

Server side scripting

préparé par

Dr. Neila Ben Lakhel

(PhD@Tokyo Institute of Technology)

E-mail: Neila.BenLakhel@gmail.com

○ PARTIE 1: PHP

■ La programmation en PHP: les principes et les notions fondamentales.

Liens

PHP : Hypertext Preprocessor

<http://www.php.net>

<http://www.phpinfo.net>

<http://www.phpfrance.com>

<http://www.developpez.com/php/>

MySQL

<http://www.mysql.com/>

<http://dev.nexen.net/docs/mysql/>

La petite histoire du PHP

- Il a été créé en 1994 par Rasmus Lerdorf pour les besoins des pages web personnelles (livre d'or, compteurs, etc.). A l'époque, PHP signifiait **Personal Home Page**.
- En 1997, PHP devient un projet collectif et son interpréteur est réécrit par Zeev Suraski et Andi Gutmans pour donner la version 3 qui s'appelle désormais PHP : **Hypertext Preprocessor** .
- C'est un langage incrusté au HTML. Il dérive du C et du Perl dont il reprend la syntaxe. Il est extensible grâce à de nombreux modules et son code source est ouvert. Comme il supporte tous les standards du web et qu'il est gratuit, il s'est rapidement répandu sur la toile.

La petite histoire du PHP

- Jusqu'à la version **PHP3**, php été **interprété**, mais a partir de la version **PHP4**, il est devenu compilé à l'**exécution côté serveur**.
- Version la plus utilisée : **PHP 5.6 (DEC 2014)**.
- Dernière version : **PhP7(MAR 2018)**

Caractéristiques de PHP

- Très populaire et très utilisé
- Très portable (fonctionne sous Windows/Unix...)
- Syntaxe héritée du C, du shell et du Perl
- Extensible par de nombreuses bibliothèques
 - calcul mathématique,
 - création dynamique d'images,
 - connexions sécurisées,
 - accès à la plupart des SGBD
- Logiciel Open Source (donc plus facilement extensible) et disponible gratuitement

Les semblables de PHP

- JSP : Java-Server Pages
 - semblable à **PHP** mais la partie dynamique est écrite en JAVA
- ASP : Active Server Pages
 - version Microsoft de **PHP** et JSP
 - contenu dynamique écrit en Visual Basic Script,
 - langage de script propriétaire de Microsoft
- Le choix entre **PHP**, JSP et ASP est plus "politique" que technique !
 - **PHP** : open source
 - JSP : Java Sun
 - ASP : Microsoft

Avantages

- Le client n'a pas accès au code source car il est interprété avant envoi (pas le cas de JavaScript)
- le client ne reçoit que le résultat de l'exécution, la qualité dynamique des pages est masquée au client
- Le code n'est pas alourdi par des commandes destinées à générer du HTML

Inconvénients

- Trous de sécurité
 - On exécute quelque chose sur le serveur, il faut bien écrire ses scripts, prévoir tous les cas de saisie du client
- Rapidité d'exécution
 - langage interprété par le serveur avec plusieurs requêtes simultanées donc pas très rapide
 - tend à s'améliorer grâce à des optimisations au niveau de l'interpréteur
- Pas d'interactivité au niveau du client (JavaScript reste nécessaire pour cela)

La petite histoire du PHP

- PHP est similaire à JavaScript, seulement c'est un langage de programmation **côté serveur**
- Le code PHP est imbriqué dans des balises HTML
- Quand une demande de page arrive, le serveur reconnaît que c'est un contenu PHP grâce à une extension de fichier (.php, .phtml, .php3, .php4, .php5)
- le serveur exécute le code PHP, et le remplace par du code HTML, la page est ensuite téléchargée vers le client
- **L'utilisateur ne voit jamais le code PHP, seule le résultat de l'exécution des commandes php.**

Intégration d'un script dans une page

- Un fichier **PHP** (.php) peut contenir :
 - du code HTML
 - du code **PHP**
 - du code JavaScript
- Conçu pour fonctionner efficacement avec le serveur Apache lui aussi en open source

Intégration d'un script dans une page

- Le code source php est directement insérer dans le fichier html grâce à la balise : **<?php ... ?>**
- *Exemple:*

```
<html>                                Index.php  
  <body>  
    <?php echo "Bonjour"; ?>  
  </body>  
</html>
```

- Autres syntaxes d'intégration :
<? ... ?>
<script language="php"> ... </script>
<% ... %>
- Néanmoins, **<?php ... ?>** est la forme la plus correcte.

Modèle

Poste client



1

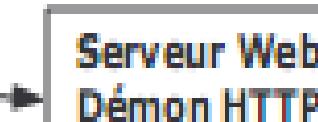
Requête HTTP
demandant index.php



4

Réponse HTTP

Site serveur



2

recherche de index.php
sur le disque

3

interprétation des
commandes PHP

5

requêtes SQL éventuelles



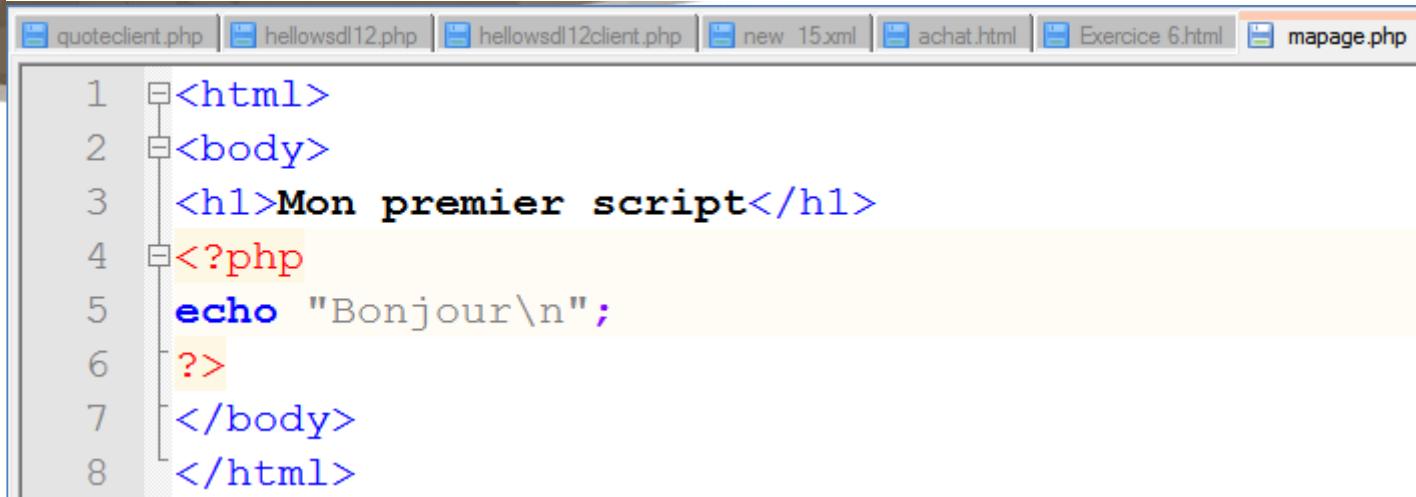
code HTML/
JavaScript



- L'interpréteur de code PHP est intégré au serveur HTTP
- Le serveur lis les instructions PHP intégrées à la page HTML, interprète ces instructions et les remplace par le résultat de leur exécution

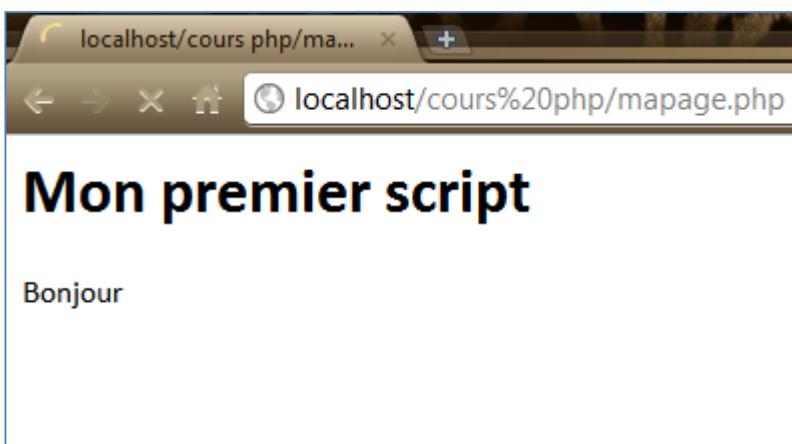
Exemple de script

Exemple de script **mapage.php**, code source (côté **serveur**) :

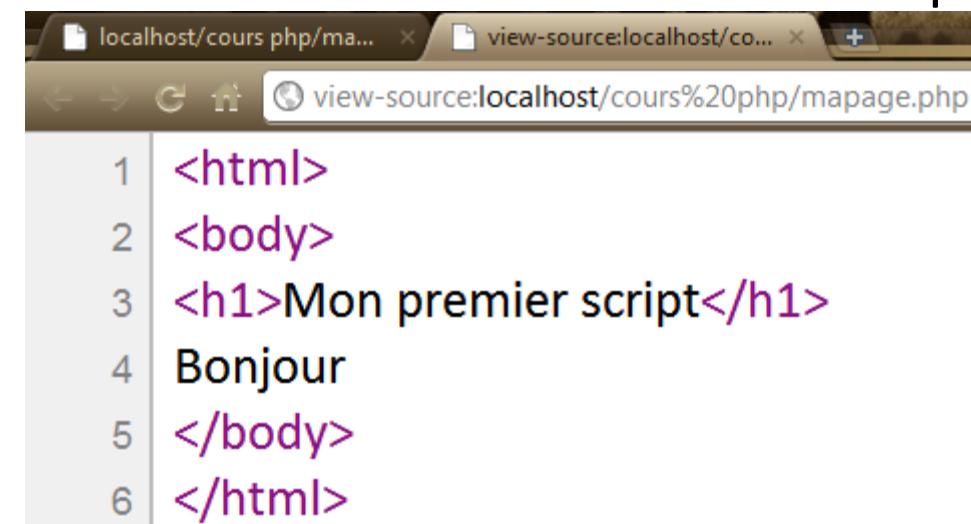


```
1 <html>
2 <body>
3 <h1>Mon premier script</h1>
4 <?php
5 echo "Bonjour\n";
6 ?>
7 </body>
8 </html>
```

Résultat affiché par le navigateur:



Code source (côté client) de la page **mapage.php**
résultant du script



```
1 <html>
2 <body>
3 <h1>Mon premier script</h1>
4 Bonjour
5 </body>
6 </html>
```

Fonctionnement de l'interpréteur PHP

- Toute ligne située à l'extérieur de <?php et ?> **n'est pas interprétée** : elle est recopiée à l'identique sur la sortie standard
- toute ligne située à l'intérieur de ces balises est interprétée par le moteur comme une instruction **PHP**
- les instructions **PHP n'apparaissent pas** dans le flux de sortie généré
- lorsqu'une erreur se produit, un message explicatif est intégré dans le flux de sortie et l'analyse du code est interrompue (sauf si warning)

Exemple: syntaxe de base de PHP

```
1 <html>
2 <!--hello.php -->
3 <head>
4 <title>
5 Hello World</title>
6 </head>
7 <body>
8 <p>This is going to be ignored by the PHP interpreter.</p>
9
10 <?php
11 echo '<p>While this is going to be parsed.</p>';
12 ?>
13 <p>This will also be ignored by PHP.</p>
14
15 <?php
16 print('<p>Hello and welcome to <i>my</i> page!</p>');
17 ?>
18 <?php
19 //This is a comment
20 /*
21 This is
22 a comment block
23 */
24 ?>
25 </body>
26 </html>
```

Commentaires

- Un script PHP se commente comme en C :
- *Exemple :*

```
<?php  
// commentaire de fin de Ligne  
/* commentaire  
sur plusieurs  
lignes */  
# commentaire de fin de Ligne comme en Shell  
?>
```

- Tout ce qui se trouve dans un commentaire est ignoré. Il est conseillé de commenter largement ses scripts.

Variables, types (I/2)

- Le typage des variables est **implicite** en php. Il n'est donc pas nécessaire de déclarer leurs types au préalable ni même de les initialiser avant leurs utilisation.
- Les variables peuvent être de type entier (**int**), réel (**double**), chaîne de caractères (**string**), tableau (**array**), objet (**object**), booléen (**boolean**).
- Les identificateurs de variable sont précédés du symbole « **\$** » (dollars).
- PHP est **sensible à la casse** pour les noms de variable.
- Syntaxe : **`$variable_name = Value;`**
- Exemple : **`$toto`**.

Variables, types (2/2)

Nomenclature des variables :

\$MaVariable;

// Format correct

\$_MaVariable;

// Format correct

\$1MaVariable;

// Format incorrect car commence par un nombre

\$MaVariable1;

// Format incorrect car se termine par un nombre

Variables, types (2/2)

- Il est possible de convertir une variable en un type primitif grâce au **cast**:

```
$str = "12";           // $str vaut La chaîne "12"  
$a_number = 4;  
$nbr = (int)$str;      // $nbr vaut Le nombre entier 12
```

- PHP fait la conversion de type automatiquement entre certains types:

```
string → int  conversion automatique  print("1" + 1) // affichera 2  
int → float  conversion automatique
```

- Une variable chaîne de caractères **n'est pas limitée en nombre de caractères**.
Elle est toujours délimitée par des simples quotes ou des doubles quotes.

```
$nom = "Victor";  
$prenom = 'Hugo';
```

Chaînes de caractères (III)

- Opérateur de concaténation de chaînes : **.** (le point) :

```
$foo = "Hello";
```

```
$bar = "World";
```

```
echo $foo.$bar; //affiche HelloWorld sans espace
```

- *Attention le **+** n'est pas un opérateur de concaténation !!*

```
echo 5 + "2 tableaux" //afficher a 7
```

```
echo 5 . "2 tableaux" //affichera "52 tableaux"
```

L'affichage

- **Echo et print()** pour l'affichage:

- **Echo N'EST PAS** une fonction : c'est une **construction du langage** php.

