d'hyperliens. Chacun de ces hyperliens enverra des paramètres différents.
Ce menu sera appelé dans une page qui réagira différemment selon le paramètre envoyé.

Fichier menu.php

```
<table width="200" border="0" cellspacing="0" cellpadding="1">
        <tr>
                <td> <a href="page.php?menu=1">Menu1</a> </td>
        </tr>
        <tr>
                <td> <a href="page.php?menu=2">Menu2</a></td>
        </tr>
        <tr>
                <td> <a href="page.php?menu=3">Menu3</a> </td>
        </tr>
</table>
```

Fichier page.php

```
<?
Include("menu.php") ;
echo "<br><br>" ;
$menu= $_GET['menu'] ;
switch ($menu)
        {
        case 1:
                echo "Ceci est la page obtenue du Menu1" ;
                break;
        case 2:
                echo "Ceci est la page obtenue du Menu2" ;
                break;
        case 3:
                echo "Ceci est la page obtenue du Menu3" ;
                break;
        default:
                echo "Ceci est la page d'accueil" ;
        }
?>
```

En exécutant à travers le serveur web page.php, la sélection de chacun des menus chargera à nouveau la page en envoyant des paramètres différents qui seront traités par la page.

3- Redirection

La fonction essentielle de la redirection consiste à sitôt que l'instruction a été trouvée de l'exécuter en redirigeant l'utilisateur vers la page spécifiée.

Header()

int **header**(« Location :string destination)

Syntaxe

```
<?
header("Location:$page_destination");
exit() ;
?>
```

Exemples

```
<?
header("Location:http://www.google.com");
exit() ;
?>
```

Le résultat obtenu sera :

Vous serez automatiquement redirectionné vers le site de google

VI-Variables persistantes: Cookies et Session

Les variables ont une durée de vie limitée : celle du script qui les appelle. Ainsi que nous l'avons vu dans le chapitre précédent l'unique moyen de transmettre ces variables de pages en pages consiste à effectuer un passage de paramètres (méthode GET ou POST) ce qui d'une manière générale est contraignant à plus d'un titre :

- Contraignant pour le développeur puisque il doit gérer par code ces passages de paramètres,
- Contraignant pour la sécurité si l'on ne désire pas que le client accède à certaines informations.

PHP offre un mécanisme de stockage d'informations de manière persistante. Autrement dit tout au long de la navigation à l'intérieur d'un site des variables seront accessibles sans avoir pour autant à les passer en paramètres. Deux types de variables persistantes existent :

- Les variables persistantes côté client : les cookies
- Les variables persistantes côté serveur : la session

1- les Cookies

Les cookies sont un mécanisme d'enregistrement d'informations sur le client, et de lecture de ces informations. Ce système permet d'authentifier et de suivre les visiteurs d'un site. PHP supporte les cookies de manière transparente.

a- setcookie() : Cette fonction permet de définir un cookie qui sera envoyé avec le reste des en–têtes.

int setcookie (string nom_variable ,[string valeur_variable],[int expiration],[string chemin],[string domaine],[int securité])

nom_variable : nom de la variable a stocker
valeur_variable : Valeur de la variable a stocker
expiration : durée pour l'expiration du cookie
chemin : le chemin du répertoire ou doit être lu le cookie
domaine : le nom domaine
securité : le type d'entête (http, https)

Tous les arguments sauf nom_variable sont optionnels.

A l'instar de la fonction header(), setcookie() doit impérativement être appelée avant tout affichage de texte.

Syntaxe

```
<?
setcookie(variable1,$valeur1, $durée,$chemin,$domaine,$securite);
?>
```

Exemples

```
<?
$valeur= "Ceci est la valeur de la variable" ;
setcookie (TestCookie,$value,time()+3600); /* expire dans une heure */
?>
```

b- lire un cookie

Syntaxe

```
<?
$c=$HTTP_COOKIE_VARS["$nom_de_la_variable"] ;
?>
```

Exemple

```
<?
echo $HTTP_COOKIE_VARS["TestCookie"];
?>
```

Le résultat obtenu sera :

Ceci est la valeur de la variable

c- Tableau de variables dans un cookie

Il est possible d'envoyer un tableau de variables à stocker dans un cookie

Exemple

```
<?
setcookie( "tcookie[trois]", "troisième cookie" );
```

```
setcookie( "tcookie[deux]", "second cookie" );
setcookie( "tcookie[un]", "premier cookie" );
?>
```

d- Limitation des cookies

Le problème majeur du cookie c'est que le client a le pouvoir de le refuser (en configurant spécifiquement son browser). Votre application risque donc de ne pas pouvoir fonctionner.
Il y a aussi des risques plus graves quant à la sécurité. L'usurpation d'identité, car ce fichier peut être recopié facilement sur un autre ordinateur et modifié puisque ce n'est qu'un fichier texte.

2- les sessions

La gestion des sessions avec PHP est un moyen de sauver des informations entre deux accès. Cela permet notamment de construire des applications personnalisées, et d'accroître l'attrait de votre site.
Chaque visiteur qui accède à votre site se voit assigner un numéro d'identifiant, appelé plus loin "identifiant de session". Celui−ci est enregistré soit dans un cookie, chez le client, soit dans l'URL.
Les sessions vous permettront d'enregistrer des variables pour les préserver et les réutiliser tout au long de la visite de votre site. Lorsqu'un visiteur accède à votre site, PHP vérifiera si une session a déjà été ouverte. Si une telle session existe déjà, l'environnement précédent sera recréé.
L'inconvénient précédemment évoqué concernant les cookies est dépassé dans la mesure ou tout est stocké sur le serveur même.

a- session_start()

session_start() permet de démarrer une session pour le client .

