

# Programmation Web

---

*Partie 3 : PHP*

# Généralités

---

- ◆ PHP est un langage de script permettant de réaliser des pages web dynamiques dont le contenu peut être complètement ou partiellement généré au moment de l'appel de la page, grâce à des informations récupérés dans un formulaire ou extraites d'une base de données.
- ◆ A la différence d'un langage comme JavaScript, où le code est exécuté côté client, le code PHP est exécuté côté serveur. Le résultat de cette exécution est intégré dans la page HTML qui est envoyée au navigateur. Ce dernier n'a aucune connaissance de l'existence du traitement qui s'est déroulé sur le serveur.
- ◆ Environnement pour tester les scripts PHP : Il existe des packages faciles à installer comprenant Apache, MySQL, PHP : [EasyPHP](#), [Xampp](#) ou [Wampserver](#).
- ◆ PHP ressemble beaucoup à C, C++ et Java.

# Généralités

---

- ◆ Un script PHP est un simple fichier texte contenant des instructions écrites à l'aide de caractères ASCII (non accentués) incluses dans un code HTML à l'aide de balises spéciales et stocké sur le serveur.
- ◆ Ce fichier doit avoir l'extension «.php» pour pouvoir être interprété par le serveur. Ainsi, lorsqu'un client (navigateur) désire accéder à une page réalisé en PHP :
  - le serveur reconnaît l'extension d'un fichier PHP et le transmet à l'interpréteur PHP ;
  - dès que l'interpréteur rencontre une balise indiquant que les lignes suivantes sont du code PHP, il exécute les instructions puis envoie les sorties éventuelles au serveur ;
  - à la fin du script, le serveur transmet le résultat au client.

# Syntaxe de base

---

- ◆ Deux conditions sont à remplir pour qu'un script soit interprété par le serveur :
  - le fichier contenant le code doit avoir l'extension *.php* et non *.html*
  - le code PHP contenu dans le code HTML doit être délimité par des balises spéciales :

*<?php ... ?>*

- ◆ Séparateurs d'instructions : ;

- ◆ Commentaires :

*//* Toute une ligne

*#* Toute une ligne

*/\** Plusieurs

lignes *\*/*

# Premier script PHP

---

```
<!DOCTYPE html>
<HTML>
  <HEAD>
    <META charset="UTF-8">
    <TITLE> Test – PHP </TITLE>
  </HEAD>
  <BODY>
    <H1>
      <?php
        echo "Bonjour le monde !";
      ?>
    </H1>
  </BODY>
</HTML>
```



Remarque : PHP inclut une balise ouvrante echo courte `<?=>` qui est un raccourci au code `<?php echo`. Exemple :

`<?='texte' ?>` est équivalent à `<?php echo 'texte' ?>`

# Inclusion de fichiers externes

---

Il est possible d'écrire du code PHP ou HTML dans des fichiers séparés puis les incorporer dans du code HTML ou d'autres scripts PHP en fonction des besoins :

- *include("nomFichier.php")*
  - \* Inclus le fichier *nomFichier.php*. Si ce fichier est déjà inclus, alors il y a un message d'erreur.
  - \* Si le fichier n'existe pas, alors il y a un 'warning' affiché et le script continue.
- *include\_once("nomFichier.php")*
  - \* Comme le *include*, à la différence que si le fichier a déjà été ajouté, il ne l'ajoutera pas de nouveau.
- *require("nomFichier.php")*
  - \* Inclus le fichier *nomFichier.php*. Si ce fichier est déjà inclus, alors il y a un message d'erreur.
  - \* Si le fichier n'existe pas, alors une erreur fatale est déclenchée et met fin au script.
- *require\_once("nomFichier.php")*
  - \* Comme le *require*, à la différence que si le fichier a déjà été ajouté, il ne l'ajoutera pas de nouveau.

# Variables et types

---

- ♦ L'identificateur de toute variable en PHP doit commencer par le symbole « \$ » (dollars) qui sera suivi du nom de la variable.  
Ex : \$pi  
Un nom de variable valide doit commencer par une lettre ou un souligné (\_), suivi de lettres, chiffres ou soulignés.
- ♦ Une variable peut référencer des valeurs différentes au cours de l'exécution.
- ♦ Pas de variable typée en PHP : une variable peut référencer un nombre, puis une chaîne, ...
- ♦ Attention : Les noms des variables sont sensibles à la casse (Ex : \$pi et \$Pi sont deux variables différentes). Cette différence entre majuscule et minuscule n'est pas applicable aux identificateurs de fonctions (Ex : ECHO et echo sont équivalentes).

# Variables et types

---

- ♦ Pas de déclaration explicite de variable en PHP. L'affectation détermine le type de la variable.

Ex :        `$i = 1;`  
             `$pi = 3.14;`  
             `$x = $i + $pi;`  
             `$lien = "www.google.fr";`

- ♦ Types de données :
  - Les entiers : int (ou integer)
  - Les flottants : float (ou double)
  - Les chaînes de caractères : string
  - Les booléens : bool (ou boolean)
  - Les tableaux (array) et les objets



# Variables et types

---

## ♦ Conversions de type :

- **Conversions implicites** : PHP convertit le type en fonction de l'opération effectuée.

**\$x + \$y** : PHP convertira \$x et \$y en numériques. (La même chose pour -, \*, / et %)

Ex :     \$z = 5 + "6";                     // z est 11  
          \$z = 3 + "11.45oujda";    // z est 14.45  
          \$z = 3 + "oujda11.45";    // z est 3

**"Valeur 1=\$x, Valeur 2=\$y"** : PHP convertit \$x et \$y en chaînes et les inclut dans la chaîne principale.

Ex :     \$z = 5.34;  
          echo "z = \$z";    // affiche : z = 5.34

- **Conversions explicites** : on indique le nom du type souhaité entre parenthèse devant l'expression à convertir.

Ex :     \$pi = 3.14;  
          \$pi = (int) \$pi;            // pi est int : 3  
          \$pi = (double) \$pi;        // pi est double : 3.0  
          \$pi = (bool) \$pi;          // pi est boolean : true

# Les booléens

---

- ♦ Les variables booléennes prennent pour valeurs TRUE (vrai) et FALSE (faux). Une valeur entière (ou décimal) nulle est automatiquement considérée comme FALSE. Tout comme un tableau vide, un objet vide ou une chaîne de caractères vide "". Ou encore les chaînes "0" et '0'.

Exemple :

```
if(0) echo 1;           // faux
if("") echo 2;          // faux
if("0") echo 3;         // faux
if("00") echo 4;
if('0') echo 5;         // faux
if('00') echo 6;
if(" ") echo 7;
```

Cet exemple affiche 467. Donc l'espace ou la chaîne "00" ne sont pas considérés castés en FALSE.

# Les tableaux

---

- ♦ Une variable tableau est de type *array*. Un tableau accepte des éléments de tout type.

Exemple :

```
$fruits = array(); // ou $fruits = [];  
$fruits[0] = "pomme";  
$fruits[1] = "banane";  
$fruits[] = "orange"; // équivaut a : $fruits[2] = "orange";  
ou : $fruits = array("pomme", "banane", "orange");  
ou : $fruits = ["pomme", "banane", "orange"];
```

- ♦ Quelques fonctions :

- *print\_r(\$tab)* : permet de visualiser un tableau.  
Ex : `print_r($fruits);` // affiche : Array ([0] => pomme [1] => banane [2] => orange)
- *count(\$tab), sizeof(\$tab)* : retournent le nombre d'éléments du tableau.
- *in\_array(\$val, \$tab)* : teste si la valeur *\$val* existe dans le tableau *\$tab*.
- *sort(\$tab) | rsort(\$tab)* : trie en ordre croissant | décroissant les éléments du tableau *\$tab*.  
Ex : `$fruits_trie = sort($fruits);`
- *array\_push(\$tab, \$val1[, \$val2,...])* : ajoute un ou plusieurs éléments spécifiés en argument à la fin du tableau *\$tab*.
- *array\_pop(\$tab)* : retire le dernier élément du tableau *\$tab*.