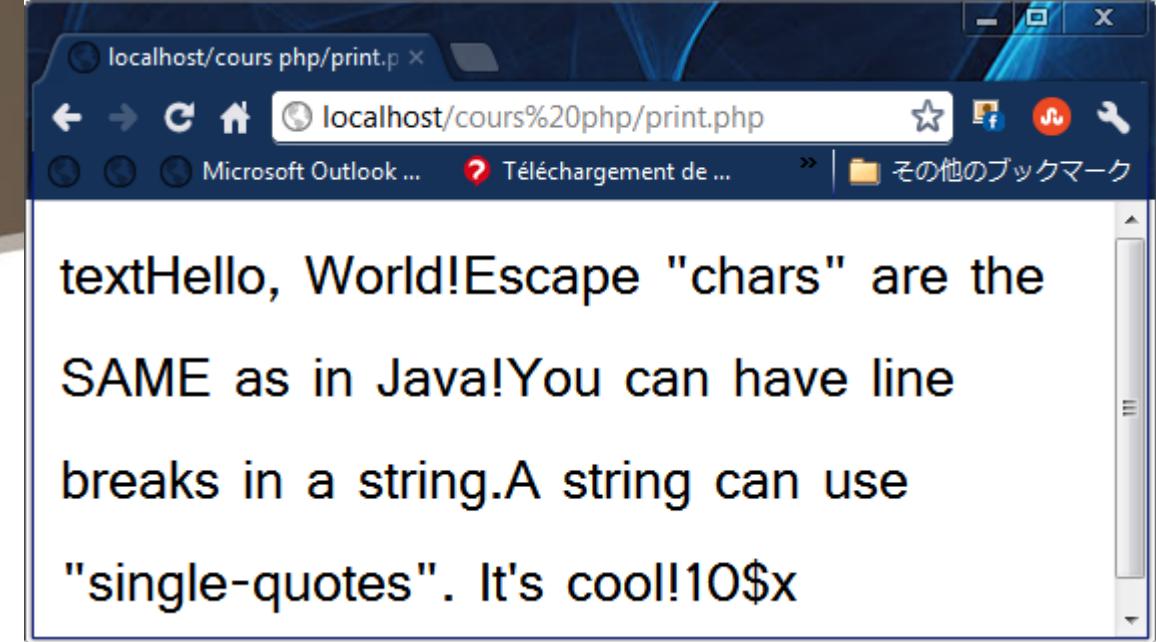
Syntaxe : echo "texte à afficher" (entre double/simple quotes)

Exemples:

```
<?php  
// This won't work because of the quotes around special!  
echo "<h5 class='special'>I love using PHP!</h5>";  
// OK because we escaped the quotes!  
echo "<h5 class=\"special\">I love using PHP!</h5>";  
// OK because we used an apostrophe '  
echo "<h5 class='special'>I love using PHP!</h5>"; ?>
```

print()

```
<?php  
print "text";  
print "Hello, World! ";  
print "Escape \"chars\" are the SAME as in Java! ";  
print "You can have line  
    breaks in a string.";  
print 'A string can use "single-quotes". It\'s cool!';  
$x=10;  
print "$x";  
print '$x';  
?>
```



L'affichage

- **echo et print()**
 - **echo** peut prendre plusieurs paramètres, séparés par des ",", contrairement à **print()**
 - **echo** est légèrement **plus rapide** que **print()**, c'est d'ailleurs pour cela que nous la privilégions.
- Voici un exemple des différentes formulations possibles :
 - **echo ' le texte ';** //affichera le texte
 - **print (' le texte ');** //affichera le texte

Echoing Variables and Text Strings

- Il est possible de mettre le nom d'une variable **entre les doubles quotes de echo**, mais ainsi vous demandez à PHP en fait d'afficher la valeur de cette variable :
- Par exemple:

```
$my_string = "Hello Bob. My name is: ";
echo "$my_string Sarah<br />";
// affichera Hello Bob. My name is: Sarah
echo "Hi, I'm Bob. Who are you? $my_string";
// affichera Hi, I'm Bob. Who are you? Hello Bob. My name is:
echo "Hi, I'm Bob. Who are you? $my_string Sarah"; ?>
//affichera Hi, I'm Bob. Who are you? Hello Bob. My name is: Sarah
```

L'affichage

- Concaténation ou paramètres ?
- Avec **echo**, on peut afficher plusieurs variables et chaines de caractères à la suite :

```
<?php $variable='du texte';  
      echo 'Ceci est ', $variable; ?>
```

- Ici, on utilise la virgule entre la chaîne de caractères et la variable. On peut également utiliser un point.
- Lorsque l'on utilise des points, c'est la concaténation.
- Avec des virgules, c'est le passage de plusieurs paramètres .

Chaînes de caractères (I)

- La différence entre l'utilisation de ' (apostrophe) ou " (double quote) est que PHP examinera ce que contient une chaîne entre " (remplacera les variables par leur valeurs), mais pas une chaîne qui est entre ' (apostrophe) qu'il affichera directement.
- Les chaînes délimitées par (apostrophe) ' sont affichées plus rapidement.

Double quotes vs simple quote

```
$Prenom = 'Romuald';
$Age = '23 ans';
$Profession = 'informaticien';
echo 'Bonjour' . $Prenom . ', tu as' . $Age . ' et ta profession est' . $Profession . '';
//Ce qui affichera sur votre navigateur : Bonjour Romuald, tu as 23 ans et ta profession est informaticien.
echo "Bonjour $Prenom, tu as $Age et ta profession est $Profession";
//Ce qui affichera sur votre navigateur : Bonjour Romuald, tu as 23 ans et ta profession est informaticien.
echo 'Bonjour $Prenom, tu as $Age et ta profession est $Profession';
//Ce qui affichera sur votre navigateur : Bonjour $Prenom, tu as $Age et ta profession est $Profession
```

//Le contenu d'une chaîne construite avec des " sera interprété par PHP et les variables éventuellement utilisées seront remplacées par leurs valeurs.

Echo vs print

echo

- 1+ paramètres séparés par ,
- Les () sont optionnelles
- N'a pas de valeur de retour

print

- 1 seul paramètre
- Les () sont obligatoires
- A une valeur de retour

Exemple :

```
$name = "Henry";
```

```
$X = print("Bonjour $name"); //affichera Bonjour Henry
```

```
echo $X; //affichera 1
```

```
$Y = echo("Bonjour $name"); //retourne une erreur car pas de  
echo $Y; //valeur de retour pour echo !!
```

Variables, types

- **Variables dynamiques** : C'est à dire des variables ayant un nom changeant au cours du script : un nom de variable qui est affecté et utilisé dynamiquement.

- **Syntaxe :** \${\$var} = valeur; // il est possible d'omettre les {}

- *Exemple 1:*

- \$toto = "foobar";
 - \${\$toto} = 2002;
 - echo \$foobar; // la variable \$foobar vaut 2002

- *Exemple 2:*

- \$a = 'bonjour';
 - \$\$a = 'monde';
 - echo "\$a \${\$a}"; // produira le même affichage que echo "\$a \$bonjour"; c'est à dire bonjourmonde.

Variables, types: fonctions prédefinies

○ Quelques fonctions :

- **empty(\$var)** : retourne la valeur **TRUE** si la variable \$var n'est pas initialisée, a la valeur **0** ou **NULL** ou la chaîne "**0**", et la valeur **FALSE** si elle a une quelconque autre valeur.
- **isset(\$var)** : retourne la valeur **FALSE** si la variable \$var n'est pas initialisée ou a la valeur **NULL** et la valeur **TRUE** si elle a une valeur quelconque.
- **unset(\$var)** : détruit une variable
- **gettype(\$var)** : retourne le type de la variable
- **settype(\$var, "type")** : convertit la variable en type **type** (cast)
- **is_long(), is_double(), is_string(), is_array(), is_object(), is_bool(), is_float(), is_numeric(), is_integer(), is_int()**...

Portée des variables

- Une variable d'un bloc **n'est pas connue** dans les fonction du même bloc.
- Une variable **locale** à une fonction **n'est pas connue** dans le reste du bloc.
- Les variables peuvent avoir une portée globale sur tout le bloc si elles sont définies **en dehors** d'une fonction. Elles peuvent être utilisées **dans une fonction** à condition de déclarer à nouveau les variables dans la fonction avec le mot-clé **global** .
- Par ailleurs, **des variables peuvent être déclarées statiques dans une fonction** (en les précédant du mot-clé **static**) afin de les utiliser récursivement dans la fonction elle-même.

Exemple : Portée des variables

```
<?php
$a = 1; /* limited variable scope */
function Test()
{
    echo $a;
    /* reference to local scope variable , hence the value of $a is undefined*/
}
Test();
?>
```

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

global

Il est possible d'utiliser dans le code même de vos fonctions des variables que vous avez déclarées dans votre script courant à l'aide du mot clé **global**.

```
<?php
function Test1()
{
    static $a = 0;
    echo $a;
    $a++;
}
Test1(); //affiche 0
Test1(); //affiche 1
Test1(); //affiche 2
?>
```

static

La variable ne perd pas sa valeur de sa dernière exécution: l'initialisation est sans effet.

Les opérateurs

- Opérateurs arithmétiques :
 - + (addition), - (soustraction), * (multiplié), / (divisé), % (modulo), ++ (incrément), --(décrément). Ces deux derniers peuvent être pré ou post fixés
- Opérateurs d'assignement :
 - = (affectation), *= (\$x*=\$y équivalent à \$x=\$x*\$y), /=, +=, -=, %=
- Opérateurs logiques :
 - and, && (et), or, || (ou), xor (ou exclusif), ! (non)
- Opérateurs de comparaison :
 - == (égalité), < (inférieur strict), <= (inférieur large), >, >=, != (différence)

Les opérateurs : exemples

○ Examples :

```
$addition = 2 + 4;  
$subtraction = 6 - 2;  
$multiplication = 5 * 3;  
$division = 15 / 3;  
$modulus = 5 % 2;  
echo "Perform addition: 2 + 4 = ".$addition."<br />";  
//Perform addition: 2 + 4 = 6  
echo "Perform subtraction: 6 - 2 = ".$subtraction."<br />";  
//Perform subtraction: 6 - 2 = 4  
echo "Perform multiplication: 5 * 3 = ".$multiplication."<br />";  
// Perform multiplication: 5 * 3 = 15  
echo "Perform division: 15 / 3 = ".$division."<br />";  
//Perform division: 15 / 3 = 5
```

Opérateurs

- **Operateurs arithmétiques:** +, -, *, /, %, ++, --
- **Operateurs d'affectation:** =, +=, -=, *=, /=, %=
- **Operateurs de comparaison:** ==, !=, >, <, >=, <=
- **Operateurs logiques:** &&, ||, !
- **Operateurs de chaine (String):** . and .= (string concatenation)

Example Is the same as

x+=y x=x+y

x-=y x=x-y

x*=y x=x*y

x/=y x=x/y

x%=y x=x%y

```
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"
```

Booléens

- Les variables booléennes prennent pour valeurs **TRUE** et **FALSE**.
- Une valeur entière nulle est automatiquement considérée comme **FALSE**. Tout comme une chaîne de caractères vide **""**. Ou encore comme les chaînes **"0"** et **'0'** castées en l'entier **0** lui même casté en **FALSE**.

```
<?php  
if(0) echo 1;      // faux  
if("") echo 2;    // faux  
if("0") echo 3;   // faux  
if("00") echo 4;  // affiche 4  
if('0') echo 5;   // faux  
if(" ") echo 7;   // L'espace vaut vrai donc affiche 7  
?>
```

- **FALSE** affichera une chaîne vide et **TRUE** affichera **1**

```
print (10==10) ;    // affichera 1  
print (10==100) ;   // n'affiche rien
```

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.

define("var",valeur) : définit la constante var (sans \$) de valeur valeur

- *Exemples :*

```
define("author","Foobar"); //constante chaine  
echo author; // affiche 'Foobar'  
define('MY_YEAR',1980); //constante int  
echo MY_YEAR; // affiche 1980
```

- les constantes sont accessibles de manière **globale**. Vous pouvez la définir n'importe où, et y accéder depuis n'importe quelle fonction.