Syntaxe

```
<?
session_start() ;
?>
```

Cette commande doit figurer dans toutes les pages elle perpétue le transfert des variables de session au cours de la navigation dans le site.
Le compilateur PHP va alors créer alors dans le répertoire de sauvegarde des sessions, un fichier dont le nom commence par sess_ et se termine par un identifiant généré de manière aléatoire.
L'identifiant de session peut être affiché par la commande session_id(). Vous pouvez également gérer vous-même ce nom de session en utilisant session_name() avant le démarrage de la session.
La durée de vie d'une session est paramétrée par session.cache_expire La session est perdue définitivement pour l'utilisateur lorsque :

1- Il n'a exécuté aucune action (POST ou GET) au delà de la durée de vie définie.
2- Il ferme son navigateur.
3- La commande session_destroy est appelée.

b- Ajouter des variables dans la session

L'ajout de variable dans l'environnement de session se fait au travers d'une affectation classique.
Toutefois la variable session définie doit être appelée vie la tableau **$_SESSION[]**

Syntaxe

```
<?
session_start() ;
$_SESSION['nom_variable'] = $valeur;
?>
```

Exemple

```
<?
session_start() ;
$_SESSION['ville'] = "Marrakech";
?>
```

Dans cet exemple une variable du non de ville contenant la valeur Marrakech a été enregistrée dans la session courante.

c- Lire une variable dans la session

Syntaxe

```
<?
session_start() ;
```

```
$variable = $_SESSION['nom_variable'] ;
?>
```

Exemple

```
<?
session_start() ;
echo ' Le contenu de la variable VILLE de la session est : '. $_SESSION['ville'] ;
?>
```

Le résultat obtenu sera :

Le contenu de la variable VILLE de la session est : Marrakech

d- Supprimer une variable de la session

Syntaxe

```
<?
session_start() ;
unset ($_SESSION['nom_de_le_variable']);
?>
```

Exemple

```
<?
session_start() ;
unset ($_SESSON['ville']);
//Verificatin de la suppression
if ( isset ($_SESSON['ville']))
{
        $resultat = "La suppression a échoué .";
 }
else
{
        $resultat = "La ville a été effacée.";
}
echo $resultat;
?>
```

Le résultat obtenu sera :

La ville a été effacée.

e- Supprimer l'environnement de session

Dans l'exemple précédent nous avons vu comment supprimer une variable de l'environnement de session. Il reste toutefois possible de supprimer tout un environnement de session donné. Pour cela il suffit de vider le tableau global des sessions en le réinitialisant.

Syntaxe

```
<?
session_start() ;
$_SESSION = array();
?>
```

VIII- Access aux dossiers

Il est possible de parcourir les dossiers grâce aux fonctions suivantes

- Chdir($str) : charge le dossier courant dans la chaine $str. Retourne TRUE si succès, ou FALSE dans le cas contraire.
- Getcwd() : retourne le nom du dossier courant en format chaîne de caractères ;
- Opendir($str) : ouvre le dossier $str, et récupère un pointeur. Elle gère aussi une erreur php.
- Closedir($id) : ferme le pointeur du dossier $id.
- Readdir($id) : lit une entrée du dossier pointée par $id. càd retour le nom de fichier contenu dans le dossier ou FALSE si il n'y a plus de fichiers.
- Rewinddir($id) : retourne à la premiere entrée du dossier.

Exemple

```
< ?php
        If ($id=@opendir('.'){
                While ($file= readdir($id){
                        Echo « $file <br> » ;
                }
        }
?>
```

On peut aussi utiliser la pseudo-classe « dir » pour accéder aux dossiers :

- les attributs sont :
  - handle : valeur du pointeur
  - path :nom du dossier
- les méthodes sont :
  - read() : lecture
  - close() : fermeture
- le constructeur :
  - dir ($str) : retourne un objet dir et ouvre le dossier $str ;

Exemple :

```
< ?php
        $d=dir('.') ;
        Echo « pointeur : ».$d->handle. »<br> ;
        Echo "Chemin :".$d->path."<br>;
        While ($ent=$d->read()){
                Echo $ent . "<br>;
        }
        $d->close();
?>
```

Ces deux exemples listent les fichiers et les sous-dossiers du dossier courant.

IX- Chargement de fichier

Les formulaires permettent aussi de télécharger les fichiers vers un serveur.
C'est la balise <input type= « file » /> qui permet le chargement des fichiers.

X- Utilisation d'une base de données MySql

1- Introduction

PHP fonctionne nativement avec une base de données MYSQL.
MYSQL est un système de gestion de base de données (SGBD) qui permet d'entreposer des données de manière structurée (Base, Tables, Champs, Enregistrements). Le noyau de ce système permet d'accéder à l'information entreposée via un langage spécifique le SQL.
Ainsi, dans les chapitres précédents nous avions vu que l'information au mieux pouvait être stockée dans des fichiers accessibles en lecture et écriture par des scripts PHP.
MYSQL vient ajouter une couche supplémentaire de stockage des données qui est plus commode, rapide et puissante d'utilisation.

Schéma de liaison entre apache/PHP/MYSQL

Voici ce qu'il peut se passer lorsque le serveur reçoit une demande d'un client de consultation d'une page en PHP qui fait appel à des données stockées sous MYSQL:

1. Le serveur WEB envoie le nom de la page PHP demandée à l'interpréteur PHP.
2. PHP exécute le script existant dans la page. Sitôt que des instructions relatives à la connexion à une base de données trouvées, PHP se charge d'envoyer les requêtes d'exécution à MYSQL.
3. MySQL exécute la requête et renvoie à PHP le jeu de données résultat.
4. PHP termine son traitement et renvoie la page HTML générée au serveur web qui la transmet à l'internaute.

2- Accéder à MYSQL via PHP

a- Connexion à un serveur MYSQL

Exemple

```
<?
$nom_serveur_MYSQL="localhost" ;
$utilisateur="root" ;
$mot_de_passe="" ;
mysql_connect("$nom_serveur_MYSQL", "$utilisateur", "$mot_de_passe");
?>
```

La fonction **mysql_connect()** retourne un booléen ; True si la connexion est possible False si elle ne l'est pas.

b- Connection à une base de données MYSQL

Suite à la connexion au serveur MYSQL, il faut choisir quelle est la base du serveur MYSQL sur laquelle nous désirons nous connecter. Un serveur MYSQL pouvant contenir plusieurs bases de données.

Exemple

```
<?
$adresse_serveur_MYSQL="localhost" ;
$utilisateur="root" ; $mot_de_passe="" ;
$nom_de_la_base="exercice" ;
mysql_connect("$adresse_serveur_MYSQL", "$utilisateur", "$mot_de_passe");
mysql_select_db("$nom_de_la_base");
?>
```

De même la fonction **mysql_select_db()** retourne un booléen ; True si la connexion est réussie, False si elle a échoué.

c- Exécution d'une requête SQL via PHP

Le lancement d'une requête SQl au travers d'un script PHP se fait par le biais de la fonction mysql_query() qui prend en paramètre la requête SQL et retourne true ou false selon que la requête ait réussi ou échoué. Pour le cas particulier d'une requête avec SELECT et si la requête a réussi alors un identifiant est retourné afin d'être exploité par d'autres fonctions.