# Les tableaux

---

## ♦ Tableaux associatifs :

- Parfois, afin de mieux se retrouver dans nos variables tableaux, il est bon de remplacer l'index lors de la définition de la variable par un nom significatif (clé). Il suffit d'écrire le nom de la variable suivi d'une clé entre crochets et guillemets.
- Pour un tableau donné, toutes les clefs doivent être différentes.
- La fonction *array* peut également être utilisée pour initialiser le tableau.

### Exemple :

```
$coordonnees = array(); // ou $coordonnees = [];  
$coordonnees["Nom"] = "Namir";  
$coordonnees["Prenom"] = "Malika";  
$coordonnees["Ville"] = "Oujda";  
// ou $coordonnees = array("Nom" => "Namir", "Prenom" => "Malika", ...);  
// ou $coordonnees = ["Nom" => "Namir", "Prenom" => "Malika", ...];  
  
echo $coordonnees["Nom"];
```

# Les constantes

---

- ♦ L'utilisateur peut définir des constantes dont la valeur est fixée une fois pour toute.
- ♦ Les constantes ne portent pas le symbole \$ (dollars) en début d'identificateur et ne sont pas modifiables.
- ♦ Définition d'une constante : *define("NOM\_CONST", valeur, boolean casse)*  
*NOM\_CONST* (sans \$) de valeur *val*. Si *casse==true*, le nom de la constante est insensible à la casse.
- ♦ La fonction *defined("NOM\_CONST")* permet de savoir si la constante *NOM\_CONST* est définie ou non.

Exemple :

```
define("PI", 3.14, true); // la constante PI est insensible à la casse
define("nl", "<br>"); // la constante nl est sensible à la casse
if(defined("PI")) {
    $x = 5 + pi;
    echo '$x = '.$x.nl;
}
```

- ♦ Depuis la version 5.3, le mot clé *const* permet aussi de définir une constante :  
*const NOM\_CONST = valeur;*

# Les opérateurs

---

## ◆ Opérateurs d'affectation

- Affectation simple :
- Affectation multiple :
- Affectation + opération :
- Affectation conditionnelle :
- Pré/Post incrémentation/décrémentation :

## Exemples

`$a = 2;`

`$a = $b = 2;`

`$a += 2;`

`$max = ($a > $b) ? $a : $b;`

`++$a;`                      `--$a;`

`$a++;`                      `$a--;`

## ◆ Affectation par référence : `$var2 = &$var1;`

La variable `$var2` devient un alias de la variable `$var1`. Les modifications opérées sur `$var1` sont répercutées sur `$var2` et inversement.

## ◆ Opérateurs arithmétiques : `+` `-` `*` `/` `%`(modulo)

# Les opérateurs

---

- ◆ Concaténation de chaînes de caractères : .

Ex :     `$ch1 = $ch2 . $ch3;`  
          `$ch1 .= "<BR>";`

- ◆ Caractères spéciaux dans les chaînes :

- Antislash :     `//`
- Dollar :       `/$`
- Guillemets :   `/"` (dans les chaînes " )
- Cotes :        `/'` (dans les chaînes ' )

- ◆ Les chaînes de caractères (string) sont écrites entre guillemets simples ' ', guillemets doubles " " ou avec la syntaxe **Heredoc** ou avec la syntaxe **Nowdoc**. Exemple :

`$exempleHeredoc = <<<END`

Ma chaîne  
sur plusieurs lignes.

`END;`

`$exempleNowdoc = <<<'FIN'`

Ma chaîne  
sur plusieurs lignes.

`FIN;`

Remarque : Syntaxe Heredoc : Après l'opérateur <<<, un identifiant est fourni, suivi d'une nouvelle ligne. La chaîne en elle-même vient ensuite, suivie du même identifiant pour fermer la notation. L'identifiant de fin *doit* commencer à la première colonne de la ligne. Un Nowdoc est spécifié de manière similaire à un Heredoc, mais l'identifiant est entouré de guillemets simples, comme <<<'END'.

# Les opérateurs

---

## ♦ Opérateurs logiques :

- ET : *and* ou *&&*
- OU : *or* ou *||*
- NON : *!*

## ♦ Opérateurs de comparaison :

- Egalité /Différence : *==* *===* *!=* *!==*
- Inférieur/Supérieur : *<* *>*
- Inférieur ou égal/Supérieur ou égal : *<=* *>=*



# Affichage

---

## ♦ Fonctions prédéfinies :

- `echo(chaine) | echo chaine1[, chaine2,...]` Ex : *echo 'Coucou !';*
- `print(chaine)` Ex : *print("Coucou !");*
- Ces deux procédures peuvent afficher des chaînes " ou ' indifféremment.

## ♦ Formatage : Utilisation des balises HTML

Ex : `echo '<H1> TITRE </H1>';`

## ♦ Affichage du contenu de variables :

- Utilisation de l'opérateur de concaténation

Ex : `echo 'PI = '.$pi.'<BR>';`

- Directement dans une chaîne " ou avec la syntaxe Heredoc (ne fonctionne pas dans une chaîne ' ou avec la syntaxe Nowdoc )

Ex : `echo "PI = $pi <BR>";`

## ♦ Affichage d'un résultat de fonction : Concaténation

Ex : `echo "Carré de PI = ".carre($pi);`

# Structures de contrôle

---

- *Test* : ***if (condition) { instruction(s) } [else { instruction(s) } ]***

Ex :     <?php  
          \$i = 0;  
          if(\$i==0) echo 'La variable i vaut 0';  
          else echo 'La variable i ne vaut pas 0';  
          ?>

- *Sélection* : ***switch(var) { les cas }***

Ex :     <?php  
          \$utilisation = 'PHP';  
          switch(\$utilisation) {  
            case 'HTML' : echo 'Vous utilisez HTML'; break;  
            case 'JavaScript' : echo 'Vous utilisez JavaScript'; break;  
            case 'PHP' : echo 'Vous utilisez PHP'; break;  
            default : echo 'Vous n'utilisez ni HTML, ni JavaScript, ni PHP';  
          }  
          ?>

# Structures de contrôle

---

- Boucle « tant que » : ***while(condition) { instruction(s) }***

Ex : `$i = 1;`  
`while($i <= 10) { echo $i++; }`

- Boucle « répéter jusqu'à » : ***do { instruction(s) } while(condition);***

Ex : `$i = 1;`  
`do { echo $i++; } while($i <= 10);`

- Boucle « pour » : ***for(expr1; expr2; expr3) { instruction(s) }***

Ex : `for($i = 1; $i <= 10; $i++) { echo $i++; }`

- Boucle *foreach* : ***foreach(tableau as valeur) { instruction(s) }***

Elle permet d'exécuter une série d'instructions pour chacun des éléments d'un tableau.

Ex : `$fruits = array( "pomme", "banane", "orange" );`  
`foreach($fruits as $element) { print($element." "); }`

- Les instructions ***break*** et ***continue*** :

- *break* déclenche la sortie forcée d'une boucle ou d'un switch.
- *continue* dirige l'exécution à la prochaine évaluation du test de continuation en sautant les éventuelles instructions complétant le corps d'une boucle.

# Fonctions

---

## 1- Déclarer une fonction :

Syntaxe :      `function nom_fn([$p1, $p2, ..., $pn]) {  
                    // Code de la fonction  
                    return val_retour; // optionnel  
                    }`

Note : si return est omis, la valeur *null* est retournée.