Les constantes

- Par convention, les identificateurs de constantes sont sensibles à la casse. Mais, **define** peut prendre un 3^{ème} argument -- optionnel celui-là -- qui, mis à **1**, permet d'utiliser la constante sans que elle soit sensible à la casse
- *Exemple :*

```
define("CST1","constante 1");  
define("CST2","constante 2", 1);  
echo CST1; //affiche "constante 1"  
echo cst1; //provoque une erreur  
echo CST2; //affiche "constante 2"  
echo cst2; //affiche "constante 2"
```

Variables par Référence

- On peut à la manière des pointeurs en C faire référence à une variable grâce à l'opérateur **&** (ET commercial).
- **Exemple :**

```
<?php  
$toto = 100; // la variable $toto est initialisée à la valeur 100  
$foobar = &$toto; // la variable $foobar fait référence à $toto  
  
$toto++; // on change la valeur de $toto  
echo $foobar; // qui est répercutée sur $foobar qui vaut 101  
?>
```

Tableaux (I)

- Une variable tableau est de type **array**. Un tableau accepte des éléments **de tout type**.
- Les éléments d'un tableau peuvent être de types différents et sont séparés d'une virgule.
- Un tableau peut être initialisé avec la syntaxe **array**.
- **Exemple:**

```
<?php  
$stab_colors = array('red', 'yellow', 'blue', 'white');  
$stab = array('foobar', 2002, 20.5, $name);  
?>
```

Tableaux (I)

- Un tableau peut aussi être initialisé au fur et à mesure :
`$prenoms[] = "Clément"; $villes[0] = "Paris";`
`$prenoms[] = "Justin"; $villes[1] = "Londres";`
`$prenoms[] = "Tanguy"; $villes[2] = "Lisbonne";`
- L'appel d'un élément du tableau se fait à partir de son indice (dont l'origine est zéro comme en C).
- `<?php echo $tab[10]; //pour accéder au 11ème élément`
`?>`
- Le type String est manipulé comme tableau:
`$favorite_food = "Tunisian";`
`print $favorite_food[2]; #affichera n`

Tableaux (II)

Parcours d'un tableau

- `$tab = array('Hugo', 'Jean', 'Mario');`
- *Exemple 1:*

```
$i=0;
while($i < count($tab)) { // count() retourne le nombre d'éléments
    echo $tab[$i].'  
';
    $i++;
}

```
- *Exemple 2:*

```
foreach($tab as $elem) {
    echo $elem;
}
```
- La variable `$elem` prend pour valeurs successives tous les éléments du tableau `$tab`.

Tableaux (III) Quelques fonctions

- **count(\$tab), sizeof** : retournent le nombre d'éléments du tableau
- **in_array(\$var,\$tab)** : dit si la valeur de **\$var** existe dans le tableau **\$tab**
- **list(\$var1,\$var2...)** : transforme une liste de variables en tableau
- **range(\$i,\$j)** : retourne un tableau contenant un intervalle de valeurs
- **shuffle(\$tab)** : mélange les éléments d'un tableau
- **sort(\$tab)** : trie alphanumérique les éléments du tableau
- **rsort(\$tab)** : trie alphanumérique inverse les éléments du tableau
- **implode(\$str,\$tab), join** : retournent une chaîne de caractères contenant les éléments du tableau **\$tab** joints par la chaîne de jointure **\$str**
- **explode(\$delim,\$str)** : retourne un tableau dont les éléments résultent du hachage de la chaîne **\$str** par le délimiteur **\$delim**
- **array_merge(\$tab1,\$tab2,\$tab3...)** : concatène les tableaux passés en arguments
- **array_rand(\$tab)** : retourne un élément du tableau au hasard

Tableaux associatifs (I)

- Un tableau associatif est appelé aussi *dictionnaire* ou *hashtable*. On associe à chacun de ses éléments une clé dont la valeur est de type chaîne de caractères.
- L'initialisation d'un tableau associatif est similaire à celle d'un tableau normal.
- *Exemple 1:*
\$personne = array("Nom"=> "Cesar", "Prenom" => "Jules");
- *Exemple 2:*
\$personne["Nom"] = "Cesar";
\$personne["Prénom"] = "Jules";
- Ici à la clé "Nom"est associée la valeur "César".

Tableaux associatifs (II)

- Parcours d'un tableau associatif.

```
$personne = array("Nom" => "Cesar", "Prenom" => "Jules");
```

- *Exemple 1 :*

```
foreach($personne as $elem){  
    echo $elem;  
}
```

- Ici on accède directement aux éléments du tableau sans passer par les clés.

Tableaux associatifs (II)

- *Exemple 2 :*

```
foreach($personne as $key => $elem) {  
    echo "$key : $elem";  
}
```

- Ici on accède simultanément aux clés et aux éléments.

Exemples de manipulation de array

```
<?php  
$arr = array("foo" => "bar", 12 =>  
true);  
echo $arr["foo"]; // bar  
echo $arr[12]; // 1  
?>
```

```
<?php  
$arr = array(5 => 43, 32, 56, "b" =>  
12);  
$arr = array(5 => 43, 6 => 32, 7 => 56,  
"b" => 12);  
?>
```

```
<?php  
$arr = array(5 => 1, 12 => 2);  
foreach ($arr as $key => $value)  
{ echo $key, '=>', $value; }  
$arr[] = 56; // the same as $arr[13] = 56;  
$arr["x"] = 42; // adds a new element  
unset($arr[5]); // removes the element  
unset($arr); // deletes the whole array  
?>
```

Si un clé n'est pas fourni, PHP utilise la dernière clé +1

`unset()` supprime la clé et sa valeur .

Tableaux

```
$array = array(1, 2, 3, 4, 5); //Création d'un tableau simple.  
print_r($array); //affichera Array ([0]=>1 [1]=>2 [2]=>3 [3]=>4 [4]=>5)  
//Maintenant, on efface tous les éléments, mais on conserve le tableau :  
foreach ($array as $i => $value) {  
    unset($array[$i]);}  
print_r($array); //affichera Array ()  
$array[] = 6; //Ajout d'un élément (notez que la nouvelle clé est 5, et non 0).  
print_r($array); //affichera Array ([5]=>6)  
$array = array_values($array); //Ré-indexation :  
$array[] = 7;  
print_r($array); //affichera Array ([0]=>6 [1]=>7)
```

Partie II

Sommaire

○ PARTIE 2

■ Les fonctions

■ Les structures de contrôle

■ L'inclusion de fichier

■ POST et GET

Fonctions (I)

- Comme tout langage de programmation, php permet l'écriture de **fonctions**.
- Les fonctions peuvent prendre des arguments dont il n'est pas besoin de spécifier le type. Elles peuvent de façon optionnelle retourner une valeur.
- Les identificateurs de fonctions sont **insensibles à la casse**.

Syntaxe : Définition de la fonction :

function mafonction(\$x,\$y,...)

{ *//code de définition de la fonction*

return \$var;}

Fonctions (II)

- Même sans paramètre, un entête de fonction doit porter des parenthèses **()**. Les différents arguments sont séparés par une virgule **,**. Et le corps de la fonction est délimité par des accolades **{}**.
- *Quelques exemples :*

```
function afficher($str1, $str2) {  
    // passage de 2 paramètres  
    echo "$str1, $str2";  
}  
  
function bonjour() {  
    // passage d'aucun paramètre  
    echo "Bonjour";  
}  
  
function GetColor() {  
    // retour d'une variable de type chaîne  
    return "black"; }
```

Fonctions (III)

```
<?php  
function mafonction($toto) {  
    $toto += 15;  
    return ($toto+10);  
}  
$nbr = MaFonction(15.1);  
echo $nbr; //affichera 40.1  
?>  
/* retourne 15.1+15+10=40.1,  
les majuscules n'ont pas d'importance */
```

Fonctions (IV)

- La déclaration de la fonction peut définir plusieurs paramètres à séparer par une virgule :

```
<?php function mySum($X, $Y) {  
    $total = $X + $Y;  
    return $total; }  
  
$myNumber = 0;  
echo "Before the function, myNumber = ". $myNumber . "<br />";  
//affichera : Before the function, myNumber = 0  
$myNumber = mySum(3,4);  
// Store the result of mySum in $myNumber  
echo "After the function, myNumber = " . $myNumber . "<br />";  
//affichera : After the function, myNumber = 7  
?>
```

L'appel de fonction

- L'appel à une fonction **peut ne pas respecter son prototypage** (nombre de paramètres).
 - Mais pour cela, dans la définition de la fonction, **tous les paramètres qui ont une valeur par défaut doivent figurer en dernier dans la liste des paramètres.**
 - Seul les paramètres avec valeur par défaut peuvent être omis dans l'appel.

```
<?php  
function ht($prix,$tax=19.6){  
    return "Prix Hors Taxes :". round($prix*(1-$tax/100),2);}  
$prix=154;  
echo "Prix TTC= $prix &euro; ",ht($prix)," &euro;<br />";  
//affichera Prix TTC= 154 € Prix Hors Taxes :123.82 €  
echo "Prix TTC= $prix &euro; ",ht($prix,19.6)," &euro;<br />";  
//affichera Prix TTC= 154 € Prix Hors Taxes :123.82 €  
?>
```

L'appel de fonction II

- Une fonction peut être définie après son appel (en PHP4 du fait de la compilation avant exécution contrairement au PHP3 qui est interprété).
- *Exemple :*

```
function foo() { // définition de la fonction foo
    echo "Foo...";}
foo(); // appel de la fonction foo définie plus haut
bar(); // appel de la fonction bar pas encore définie
function bar() { // définition de la fonction bar
    echo "bar!";
}
```

- Cet exemple affichera : Foo...bar!

Par référence vs par valeur

- Par défaut, les paramètres **sont passés par copie** et les **résultats sont retournés par copie** (sauf à forcer la référence).

- Deux types de passage par référence
 - de façon permanente en ajoutant un **&** devant le nom de la variable **dans la définition de la fonction**
 - de façon ponctuelle en ajoutant un **&** devant le nom de la variable **lors de l'appel de la fonction**

Par référence vs par valeur (cont.)

○ Passage de paramètres :

- \$arg : **par valeur**
- &\$arg : **par référence**
- Forcer le passage de paramètre par référence (équivalent à user de global):

○ Exemple :

```
<?php
function change(&$var) { // force le passage systématique par référence
    $var += 100;           // incrémentation de +100
}
$toto = 12;              // $toto vaut 12
change($toto); // passage par valeur mais la fonction la prend en référence
echo $toto;           // $toto vaut 112
?>
```

Exemples: par référence/ par valeur

```
<?php
function plus3($a) {
$a += 3;
}
function plus5(&$a) {
$a += 5;
}
// passage par valeur
$x = 1; plus3($x); echo $x; // affiche "1"
// passage par référence ponctuel
$y = 1; plus3(&$y); echo $y; // affiche "4"
// passage par référence "forcé"
$z = 1; plus5($z); echo $z; // affiche "6"
?>
```

Exemples: par référence/ par valeur

```
<?php  
function dire_texte($qui, &$texte)  
{ $texte = "Bienvenue $qui" ;}  
$chaine = "Bonjour ";  
dire_texte("cher phpeur",$chaine);  
echo $chaine; // affiche "Bienvenue cher phpeur"  
echo " <br/>"  
function dire_texte_par_valeur($qui, $texte_par_valeur)  
{ $texte_par_valeur = "Bienvenue $qui";}  
$chaine_par_valeur = "Bonjour ";  
dire_texte_par_valeur("cher phpeur",$chaine_par_valeur);  
echo $chaine_par_valeur; // affiche "Bonjour"  
?>
```

Un tableau comme valeur de retour

- PHP n'offre pas la possibilité de retourner explicitement plusieurs variables
- Pour pallier cet inconvénient, il suffit de retourner une variable de type **array** contenant autant de valeurs que désiré.

```
<?php
    function dofoo() {
        $tab["a"] = "Foo";
        $tab["b"] = "Bar";
        $tab["c"] = "Baz";
        return $tab;
        // on retourne les valeurs dans un tableau
    }
?>
```

Les fonctions dynamiques

- PHP offre la possibilité de travailler avec des noms de fonctions dynamiques, qui peuvent être variables et donc dépendants de l'utilisateur du site ou de l'interrogation d'une base de données.
- Exemple : Le code suivant :

```
<?php  
$ch ="phpinfo";  
$ch(); ?>
```

Est équivalent au code :

```
<?php phpinfo(); ?>
```

Structures de contrôle (I)

- Structures conditionnelles (même syntaxe qu'en langage C) :

```
if( ... ) {
```

```
    ...
```

```
} elseif {
```

```
    ...
```

```
} else {
```

```
    ...
```

```
}
```

Exemple :

```
if ($expr) {  
    echo "$expr est vrai" ;  
}  
elseif ($expr2) {  
    echo "$expr2 est vrai" ;  
}  
else {  
    echo "tout est faux" ;  
}
```

Structures de contrôle (II)

- Structures de boucle (même syntaxe qu'en langage C) :

La boucle while:

```
$i = 1;  
while ($i <= 100){  
    echo $i;  
    $i++;  
}
```

La boucle do while:

```
$i = 1;  
do{  
    echo $i;  
    $i++;  
}  
while ($i <= 100);
```

La boucle for:

```
for($i = 0; $i < 10; $i++){  
    echo $i;  
}
```

Structures de contrôle (III)

```
switch(...){  
    case ...:{ ... } break  
    ...  
    default:{ ... }  
}
```

```
<html><head></head>  
<body>  
<?php  
$x = rand(1,5); // random integer  
echo "x = $x <br/><br/>";  
switch ($x)  
{  
    case 1:  
        echo "Number 1";  
        break;  
    case 2:  
        echo "Number 2";  
        break;  
    case 3:  
        echo "Number 3";  
        break;  
    default:  
        echo "No number between 1 and 3";  
        break;  
}  
?></body></html>
```

Structures de contrôle (IV)

- L'instruction **break** permet de quitter prématurément une boucle.
- *Exemple :*

```
while($nbr = $tab[$i++]) {  
    echo $nbr."<br />";  
    if($nbr == $stop)  
        break;  
}
```

Structures de contrôle (V)

- L'instruction **continue** permet de sauter les instructions suivantes de l'itération courante de la boucle pour passer à la suivante.