Exemple

```
<?
$adresse_serveur_MYSQL="localhost" ;
$utilisateur="root" ; $mot_de_passe="" ;
$nom_de_la_base="exercice" ;
$requete_sql="INSERT INTO t_personne (id, Nom, Prenom, Age) VALUES (NULL,'Joulak', 'Tarak', 32)" ;
mysql_connect("$adresse_serveur_MYSQL", "$utilisateur", "$mot_de_passe");
mysql_select_db("$nom_de_la_base");
$res = mysql_query("$requete_sql");
?>
```

Dans l'exemple précédent une ligne d'enregistrement a été ajoutée à la table t_personne contenant Tarak Joulak 32. En remplaçant la requête de l'exemple précédent par les requêtes SQL de modification et suppression on obtient les résultats déjà observés en exécutant directement la requête SQL.

d- Parcourir le résultat d'un SELECT

Une requête SQL de consultation peut retourner plusieurs enregistrements. De ce fait il y a besoin de pouvoir les consulter enregistrement par enregistrement et champs par champs.

Trois fonctions peuvent être utilisées pour récupérer les informations référencées par $resultat, après appel de mysql_query() sur une requête de type SELECT : mysql_fetch_array(), mysql_fetch_row(), mysql_fetch_object().

`$ligne = mysql_fetch_row($resultat);`

Cette fonction retourne un tableau indicé nommé ici `$ligne`, indicé à partir de 0, dont les éléments contiennent les valeurs des champs de la ligne courante pointée par *$resultat*, ou bien `FALSE` s'il ne reste plus de ligne.

Les appels consécutifs à **mysql_fetch_row()** déplace un pointeur vers la ligne suivante, si elle existe. Insérées dans une boucle *while*, ces appels permettent de décrire en séquence toutes les lignes du résultat.

Exemple

```
<?php
$result = mysql_query("SELECT id,email FROM t_personne WHERE id =
'42'");
if (!$result) {
   echo 'Impossible d\'exécuter la requête : ' . mysql_error();
   exit;
}
$row = mysql_fetch_row($result);
echo $row[0]; // 42
echo $row[1]; // la valeur du champ email
?>
```

`$ligne = mysql_fetch_array($resultat);`

Cette fonction est une version étendue de mysql_fetch_row().

Ici `$ligne` désigne un tableau associatif dont les indices peuvent être remplacés par les noms des champs, ce qui permet de s'adresser directement à la valeur d'une colonne du résultat.

Ainsi la valeur du champ `nom` du tableau `$ligne` s'obtient avec `$ligne[nom]`.

Exemple

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
die("Impossible de se connecter : " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM t_personne ");
while ($row = mysql_fetch_array($result) {
   echo $row["id"] ;
   echo $row["name"];
}
mysql_free_result($result);
?>
```

`$ligne= mysql_fetch_object($resultat);`

On obtient un objet construit à partir d'une ligne du resultat, ou FALSE si on a atteint la fin de la table

Cet objet `$ligne` possède pour attributs tous les champs de la table, initialisés aux valeurs actuelles de cette ligne. Ainsi la valeur du champ `nom` de l'objet `$ligne` s'obtient avec la syntaxe `$ligne->nom`.

Exemple

```
<?php
mysql_connect("hostname", "user", "password");
mysql_select_db("mydb");
$result = mysql_query("select * from t_personne ");
while ($row = mysql_fetch_object($result)) {
   echo $row->id;
   echo $row->nom;
}
mysql_free_result($result);
?>
```

IX- La programmation orientée objet

Un objet est un format complexe de variable regroupant des variables et des fonctions. La programmation orientée objet s'éffectue en deux étapes : la définition des classes puis leur utilisation. Une fois la classe définie, il sera possible de créer une infinité d'objets du format de la classe définie.
Les fonctions d'une classe sont appelées les méthodes et les variables les propriétés

1- Définition des classes

Définir une classe adopte la syntaxe suivante :

```
class nomObjet
{
  var $variable1;
  var $variable2;
  ...
  function maFonction1()
        {
          ...code
        }
  function maFonction2()
        {
        }
}
```

Il est possible d'attribuer une valeur par défaut. Le code dans la classe est alors var $variable1 = valeur ;
La définition de fonctions dans une classe est identique à celle de n'importe quelle fonction à la différence que lorsqu'elle fait référence à une variable de la classe, $variable doit être $this->variable. De même pour exécuter une fonction de la classe. ex :

```
class client
{
  var $aDitBonjour = false;
  function direBonjour()
        {
          $this->message("Bonjour");
        }
  function message($message)
        {
          echo $message;
          $this->aDitBonjour = true;
        }
}
```

2- Utilisation d'un objet

Attention : la classe est la définition d'un format de variable personnalisable. Le code n'est pas exécuté et il est impensable d'introduire le code suivant qui n'aurait aucun sens :

```
class client
{
  for ($i=0; $i<5; $i++)
  echo "$i\n";
}
```

Une fois la classe définie, il va falloir créer des variables objet du format de la classe définie. On crée un objet par le code suivant :

```
$objet = new client();
```

Il faut bien entendu avoir préalablement définit la classe client. La variable $objet contient donc un objet. Pour accéder à une variable pour lui faire subir des modifications, il suffit d'entrer le code suivant :

```
$objet->variable1 = "Hello world";
```

Il est possible de lui faire subir les mêmes opérations qu'à une variable normale. De même pour exécuter une fonction :

```
$objet->maFonction();
```

Autant les méthodes une fois définies ne peuvent pas être modifiées, autant il est possible d'ajouter ou de supprimer des variables dans l'objet :

```
$objet->variable = "valeur"; //définition de variable
unset( $objet->variable ); //suppressions
```

L'objet est unique, de sorte que s'il est enregistré dans une autre variable et qu'une modification lui est faite, elle sera visible pour les deux variables :

```
//Le code reprend l'ancien script
$objet = new client();
$objet2 = $objet;
$objet2->direBonjour();
echo $objet->aDitBonjour;
//affiche true
```

Pour dupliquer une variable, il faut donc entrer le code suivant :

```
$objet2 = clone $objet;
```

La nouvelle variable sera différente de l'ancienne mais aura les mêmes valeurs.
Il est également possible d'exécuter la méthode d'un objet sans avoir créé de variable auparavent:

```
class Message
{
  function direBonjour()
  {
            echo "salut";
  }
}
/* Exécute la méthode */
Message::direBonjour();
```

3- Les méthodes prédéfinies

Il existe quelques méthodes pré-définies qui s'exécutent à des périodes de la vie de l'objet :
Constructeur et destructeur
__construct()

Cette méthode s'exécute lors de la création de l'objet. On entre alors les attributs potentiels de la fonction lors de sa création. Cette méthode est appelée "le constructeur"

__destruct()

Cette méthode s'exécute au contraire au moment de la destruction de la variable. Elle est appelée "le destructeur".

Attention, ces deux méthodes sont précédées de deux underscores. Voici un exemple utilisant les méthodes :
Exemple :

```
//Définition de la classe
class Humain
{
  public $homme = false;
  public $femme = false;
  function __construct($type)
  {
            if ($type=="homme")
              $this->homme=true;
            if ($type=="femme")
              $this->femme=true;
  }

  function extremeOnction()
  {
             echo 'Amen';
  }
   function __destruct()
  {
            $this->extremeOnction();
  }
}
//C'est un garçon !
$homme = new Humain("homme");
if ($homme->homme)
```

```
{
        echo "C'est un homme";
}
elseif ($homme->femme)
{
        echo "C'est une femme";
}
//mort de l'homme
unset($homme);
```

4- Héritage

Il est possible de faire hériter une classe des propriétés et méthodes d'un autre classe. Ex :

```
class parent
{
        var $varParent;
        function functionParent()
        {
        }
}
class enfant extends parent
{
        var $varEnfant;
        function functionEnfant()
        {
        }
}
```

L'objet enfant bénéficiera des propriétés et des méthodes de la classe parent.
Il est possible d'accéder à une méthode ou un membre surchargé avec l'opérateur parent::
Exemple : Héritage d'une classe simple

```
<?php
// Rappel
class SimpleClass
{
   // déclaration d'un membre
   public $var = 'une valeur par défaut';

   // déclaration de la méthode
   public function displayVar() {
     echo $this->var;
   }
}

// extension de la classe
class ExtendClass extends SimpleClass
{
  // Redéfinition de la méthode parent
  function displayVar()
  {
    echo "Classe étendue\n";
    parent::displayVar();
  }
}

$extended = new ExtendClass();
$extended->displayVar();
?>
```

Le résultat obtenu sera :

```
Classe étendue
une valeur par défaut
```

5- Visibilité

La visibilité d'une propriété ou d'une méthode peut être définie en préfixant la déclaration avec un mot-clé : **public**, **protected** ou **private**. Les éléments déclarés publics (**public**) peuvent être utilisés par n'importe quelle partie du programme. L'accès aux éléments protégés (**protected**) est limité aux classes et parents hérités (et à la classe qui a défini l'élément). L'accès aux éléments privés (**private**) est uniquement réservé à la classe qui les a définis.

a- Visibilité des membres

Les classes membres doivent être définies comme publiques, protégées ou privées.

Exemple 1

```
class CompteEnBanque
{
        private $argent = 0;
        private function ajouterArgent($valeur)
        {
                $this->argent += $valeur;
        }
        function gagnerArgent($valeur)
        {
                $this->ajouterArgent($valeur);
        }
}


        $compte = new CompteEnBanque()
        //les actions suivantes sont impossibles:
        $compte->argent = 3000;
        $compte->ajouterArgent(3000);
        //l'action suivante est possible
        $compte->gagnerArgent(3000);
```

Exemple 2 : Déclaration des membres

```
<?php
/**
 * Définition de MyClass
 */
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->private;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Fonctionne
echo $obj->protected; // Erreur fatale
echo $obj->private; // Erreur fatale
$obj->printHello(); // Affiche Public, Protected et Private


/**
 * Définition de MyClass2
 */
class MyClass2 extends MyClass
{
```

```
    // On peut redéclarer les éléments publics ou protégés, mais pas ceux privés
    protected $protected = 'Protected2';

    function printHello()
    {
      echo $this->public;
      echo $this->protected;
      echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj->public; // Fonctionne
echo $obj2->private; // Indéfini
echo $obj2->protected; // Erreur fatale
$obj2->printHello(); // Affiche Public, Protected2 et non Private
?>
```

Note: La méthode de déclaration de variable en PHP 4 avec le mot-clé ***var*** est toujours supportée pour des raisons de compatibilité (en tant que synonyme du mot-clé public). Depuis PHP 5.1.3, son utilisation génère une erreur.

b- Visibilité des méthodes

Les méthodes des classes doivent être définies en tant que publiques, privées ou protégées. Les méthodes sans déclaration seront automatiquement définies comme étant publiques.

Exemple : Déclaration d'une méthode

```
<?php
/**
 * Définition de MyClass
 */
class MyClass
{
    // Les contructeurs doivent être publics
    public function __construct() { }

    // Déclaration d'une méthode publique
    public function MyPublic() { }

    // Déclaration d'une méthode protégée
    protected function MyProtected() { }

    // Déclaration d'une méthode privée
    private function MyPrivate() { }

    // Celle-ci sera publique
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Fonctionne
$myclass->MyProtected(); // Erreur fatale
$myclass->MyPrivate(); // Erreur fatale
$myclass->Foo(); // Public, Protected et Private fonctionnent
```

```
/**
 * Définition de MyClass2
 */
class MyClass2 extends MyClass
{
    // Celle-ci sera publique
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Erreur fatale
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Fonctionne
$myclass2->Foo2(); // Public et Protected fonctionnent, non pas Private

class Bar
{
    public function test() {
        $this->testPrivate();
        $this->testPublic();
    }

    public function testPublic() {
        echo "Bar::testPublic\n";
    }

    private function testPrivate() {
        echo "Bar::testPrivate\n";
    }
}

class Foo extends Bar
{
    public function testPublic() {
        echo "Foo::testPublic\n";
    }

    private function testPrivate() {
        echo "Foo::testPrivate\n";
    }
}

$myFoo = new foo();
$myFoo->test(); // Bar::testPrivate
                // Foo::testPublic
?>
```

5- L'opérateur de résolution de portée (::)

L'opérateur de résolution de portée (aussi appelé Paamayim Nekudotayim) ou, en termes plus simples, le symbole "double deux points" (::), fournit un moyen d'accéder aux membres statiques ou constants ainsi qu'aux éléments redéfinis par la classe.
Lorsque vous référencez ces éléments en dehors de la définition de la classe, utilisez le nom de la classe.

Paamayim Nekudotayim peut sembler un choix étrange pour un double deux points. Cependant, au moment de l'écriture du Zend Engine 0.5 (fourni avec PHP 3), c'est le nom choisi par le groupe Zend. En fait, cela signifie un double deux points... en hébreu !

Exemple : en dehors de la définition de la classe