Exemple :      `function my_max($n1, $n2) { return ($n1>$n2) ? $n1 : $n2; }  
                    echo my_max(3, 17);`

## 2- Fonction anonyme assignée à une variable :

Syntaxe :      `$variable = function ([$p1, $p2, ..., $pn]) { ... };`

Exemple :      `$texte = function() { echo 'Fonction anonyme bien exécutée'; };  
                    $carre = function($x) { return $x* $x; };  
                    $texte();  
                    echo '<br>'.$carre(3);`

# Fonctions

---

- **Passage de paramètres par référence : &**

Remarque : par défaut, les paramètres sont passés par valeur (sauf les objets).

Exemple :

```
function test(&$a) { $a++; }  
$c = 5;  
test($c);  
echo $c; // affiche : 6
```

- **Paramètres par défaut :** il est possible de définir des valeurs par défaut pour les arguments. (La valeur par défaut d'un argument doit obligatoirement être une constante, et ne peut être ni une variable, ni un membre de classe, ni un appel de fonction.)

Exemple :

```
function cafe($type = "expresso") {  
    return "Je fais un" . $type . "<BR>";  
}  
echo cafe(); echo cafe("capucino");
```

# Fonctions

---

## ■ Déclarations de type :

Depuis la version 7.4, les déclarations de types peuvent être ajoutées aux arguments des fonctions et aux valeurs de retour. Elles assurent que les valeurs sont du type spécifié au temps de l'appel.

Exemple :

```
function affiche(string $s) { echo "<b>$s</b><br>"; }  
function somme($a, $b) : float { return $a + $b; }  
$carre = function(int $x) : int { return $x*$x; };  
$produit = function(float $a, float $b) : float { return $a*$b; } ;  
  
affiche("PHP");  
affiche(somme(3.5, 1.2));  
echo somme(2,"8.8")." --- ".$produit(0.5,'4.6')." --- ".$carre(9.3);
```

```
PHP  
4.7  
10.8 --- 2.3 --- 81
```

# Les fonctions prédéfinies les plus utiles (fonctions pour les variables)

---

- ♦ ***isset(\$var)*** : permet de tester si une variable est définie. Cette fonction retourne FALSE si la variable n'est pas initialisée ou vaut NULL et la valeur TRUE si elle a une quelconque autre valeur.
- ♦ ***empty(\$var)*** : détermine si une variable est considérée comme vide. Elle retourne TRUE si \$var n'est pas initialisée, vaut "" , 0, NULL ou "0" et FALSE si elle a une quelconque autre valeur.
- ♦ ***unset(\$var)*** : permet de supprimer la variable, et de désallouer la mémoire utilisée.

Ex :     \$a = "test";  
          echo isset(\$a);   // --> 1 (vrai)  
          unset(\$a);  
          echo isset(\$a);   // --> 0 (faux)

- ♦ ***gettype(\$var)*** : permet de connaître le type de la variable. Elle renvoie "string" ou "integer" ou "double" ou "array" ou "object".

Ex :     \$a = 12;  
          echo gettype(\$a);   // --> "integer"  
          \$a = \$a/2.1;  
          echo gettype(\$a);   // --> "double"

# Les fonctions prédéfinies les plus utiles (fonctions mathématiques)

---

*abs(\$n)* : renvoie la valeur absolue de  $n$  qui peut être un entier ou un réel.

*sqrt(\$f)* : renvoie la racine carrée de  $f$ .

*ceil(\$f)* : renvoie le plus petit entier supérieur ou égal à  $f$ .

*floor(\$f)* : renvoie le plus grand entier inférieur ou égal à  $f$ .

*round(\$f)* : renvoie l'entier le plus proche de  $f$ .

*max(\$v1, ..., \$vn)* : renvoie la plus grande des valeurs passées en paramètre.

*min(\$v1, ..., \$vn)* : renvoie la plus petite des valeurs passées en paramètre.

*rand(\$i, \$j)* : renvoie une valeur aléatoire entre  $i$  (inclus) et  $j$  (inclus).

*is\_numeric(\$x)* : teste si  $x$  est un nombre.

Il y a aussi : *cos*, *sin*, *tan*, *exp*, *log*,... ainsi que des constantes tels que *M\_PI*, *M\_E*



# Les fonctions prédéfinies les plus utiles (fonctions pour les chaînes de caractères)

---

***explode(\$sep, \$ch)*** : divise la chaîne *\$ch* en valeurs séparées par *\$sep* et renvoie le tableau de ces valeurs.

***implode(\$sep, \$tableau)*** : renvoie une chaîne de caractères avec les valeurs de *\$tableau* séparées par *\$sep*.

***strlen(\$ch)*** : renvoie la longueur de la chaîne *\$ch*.

***strtoupper(\$ch)*** : retourne la chaîne *\$ch* en majuscule.

***strtolower(\$ch)*** : retourne la chaîne *\$ch* en minuscule.

***substr(\$ch, \$deb, \$long)*** : renvoie la sous-chaîne de *\$ch* de longueur *\$long* à partir de *\$deb*.

***ucwords(\$ch)*** : retourne la chaîne *\$ch* avec toutes les initiales des mots qui la composent en majuscules.

***ucfirst(\$ch)*** : retourne la chaîne *\$ch* avec uniquement la première lettre en majuscule.

# Les fonctions prédéfinies les plus utiles (fonctions pour les chaînes de caractères)

---

***substr\_count(\$ch, \$sch)*** : retourne le nombre d'occurrences de la sous chaîne *\$sch* dans la chaîne *\$ch*.

***str\_replace(\$sch1, \$sch2, \$ch)*** : retourne la chaîne *\$ch* ou toutes les occurrences de *\$sch1* sont remplacées par *\$sch2*.

***strpos(\$ch, \$sch)*** : retourne la position du premier caractère de la première occurrence d'une sous chaîne *\$sch* dans la chaîne *\$ch* ou FALSE si *\$sch* ne figure pas dans *\$ch*.

***strcmp(\$ch1, \$ch2)***, ***strcasecmp(\$ch1, \$ch2)*** : retourne un nombre négatif si *\$ch1* < *\$ch2* et un nombre positif dans le cas contraire et 0 en cas d'égalité. *strcasecmp* n'est pas sensible à la casse.

***trim(\$ch)*** : renvoie la chaîne *\$ch* sans les espaces de début et de fin de chaîne.

***addslashes(\$ch)*** : ajoute dans la chaîne *\$ch* le caractère d'échappement \ devant les caractères spéciaux tels que ' , " , \ et NULL.

***stripslashes(\$ch)*** : retire les \ devant les apostrophes, les guillemets et les antislashes de *\$ch*.

# Les fonctions prédéfinies les plus utiles (la fonction date)

---

*date("format")* : renvoie la date et l'heure courante selon le format indiqué.

Il existe plusieurs options de formatage, par exemple :

*d* : jour du mois (valeurs de "01" à "31")

*m* : mois (valeurs de "01" à "12")

*Y* : année sur 4 chiffres

*y* : année sur deux chiffres

*H* : heure sur 24 heures (valeurs de "00" à "23")

*h* : heure sur 12 heures

*i* : minutes (valeurs de "00" à "59")

*s* : secondes (valeurs de "00" à "59")

*W* : numéro de la semaine dans l'année

*w* : jour de la semaine (valeurs de 0 à 6, 0 pour le dimanche)

Ex : `<?php print( date("d/m/Y H:i:s") ); ?>` ==> affichage formaté : 13/01/2022 22:46:39

*time()* : renvoie le nombre de secondes depuis le 1<sup>er</sup> janvier 1970.

# Variables statiques

- Par défaut, les variables locales d'une fonction sont réinitialisées à chaque appel de la fonction. L'instruction *static* permet de définir des variables locales statiques qui ont pour propriété de conserver leur valeur d'un appel à l'autre de la fonction, *pendant la durée du script*.