- *Exemple :*

```
for($i=1; $i<=10; $i++){  
    if($tab[$i] == $val)  
        continue;  
    echo $tab[$i];  
}
```

Inclusions

- On peut inclure dans un script php le contenu d'un autre fichier en utilisant **include** (équivaut à #include de C) ou **require** .
- Leur fonctionnement est le même mais la différence est que:
 - **include** renverra une erreur de type WARNING si elle n'arrive pas à ouvrir le fichier en question. De ce fait l'exécution du code qui suit dans la page sera exécuté.
 - **require** affichera une erreur de type FATAL qui interrompt l'exécution du script.
- **include** et **require** évaluent et insèrent à chaque appel (même dans une boucle) le contenu du fichier passé en argument.
- *Exemples : les 3 écritures suivantes sont équivalentes*
- `include("fichier.php");`
- `include('fichier.php');`
- `include 'fichier.php';`

Exemple : include

```
?php  
include("include_inc.php");  
  
$resultat = cube(3);  
echo $resultat;  
?  
?
```

include.php fait appel à :

```
<?php  
function cube($x) {  
    return $x*$x*$x;  
}  
?  
?
```

include_inc.php

est équivalent à :

```
<?php  
function cube($x) {  
    return $x*$x*$x;  
}  
?
```

```
$resultat = cube(3);  
echo $resultat;  
?  
?
```

Inclusions

- Inclure le fichier "fichier.php" se trouvant dans le répertoire "dossier" situé au même niveau que le script "../".
- Grâce à ../ nous remontons l'arborescence de fichier d'un niveau (retour dans le dossier parent).
- Dans la seconde importation, nous souhaitons importer le fichier "fichier2.php" se trouvant au même niveau que ce script.

<?php

//Importations avec require()

require('../dossier/fichier.php');

require 'fichier2.php';

//Importations avec include()

include('../dossier/fichier.php');

include 'fichier2.php';

?>

Include_once() et require_once()

- Lorsque des scripts inclus, incluent eux-mêmes d'autres scripts, il peut arriver que, vous incluez plusieurs fois le même script. Si ce dernier contient une déclaration de fonction ou de classe, cela conduira inévitablement à une erreur de redéclaration.
- Pour éviter ce problème, vous pouvez utiliser *include_once()* à la place d'*include()* et *require()* au lieu de *require_once()*.
- Son comportement est identique si ce n'est qu'un script déjà inclus ne sera pas inclus une seconde fois.

Différence entre include() et require()

- Lorsqu'un script inclus par *include()* ou *include_once()* lève une erreur, le script principal poursuit son exécution comme si de rien n'était.
- Si vous préférez que le script principal s'arrête en cas d'échec du script inclus privilégiez l'instruction *require()* ou *require_once()*.

Exemple de include et require

```
//Listing du fichier "config.php"
<?php
// Définition des variables
$a = 15;
$b = 5;
// Affichage d'un texte
echo 'Un peu de mathématiques...';
?>
```

```
//Listing du fichier "programme.php"
<?php
// Importation et exécution du fichier
require('config.php');
// Calcul de la somme
$somme = $a + $b;
// Affichage de la somme
echo 'Somme de $a + $b = ', $somme;
?>
```

- Note :

echo 'Somme de \$a + \$b = ', \$somme; //affichera: Somme de \$a + \$b = 20

echo 'Somme de \$a + \$b = '. \$somme; //affichera: Somme de \$a + \$b = 20

echo "Somme de \$a + \$b = \$somme"; //affichera : Somme de 15+5 = 20

Arrêt prématué

- Pour stopper prématièrement un script, il existe deux fonctions:
die("message") et **exit()**:
- **die** arrête un script et **affiche un message d'erreur :**

```
<?php If($requet== false)  
 die("Erreur ds la requête : <br />$requet''); ?>
```

- **exit** l'arrête aussi mais **sans afficher de message d'erreur:**

```
function foobar() {  
 exit(); }
```

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



○ GET et POST: Traitement des formulaires avec php

Les formulaires : rappel

- **action="nom_de_fichier.php" est obligatoire.**
- **Il designe le fichier qui va traiter, sur le serveur, les informations saisies dans le formulaire.**
- Si le fichier se trouve dans un autre dossier, voire sur un autre serveur, il faut utiliser une adresse absolue, en écrivant, par exemple :
- **action=**
"http://www.abcphp.com/dossier/nom_de_fichier.php"

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml"  
xml:lang="fr">  
<head><title>Titre de la page</title>  
</head>  
<body>  
<form method="post" action="nomdefichier.php">  
<fieldset>  
<legend>Titre du formulaire</legend>  
<!-- Corps du formulaire -->  
</fieldset>  
</form>  
</body>  
</html>
```

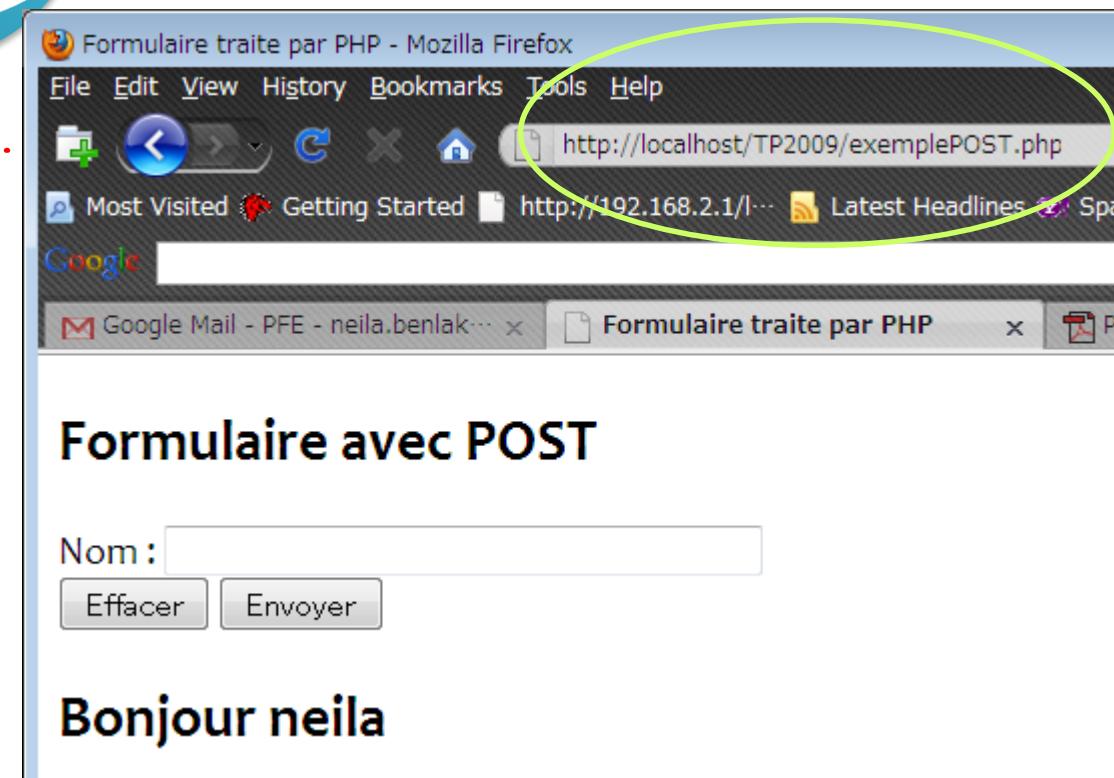
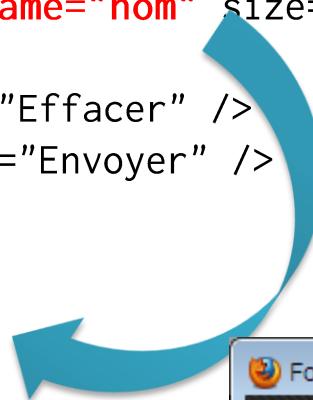
Que se passe-t-il lorsque l'utilisateur clique sur le bouton d'envoi

- Une requête HTTP est envoyée au serveur à destination du script désigné de l'élément <form>.
- La requête HTTP contient toutes les associations entre les noms des champs et leur valeur:
 - si la méthode **POST** est utilisée : Ces associations se trouvent dans l'en-tête HTTP et dans le tableau superglobal **`$_POST`**.
 - si la méthode **GET** est utilisée : Ces associations se trouvent dans l'URL et dans le tableau superglobal **`$_GET`**.
- Les clés d'accès de ces tableaux sont les noms associés aux champs par l'attribut **name**.

POST

```
<form action= "exemplePOST.php"
method="post"
<div>
Nom : <input type="text" name="nom" size="40" />
<br />
<input type="reset" value="Effacer" />
<input type="submit" value="Envoyer" />
</div>
</form>
<?php
if(isset($_POST["nom"]))
{
echo "<h2> Bonjour ". $_POST["nom"]. "</h2>";
}
?>
```

*La clé du tableau associatif est la valeur de l'attribut **name***



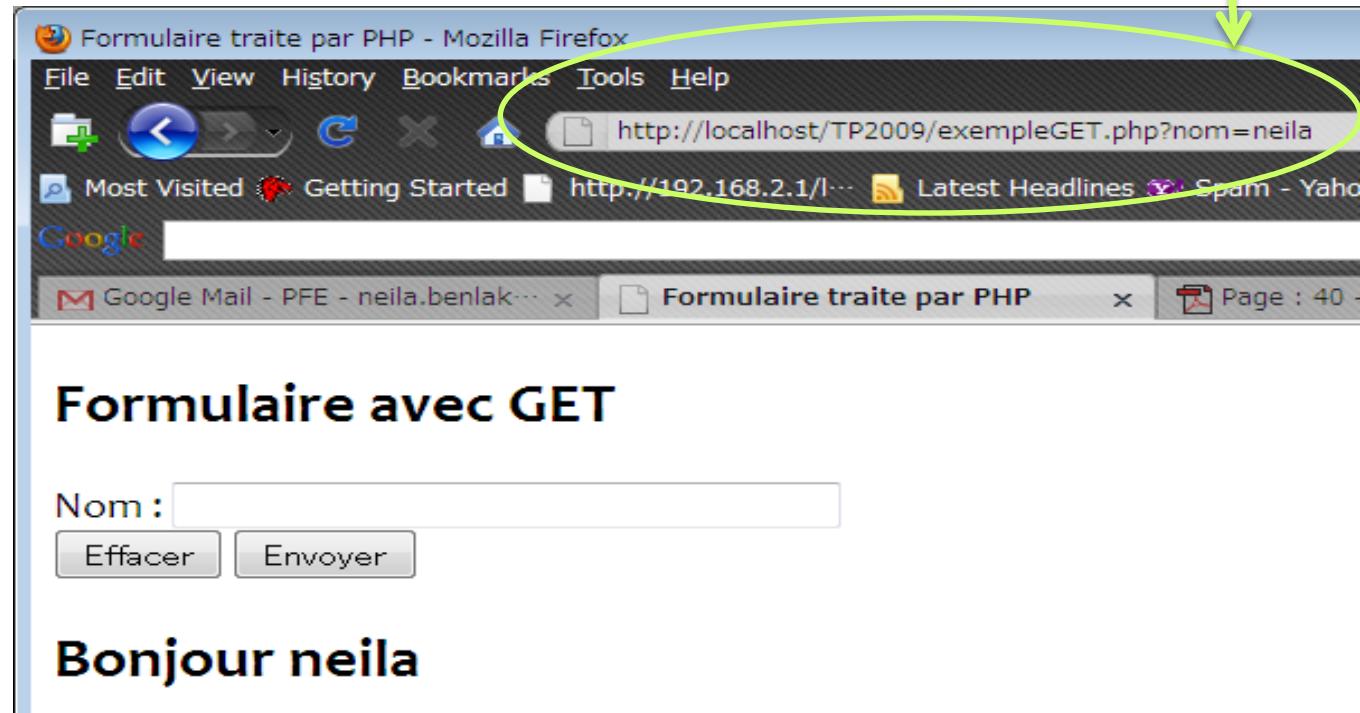
- POST utilisée en général avec un formulaire

GET

```
<form action= "exempleGET.php" method="get">
<div>
Nom : <input type="text" name="nom" size="40" />
<br />
<input type="reset" value="Effacer" />
<input type="submit" value="Envoyer" />
</div>
</form>
<?php
if(isset($_GET["nom"]))
{echo "<h2> Bonjour ". $_GET["nom"]. "</h2>";}
?>
```

✿ GET est le moyen le plus simple pour transférer des données à un script

La chaîne ?nom=neila est ajoutée à l'adresse



POST

Order.html

```
...<form action="process.php" method="post">  
<input name="item"> ...  
<input name="quantity" type="text" /> ...
```

- Ce code HTML spécifie que les données du formulaire seront soumis à la page Web **process.php** en utilisant la méthode POST.
- Pour ce faire, PHP va en fait stocker toutes les valeurs «postées» dans un tableau associatif appelé **"\$_POST"**.
- Les noms des différents champs du formulaire (attributs **name**) représentent les clés d'accès aux valeurs des champs dans le tableau associatif **superglobal. "\$_POST"** :

```
$_POST['quantity']  
$_POST['item'];
```

GET

- La méthode alternative à POST est la méthode GET, elle passe les variables à la page web process.php en les ajoutant à la fin de l'URL. L'URL, après avoir cliqué sur soumettre, l'expression suivante s'ajoutera à la fin de l'URL :

```
"? item = # # & quantity = # #"
```

- "?" indique au navigateur que les éléments suivants sont des variables. Maintenant que nous avons changé la méthode de transmission des renseignements sur "order.html", nous devons changer code de process.php pour utiliser le tableau associatif "\$_GET".

```
$_GET['quantity']  
$_GET['item'];
```

Les cases à cocher



```
<form action="choice.php">
```

Which vegetable you like ?