```
<?php
class MyClass {
    const CONST_VALUE = 'Une valeur constante';
}
echo MyClass::CONST_VALUE;
?>
```

Deux mots-clé spéciaux, *self* et *parent*, sont utilisés pour accéder aux membres ou aux méthodes depuis la définition de la classe.

Exemple : depuis la définition de la classe

```
<?php
class OtherClass extends MyClass
{
    public static $my_static = 'variable statique';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}
OtherClass::doubleColon();
?>
```

Lorsqu'une classe étendue redéfinit une méthode de la classe parente, PHP n'appellera pas la méthode d'origine. Il appartient à la méthode dérivée d'appeler la méthode d'origine en cas de besoin. Cela est également valable pour les définitions des constructeurs et destructeurs, les surcharges et les méthodes magiques.

Exemple : Appel d'une méthode parente

```
<?php
class MyClass
{
    protected function myFunc() {
        echo "MyClass::myFunc()\n";
  }
}

class OtherClass extends MyClass
{
    // Dépassement de la définition parent
    public function myFunc() {

      // Mais appel de la fonction parent
      parent::myFunc();
      echo "OtherClass::myFunc()\n";
  }
}
$class = new OtherClass();
$class->myFunc();
?>
```

6- Statique

Le fait de déclarer des membres ou des méthodes comme statiques vous permet d'y accéder sans avoir besoin d'instancier la classe. On ne peut accéder à un membre déclaré comme statique avec l'objet instancié d'une classe (bien qu'une méthode statique le peut).

La déclaration static doit être faite après la déclaration de visibilité. Pour des raisons de compatibilité avec PHP 4, si aucune déclaration de visibilité n'est spécifiée, alors le membre ou la méthode sera automatiquement spécifié comme public.

Comme les méthodes statiques peuvent être appelées sans objet, la pseudo-variable *$this* n'est pas disponible dans la méthode déclarée en tant que statique.
En fait, les appels de méthodes statiques sont résolus au moment de la compilation. Lorsque l'on utilise un nom de classe explicite, la méthode est déjà identifiée complètement et aucune notion d'héritage n'est appliquée. Si l'appel est effectué par le mot-clé self, alors self est traduit en la classe courante, qui est la classe appartenant au code. Ici aussi, aucune notion d'héritage n'est appliquée.
On ne peut pas accéder à des propriétés statiques à travers l'objet en utilisant l'opérateur ->.
L'appel non-statique à des méthodes statiques génère une alerte.
Exemple : Exemple avec un membre statique

```
<?php
class Foo
{
    public static $my_static = 'foo';

    public function staticValue() {
        return self::$my_static;
    }
}

class Bar extends Foo
{

    public function fooStatic() {
        return parent::$my_static;
    }
}


print Foo::$my_static . "\n";

$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static . "\n";      // propriété my_static non définie

// $foo::my_static n'est pas possible

print Bar::$my_static . "\n";
$bar = new Bar();
print $bar->fooStatic() . "\n";
?>
```

Exemple : Exemple avec une méthode statique

```
<?php
class Foo
{
    public static function aStaticMethod() {
        // ...
    }
}

Foo::aStaticMethod();
?>
```

7- Abstraction de classes

PHP 5 introduit les classes et les méthodes abstraites. Il n'est pas autorisé de créer une instance d'une classe définie comme abstraite. Toutes les classes contenant au moins une méthode abstraite doivent également être abstraites. Pour définir une méthode abstraite, il faut simplement déclarer la signature de la méthode et ne fournir aucune implémentation.

Lors de l'héritage d'une classe abstraite, toutes les méthodes marquées comme abstraites dans la déclaration de la classe parent doivent être définies par l'enfant ; de plus, ces méthodes doivent être définies avec la même visibilité, ou une visibilité plus faible. Par exemple, si la méthode abstraite est définie comme protégée, l'implémentation de la fonction doit être définie comme protégée ou publique, mais non privée.

Exemple : Exemple de classe abstraite

```
<?php
abstract class AbstractClass
{
    // Force la classe étendue à définir cette méthode
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // méthode commune
    public function printOut() {
        print $this->getValue() . "\n";
   }
}

class ConcreteClass1 extends AbstractClass
{
     protected function getValue() {
       return "ConcreteClass1";
     }

     public function prefixValue($prefix) {
       return "{$prefix}ConcreteClass1";
    }
}

class ConcreteClass2 extends AbstractClass
{
     public function getValue() {
       return "ConcreteClass2";
     }

     public function prefixValue($prefix) {
       return "{$prefix}ConcreteClass2";
    }
}

$class1 = new ConcreteClass1;
$class1->printOut();
echo $class1->prefixValue('FOO_') ."\n";

$class2 = new ConcreteClass2;
$class2->printOut();
echo $class2->prefixValue('FOO_') ."\n";
?>
```

Le résultat obtenu sera :

```
ConcreteClass1
FOO_ConcreteClass1
ConcreteClass2
FOO_ConcreteClass2
```

8- Interfaces

Les interfaces objet permettent de créer du code qui spécifie quelles méthodes une classe doit implémenter.

Les interfaces sont définies en utilisant le mot-clé interface, de la même façon qu'une classe standard mais sans aucun contenu de méthode.
Toutes les méthodes déclarées dans une interface doivent être publiques.

implements

Pour implémenter une interface, l'opérateur implements est utilisé. Toutes les méthodes de l'interface doivent être implémentées dans une classe ; si ce n'est pas le cas, une erreur fatale sera émise. Les classes peuvent implémenter plus d'une interface en séparant chaque interface par une virgule.
Note: Une classe ne peut implémenter deux interfaces qui partagent des noms de fonctions, puisque cela causerait une ambiguïté.
Exemple :

```
<?php
// Declaration de l'interface 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

// Implémentation de l'interface
// Ceci va fonctionner
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }

        return $template;
    }
}
// Ceci ne fonctionnera pas
// Fatal error: Class BadTemplate contains 1 abstract methods
// and must therefore be declared abstract (iTemplate::getHtml)
class BadTemplate implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }
}
?>
```

9- Surcharge

Les appels de méthodes et l'accès aux membres peuvent être surchargés via les méthodes **__call()**, **__get()** et **__set()**. Ces méthodes ne seront déclenchées que si votre objet, hérité ou non, ne contient pas le membre ou la méthode auquel vous tentez d'accéder. Toutes les méthodes surchargées ne doivent pas être définies en tant que static. Toutes les méthodes surchargées doivent être définies en tant que public.

Depuis PHP 5.1.0, il est également possible de surcharger les fonctions isset() et unset() via, respectivement, les méthodes __isset et __unset. La méthode __isset est également appelée avec la fonction empty().

Surcharge de membre