Exemple :

```
<?php
function compteur() {
    $a = 0;
    echo $a;
    $a++;
}
?>
```

- ➡ Cette fonction est inutile car à chaque appel, elle initialise *\$a* à 0 et affiche "0".

```
<?php
function compteur() {
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

Maintenant, à chaque fois que la fonction *compteur()* est appelée, elle affichera une valeur de *\$a* incrémentée de 1.

# Portée des variables

---

- Par défaut, les variables globales ne sont pas connues à l'intérieur du corps d'une fonction. On peut cependant y accéder à l'aide du mot clé *global*.

Exemple 1 : sans le mot clé global

```
<?php
    $a = 1; /* portée globale */
    function test() {echo $a; /* portée locale */ }
    test();
?>
```

Le script n'affichera rien à l'écran car l'instruction *echo* utilise la variable locale *\$a*, et celle-ci n'a pas été assignée préalablement dans la fonction.

Exemple 2 : avec le mot clé global

```
<?php
    $a = 1; $b = 2;
    function somme() {
        global $a, $b;
        $b = $a + $b;
    }
    somme(); echo $b;
?>
```

Le script ci-dessus va afficher la valeur 3.

# Portée des variables

---

- Une deuxième méthode pour accéder aux variables globales est d'utiliser le tableau associatif prédéfini **\$GLOBALS**. Le précédent exemple peut être réécrit de la manière suivante :

```
<?php
    $a = 1; $b = 2;
    function somme() {
        $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
    }
    somme(); echo $b;
?>
```

- Les fonctions anonymes peuvent hériter des variables du contexte de leur parent. Ces variables doivent alors être passées dans la construction de langage **use**. Ces variables ne doivent pas inclure des variables ayant le même nom qu'un paramètre. Une déclaration de type de retour pour la fonction doit être placée après la clause use.

Exemple : <?php

```
    $a = 1;
    $test = function() use ($a) { echo $a; };
    $test();           // affiche : 1
?>
```

# Les constantes et les fonctions

---

- La portée des constantes est le script dans le lequel elles sont définies.
- À la différence des variables, cette portée s'étend aux fonctions sans qu'elle soit déclarée globale.
- Réciproquement, une constante définie dans une fonction peut être utilisée dans un script, après appel de la fonction.

Exemple : <?php

```
define(CONST_SCRIPT, 'valConstScript');  
function f() {  
    define(CONST_FONCT, 'valConstFonct');  
    echo 'Dans la fonction, CONST_SCRIPT = ', CONST_SCRIPT, '<br>';  
}  
f();  
echo 'Dans le script, CONST_FONCT = ', CONST_FONCT, '<br>';  
?>
```

Résultat : Dans la fonction, CONST\_SCRIPT = valConstScript  
Dans le script, CONST\_FONCT = valConstFonct

# Arrêt prématuré du script

---

- ◆ Les instructions *exit* et *die* permettent d'interrompre l'exécution du script (*die* est un alias d'*exit*).

- ◆ Syntaxe :

<code>exit [(string message)]</code>		<code>exit [(int statut)]</code>
<code>die [(string message)]</code>		<code>die [(int statut)]</code>

  - *message* : message à afficher avant d'interrompre le script
  - *statut* : statut de retour (entier entre 1 et 254)

Exemple 1 :

```
<?php
    echo 'Bonjour';
    if($nom == NULL) { exit(1); }
    echo $nom;
?>                                ==> Résultat : Bonjour
```

Exemple 2 :

```
<?php
    echo 'Bonjour';
    if($nom == NULL) { exit('<i>Utilisateur inconnu.</i>'); }
    echo $nom;
?>                                ==> Résultat : Bonjour Utilisateur inconnu.
```

- ◆ Ces instructions stoppent tout le script, pas seulement le bloc en cours.



# Les variables d'environnements

---

- ◆ PHP propose toute une série de variables qui sont déjà présentes dans le langage sans que nous n'ayons besoin à les déclarer. Ces variables s'écrivent toujours en majuscules et nous fournissent divers renseignements :

`$_SERVER['HTTP_USER_AGENT']` : permet d'avoir des informations sur le type de navigateur utilisé par le client, ainsi que son système d'exploitation

`$_SERVER['REMOTE_ADDR']` : adresse IP du client qui demande la page courante

`$_SERVER['REMOTE_PORT']` : port utilisé par la machine cliente pour communiquer avec le serveur web

`$_SERVER['REQUEST_URI']` : chemin du script

`$_SERVER['REQUEST_METHOD']` : méthode d'appel du script

`$_SERVER['SERVER_ADDR']` : adresse IP du serveur

`$_SERVER['QUERY_STRING']` : liste des paramètres passés au script

`$_SERVER['PHP_SELF']` : nom du fichier du script en cours d'exécution

`$_SERVER['SERVER_NAME']` : nom donné au serveur en local

Remarque : L'exécution de la fonction `phpinfo()` permet de lister les variables d'environnement supportés par le serveur. `<?php phpinfo(); ?>`

# Les variables d'environnement

---

Exemple :

```
<!DOCTYPE html>
<HTML>
  <HEAD>
    <META charset="UTF-8">
    <TITLE> PHP </TITLE>
  </HEAD>
  <BODY>
    <?php
      echo "Navigateur du client : ".$_SERVER['HTTP_USER_AGENT']."<BR>";
      echo "Adresse IP du client : ".$_SERVER['REMOTE_ADDR']."<BR>";
      echo "Nom du serveur : ".$_SERVER['SERVER_NAME'];
    ?>
  </BODY>
</HTML>
```

# La gestion des formulaires

---

- ♦ PHP peut intervenir à deux endroits par rapport au formulaire :
  - Pour la construction du formulaire, si ce dernier doit contenir des informations dynamiques.
  - Pour le traitement du formulaire (c.à.d des données saisies par l'utilisateur dans le formulaire).
- ♦ Trois méthodes sont utilisables pour faire interagir un formulaire et un script PHP :
  - Placer le formulaire dans un document HTML pur (*.htm* ou *.html*), et indiquer le nom du script PHP qui doit traiter le formulaire dans l'attribut *action* de la balise `<form>`.
  - Placer le formulaire dans un script PHP (par exemple, pour construire une partie dynamiquement) et faire traiter le formulaire par un autre script PHP (mentionné dans l'attribut *action* de la balise `<form>`).
  - Placer le formulaire dans un script PHP (par exemple, pour construire une partie dynamiquement) et le faire traiter par le même script PHP (mentionné dans l'attribut *action* de la balise `<form>` ou appelé par défaut si cet attribut n'est pas présent).

# La gestion des formulaires

---

- ♦ Les paramètres d'un formulaire envoyés depuis le navigateur vers le serveur (par POST ou GET) sont accessibles directement par leur nom dans le script qui les reçoit.
- ♦ PHP crée automatiquement une variable pour chaque champ du formulaire. Ainsi, ces variables peuvent être utilisées dans le fichier référencé par l'action du formulaire.
- ♦ Si le formulaire est soumis par *POST*, les variables sont disponibles en PHP dans le tableau associatif *\$\_POST*.
- ♦ Si le formulaire est soumis par *GET*, les variables sont disponibles dans le tableau associatif *\$\_GET*.
- ♦ Dans les deux cas, les variables sont aussi disponibles dans le tableau associatif *\$\_REQUEST*.

# La gestion des formulaires

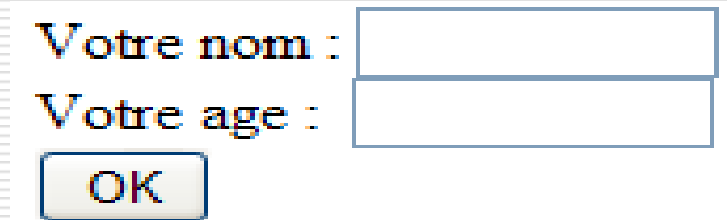
---

## Exemple :

Écrire un formulaire qui demande le nom et l'âge de l'utilisateur.

Le bouton OK (de type *submit*) de ce formulaire provoquera l'affichage d'une page qui saluera l'utilisateur avec cette phrase :

« Bonjour machin, vous avez xx ans »



Votre nom :

Votre age :

# La gestion des formulaires

---

Exemple (suite) :

- [formulaire.html](#) :