```
<input type="checkbox" name="lettuce" /> Lettuce
```

```
<input type="checkbox" name="tomato" checked="checked" /> Tomato
```

```
<input type="checkbox" name="pickles" checked="checked" /> Pickles
```

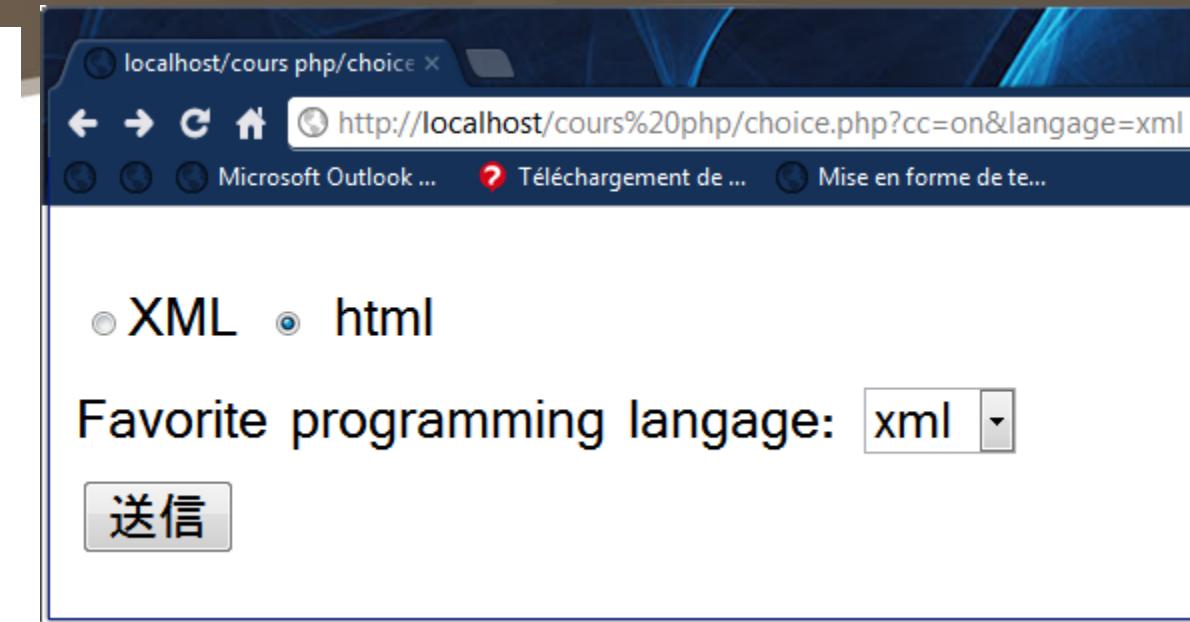
```
<input type="submit" />
```

```
</form>
```

- Les cases à cocher qui ont été sélectionnés sont envoyé avec la valeur **on** au serveur :
- <http://localhost/cours%20php/choice.php?tomato=on&pickles=on>

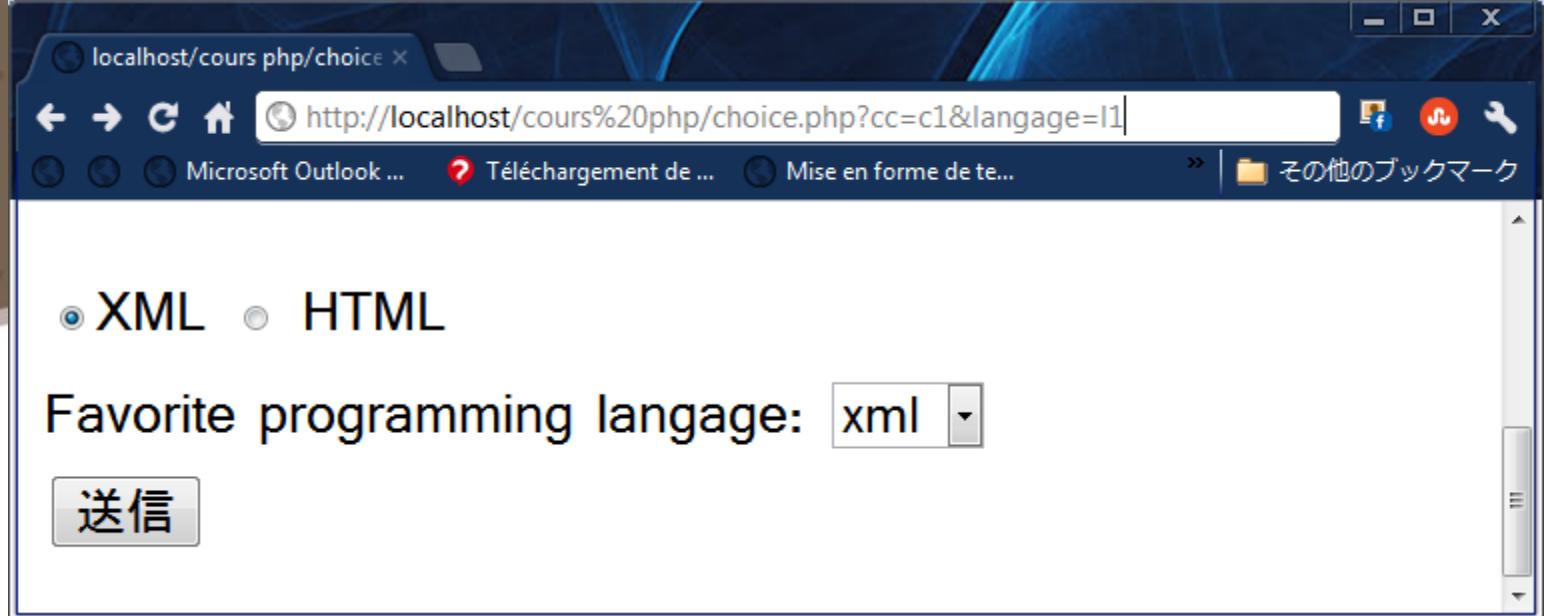
Les boutons radio et les zones select

- Si on utilise PAS l'attribut **value** : il n'est plus possible de distinguer le choix de l'utilisateur !!



<http://localhost/cours%20php/choice.php?cc=on&langage=xml>

```
<form action="choice.php">  
<label><input type="radio" name="cc" /> XML</label>  
<label><input type="radio" name="cc" /> HTML</label> <br />  
Favorite programming langage:  
<select name="langage">  
    <option>xml</option><option>html</option>  
</select> <br/><input type="submit" /></form>
```



<http://localhost/cours%20php/choice.php?cc=c1&langage=l1>

```
<form action="choice.php">
<label><input type="radio" name="cc" value="c1"/> XML</label>
<label><input type="radio" name="cc" value="c2"/> HTML</label> <br />
Favorite programming langage:
<select name="langage">
  <option value="l1">xm1</option>
  <option value="l2">html</option></select> <br /><input type="submit" />
```

Exemple simple et concret de traitement de formulaire

- Un formulaire d'identification a besoin de deux éléments : un champ texte qui reçoit le login et un champ password qui reçoit le mot de passe du visiteur.
- Nous traiterons les données **dans la même page** avant la première balise HTML du document. Notre formulaire **devra donc s'appeler lui même**.
- Les identifiants de comparaison seront stockés dans deux constantes.
- Les erreurs seront signalées à l'utilisateur.
- Si toutes les informations sont vérifiées, un message de réussite simulera l'ouverture de la session du membre.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
<head><title>Formulaire d'identification</title></head>
<?php
define('LOGIN', 'User'); // Login correct
define('PASSWORD', 'User'); // Mot de passe correct
$message = ' ' ; // Message à afficher à l'utilisateur
//***** Vérification du formulaire*****
// Si le tableau $_POST existe alors le formulaire a été envoyé
if(!empty($_POST)) // Le login est-il rempli ?
{ if(empty($_POST['login']))
    {$message = 'Veuillez indiquer votre login svp !';}
    elseif(empty($_POST['motDePasse'])) // Le mot de passe est-il rempli ?
        {$message = 'Veuillez indiquer votre mot de passe svp !';}
    elseif($_POST['login'] !== LOGIN) // Le login est-il correct ?
        {$message = 'Votre login est faux !';}
    elseif($_POST['motDePasse'] !== PASSWORD) // Le mot de passe correct ?
        {$message = 'Votre mot de passe est faux !';}
    else
        {$message = 'Bienvenue '. LOGIN . ' !'; // L'identification a réussi
    } }?>
```

```
<body>
<?php
if(!empty($message))
    echo $message;
?>
<form action="formulaire.php" method="post">
    <fieldset>
        <legend>Identifiant</legend>
        <p><label>Login :</label>
            <input type="text" name="login" id="login" value="" />
        </p>
        <p><label>Mot de passe :</label>
            <input type="password" name="motDePasse" id="password" value="" />
            <input type="submit" name="submit" value="Identification" />
        </p>
    </fieldset>
</form></body>
</html>
```

POST ou GET

○ Envoie de formulaire avec POST ou GET ?

- Utiliser GET si les données contenu dans le formulaire ne dépassent PAS les 1024 caractères.
- jamais GET pour des informations confidentielles (mot de passe)
- Utiliser POST pour l'envoie de fichiers (données binaires)
- Favoriser l'utilisation de POST pour insérer des données dans une base de données (raison de sécurité), avec le GET risque de SQL injection.

Sommaire

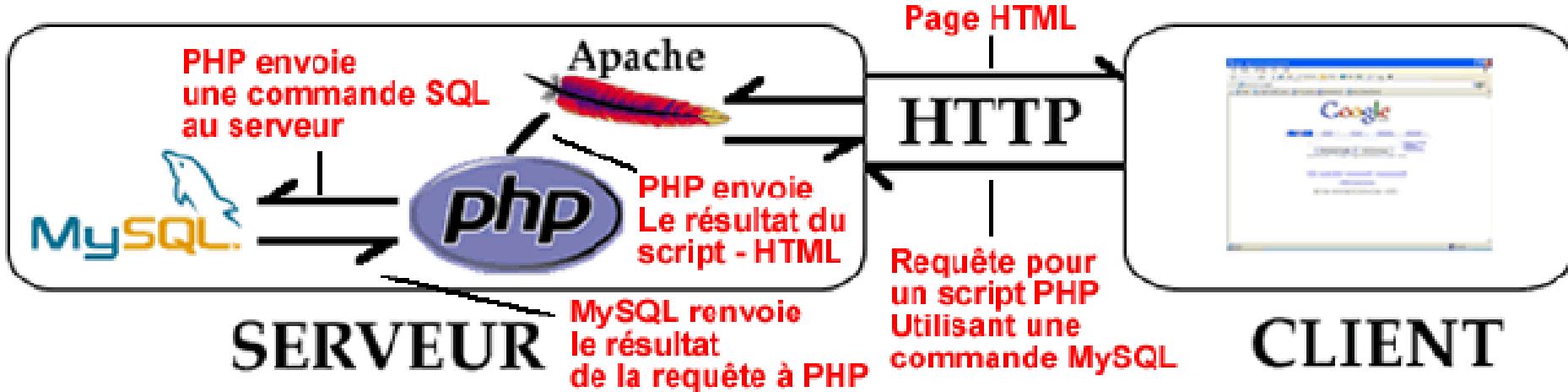
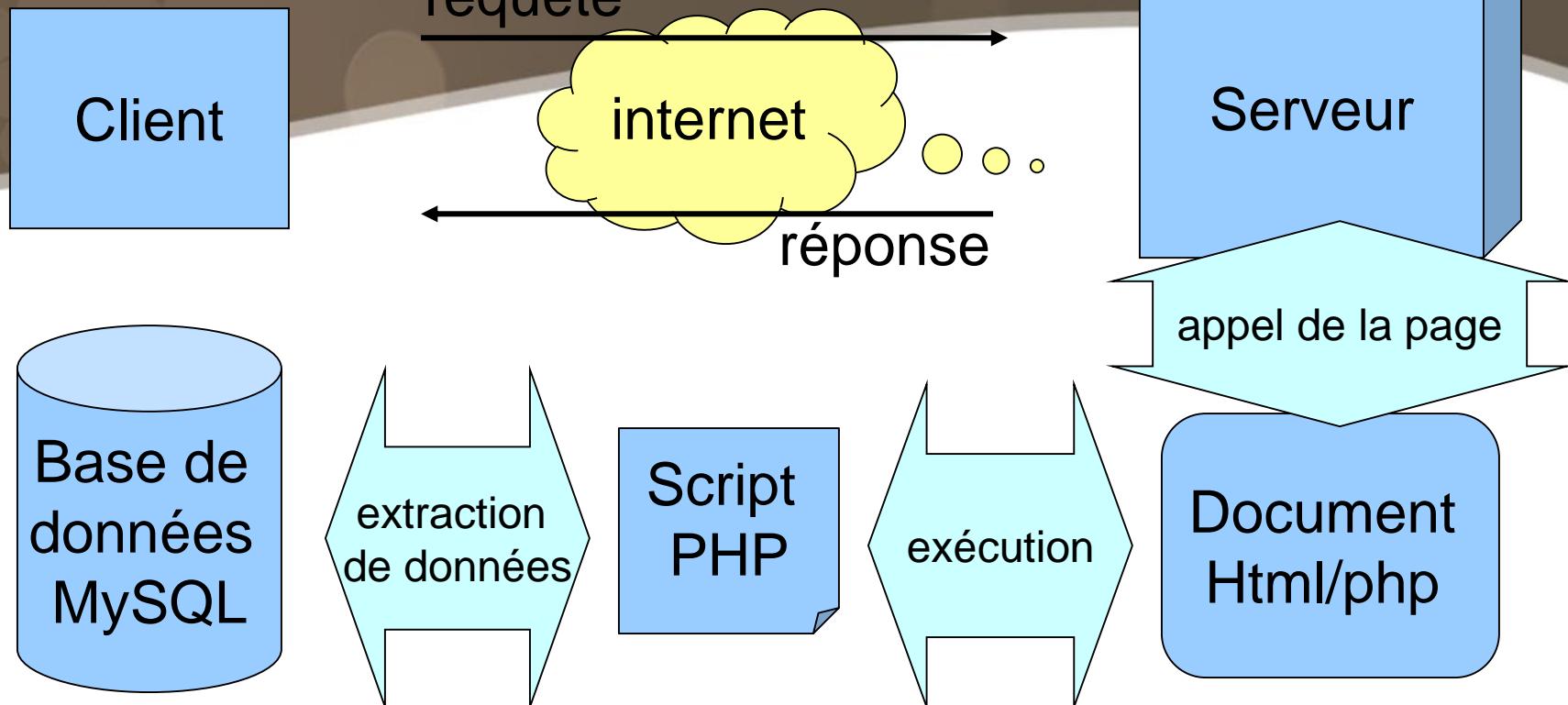
○ PARTIE 3 :

■ La création et l'interrogation d'une base de données MySQL avec PHP

Présentation

- MySQL est une base de données implémentant le langage de requête SQL un langage relationnel très connu. Cette partie suppose connue les principes des bases de données relationnelles.
- Il existe un outil libre et gratuit développé par la communauté des programmeurs libres : **phpMyAdmin** qui permet l'administration aisée des bases de données MySQL avec php. Il est disponible avec l'environnement WAMP.
- Avec MySQL vous pouvez créer plusieurs bases de données sur un serveur. Une base est composée de tables contenant des enregistrements.