```
void __set ( string $name , mixed $value )
mixed __get ( string $name )
bool __isset ( string $name )
void __unset ( string $name )
```

Les membres d'une classe peuvent être surchargés afin d'exécuter un code spécial contenu dans vos méthodes __set() et __get(). Le paramètre $name est le nom de la variable qui doit être définie ou recherchée. Le paramètre $valeur de la méthode __set() spécifie la nouvelle valeur à donner à la variable *$name*.

Exemple : Exemple de surcharge avec __get, __set, __isset et __unset

```
<?php
class Setter
{
  public $n;
  private $x = array("a" => 1, "b" => 2, "c" => 3);

  public function __get($nm)
  {
    echo "Récupération de [$nm]\n";

    if (isset($this->x[$nm])) {
      $r = $this->x[$nm];
      echo "Retour : $r\n";
      return $r;
    } else {
      echo "Rien!\n";
    }
  }

  public function __set($nm, $val)
  {
    echo "Définition de [$nm] à $val\n";

    if (isset($this->x[$nm])) {
      $this->x[$nm] = $val;
      echo "OK!\n";
    } else {
      echo "Pas OK!\n";
    }
   }

  public function __isset($nm)
  {
    echo "Vérifie si $nm est défini\n";

    return isset($this->x[$nm]);
  }

  public function __unset($nm)
  {
    echo "Libération de $nm\n";

    unset($this->x[$nm]);
  }
}
```

```
$foo = new Setter();
$foo->n = 1;
$foo->a = 100;
$foo->a++;
$foo->z++;
var_dump($foo);

var_dump(isset($foo->a)); //true
unset($foo->a);
var_dump(isset($foo->a)); //false

// ceci ne passera pas via la méthode __isset()
// parce que 'n' est une propriété publique
var_dump(isset($foo->n));

var_dump($foo);
?>
```

Le résultat obtenu sera :

```
Définition de [a] à 100
OK!
Récupération de [a]
Retour : 100
Définition de [a] à 101
OK!
Récupération de [z]
Rien!
Définition de [z] à 1
Pas OK!

Vérifie si a est défini
bool(true)
Libération de a
Vérifie si a est défini
bool(false)
bool(true)

object(Setter)#1 (2) {
  ["n"]=>
  int(1)
  ["x:private"]=>
  array(2) {
    ["b"]=>
    int(2)
    ["c"]=>
    int(3)
  }
}
```

Surcharge de méthode

mixed __call (string $name , array $arguments)

La méthode magique **__call()** permet la capture des appels à des méthodes inexistantes. De cette façon, **__call()** peut être utilisée pour implémenter un gestionnaire de méthodes défini par l'utilisateur qui dépend du nom de la méthode réellement appelée. Ceci est particulièrement pratique pour implémenter une classe proxy, par exemple. Les arguments qui sont passés à la fonction devront être définis dans un tableau via le paramètre *$arguments*. La valeur retournée par la méthode **__call()** sera retournée à la méthode appelante.

Exemple : Exemple de surcharge avec __call()

```
<?php
class Caller
```

```
{
  private $x = array(1, 2, 3);
  public function __call($m, $a)
  {
    print "Méthode $m appelée :\n";
    var_dump($a);
    return $this->x;
  }
}
$foo = new Caller();
$a = $foo->test(1, "2", 3.4, true);
var_dump($a);
?>
```

Le résultat obtenu sera :

```
Méthode test appelée :
array(4) {
  [0]=>
  int(1)
  [1]=>
  string(1) "2"
  [2]=>
  float(3.4)
  [3]=>
  bool(true)
}
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(2)
  [2]=>
  int(3)
}
```

10- Parcours d'objets

PHP 5 fournit une façon de définir les objets de manière à ce qu'on puisse parcourir une liste de membres avec une structure **foreach**. Par défaut, toutes les propriétés visibles seront utilisées pour le parcours.

Exemple : Parcours d'objet simple

```
<?php
class MyClass
{
  public $var1 = 'valeur 1';
  public $var2 = 'valeur 2';
  public $var3 = 'valeur 3';

  protected $protected = 'variable protégée';
  private   $private   = 'variable privée';

  function iterateVisible() {
     echo "MyClass::iterateVisible:\n";
     foreach($this as $key => $value) {
         print "$key => $value\n";
     }
  }
}
$class = new MyClass();

foreach($class as $key => $value) {
```

```
    print "$key => $value\n";
}
echo "\n";
$class->iterateVisible();
?>
```

Le résultat obtenu sera :

```
var1 => valeur 1
var2 => valeur 2
var3 => valeur 3
MyClass::iterateVisible:
var1 => valeur 1
var2 => valeur 2
var3 => valeur 3
protected => variable protégée
private => variable privée
```

11- Méthodes magiques

Les noms de fonction **__construct**, **__destruct** (voir les Constructeurs et Destructeurs), **__call**, **__get**, **__set**, **__isset**, **__unset** (voir la surcharge), **__sleep**, **__wakeup**, **__toString**, **__set_state**, **__clone** et **__autoload** sont magiques dans les classes PHP. Vous ne pouvez pas utiliser ces noms de fonction dans aucune de vos classes sauf si vous voulez modifier le comportement associé à ces fonctions magiques.

NB : PHP réserve tous les noms de fonctions commençant par __ pour les fonctions magiques. Il est recommandé de ne pas utiliser de noms de fonctions commençant par __.

a- __sleep et __wakeup

La fonction serialize() vérifie si votre classe a une fonction avec le nom magique __sleep. Si c'est le cas, cette fonction sera exécutée avant toute linéarisation. Elle peut nettoyer l'objet et elle est supposée retourner un tableau avec les noms de toutes les variables de l'objet qui doivent être linéarisées. Si la méthode ne retourne rien, alors NULL est linéarisé et une alerte de type E_NOTICE est émise.

Le but avoué de __sleep est de valider des données en attente ou d'effectuer les opérations de nettoyage. De plus, cette fonction est utile si vous avez de très gros objets qui n'ont pas besoin d'être sauvegardés en totalité.

Réciproquement, la fonction unserialize() vérifie la présence d'une fonction dont le nom est le nom magique __wakeup. Si elle est présente, cette fonction peut reconstruire toute ressource que l'objet possède.

Le but avoué de __wakeup est de rétablir toute connexion base de données qui aurait été perdue durant la linéarisation et d'effectuer des tâches de réinitialisation.

Exemple : Utilisation de **sleep() et wakeup()**