```
<FORM ACTION="age.php" METHOD="POST">  
  Votre nom : <INPUT TYPE="text" NAME="nom" SIZE="10"> <BR>  
  Votre age : <INPUT TYPE="text" NAME="age" SIZE="10"> <BR>  
  <INPUT TYPE="submit" VALUE="OK">  
</FORM>
```

- [age.php](#) :

```
<?php  
  $nom = $_POST["nom"];  
  $age = $_POST["age"];  
  print("Bonjour $nom, vous avez $age ans");  
?>
```

# La gestion des formulaires

---

- ◆ Les balises *input* transmettent l'association entre la valeur de l'attribut *name* et leur attribut *value*.

Ex :

```
<input type="checkbox" name="choix" value="oui">
```

Dans le script, `$_REQUEST["choix"]` vaut "oui"

- ◆ Les balises *select* transmettent l'association entre la valeur de leur attribut *name* et la valeur de l'attribut *value* de la balise option sélectionnée.

Ex :

```
<select name="monMenu">
```

```
    <option value="n1"> Choix 1 </option>
```

```
</select>
```

Dans le script, `$_REQUEST["monMenu"]` vaut "n1"

- ◆ Les balises *textarea* transmettent l'association entre la valeur de leur attribut *name* et le contenu de la zone de texte.

# La gestion des formulaires

---

- ♦ Les balises *input* de type *image* envoient les coordonnées du clic en pixel :

Ex :

```
<input type="image" name="monImage" src=... alt=... >
```

Dans le script, on aura `$_REQUEST["monImage_x"]` valant la coordonnée horizontale.

- ♦ On peut associer les valeurs du formulaire à un tableau plutôt qu'à une variable simple :

Ex :

```
<select name="menu[]" id="menu" multiple> ...
```

Les valeurs des champs sélectionnés apparaîtront dans :

```
$_REQUEST["menu"][0], $_REQUEST["menu"][1], ...
```



# La gestion des formulaires

---

- ◆ Pour pouvoir télécharger des fichiers, il est nécessaire que l'attribut *enctype* de la balise `<form>` soit *"multipart/form-data"*.

Avec une balise : `<input type="file" name="fichier">` la variable `$_FILES["fichier"]` est un tableau contenant :

`$_FILES["fichier"]["name"]` : le nom original du fichier, tel que sur la machine du client

`$_FILES["fichier"]["type"]` : son type MIME

`$_FILES["fichier"]["size"]` : sa taille en octets

`$_FILES["fichier"]["tmp_name"]` : le nom temporaire sur le serveur (avec son chemin)

`$_FILES["fichier"]["error"]` : le code d'erreur (0 : pas d'erreur)

- ◆ `move_uploaded_file($fichier, $dest)` : déplace un fichier jusqu'à *\$dest* seulement s'il a été téléchargé par http POST.

# La gestion des formulaires

Exemple :

-----HTML-----

```
<form action="testFichier.php" method="post" enctype="multipart/form-data">
  <label for="photo"> Insérez une photo : </label>
  <input type="file" name="photo" id="photo">
  <input type="submit" value="envoi">
</form>
```

Insérez une photo :

-----PHP-----

```
<?php
$fichier = $_FILES['photo'];
if($fichier['error'] == 0) {
  $src = $fichier['tmp_name']; $dest = "./photo_".$fichier['name'];
  move_uploaded_file($src, $dest); // ou copy($src, $dest);
  echo "Votre photo :<br><img src=\"".$dest.\" alt=\"Votre photo\" >";
}
?>
```



# La gestion des fichiers

---

## ◆ Principe :

- PHP prend en charge l'accès au système de fichiers du système d'exploitation du serveur.
  - Les opérations sur les fichiers concernent la création, l'ouverture, la suppression, la copie, la lecture et l'écriture de fichiers.
  - La communication entre le script PHP et le fichier est repérée par une variable, indiquant l'état du fichier et qui est passée en paramètre aux fonctions spécialisées pour le manipuler.
- ◆ Il existe une multitude de fonctions dédiées à l'utilisation des fichiers. Les plus simples sont :
- la fonction `file_put_contents("nomDuFichier", $chaine)` : écrit le contenu `$chaine` dans le fichier. Si ce dernier n'existe pas, il sera créé. Sinon, le fichier existant sera écrasé.
  - la fonction `file_get_contents("nomDuFichier")` : elle prend comme paramètre le nom du fichier et elle affecte son contenu à une variable de type chaîne de caractères.

Ex : Soit le fichier *test.txt* contenant le texte suivant : *Bonjour, c'est un test ...*

Pour afficher le contenu de ce fichier sur une page web :

```
$str = file_get_contents("test.txt");  
echo $str; // affiche : Bonjour, c'est un test ...
```

# La gestion des fichiers

---

## ♦ Ouverture de fichiers :

- La fonction *fopen("nom du fichier", "mode")* permet d'ouvrir un fichier, que ce soit pour le lire, le créer ou y écrire. Elle retourne un identificateur de fichier ou FALSE si échec.

*mode* : indique le type d'opération qu'il sera possible d'effectuer sur le fichier après ouverture. Il s'agit d'une lettre indiquant l'opération possible :

- \* **r** : indique une ouverture en lecture seulement.
- \* **w** : indique une ouverture en écriture seulement (la fonction crée le fichier s'il n'existe pas).
- \* **a** : indique une ouverture en écriture seulement avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas).

Lorsque le mode est suivie du caractère **+** celui-ci peut être lu et écrit.

# La gestion des fichiers

---

## ◆ Quelques fonctions :

- *fgetc(\$fp)* : lit un caractère (avec *\$fp* est l'identificateur retourné par la fonction *fopen*). Elle retourne FALSE (ou une valeur équivalente) à la fin du fichier.
- *fgets(\$fp)* : lit la ligne courante. Elle retourne FALSE (ou une valeur équivalente) à la fin du fichier.
- *fgets(\$fp, \$length)* : lit une ligne de *\$length* caractères au maximum. Si la longueur de la ligne est inférieur à *\$length*, lit jusqu'à la fin de la ligne. Elle retourne FALSE (ou une valeur équivalente) à la fin du fichier.
- *fputs(\$fp, \$str)*, *fwrite(\$fp, \$str)* : écrit la chaîne *\$str* dans le fichier identifié par *\$fp*. Elle retourne une valeur de type int qui correspond au nombre de bytes écrites ou FALSE en cas d'erreur.
- *fclose(\$fp)* : ferme le fichier identifié par *\$fp*.
- *feof(\$fp)* : teste la fin du fichier, retourne TRUE ou FALSE.
- *file\_exists(\$filename)* : vérifie si un fichier ou un dossier existe . Elle retourne TRUE ou FALSE. Le nom du fichier peut être une arborescence de répertoire.

# La gestion des fichiers

---

- *filesize(\$filename)* : retourne la taille du fichier *\$filename* en octet.
- *unlink(\$filename)* : détruit le fichier *\$filename*.
- *copy(\$source, \$dest)* : copie le fichier *\$source* vers *\$dest*.
- *readfile(\$filename)* : affiche le fichier *\$filename*.
- *rename(\$old, \$new)* : renomme le fichier *\$old* en *\$new*.
- *is\_file(\$filename)* : permet de savoir si *\$filename* correspond à un fichier existant.
- *file(\$filename)* : retourne le fichier *\$filename* dans un tableau. Chaque élément du tableau correspond à une ligne du fichier.

## Remarque :

La fonction *fopen()* permet d'ouvrir des fichiers dont le chemin est relatif ou absolu. Elle renvoie FALSE si l'ouverture échoue.

Ex : `$fp = fopen("../docs/rapport.txt", "r");`  
`$fp = fopen("http://www.php.net/", "r");`

# La gestion des fichiers

---

Exemple :

