

## Chapitre 4 : PHP5

# Plan

---

- I. Introduction
- II. Éléments de base de PHP
- III. Traitements prédéfinis
- IV. Récupération des données du formulaire
- V. Authentification et redirection
- VI. Cookies et Sessions
- VII. Programmation orientée objet en PHP
- VIII. Accès à la BD MySQL avec PHP
- IX. L'architecture MVC
- X. Les Frameworks php (Laravel)

# I. Introduction

## 1. Le langage PHP

- ▶ PHP (*Hypertext PreProcessor*) est **un langage de programmation préprocesseur** conçu pour permettre la création **des applications Web dynamiques** coté serveur.
- ▶ Ce langage de script est conçu spécifiquement pour être exécuté côté serveur.
- ▶ Grâce à un serveur Web tel que Apache, PHP permet, généralement, de réaliser les traitements suivants :
  - ▶ **Récupérer des données envoyées** par le navigateur afin d'être interprétées ou stockées pour une utilisation ultérieure.
  - ▶ **Récupérer des informations issues d'une base de données**, d'un système de fichiers (contenu de fichiers et de l'arborescence)
- ▶ Les capacités de PHP ne s'arrêtent pas à la création de pages web. Il est aussi possible de manipuler des images, de créer des fichiers PDF, etc.
- ▶ PHP est aujourd'hui l'un des langages de programmation Web les plus populaires et les plus utilisés dans le monde, il a été utilisé pour *créer un grand nombre de sites web* célèbres, comme **Facebook, Wikipédia**, etc.

## 2. Principe de fonctionnement

---

- ▶ Lorsqu'un visiteur demande à consulter une page de site internet, son navigateur envoie une requête au serveur HTTP correspondant. Si la page est identifiée comme un script PHP (généralement grâce à l'extension .php), le serveur appelle l'interpréteur PHP qui va traiter et générer le code final de la page (constitué généralement d'HTML, mais aussi souvent de CSS et de JS).
- ▶ Ce contenu est renvoyé au serveur HTTP, qui l'envoie finalement au client.
- ▶ Une étape supplémentaire est souvent ajoutée : celle du dialogue entre PHP et la base de données. Classiquement, PHP ouvre une connexion au serveur de SGBD voulu, lui transmet des requêtes et en récupère le résultat, avant de fermer la connexion.

### 3. Syntaxe de base

---

- ▶ PHP appartient à la grande famille des descendants du C, dont la syntaxe est très proche. Il s'agit d'un langage de script qui peut facilement être mélangé avec du code HTML au sein d'un fichier PHP.
- ▶ Un script PHP peut être placé n'importe où dans un document Web. Il commence par **<?php** et se termine par **?>** :

```
<?php  
// le code PHP s'insère ici  
?>
```

- ▶ L'extension des fichiers PHP est ".php". Un fichier PHP peut contenir du code de script PHP mais aussi des balises HTML.
- ▶ Exemple: un script PHP qui utilise une fonction PHP intégré "echo" pour afficher le texte "Hello World !" sur une page web:

```
<!DOCTYPE html>  
<html>  
<body>  
<h1>My first PHP page</h1>  
<?php  
echo "Hello World!";  
?>  
</body>  
</html>
```

## 4. PHP5

---

- ▶ Dès qu'on parle de PHP5, on pense implicitement au PHP Orienté Objet.
- ▶ Depuis la version 5, PHP intègre les concepts de l'orientée objet pour bénéficier des avantages offerts par cette approche de programmation tels que l'*encapsulation*, l'*abstraction*, l'*héritage* ...
- ▶ Bien qu'on peut continuer à programmer en procédural, pourtant, le concept de OO est de plus en plus utilisé par les développeurs. D'ailleurs, c'est l'approche utilisée dans les Framework et les CMS.
- ▶ La POO apporte quelques avantages non négligeables au langage PHP à savoir :
  - ▶ *Réutilisation du code* : les objets créés peuvent servir de base pour d'autres objets. Cela permet de regrouper les objets similaires pour **ne pas avoir à écrire plusieurs fois le même code**. De plus, il est possible de réutiliser le code dans différents projets.
  - ▶ *Modularité du code*: facilité de **rajouter** des éléments **sans** avoir à **modifier** tout le code.
  - ▶ *Clarté du code*: le concept impose **un code plus lisible et mieux organisé**. Il permet aussi d'avoir un code plus compréhensible pour les autres développeurs amenés à reprendre votre code.