Modèle



L'extension MySQLi au lieu de MySQL

- PHP MySQLi = PHP MySQL **Improved**
- Depuis la version PHP 5.5.0, Il est recommandé d'utiliser soit l'extension mysqli, soit l'extension PDO MySQL pour accéder à une BD MySQL
- Il n'est pas recommandé d'utiliser l'ancienne extension mysql pour de nouveaux développements sachant qu'elle est obsolète et sera supprimée dans un futur proche.
- PHP peut accéder à une BD mySQL :
 - En utilisant MySQLi Orientée objet ou la forme procédurale
- PDO fonctionne avec 12 SGBD différents ; MySQL est limité à MYSQL

Connexion à un SGBD MySQL

- Pour se connecter à une BD MySQL depuis un script php, il faut spécifier un **nom de serveur**, un **nom d'utilisateur**, un **mot de passe** et un **nom de base**.

- **Signature de la fonction de connexion**

```
int mysqli_connect($servername, $username, $password)
```

- permet de se connecter au serveur **\$server** en tant qu'utilisateur **\$user** avec le mot de passe **\$password**, retourne **l'identifiant** de la connexion si succès, **FALSE** sinon
- En fait il y a d'autres paramètres, mais ces paramètres sont les plus importants.

Exemple de Connexion

- Dans l'exemple suivant, une connexion est stockée dans la variable **\$con** pour être utiliser ultérieurement dans un script. "die" va être appeler au cas où la connexion échoue :

Example :

```
<?php
$con = mysqli_connect("localhost", "peter", "abc123");
if (!$con)
{
    die('Could not connect: ' . mysqli_connect_error());
}
echo "Connected successfully";
// some code
?>
```

Connexion: forme OO

- <?php

```
$con = new mysqli($servername, $username, $password);

// Check connection

if ($con->connect_error) {
    die("Could not connected: " . $con->connect_error);
}
echo "Connected successfully";

// some code
?>
```

Fermeture d'une Connexion

- La connexion va être fermée automatiquement quand le script se termine. Si l'on souhaite fermer la connexion plutôt, utilisez la fonction **mysql_close()** :
- Boolean **mysqli_close([\$id])** : permet de fermer la connexion **\$id**

Exemple :

```
<?php
$con = mysqli_connect("localhost", "peter", "abc123");
if (!$con)
{
    die('Could not connect: ' . mysqli_connect_error());
}
mysqli_close($con);
// some code
?>
```

Forme orientée objet :

\$con->close();

Interrogation

- Pour envoyer une requête à une BD, il existe la fonction :

resource `mysql_query(string $query[, resource $link_identifier])`

- **\$query** : chaîne de caractères qui contient la requête écrite en SQL et retourne un identificateur de résultat ou FALSE si échec.
 - Envoie une requête SQL au serveur
 - Renvoie un identificateur de résultat
- Les requêtes les plus couramment utilisées sont : **CREATE** (création d'une table), **SELECT** (sélection), **INSERT** (insertion), **UPDATE** (mise à jour des données), **DELETE** (suppression), **ALTER** (modification d'une table), etc.
- Attention, contrairement à Oracle SQL, les requêtes MySQL ne se terminent pas par un point virgule ';' et n'autorisent pas les **SELECT** imbriqués.

Create a Database

- **CREATE DATABASE database_name** permet de créer une BD dans MySQL. Pour exécuter cette instruction à travers Php, **mysqli_query()** est utilisée.
- *Exemple* : cet exemple crée une BD nommée "my_db":

```
<?php  
$con = mysqli_connect("localhost", "peter", "abc123") ;  
if (!$con)  
{ die('Could not connect: ' . mysqli_connect_error());}  
if (mysqli_query("CREATE DATABASE my_db", $con))  
{ echo "Database created" ; }  
else  
{ echo "Error creating database: " . mysqli_error($con) ; }  
mysqli_close($con);  
?>
```

OO syntaxe

```
<?php  
$con = new mysqli($servername, $username, $password);  
// Check connection  
if ($con->connect_error) {  
    die("Connection failed: " . $con->connect_error);  
}  
  
// Create database  
  
if ($con->query("CREATE DATABASE myDB") == TRUE) {  
    echo "Database created successfully";  
} else {  
    echo "Error creating database: " . $con->error;  
}  
  
$con->close();  
?>
```

Create a Table

- L'instruction **CREATE TABLE** est utilisée dans MySQL pour créer une table.
- ```
CREATE TABLE table_name
(
 column_name1 data_type,
 column_name2 data_type,
 column_name3 data_type,

)
```
- Pour exécuter cette instruction à travers Php, **mysqli\_query()** est utilisée.

# Exemple : création d'une table

- Cet exemple créer une table "Persons", avec 3 colonnes : "FirstName", "LastName" et "Age":

<?php

```
$con = mysqli_connect("localhost", "peter", "abc123", "myDB");
if (!$con) {
 die('Could not connect: ' . mysqli_connect_error());
}
// Create table
$sql = "CREATE TABLE Persons
 (FirstName varchar(15), LastName varchar(15), Age int)";
// Execute query
if (mysqli_query($con, $sql)) {
 echo "Table created successfully";
} else {
 echo "Error creating table: " . mysqli_error($con);
}mysqli_close($con);?>
```

# Clé primaire

- Une table doit définir une clé primaire (**primary key**).

*Example :*

```
<?php $sql = "CREATE TABLE Persons
(
 personID int NOT NULL AUTO_INCREMENT,
 PRIMARY KEY(personID) ,
 FirstName varchar(15) ,
 LastName varchar(15) ,
 Age int
) ";
```

```
mysqli_query($sql,$con) ;
```

```
?>
```

# Insert Data Into a Database Table

*Exemple : Insertion d'enregistrement dans la table "Persons" :*

```
<?php
 $con = mysqli_connect("localhost", "peter", "abc123", "myDB");
 if (!$con)
 { die('Could not connect: ' . mysqli_connect_error()); }

 //first record to be inserted
 mysqli_query("INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('Peter', 'Green', '35')");
 mysqli_query("INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('Glenn', 'Blue', '33')");
 mysqli_close($con);
?>
```

# Insert Data Into a Database Table (insertion multiple)

*Exemple : Insertion d'enregistrement dans la table "Persons" :*

```
<?php
 $con = mysqli_connect("localhost", "peter", "abc123", "myDB");
 if (!$con)
 { die('Could not connect: ' . mysqli_connect_error()); }
 $sql= "INSERT INTO Persons (FirstName, LastName, Age) VALUES ('Peter', 'Green', '35') ";
 $sql .= "INSERT INTO Persons (FirstName, LastName, Age) VALUES ('Glenn', 'Blue', '33')";
 if (mysqli_multi_query($con, $sql)) {
 echo "New recordS created successfully";
 } else {
 echo "Error: " . $sql . "
" . mysqli_error($con);
 }
 mysqli_close($con);
?>
```

# Insert Data From a Form Into a Database

- Insertion d'enregistrement dans une table à partir d'un formulaire HTML :

*Exemple : formulaire pour la table "Persons".*

```
<html><body>
 <form action="insert.php" method="post">
 Firstname: <input type="text" name="firstname" />
 Lastname: <input type="text" name="lastname" />
 Age: <input type="text" name="age" />
 <input type="submit" />
 </form>
</body></html>
```

- Le formulaire est envoyé vers "insert.php".
- Le script "insert.php" va connecter à la BD, et extraire les données du formulaire en utilisant les variables `$_POST`.
- Ensuite, la fonction `mysqli_query()` exécute la requête `INSERT INTO`.

# Insert Data From a Form Into a Database

- Le contenu de "insert.php" :

```
<?php
$con = mysqli_connect("localhost","peter","abc123","myDB");
if (!$con)
{ die('Could not connect: ' . mysqli_error()); }
$firstname=$_POST['firstname'];
$lastname=$_POST['lastname'];
$age=$_POST['age'];
$sql="INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('$firstname','$lastname','$age')";
if (!mysqli_query($con,$sql)) {
die('Error: ' . mysqli_error());
}
echo "1 record added";
mysqli_close($con);?>
```

# Interrogation

- **mysqli\_fetch\_row(\$result)** :— Retourne une ligne de résultat MySQL sous la forme d'un tableau **indexé numériquement**. Retourne FALSE s'il n'y a plus aucune ligne.
- **mysqli\_fetch\_assoc(\$result)** :— Retourne une ligne de résultat MySQL sous la forme d'un tableau **associatif**. Retourne FALSE s'il n'y a plus aucune ligne.
- **mysqli\_fetch\_array (\$result)** :— Retourne une ligne de résultat MySQL sous la forme d'un tableau associatif indexé par les champs de la sélection, d'un tableau indexé par des entiers, ou les deux,
- Prend en paramètre un identificateur de résultat **\$result**,

# Interrogation (OO)

- `while($row = $result->fetch_assoc())`
- `while($row = $result->fetch_array())`

# interrogation

- *Exemple :*

```
<?php
$requete="SELECT Nom,Prenom FROM Personne";
if($result=mysqli_query($requete)){
while($ligne=mysqli_fetch_row($result)){
echo "Nom : ".$ligne[0]."
";
echo "Prénom : ".$ligne[1]. "
";
}
}
?>
```

# Interrogation

- Example :

- Les données retournées par la fonction `mysqli_query()` sont stockées dans la variable `$result`.
- Ensuite, `mysqli_fetch_array()` est utilisée pour retourner la première ligne de l'ensemble d'enregistrement retournée comme tableau.

```
<?php
```

```
$con = mysqli_connect("localhost","peter","abc123","myDB");
if (!$con)
{
 die('Could not connect: ' . mysqli_error());
}
$result = mysqli_query("SELECT * FROM Persons");
while($row = mysqli_fetch_assoc($result))
{
 echo $row['FirstName'] . " " . $row['LastName'];
 echo "
";
}
mysqli_close($con);
```

```
?>
```

# Exemple : Affichage des résultats dans un tableau HTML

```
<?php
 $con = mysql_connect("localhost", "peter", "abc123", "myDB");
 if (!$con)
 { die('Could not connect: ' . mysqli_connect_error()); }
 $result = mysqli_query("SELECT * FROM Persons");
 echo "<table border='1'>
 <tr><th>Firstname</th><th>Lastname</th></tr>";
 while($row = mysqli_fetch_assoc($result))
 { echo "<tr>";
 echo "<td>" . $row['FirstName'] . "</td>";
 echo "<td>" . $row['LastName'] . "</td>";
 echo "</tr>";
 }
 echo "</table>";
 mysqli_close($con); ?>
```

# Interrogation : Where

- **WHERE** est utilisée pour sélectionner les enregistrements qui vérifient une certaine condition.

## Syntax

- ```
SELECT column_name(s)
      FROM table_name
      WHERE column_name operator value
```
- Pour exécuter cette instruction à travers PHP,
mysql_query() est utilisée .

Interrogation : Where

- L'exemple suivant sélectionne toutes les lignes de la table "Persons" table where "FirstName='Peter':

```
<?php $con=mysql_connect("localhost","peter","abc123","myDB");
if(!$con)
{ die('Could not connect: '.mysql_connect_error()); }
$result=mysqli_query("SELECT * FROM Persons WHERE
FirstName='Peter'");
while($row=mysqli_fetch_array($result))
{ echo $row['FirstName']."'". $row['LastName'];
echo "<br/>";
}?
?>
```

- Le résultat de l'exécution de cette exemple est : Peter Green

Fonctions additionnelles

- Quelques fonctions supplémentaires très utiles :
- **mysqli_free_result(\$result)** : efface de la mémoire du serveur les lignes de résultat de la requête identifiées par \$requet. Très utile pour améliorer les performances du serveur.
- **mysqli_insert_id([\$id])** : retourne l'identifiant d'un attribut clé primaire AUTO_INCREMENT de la dernière insertion.
- **mysqli_num_fields(\$result)** : retourne le nombre d'attributs du résultats.
- **mysqli_num_rows(\$result)** : retourne le nombre de lignes du résultats. Et ainsi permet de remplacer le **while** par un **for**.
- Penser à bien tester la valeur de retour des fonctions (**mysqli_query** et les autres) afin de détecter toute erreur et d'éviter de polluer votre page avec des *Warnings*.