```
<?php
    $fich = "fichier.txt";
    if($fd = fopen($fich, "r")) {           // ouverture du fichier en lecture seule
        while( ! feof($fd) ) {             // teste la fin de fichier
            $str .= fgets($fd, 500);
            /* lecture jusqu'à fin de ligne où des 500 premiers caractères */
        }
        fclose($fd);                       // fermeture du fichier
        echo $str;                          // affichage
    } else {
        die("Ouverture du fichier <b>$fich</b> impossible.");
    }
?>
```

# PHP et les bases de données (MySQL)

---

- ◆ L'utilisation en général d'un SGBD (tel que MySQL) avec PHP s'effectue en 5 étapes :

1. Connexion au serveur de données
2. Sélection de la base de données
3. Envoi de requêtes
- [4. Exploitation des résultats de requêtes]
5. Fermeture de la connexion



# PHP et les bases de données (MySQL)

---

## ♦ *Connexion au serveur de données :*

La fonction *mysqli\_connect* permet de se connecter au serveur de données.

Syntaxe :

*mysqli\_connect("NomHôte|adresseIP", "NomUtilisateur", "MotDePasse", ["NomDeLaBase"])*

Cette fonction retourne un identifiant de connexion ou, en cas d'erreur, la valeur FALSE.

## ♦ *Sélection de la base de données :*

Pour faire cette sélection, vous pouvez utiliser la fonction ci-dessus (*mysqli\_connect*) ou la fonction *mysqli\_select\_db* et vous lui passez en paramètre, le nom de la base de données.

Syntaxe :

*mysqli\_select\_db("NuméroDeConnexion", "NomDeLaBase" )*

- *NuméroDeConnexion* : celui renvoyé par *mysqli\_connect*.
- Cette méthode retourne TRUE en cas de succès et FALSE en cas d'erreur.

Exemple :

```
$c = mysqli_connect("localhost", "root", "" );  
mysqli_select_db($c, "ma_base");
```

# PHP et les bases de données (MySQL)

---

## ♦ *Envoi d'une requête SQL :*

La fonction *mysqli\_query* permet d'envoyer une requête à une base de données sélectionnée.

Syntaxe :

*mysqli\_query("NuméroDeConnexion", "Requête")*

- *Requête* : une chaîne de caractères, syntaxe SQL
- *NuméroDeConnexion* : celui renvoyé par *mysqli\_connect*.

Exemple :

```
<?php
//...
$requete = "SELECT * FROM maTable";
$result = mysqli_query($c, $requete);
?>
```

# PHP et les bases de données (MySQL)

---

## ♦ *Exploitation des requêtes :*

Après l'exécution d'une requête de sélection, les données ne sont pas "affichées", elles sont simplement mises en mémoire, il faut les chercher, enregistrement après enregistrement, et les afficher.

PHP gère un pointeur de résultat, c'est celui qui est pointé qui sera retourné. Lorsque vous utilisez une fonction de lecture, le pointeur est déplacé sur l'enregistrement suivant et ainsi de suite jusqu'à ce qu'il n'y en ait plus.

Les fonctions qui retournent un enregistrement sont :

- *`mysqli_fetch_row($result)`* : retourne une ligne de résultat sous la forme d'un tableau simple. Les éléments du tableau étant les valeurs des attributs de la ligne. Retourne FALSE s'il n'y a plus aucune ligne.
- *`mysqli_fetch_assoc($result)`* : retourne un tableau associatif. Les clés étant les noms des attributs et leurs valeurs associées leurs valeurs respectives. Retourne FALSE s'il n'y a plus aucune ligne.
- *`mysqli_fetch_object($result)`* : retourne un objet. Les attributs de l'objet correspondent à ceux de la ligne de résultat. Et les valeurs des attributs de l'objet correspondent à ceux de la ligne de résultat. Retourne FALSE s'il n'y a plus aucune ligne.

# PHP et les bases de données (MySQL)

---

Exemple avec la fonction *mysqli\_fetch\_row* :

```
<?php
    //...
    $requete = "SELECT nom, prenom, age FROM clients";
    if($result = mysqli_query($c, $requete)) {
        while($ligne = mysqli_fetch_row($result)) {
            echo $ligne[0].' '.$ligne[1].' '.$ligne[2].'<br>';
        }
    } else {
        echo "Erreur de requête de la base de données.";
    }
?>
```

# PHP et les bases de données (MySQL)

---

Le même exemple avec la fonction *mysqli\_fetch\_assoc* :

```
<?php
    //...
    $requete = "SELECT nom, prenom, age FROM clients";
    if($result = mysqli_query($c, $requete)) {
        while($ligne = mysqli_fetch_assoc($result)) {
            echo $ligne['nom'].' '.$ligne['prenom'].' '.$ligne['age'].'<br>';
        }
    } else {
        echo "Erreur de requête de la base de données.";
    }
?>
```

# PHP et les bases de données (MySQL)

---

Le même exemple avec la fonction *mysqli\_fetch\_object* :

```
<?php
    //...
    $requete = "SELECT nom, prenom, age FROM clients";
    if($result = mysqli_query($c, $requete)) {
        while($ligne = mysqli_fetch_object($result)) {
            echo $ligne->nom.' '.$ligne->prenom.' '.$ligne->age.'<br>';
        }
    } else {
        echo "Erreur de requête de la base de données.";
    }
?>
```

# PHP et les bases de données (MySQL)

---

## ♦ *Fermeture de la connexion :*

Vous pouvez fermer la connexion au moyen de la fonction *mysqli\_close*.

Syntaxe : *mysqli\_close(NuméroDeConnexion);*

- *NuméroDeConnexion* : celui renvoyé par *mysqli\_connect*.

# PHP et les bases de données (MySQL)

---

## ♦ *Quelques fonctions supplémentaires utiles :*

- *mysqli\_num\_fields(\$result)* : retourne le nombre d'attributs du résultat.
- *mysqli\_num\_rows(\$result)* : retourne le nombre de lignes du résultat.  
Et ainsi permet de remplacer le *while* par un *for*.
- *mysqli\_free\_result(\$result)* : efface de la mémoire du serveur les lignes de résultat de la requête identifiées par *\$requete*.
- *mysqli\_affected\_rows(\$c)* : retourne le nombre de lignes affectées lors de la dernière requête SQL. (utilisée dans le cas des requêtes de manipulation de données comme INSERT, UPDATE, DELETE, ...).
- *mysqli\_error(\$c)* : retourne le texte associé avec l'erreur générée lors de la dernière requête.



# PHP et les bases de données (MySQL)

---

## Exemple complet :

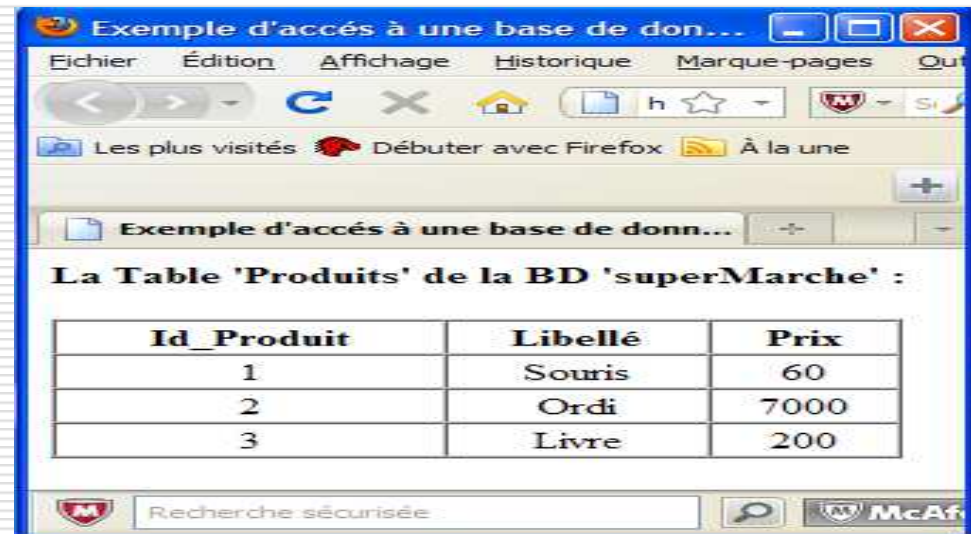
Soit la base de données *superMarche* contenant les tables suivantes (voir TP) :

*Acheteurs(Id\_Acheteur, Nom, Compte)*

*Produits(Id\_Produit, Libelle, Prix)*

*Achats(Id\_Achat, \*Id\_Acheteur, \*Id\_Produit)*

Question : afficher le contenu de la table *Produits* dans un tableau à l'aide d'un script PHP comme ci-après :



Exemple d'accès à une base de don...

La Table 'Produits' de la BD 'superMarche' :

Id_Produit	Libellé	Prix
1	Souris	60
2	Ordi	7000
3	Livre	200

Recherche sécurisée McAfee

# PHP et les bases de données (MySQL)

---

## Réponse :