```
<?php
class Connection {
    protected $link;
    private $server, $username, $password, $db;

    public function __construct($server, $username, $password, $db)
    {
        $this->server = $server;
        $this->username = $username;
        $this->password = $password;
        $this->db = $db;
        $this->connect();
    }

    private function connect()
    {
        $this->link = mysql_connect($this->server, $this->username, $this-
>password);
        mysql_select_db($this->db, $this->link);
    }

    public function __sleep()
    {
```

```
        return array('server', 'username', 'password', 'db');
    }

    public function __wakeup()
    {
        $this->connect();
    }
}
?>
```

b- __toString

La méthode __toString détermine comment la classe doit réagir lorsqu'elle est convertie en chaîne de caractères.

Exemple :

```
<?php
// Déclaration d'une classe simple
class ClasseTest
{
    public $foo;

    public function __construct($foo) {
        $this->foo = $foo;
    }

    public function __toString() {
        return $this->foo;
    }
}

$class = new ClasseTest('Bonjour');
echo $class;
?>
```

Le résultat obtenu sera :

Bonjour

Il est important de noter qu'avant PHP 5.2.0, la méthode __toString n'était appelée que si elle était directement combinée avec echo() ou print().

12- Mot-clé "final"

PHP 5 introduit le mot-clé "final" qui empêche les classes filles de surcharger une méthode en en préfixant la définition par le mot-clé "final". Si la classe elle-même est définie comme finale, elle ne pourra pas être étendue.

Exemple 1:

```
<?php
class BaseClass {
   public function test() {
       echo "BaseClass::test() appelé\n";
   }

   final public function moreTesting() {
       echo "BaseClass::moreTesting() appelé\n";
   }
}
class ChildClass extends BaseClass {
   public function moreTesting() {
       echo "ChildClass::moreTesting() appelé\n";
   }
}
// Résultat : Fatal error: Cannot override final method BaseClass::moreTesting
()
?>
```

Exemple 2:

```
<?php
final class BaseClass {
   public function test() {
       echo "BaseClass::test() appelé\n";
   }

   // Ici, peut importe si vous spécifiez la fonction en finale ou pas
   final public function moreTesting() {
       echo "BaseClass::moreTesting() appelé\n";
   }
}
class ChildClass extends BaseClass {
}
// Résultat : Fatal error: Class ChildClass may not inherit from final class (
BaseClass)
?>
```

X- PHP et XML

1- Notions de XML

XML est un langage de structuration de contenu au moyen de balises. Pour délimiter des elements ayant un contenu explicite, XML utilise les structures générales suivants :

```
<balise> contenu de l'élément </balise>
```

Pour des éléments "vides", dans lesquels l'information est donnée dans un ou plusieurs attributs, il utilise les balises suivantes :

```
<balise attribut= « valeur » />
```

L'idée essentielle de XML est de structurer l'information en séparant le contenu de sa présentation. Le contenu dans un fichier XML et la présentation dans un CSS, XSLT ou PHP…

Syntaxe:

```
< ? xml version=  « 1.0 » encoding = « ISO-8859-1 » standalone = « yes »>
      <element>
        <souselement> contenu </ souselement>
      </element>
```

Exemple: biblio.xml

```
<? xml version=« 1.0 » encoding=«ISO-8859-1» standalone=«yes»>
       <biblio>
            <livre>
                  <titre> XML : langage et applications </titre>
                  <auteur> Mohamed </auteur>
                  <dat> 2000 <dat>
            </livre>
            <livre>
                  <titre> PHP 5 : cours  et exercices </titre>
                  <auteur> Brahim </auteur>
                  <dat> 2006 <dat>
            </livre>
            <livre>
                  <titre> PHP et MySQL </titre>
                  <auteur> Farida </auteur>
                  <dat> 2001 <dat>
            </livre>
```

```
</biblio>
```

2- Acceder au contenu d'un fichier XML

Pour accéder au contenu du fichier xml via php, la fonction **simplexml_load_file()** ; permet de charger le contenu dans un objet.

Exemple : affiche.php

```
< ?php
        $x= simplexml_load_file(« biblio.xml ») ;
        echo $x->livre[0]->titre ;
        echo $x->livre[0]->auteur ;
        echo $x->livre[0]->dat ;
?>
```

La fonction **get_object_vars($x)** retourne un tableau de toutes les propriétés de l'objet $x. en appliquant la fonction count() à ce tableau, on ontient le nombre de propriétés de l'objet.

Exemple affichecomplet.php

```
<?php
        $x=simplexml_load_file("biblio.xml");
        $t=get_object_vars($x);
        $n=count($t[livre]);
        echo $n."<br>";
        for ($i=0; $i<$n; $i++){
        echo $x->livre[$i]->auteur."-------";
        echo $x->livre[$i]->titre."------";
        echo $x->livre[$i]->dat."<br>";}
?>
```

3- lecture des attributs

Exemple : biblio2.xml

```
<? xml version=« 1.0 » encoding=«ISO-8859-1» standalone=«yes»>
    <biblio>
        <livre editeur=«Eyrolles» prix=«320.00» >
            <titre> XML : langage et applications </titre>
            <auteur> Mohamed </auteur>
            <dat> 2000 <dat>
        </livre>
        <livre editeur=«OEM» prix=«220.00» >
            <titre> PHP 5 : cours  et exercices </titre>
            <auteur> Brahim </auteur>
            <dat> 2006 <dat>
        </livre>
        <livre editeur=«Eyrolles» prix=«192.00» >
            <titre> PHP et MySQL </titre>
            <auteur> Farida </auteur>
            <dat> 2001 <dat>
        </livre>
    </biblio>
```

Fichier : Afficheattributs.php

```
<?php
        $x=simplexml_load_file("biblio2.xml");
        forach( $x->livre as $val)
        {
            echo «$val->titre de $val->auteur Paru en $val->dat» ;
            foreach ($val-attributes() as $at=>$valatt)
                {
```

```
                                    echo « $at : $ valatt » ;
                                }
                            echo « <hr/> ;
                        }
            ?>
```

4- Ajout d'éléments à un document

Le moteur Simple XML permet aussi d'ajouter des éléments au fichier XML.
Supposons le document xml 'population.xml' suivant:

```
<?xml version="1.0"?>
<population>
        <individu>
            <prenom>Damien</prenom>
            <nom>MATHIEU</nom>
            <email>personne@phportail.net</email>
        </individu>
        <individu>
            <prenom>Agnes</prenom>
            <nom>SCIROCCO</nom>
            <email>nobody@phportail.net</email>
        </individu>
</population>
```

Supposons maintenant que nous voulions ajouter une troisième personne : Abderrahim BENBOUNA, dont l'e-mail sera abenbouna@gmail.com.
Le code php d'ajout d'élément est :

```
<?php
$population = simplexml_load_file('population.xml');
//On ajoute un élément individu
$individu = $population->addChild('individu');

//On ajoute les valeurs
$individu->addChild('prenom','Abderrahim');
$individu->addChild('nom','BENBOUNA');
$individu->addChild('email', 'abenbouna@gmail.com');

//On génère la chose en xml
$xml = $population->asXML();

//Et on place ce contenu dans notre fichier
file_put_contents('population.xml', $xml);
?>
```

4- Ajout des attributs à un élément

Supposons le même document xml que celui utilisé dans le premier article sur le sujet :
On désire maintenant ajouter un élément mais avec id comma attribut à l'élément « individu »
Le code php sera le suivant :

```
<?php
$population = simplexml_load_file('population.xml');
```

```
//On ajoute un élément individu
$individu = $population->addChild('individu');
//On ajoute de l'attribut Id à l'élément individu

$individu->addAttribute('Id', '4');
//On ajoute les valeurs
$individu->addChild('prenom','Hamza');
$individu->addChild('nom','Barodi');
$individu->addChild('email', 'hbarodi@gmail.com');

//On génère la chose en xml
echo $xml = $population->asXML();

//Et on place ce contenu dans notre fichier
file_put_contents('population.xml', $xml);
?>
```

XI -LES IMAGES DYNAMIQUES

Sur le serveur local installé par Wamp, l'extension GD n'est pas installée par défaut. Vous devez décommenter, dans le fichier php.ini qui se trouve dans \wamp\Apache2\bin, la ligne suivante en supprimant le point-virgule placé au début de la ligne : extension=php_gd2.dll

1- Traçage de lignes

bool imageline (resource image , int x1 , int y1 , int x2 , int y2 , int color)

imageline dessine une ligne depuis le point (x1 , y1) jusqu'au point (x2 , y2) (le coin supérieur gauche est l'origine (0,0)) dans l'image image et avec la couleur color .

Exemple :

```
<?php
      header("content-type:image/png");
      $idimg=imagecreate(800,400);
      $fond=imagecolorallocate($idimg,255,255,51);
      $noir=imagecolorallocate($idimg,0,0,0);
      imagesetthickness($idimg,2); /*épaisseur du trait*/
      for($x=0;$x<800;$x+=10)
      {
         imageline($idimg,400,399,$x,0,$noir); /*tracé lignes*/
      }
      imagepng($idimg,"rayons.png"); /*enregistré l' image*/
      imagepng($idimg); /*envoi au navigateur*/
      imagedestroy($idimg); /*libérer la mémoire*/
?>
```

2- Création des histogrammes

bool imagerectangle (resource image , int x1 , int y1 , int x2 , int y2 , int col)

imagerectangle dessine un rectangle dans la couleur color , dans l'image image , en commençant au point supérieur gauche (x1 , y1), et en finissant au

point inférieur droit (x2 , y2). Le coin supérieur gauche est l'origine (0,0).

bool imagefill (resource image , int x , int y , int color)

imagefill effectue un remplissage avec la couleur color , dans l'image image , à partir du point de coordonnées (x , y) (le coin supérieur gauche est l'origine (0,0)).

array imagefttext (resource image , float size , float angle , int x , int y , int col , string font_file , string text , array extrainfo)

imagettftext retourne un tableau de 8 éléments représentant quatre points marquants les limites du texte. L'ordre des points est : inférieur gauche, inférieur droit, supérieur droit, supérieur gauche. Les points sont relatifs au texte par rapport à l'angle, donc, "supérieur gauche" signifie dans le coin en haut à gauche lorsque vous regardez le texte horizontallement.

Exemple :

Fichier histo.php

```
<?php
function histo($x,$y,$donn,$texte,$titre)
{
   $image=imagecreate($x,$y);
   $o=imagecolorallocate($image,195,155,125);
   $bc=imagecolorallocate($image,255,255,255);
   $bl=imagecolorallocate($image,50,0,255);
   $n=imagecolorallocate($image,0,0,0);
   $j=imagecolorallocate($image,255,255,0);
   $nb=count($donn);
   $mx=0;
   for ($i=0;$i<$nb;$i++)
   {
         $mx=max($mx,$donn[$i]);
   }
   $coef=($y*0.8)/$mx;
   $xo=10;
   $yo=$y-50;
   $larg=($x-20)/$nb;
   for ($i=0;$i<$nb;$i++)
   {
         $tabX[$i]=$xo+$larg*$i;
         $tabY[$i]=$yo-$coef*$donn[$i];
   }
   for($i=0;$i<$nb;$i++)
   {

   imagerectangle($image,$tabX[$i],$tabY[$i],$tabX[$i]+$larg,$yo,$n);
         imagefill($image,$tabX[$i]+5,$yo-5,$j);
         imagettftext($image,15,0,$tabX[$i]+20,$tabY[$i]-
5,$n,"vivaldii.ttf",$donn[$i]);
         imagettftext($image,15,0,$tabX[$i]+20,$y-
55,$bl,"elephnti.ttf",$texte[$i]);
   }
   imagettftext($image,20,0,200,$y-23,$bc,"elephnti.ttf",$titre);
   imagepng($image,"histo.png");
   imagedestroy($image);
}
?>
```

Fichier creehistogramme.php

Ce fichier fait appel à la fonction histo() du fichier histo.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1" />
<title>Histogramme dynamique</title>
</head>

<body>
<h2>Résultats des ventes de la semaine</h2>
<?php
     include("histo.php");
     $donn=array(8500,2500,4050,1730,1070,2590,2150);
     $texte=array("Lun","Mar","Mer","Jeu","Ven","Sam","Dim");
     $titre="Ventes hebdomadaires";
     histo(700,400,$donn,$texte,$titre);
?>
<img src="histo.png" alt="Ventes hebdomadaires">
</body>
</html>
```