Sommaire

- PARTIE 4 :
 - L'envoi de fichiers

Le formulaire d'envoi de fichier

```
<html>
  <head>
    <title>Upload file</title>
  </head>
  <body>
    <form action="uploadFile.php" method="post"
      enctype="multipart/form-data">
      <label for="file">Filename:</label>
      <input type="file" name="file" id="file" /> <br />
      <input type="submit" name="submit" value="envoyer" />
    </form>
  </body>
</html>
```



uploadForm.php

Chargement de fichiers

- Les formulaires permettent à un internaute de transférer un fichier vers le serveur.
- C'est la balise HTML suivante : <**input type="file"**> qui permet le chargement de fichiers.
- La balise FORM doit nécessairement posséder l'attribut **ENCTYPE** de valeur "**multipart/form-data**". La méthode utilisée sera **POST**.
- L'attribut **ENCTYPE** (content-type) définit le type du contenu qui va être transmit: "**multipart/form-data**" est utilisée quand une formulaire envoie des **données binaires**, telle est le cas du contenu d'un fichier.

Configuration de PHP pour permettre l'upload

- Le fichier de configuration `php.ini` contient des directives permettant d'autoriser ou non l'envoi de fichiers via un formulaire ainsi que de le paramétrier :
 - `file_uploads=On/Off` permet d'autoriser ou non l'envoi de fichiers.
 - `upload_tmp_dir = répertoire` permet de définir le répertoire temporaire permettant d'accueillir le fichier uploadé.
 - `upload_max_filesize = 2M` permet de définir la taille maximale autorisée pour le fichier. Si cette limite est dépassée, le serveur enverra un code d'erreur.
 - `post_max_size` indique la taille maximale des données envoyées par un formulaire. Cette directive prime sur `upload_max_filesize`, il faut donc s'assurer d'avoir `post_max_size` supérieure à `upload_max_filesize`.
 - Si vous n'avez pas accès à la configuration (cas d'un site hébergé sur le serveur du fournisseur d'accès ou dans le cas d'un hébergement mutualisé), vous pouvez vérifier la configuration grâce à la fonction `phpinfo()` : `<? phpinfo(); ?>`

formulaire d'envoi de fichier : exemple 2

- Il est utile d'imposer au navigateur une taille de fichier limitée par le paramètre **MAX_FILE_SIZE** dont la valeur numérique a pour unité l'octet.
- Exemple :*

```
<?php  
echo "<form action=\"PHP_SELF\" method=\"POST\""  
ENCTYPE=\"multipart/form-data\">\n  
<input type=\"hidden\" name=\"MAX_FILE_SIZE\"  
value=\"1024000\"/>\n  
<input type=\"file\" name=\"mon_fichier\" /><br />\n<input type=\"submit\" value=\"envoyer\" />\n</form>\n";?>
```

Chargement de fichiers : explication

- Le fichier, ainsi que les informations le concernant, sont accessibles via la variable super-globale **`$_FILES[]`**, qui est un tableau associatif:
 - **`$_FILES['file']['name']`** : Le nom du fichier original sur la machine source (ex : image.gif).
 - **`$_FILES['file']['type']`** : Le type MIME du fichier, si le navigateur a fourni cette information (ex : image/gif).
 - **`$_FILES['file']['size']`** : La taille du fichier envoyé, en octets (ex : 2543).
 - **`$_FILES['file']['tmp_name']`** : Le nom temporaire du fichier qui sera chargé sur la machine serveur

Chargement de fichiers : script

- Le contenu du fichier "uploadFile.php" pour le chargement d'un fichier (partie 1) :

```
<?php  
if ($_FILES["file"]["error"] > 0)  
{  
    echo "Error: " . $_FILES["file"]["error"] . "<br />";  
}  
else  
{  
    echo "Upload: " . $_FILES["file"]["name"] . "<br />";  
    echo "Type: " . $_FILES["file"]["type"] . "<br />";  
    echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";  
    echo "Stored in: " . $_FILES["file"]["tmp_name"];  
}?>
```

uploadFile.php

Chargement de fichier avec Restrictions

*For IE to recognize jpg files the type must be pjpeg, for FireFox it must be jpeg.

- Les restrictions : l'utilisateur ne peut charger que des fichiers du type .gif ou .jpeg * et la taille du fichier supérieure à 20 kb:

```
<?php  
if ((($_FILES["file"]["type"] == "image/gif")  
|| ($_FILES["file"]["type"] == "image/jpeg")  
|| ($_FILES["file"]["type"] == "image/pjpeg"))  
&& ($_FILES["file"]["size"] > 20000))  
{  
if ($_FILES["file"]["error"] > 0)  
{ echo "Error: " . $_FILES["file"]["error"] . "<br />"; }  
else  
{ echo "Upload: " . $_FILES["file"]["name"] . "<br />";  
echo "Type: " . $_FILES["file"]["type"] . "<br />";  
echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />"; // car size en octets  
echo "Stored in: " . $_FILES["file"]["tmp_name"];  
}  
}  
}else { echo "Invalid file"; } ?>
```

uploadFile.php

Saving the Uploaded File

- Les fichiers temporaires copiés disparaissent quand le script se termine, pour garder les fichiers chargés, il faut les copier du répertoire temporaire (PHP temp) vers un répertoire de destination :

```
<?php if(file_exists("upload/" . $_FILES["file"]["name"]))  
    { echo $_FILES["file"]["name"] . " already exists. "; }  
else  
    { move_uploaded_file($_FILES["file"]["tmp_name"],  
        "upload/" . $_FILES["file"]["name"]);  
    echo "Stored in: " . "upload/" . $_FILES["file"]["name"];  
    }  
}  
else { echo "Invalid file"; } ?>
```

uploadFile.php

Cookies et Sessions

Cookies et Sessions

- HTTP est un protocole de transmission **sans état** :
 - Quand vous entrez l'adresse d'un site dans votre navigateur, HTTP la transmet au serveur puis vous renvoie le fichier correspondant avant de passer aussitôt à autre chose.
 - Le serveur ne distingue pas les connexions successives d'un même utilisateur !
- Si, a partir de la page d'accueil, vous cliquez sur un lien vers une autre page du même site, rien ne lui permet de savoir que ces 2 requêtes émanent du même poste client.
 - Comment identifier un utilisateur ?
 - Comment faire pour que les données relatives à un utilisateur soit disponibles pour tous les scripts d'un même site web ?

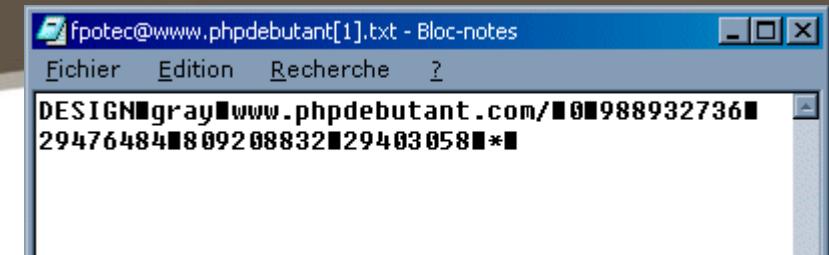
Les cookies en php

- Un cookie est un petit **fichier texte** (faisant au maximum 65 Ko) stocké **sur le disque dur du visiteur du site**. Ce fichier texte permet de **sauvegarder** diverses informations concernant ce visiteur **afin de pouvoir les réutiliser** (les informations) lors de la prochaine visite du visiteur sur ce même site.
- Par exemple, on pourrait très bien stocker dans ce cookie le **nom du visiteur** et par la suite, **afficher son nom** à chaque fois qu'il se connectera sur le site : **ceci n'est possible que si le visiteur à entré lui-même ses informations dans un formulaire sur le site**.
- Les cookies sont stockés, selon votre navigateur Internet, à un certain endroit de votre disque dur. Par exemple, avec un système composé de **Windows** et du navigateur **IE**, les cookies sont stockés dans le répertoire **C:WindowsTemporary Internet Files** .

Les cookies en php

- Un cookie n'est accessible que par le site qui l'a écrit (sauf indication).
- L'utilisation des cookies : stockage d'information de petite taille, comme le nom, le code d'accès, l'adresse ou les préférences de l'utilisateur.
- **Exp :** Stocker les coordonnées qu'un visiteur a saisies dans un formulaire et de les lire lors d'une prochaine connexion pour remplir automatiquement le même formulaire, permettant ainsi aux visiteurs de gagner du temps.

Exemple de cookie



- Exemple: (fpotec@www.phpdebutant[1].txt), la valeur entre crochets: [1] indique combien de cookies sont stockés dans le fichier :
 - DESIGN est le nom du cookie
 - gray est la valeur assignée au cookie DESIGN
 - www.phpdebutant.com/ indique d'où provient le cookie et par quel site il est utilisé.
 - Les valeurs suivantes sont des valeurs par défaut (à moins que vous ne les spécifiez lors de la génération du cookie comme la date d'expiration).
- Plusieurs cookies du même site sont tous stockés dans le même fichier (ici: Cookie: fpotec@www.phpdebutant.com/)

Creation de cookie

- Pour écrire un cookie, vous utilisez la fonction **setcookie()** :

boolean setcookie(string nom_cookie,[string valeur, integer datefin,string chemin, string domaine, integer securite])

- **nom_cookie** est une chaîne définissant le nom du cookie.
- **valeur** contient la valeur associée au cookie
- **datefin** est un timestamp (en sec). Si ce paramètre est omis, le cookie n'est valable que pendant le temps de connexion du visiteur sur le site.
- **chemin** définit dans une chaîne le chemin d'accès aux dossiers qui contiennent les scripts autorisés à accéder au cookie
- **domaine** définit le nom entier du domaine à partir duquel vous pouvez accéder au cookie
- **securite** : valeur booléenne pour indiquer si connexion sécurisé (SSL)

Les cookies en php

- Comment créer de tels cookies, grâce à la fonction **setcookie()**

<?php

```
$temps = 365*24*3600; //on définit une durée de vie de notre cookie (en secondes)
// on envoie un cookie de nom pseudo portant la valeur TOTO
setcookie ("pseudo", "TOTO", time() + $temps);
?>
```

- Grâce à ce code, nous venons d'envoyer, chez le client (donc le visiteur du site) un cookie de nom **\$pseudo** portant la valeur **TOTO**.
- Nous imposons que le cookie ait une durée de vie de un an (soit en fait l'instant présent plus un an).
- Enfin, maintenant, si le visiteur ne supprime pas ce cookie, et bien, dans toutes les pages WEB de notre site, on pourra accéder à la variable **\$pseudo** qui contiendra la chaîne de caractères **TOTO** pour toute une année.

Exemple

```
<?php  
//cookie valable uniquement pour la session  
setcookie("prenom","Jan");  
setcookie("nom","Geelsen",time()+86400); //cookie valable 24 heures  
//Ce cookie utilise tous les paramètres  
setcookie("CB","5612 1234 5678  
1234",time()+86400,"/client/paiement/",  
        "www.funhtml.com",TRUE);  
//+ieurs valeurs sous un même nom de cookie en utilisant des tableaux.  
setcookie("client['prenom']","Jan",time()+3600);  
setcookie("client['nom']","Geelsen",time()+3600);  
setcookie("client['ville']","Paris",time()+3600);?>
```

Lecture de Cookies

- Il existe 2 manières de récupérer la ou les valeurs d'un cookie:
 - dans une variable de même nom que celui du cookie (obsolète)
 - ou dans le tableau associatif superglobal **`$_COOKIE`**.
- *Exp 1:* `setcookie("nom", "Geelsen", time() + 1000)`
vous récupérez le contenu de la cookie dans : `$_COOKIE["nom"]`
- *Exp2:* Pour le cookie sous forme de tableau suivant:
`setcookie("achat['premier']", "livre", time() + 3600);
setcookie("achat['deuxieme']", "CD", time() + 3600);
setcookie("achat['troisieme']", "video", time() + 3600);`
- Pour récupérer la valeur "livre": `$_COOKIE["achat"]["premier"]`

Exemple

- *Exemple:* sur une page de notre site WEB, nous souhaitons faire en sorte que si le visiteur vient pour la première fois (ou qu'il a supprimer ses cookies), il aurait alors, la possibilité de saisir son nom dans un formulaire, ou bien s'il ne s'agit pas de sa première visite, d'afficher tout simplement Bonjour puis son nom.