```
... <body>
<h4> La Table 'Produits' de la BD 'superMarche' : </h4>
<table border="1" cellspacing="0" width="300">
  <tr> <th> Id_Produit </th> <th> Libellé </th> <th> Prix </th> </tr>
  <?php
    $connexion = mysqli_connect("localhost","root","");
    if( !$connexion) { echo "Désolé, connexion à localhost impossible"; exit; }
    if( !mysqli_select_db($connexion,'superMarche')) { echo "Désolé, accès à la base superMarche impossible"; exit; }
    $resultat = mysqli_query($connexion,"SELECT * FROM Produits");
    if($resultat) {
      while($ligne = mysqli_fetch_object($resultat)) {
        echo '<tr align="center"><td>'. $ligne->Id_Produit. ' </td><td>'.
          $ligne->Libelle.' </td><td>'. $ligne->Prix.' </td></tr>';
      }
    }
    else { echo "Erreur dans l'exécution de la requête. ". mysqli_error($connexion); }
    mysqli_close($connexion);
  ?>
</table> </body> </html>
```

# Les cookies

---

- ♦ Un cookie est un fichier texte créé par un script et stocké, par le navigateur, sur le disque dur du poste client. Il contient des informations spécifiques à l'utilisateur.
- ♦ Lorsque l'utilisateur visite une autre page du site web d'où provient le cookie, les informations sont automatiquement renvoyées au serveur web par le navigateur.
- ♦ L'information contenu dans le cookie n'est accessible que par celui qui a créé le cookie.
- ♦ La fonction *setcookie* permet de déposer un cookie sur le poste client :  
*setcookie('nom\_du\_cookie', 'contenu\_du\_cookie', date d'expiration);*
  - Seul le nom du cookie est obligatoire.
  - *date d'expiration* : le temps en secondes depuis 1/1/1970 → *time()+nbsecondes*.
  - Par défaut, sans date spécifiée, le cookie expire à la fin de la session (c.à.d lorsque l'utilisateur quitte le site).
  - Il faut appeler la fonction *setcookie* avant de générer le code HTML.
- ♦ L'accès aux cookies est réalisé par le tableau associatif super global *\$\_COOKIE* (on peut aussi utiliser le tableau associatif super global *\$\_REQUEST*).

# Les cookies

---

- ♦ Il est possible de passer plusieurs valeurs à un cookie en utilisant une notation de type tableau :

```
setcookie('nom_du_cookie[0]', 'valeur1', time()+...);  
setcookie('nom_du_cookie[1]', 'valeur2', time()+...);  
....
```

- Pour accéder à une valeur de ce cookie : `$_COOKIE['nom_du_cookie'][indice]`

- ♦ Pour supprimer un cookie, on spécifie que le paramètre nom à la fonction *setcookie* :

```
setcookie('nom_du_cookie');
```

- Une autre méthode consiste à envoyer un cookie dont la date d'expiration est passée :

```
setcookie('nom_du_cookie', '', time()-1);
```

- ♦ Plusieurs risques sont liés à l'utilisation des cookies :

- Un utilisateur peut désactiver sur son navigateur l'utilisation des cookies.
  - Un utilisateur peut avoir effacé les cookies, d'un jour à l'autre vous perdez les informations.
  - Un utilisateur peut lire le contenu de ses cookies.

# Les cookies

---

## Exemple 1:

- Envoi d'un cookie :

```
<?php
    $value = 'test';
    $duree = 60*60; // durée du cookie une heure
    setcookie("monCookie", $value, time()+$duree);
?>
```

- Afficher ce cookie :

```
<?php
    if(isset($_COOKIE['monCookie']))
        echo 'La valeur de monCookie est : ' . $_COOKIE['monCookie'] . '<br>';
    else echo 'Le cookie est inexistant.<br>';
?>
```

- Effacer ce cookie :

```
<?php
    setcookie("monCookie", 'test', -1);
?>
```

# Les cookies

---

## Exemple 2 :

Nous voulons qu'un script stocke dans un cookie appelé *DerniereVisite* la date où l'utilisateur a visité une page web. Si c'est sa première visite, on lui souhaite la bienvenue et on crée le cookie avec la date d'aujourd'hui. Si ce n'est pas sa première visite, on lui indique à quelle date il est venu la dernière fois, et on met à jour le cookie pour qu'il contienne maintenant la date d'aujourd'hui. Le cookie devra rester actif pendant trente jours.

```
if(isset($_COOKIE['DerniereVisite'])) {  
    $vLaDate = $_COOKIE['DerniereVisite'];  
    echo 'Votre dernière visite était le '.$vLaDate.'<br>';  
}  
else {  
    echo 'Bienvenue à cette première visite !<br>';  
}  
// On ajoute 30 jours (30jrs*24h*60min*60sec = 2592000 secondes!)  
$vUnMois = time() + 2592000;  
setcookie('DerniereVisite', date('d/m/Y'), $vUnMois);
```

# Les sessions

---

- ♦ PHP propose un système intégré pour gérer le passage d'information. Il est basé sur la notion de session :
  - Chaque visiteur accédant à un site se voit assigner un identifiant unique « identifiant de session ».
  - Cet identifiant est soit stocké dans un cookie (nommé par défaut *PHPSESSID*) qui s'efface lorsque le navigateur est fermé, soit propagé dans l'URL.
- ♦ Les données de session sont stockées dans un fichier sur le serveur.
- ♦ Les données de session sont donc moins sensibles que celles des cookies puisqu'on ne peut pas y avoir accès du côté client.

Finalement, une session c'est un peu comme un cookie mais coté serveur.
- ♦ Par défaut la durée de vie d'une session est limitée à l'ouverture du navigateur.

Il est possible de fixer une autre durée de vie pour le cookie de la session dans *php.ini* avec *session.cookie.lifetime* (la durée est exprimée en secondes. La valeur 0 signifie «jusqu'à ce que le navigateur soit fermé ». Par défaut 0).

# Les sessions

---

## ♦ Syntaxe :

### ▪ Démarrage d'une session :

Pour démarrer une session, il faut à chaque page, avant tout affichage, appeler la fonction :

`session_start();`

- Elle essaye de trouver un identificateur de session existant, sinon une nouvelle session est créée et un identificateur unique est assigné par PHP.
- Remarque : La session, à proprement parlé, est stockée côté serveur (seul l'identificateur de la session se trouve côté client). La fonction `session_start()` essaye donc de trouver un identificateur de session dans les cookies. Sinon, elle regardera si l'identificateur a été passé en paramètre par la méthode GET (directement dans l'URL). Par défaut, l'identificateur de session est passé par la variable `PHPSESSID` et a la forme suivante :  
`2c51f0fbc54e07c9c14c7fd82aa2598b`

### ▪ Création et utilisation d'une variable de session :

- Utilisation du tableau super global `$_SESSION` : `$_SESSION['variable']`

Pour savoir si la variable de session *variable* existe, il suffit de faire :

`if(isset($_SESSION['variable'])) ...`

- Suppression d'une variable de session : `unset($_SESSION['variable'])`



# Les sessions

---

Exemple :

-- page1.php --

```
<?php
    session_start();
    $_SESSION['testSession'] = 'test';
    $_SESSION['heure'] = date('H:i:s');
    echo 'Bienvenue à la page numéro 1';
    echo '<br> <a href="page2.php"> page 2 </a>';
?>
```

-- page2.php --

```
<?php
    session_start();
    echo 'Bienvenue à la page numéro 2 <br>';
    echo $_SESSION['testSession'].'<br>';
    echo $_SESSION['heure'];
?>
```

# Les sessions

---

## ♦ Quelques fonctions utiles :

- `session_id([id])` : retourne (ou éventuellement modifie) l'identifiant de la session.
- `session_name([nom])` : retourne (ou éventuellement modifie) le nom de la variable utilisée pour stocker l'identifiant de la session. Par défaut le nom est *PHPSESSID*.
- `session_regenerate_id()` : crée un nouvel identifiant et remplace l'existant dans une session ouverte, sans perdre une seule variable de session. Cette fonction peut être utilisée pour brasser plus fréquemment les identifiants de session et rendre certaines techniques de 'vol' d'identifiant plus délicates.

## ♦ Détruire une session :

La fonction `session_destroy()` ferme la session. Elle ne supprime pas les données de session dans le script en cours, et de même elle ne détruit pas le cookie de session.

- Pour vider le contenu de la session : `$_SESSION = array();`
- Pour détruire le cookie de session : `setcookie(session_name());`

# Authentication et sessions

---

- ♦ Un formulaire demande le login et le mot de passe
- ♦ Un script de traitement de ce formulaire, contrôle que le login et le mot de passe sont corrects (par exemple à l'aide d'une table MySQL) :
  - Si c'est le cas :
    - On crée une session avec `session_start()` et on y ajoute un paramètre  
Par exemple : `$_SESSION['validUser'] = ...;`
    - On redirige vers les pages de contenu
  - Sinon :
    - On redirige vers une page d'erreur ou on retourne au formulaire
- ♦ Sur les autres pages (pages auxquelles les utilisateurs authentifiés et seulement eux ont accès) :
  - on commence par un `session_start()`
  - on contrôle si l'utilisateur est identifié (`if(isset($_SESSION['validUser'])) {...}`)  
et sinon on redirige vers la page de formulaire
- ♦ Pour la déconnexion : `session_destroy()`

# Authentication et sessions

## Exemple :

Dans cet exemple, on utilise une session pour stocker les informations relatives à l'identification d'un utilisateur et les transmettre de page en page.

On suppose que le login valide est « **toto** » et le mot de passe valide est « **pwdtoto** ».



Si le login et le mot de passe sont valides



Si le login et le mot de passe ne sont pas valides



# Authentication et sessions

---

## login.htm :

Création du formulaire pour se connecter.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title> Page login </title>
  </head>
  <body>
    <form action="login.php" method="post">
      Login : <input type="text" name="login" size="35"> <br>
      Mot de passe : <input type="password" name="pwd" size="35"><br>
      <input type="submit" value="Connexion">
    </form>
  </body>
</html>
```

# Authentication et sessions

---

## login.php :

Traitement des données issues du formulaire de connexion (si l'utilisateur est valide, on le redirige vers la page membre, sinon, on le redirige vers la page de login).

```
<?php
    $login_valide = "toto";
    $pwd_valide = "pwdtoto";
    if(($_POST['login'] == $login_valide) && ($_POST['pwd'] == $pwd_valide)) {
        session_start();
        $_SESSION['monlogin'] = $_POST['login'];
        header('location: membre.php');
    }
    else {
        header('location: login.htm');
    }
?>
```

# Authentication et sessions

---

## membre.php :

On essaye de récupérer une session existante. Si elle n'existe pas, on redirige vers la page de login. Sinon, on affiche les données et un lien pour se déconnecter.

```
<?php
    session_start();
    if(!isset($_SESSION['monlogin'])) header('location: ./login.htm');
?>
<html>
    <head>
        <meta charset="UTF-8">
        <title> Page des membres </title>
    </head>
    <body>
        <?php
            echo "Bonjour <b>".$_SESSION['monlogin']."'</b> !<br>";
            echo '<a href="./deconnexion.php"> Déconnexion </a></p>';
        ?>
    </body></html>
```

# Authentication et sessions

---

## **deconnexion.php :**

On récupère la session et on la vide des données qu'elle contient. Enfin, on détruit la session et on redirige l'utilisateur vers la page de connexion.

```
<?php
    session_start();
    $_SESSION = array();
    setcookie(session_name(), ' ', time()-1);
    session_destroy();
    header('location: login.htm');
?>
```



# Les redirections

---

- ♦ Il est souvent très utile de rediriger le visiteur vers une autre page. Par exemple à la suite du traitement d'un formulaire, pour rediriger vers une autre page avec un message de confirmation.  
Il existe plusieurs méthodes pour rediriger le visiteur : avec du PHP, du JavaScript ou bien une simple balise `<meta>` du HTML.

- ♦ En PHP, la fonction *header()* permet de faire plusieurs choses et, en particulier, de rediriger vers une autre page.

Ex :            `<?php`  
                 `header("Location: http://www.site.com");`  
                 `?>`

Cette fonction doit par contre être utilisée avant d'envoyer des données html, sous peine de produire une erreur.

# Les redirections

---

## Exemple de redirection avec PHP, JS et HTML :

- Avec PHP :

```
<?php  
    header("Location: ../page.htm");  
?>
```

- Avec JavaScript :

```
<script type="text/javascript">  
    window.location.replace("../page.htm");  
</script>
```

- Avec la balise META du HTML :

```
<html>  
    <head>  
        <meta http-equiv="refresh" content="1; url= ../page.htm">  
    </head>  
    <body> ... </body>  
</html>
```

# Autres utilisations de la fonction header

---

- ◆ Interdiction de mise en cache :

Pour résoudre le problème des pages non mises à jour car toujours présentes dans le cache du navigateur, il faut envoyer des en-têtes complémentaires dans la page HTML à l'aide de la fonction *header* :

*header("Cache-Control: no-store, no-cache, must-revalidate");*

- ◆ Redirection (déjà vu plus haut) :

*header('Location: login.php');*

- ◆ Redirection temporisée :

*header('Refresh: 5; url=http://www.monsite.com/');*

- ◆ Spécification de la langue utilisée dans la page (fr : français):

*header('Content-language: fr');*