On aurait alors le code suivant pour notre page (par exemple **index.php**) :

Index.php

```
<html><body>

<?php // on teste la déclaration de notre cookie
if (isset($_COOKIE['pseudo'])) {
    echo 'Bonjour ' . $_COOKIE['pseudo'] . ' !'; //cookie existe
}
else {
    echo 'Notre cookie n\'est pas encore déclaré.';
    // si le cookie n'existe pas, on affiche un formulaire
    // permettant au visiteur de saisir son nom
    echo '<form action="traitement.php" method="post">';
    echo 'Votre nom : <input type = "texte" name = "nom"><br />';
    echo '<input type = "submit" value = "Envoyer">;'
?>

</body></html>
```

Traitement.php

<?php

```
If (isset($_POST['nom'])) {
```

```
$temps = 365*24*3600;
```

// on définit la durée de vie de notre cookie

//(en secondes), 1 an dans notre cas

```
setcookie ("pseudo", $_POST['nom'], time() + $temps);
```

echo "une cookie ayant pour contenu". \$_POST['nom']. "a été stocké
sur votre machine";

// on envoie un cookie de nom pseudo portant la valeur de la

// variable \$nom, c'est-à-dire la valeur qu'a saisi la

// personne qui a rempli le formulaire

```
}
```

```
else echo 'La variable du formulaire n\'est pas déclarée.';?>
```

Suppression d'un cookie

- Pour effacer un cookie, il suffit d'envoyer un cookie du même nom mais sans valeur : `setcookie ('nom_du_cookie')`
- Exp:

```
<?php  
// Si le cookie compteur de visite existe,  
if ( isset( $_COOKIE['visites'] ) ) {  
// on demande au navigateur d'effacer son cookie  
setcookie('visites') ;  
// et on en efface la valeur en local pour éviter  
// de l'utiliser par erreur dans la suite de notre script  
unset($_COOKIE['visites']) ; }?>
```

Modifier les valeurs d'un cookie

- Pour modifier un cookie, il vous suffit de refaire appel à la fonction `setcookie()` avec le nom du cookie à modifier et sa nouvelle valeur. Il remplacera le précédent de même nom

Exemple : compteur de visiteurs

C:\wamp\www\exemples php\compteur.php - Notepad++

Fichier Edition Recherche Affichage Encodage Langage Paramétrage Macro Exécution TextFX Compléments Documents ?

cookiebonjour.php traitement.php compteur.php

```
1 <?php
2 $message = array() ;
3 if (! isset($_COOKIE['visites']) ) {
4     $message[] = '$_COOKIE[\'visites\'] est vide' ;
5     $message[] = 'le cookie n\'a pas été reçu par le serveur' ;
6     $message[] = 'on envoie le cookie avec la valeur 1' ;
7     setcookie('visites', 1) ;
8 } else {
9     $message[] = '$_COOKIE[\'visites\'] n\'est pas vide' ;
10    $message[] = 'la valeur reçue est ' . $_COOKIE['visites'] ;
11    $message[] = 'on envoie un nouveau cookie avec la valeur ' . ( $_COOKIE['visites'] +1 )
12    setcookie('visites', $_COOKIE['visites'] +1) ;
13    $message[] = 'le navigateur va modifier le cookie pour lui'
14    .' donner la nouvelle valeur ' . ( $_COOKIE['visites'] +1) ;
15 }
16 $message[] = 'vous pouvez recharger la page pour voir l\'évolution';
17 echo join('<br>', $message) ;
18 ?>
```

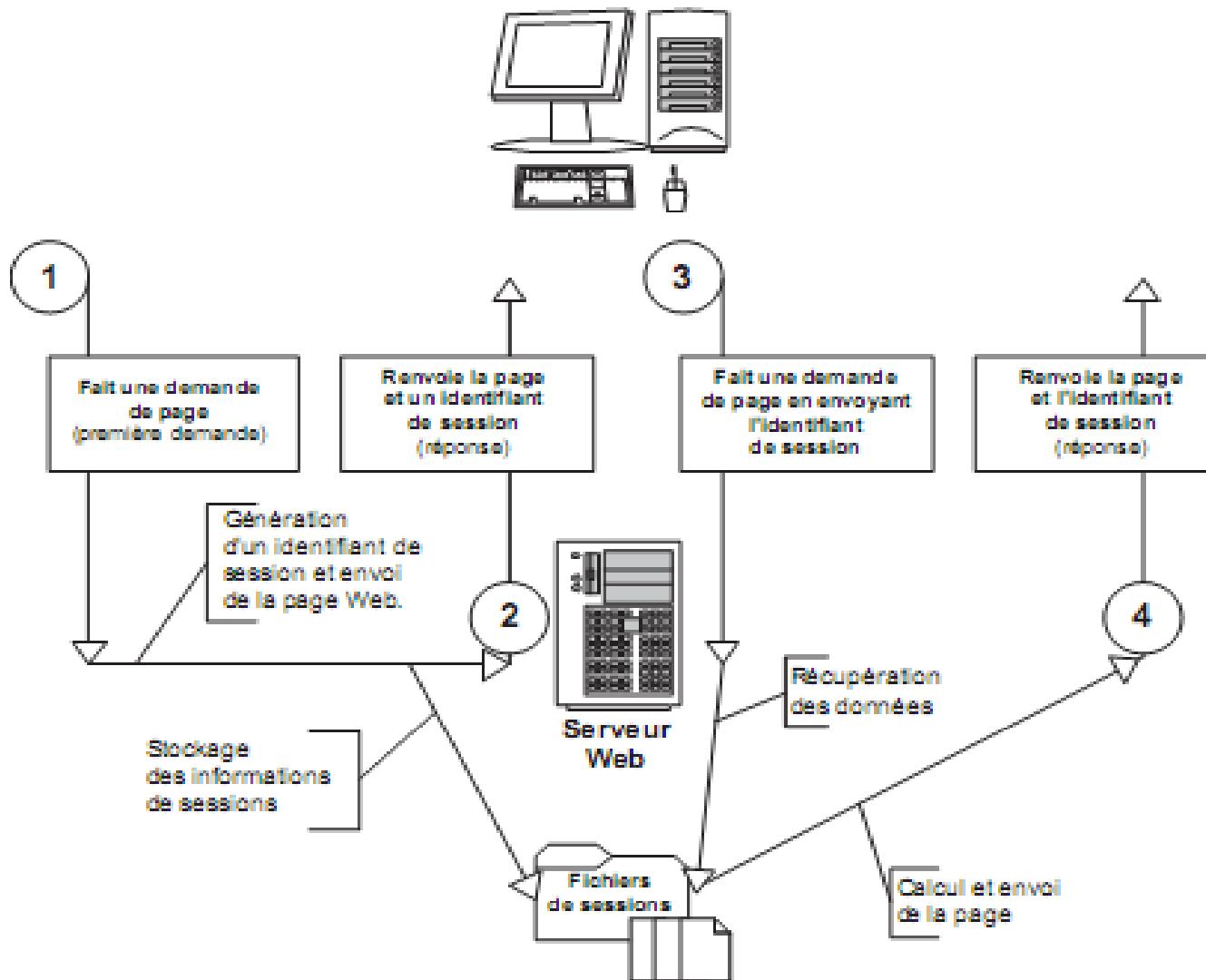
Limitations dues aux navigateurs et sécurité

- Les spécifications définissent une limite de 20 cookies par domaine et de 4 Ko par cookie, nom du cookie compris.
- Il est donc hors de question d'utiliser les cookies pour stocker de longs textes ou des images.
- Les cookies sont placées vous avec toutes les informations qui devaient être contenues dans votre machine : L'utilisateur pourrait alors modifier ces informations en écrivant dans ses cookies!!!.
- Les **sessions** permettront de pallier à ces problèmes.

Sessions (I)

- Session : un moyen pour les serveurs pour distinguer un utilisateur d'un autre .
- Les sessions sont un moyen de sauvegarder et de modifier des variables tout au cours de la visite d'un internaute **sans qu'elles ne soient visibles dans l'URL et quelque soient leurs types** (tableau, objet...).
- Cette méthode permet de sécuriser un site, d'espionner le visiteur et de **l'identifier**, de sauvegarder son panier (e-commerce), etc.
- **Les informations de sessions** sont conservées **en local sur le serveur** tandis qu'un **identifiant de session** est posté sous la forme d'un **cookie chez le client** (ou via l'URL si le client refuse les cookies).

Fonctionnement des sessions



Mécanisme des sessions : les étapes

- Ouverture d'une session dans chaque page ayant accès aux données à l'aide de la fonction **session_start()**
syntaxe : **boolean session_start()**.

Cet appel est la première instruction du script.

- Chaque utilisateur se voit attribuer un identifiant de session, qui est **une suite de 26 caractères aléatoires**. Lié à la session en cours, et donc différent lors d'une autre connexion, cet identifiant est transmis d'une page à une autre de 2 manières différentes :
 - **En étant écrit dans un cookie sur le poste client.**
 - **En étant ajouté à l'URL de la page cible d'un lien.**

Mécanisme des sessions : les étapes

- Définition des **variables de session** : des valeurs qui seront accessibles dans **toutes les pages du site** qui utilisent la fonction `session_start()` en utilisant le tableau superglobal `$_SESSION`, les clés sont les noms des variables.
- **Les noms et valeurs des variables sont stockés sur le serveur.**
- Lecture des variables de session dans chaque page en fonction des besoins à l'aide de `$_SESSION`.
- Fermeture de la session après destruction éventuelle des variables de session.

Starting a PHP Session

- Avant toute chose, vous devez commencer une session PHP
- **Note:** la fonction **session_start()** doit être placer **AVANT** la balise `<html>`:

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

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

...

```
</body>
```

```
</html>
```

Ce code va enregistrer la session de cette utilisateur sur le serveur en tant que **cookie**, et lui attribuer un **UID** (user's session ID) suite de 26 caractères.

Storing a Session Variable

- Utiliser la variable `$_SESSION` pour sauvegarder des variables dans une session :

```
<?php
    session_start(); // store session data
    $_SESSION['views']=1; // page-view counters
?>
<html><body>
<?php
    //retrieve session data
    echo "Pageviews=". $_SESSION['views'];
    // affiche Pageviews=1
?></body></html>
```

Storing a Session Variable

- Cet exemple comptabilise le nombre d'accès à la page.
- La fonction `isset()` vérifie si la variable "views" a été déjà définie. Si oui, elle est incrémenté. Sinon et si elle n'existe pas la variable "views" est créé et instanciée à 1:

```
<?php
    session_start();
    if(isset($_SESSION['views']))
        {$_SESSION['views']= $_SESSION['views']+1;}
    else
        {$_SESSION['views']=1;}
    echo "Views=". $_SESSION['views'];
?>
```

Destroying a Session

- Utiliser **unset()** et **session_destroy()** pour détruire une variables de session.
- La fonction **unset()** est utilisée pour libérer une variable de session spécifiée:

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

- Vous pouvez détruire totalement une session en appelant **session_destroy()** directement:

```
<?php  
session_destroy();?>
```

Passage de variable entre 2 pages

//La première page « un.php » démarre une session puis enregistre une variable de session et crée un lien vers la 2eme page, "deux.php" .

```
<?php session_start();  
$nom="Jean";  
$_SESSION[ 'nom' ]=$nom;  
echo "<a href=\"deux.php\">Vers la page DEUX </a>" ;?>
```

un.php

// La 2me page démarre également une session puis a accès à la variable de session de la page précédente :

```
<?php session_start();  
echo "<br /> Bonjour ",$_SESSION[ 'nom' ]; ?>
```

deux.php

NB: Cette exp ne fonctionne pas si les cookies sont désactivés dans le navigateur!!

Les sessions sans cookie

- Si PHP ne peut plus enregistrer l'identifiant de session dans un cookie sur le poste client, Il vous faut transmettre cet identifiant entre toutes les pages du site en utilisant l'URL:
- La transmission du nom et de la valeur de l'identifiant se fait en ajoutant à la fin de chaque adresse définie dans un lien le caractère ? suivi de la valeur de la constante prédefinie **SID**.
- *Exemple:*

```
<a href="pagehtml2.php?<?php echo SID?>">Page XHTML </a>
```

- Le nom de la session (**PHPSESSID**) est récupérable en appelant la fonction **session_name()** sans paramètre
- L'identifiant de session (**SID**) en appelant la fonction **session_id()**, qui retourne une chaîne de caractères.

Changement du nom de la session

```
<?php  
session_start(); // On initialise et utilise la session  
$_SESSION['nom'] = 'Sarah';  
echo $_SESSION['nom']; // affiche Sarah  
// On récupère les informations de la session :  
echo 'nom de la session : ', session_name() , '<br/>';  
#affiche PHPSESSID  
echo 'identifiant utilisé : ', session_id() , '<br/>' ;  
session_name('client') ; // Définition de 'client' comme nom de session  
// On récupère les informations de la session :  
echo 'le nouveau nom de la session : ', session_name() , '<br/>';  
#affiche client  
echo 'identifiant utilisé : ', session_id() , '<br/>' ;?>
```

Sessions

- Sauvegarder des variables de type objet dans une session est la méthode de sécurisation maximum des données : elles n'apparaîtront pas dans l'URL et ne pourront pas être forcées par un passage manuel d'arguments au script dans la barre d'adresse du navigateur.
- Les données de session étant sauvegardées sur le serveur, l'accès aux pages n'est pas ralenti même si des données volumineuses sont stockées.
- Une session est automatiquement fermée si aucune requête n'a été envoyée au serveur par le client durant un certain temps (2 heures par défaut dans les fichiers de configuration Apache).
- Une session est un moyen simple de suivre un internaute de page en page (sans qu'il s'en rende compte). On peut ainsi sauvegarder son parcours, établir son profil et établir des statistiques précises sur la fréquentation du site, la visibilité de certaines pages, l'efficacité du système de navigation...

Cookies dans php

- Vous n'avez pas à coder vous-même dans le script l'écriture du cookie, PHP se chargeant de l'envoyer immédiatement quand vous appelez la fonction `session_start()` pour la première fois.
- Si ces conditions sont remplies, vous n'avez à vous préoccuper de rien, si ce n'est de faire commencer chaque page par l'appel de la fonction `session_start()`.

PDO (PHP Data Object)

- En PHP4, pour chaque type de SGBD, il fallait utiliser une extension native dédiée :
 - mysqli pour MySQL,
 - oci8 pour Oracle...
- Bien que ces extensions aient des similitudes entre elles, vous aurez alors à manipuler des fonctions spécifiques différentes selon votre SGBD.
- Solution à partir de PHP 5 est d'utiliser **PDO (PHP Data Object)** est la principale nouveauté de PHP 5.1 : Il s'agit d'une extension qui vous permet de travailler de manière unifiée quel que soit votre SGBD..

Bases de données supportées par PDO

- PDO inclut une compatibilité avec les principales bases de données avec lesquelles PHP peut communiquer. Dans le cas où la compatibilité native n'est pas supportée, vous pouvez utiliser un pont ODBC. Vous retrouverez entre autres:
 - MySQL 3, 4 et 5 (pdo_mysql) ;
 - PostgreSQL (pdo_pgsql) ;
 - SQLite 2 et 3 (pdo_sqlite) ;
 - Oracle (pdo_oci) ;
 - ODBC (pdo_odbc).