# II. Éléments de base de PHP

## 1. Présentation

---

- ▶ Comme tout langage de programmation, PHP manipule des données. Cette manipulation nécessite l'utilisation des variables.
- ▶ Ces variables peuvent être de différents types; les plus employés sont :
  - ▶ sous forme de chaîne de caractères,
  - ▶ sous forme de nombres entiers ou décimaux,
  - ▶ sous forme de valeurs booléennes vrai ou faux (TRUE ou FALSE).
- ▶ Cependant, il en existe d'autres, qui peuvent être des types composés, comme:
  - ▶ les tableaux
  - ▶ les objets,
  - ▶ des types particuliers, comme NULL.

## 2. Les variables, constantes et leur déclaration

---

### ► Variables :

- Une variable est le conteneur d'une valeur d'un des types utilisés par PHP (entiers, flottants, chaînes de caractères, ...).
- En PHP, une variable commence toujours par le caractère dollar (\$) suivi du nom de variable. Exemples : `$var`, `$_var`, `$var2`, `$M1`, `$_123`
- PHP c'est **un langage peu typé**, la déclaration des variables n'est pas obligatoire en début de script.

### ► Constantes :

- Les constantes sont souvent utilisés en PHP pour conserver des données répétitive dans toutes les pages d'un même site.
- **Pour définir une constate**, PHP fournit la fonction `define()`, dont la syntaxe est la suivante : *boolean define(string nom\_cte, divers valeur\_cte, boolean casse).*

```
<?php
define("PI",3.1415926535,TRUE);
echo "La constante PI vaut ",PI,"<br />";
if(define("site","http://www.ensah.ma",FALSE))
{
echo "<a href=\" ",site," \">Lien vers le site de ENSAH</ a>";
}
?>
```



### 3. Affectation par valeur et par référence

- ▶ En PHP, l'opération d'affectation se fait de la même façon que d'autres langages de programmation comme C/C++ ou java. Il est possible de faire affecter une variable par valeur ou par référence. L'opérateur égale ( = ) est utilisé pour une telle affectation. Exemples :
  - ▶ *Affectation par valeur* : `$var = valeur;`
  - ▶ *Affectation par référence* : `$var2=&$var1;`
- ▶ PHP accepte aussi les opérateurs d'affectation combinée comme `+=` (`$x += $y` équivaut à `$x = $x + $y`), `*=`, ... En plus des opérations arithmétiques, il existe **un** autre **opérateur** bien particulier au langage PHP qui est le point (`.`), il permet d'**effectuer une concaténation**. Il est possible de l'utiliser de la façon suivante (`.=`) .
- ▶ Exemples :

```
<!DOCTYPE html>
<html>
<body>
<?php
$x=2;
$y=3;
echo "La somme : $x+$y =", $x+$y, "<br />";
?>
</body>
</html>
```

```
<?php
$nom="Meachel";
$prénom="Jean";
echo "ton nom et prénom est :", $prénom, " ", $nom, "<br />";
?>
```

## 4. Les opérateurs de comparaison

---

- ▶ == Teste l'égalité de deux valeurs.
- ▶ != ou <> Teste l'inégalité de deux valeurs.
- ▶ === Teste l'**identité** des valeurs et des **types** de deux expressions.
- ▶ !== Teste la non-identité de deux expressions.
- ▶ < Teste si le premier opérande est strictement inférieur au second.
- ▶ <= Teste si le premier opérande est inférieur ou égal au second.
- ▶ > Teste si le premier opérande est strictement supérieur au second.
- ▶ >= Teste si le premier opérande est supérieur ou égal au second.
- ▶ AND Teste si les deux opérandes valent TRUE en même temps. && Équivaut à l'opérateur AND mais n'a pas la même priorité.
- ▶ OR Teste si l'un au moins des opérandes a la valeur TRUE. || équivaut à l'opérateur OR mais n'a pas la même priorité.
- ▶ XOR Teste si un et un seul des opérandes a la valeur TRUE.
- ▶ ! Opérateur unaire de négation, qui inverse la valeur de l'opérande.

## 5. Les instructions conditionnelles

---

- ▶ Comme tout langage, PHP dispose d'instructions conditionnelles qui permettent d'orienter le déroulement d'un script en fonction de la valeur de données.
- ▶ La syntaxe utilisée est identique à celle utilisée par d'autres langages de programmation comme C, C++, Java...

```
<?php
if (condition1) {
    // Instructions
} elseif (condition2) {
    // Instructions
} elseif (condition3) {
    // Instructions
} else {
    // Instructions
}
?>
```

```
<?php
switch ($variable) {
    case valeur1:
        // Instructions
        break;
    case valeur2:
        // Instructions
        break;
    // Autres cases...
    default:
        // Instructions
}
?>
```

```
$result = match (expression) {
    valeur1 => résultat1,
    valeur2 => résultat2,
    // ...
    default => valeurParDéfaut,
};
```

## 6. Boucles

- ▶ De même pour les boucles la syntaxe est inspirée aussi de langage C.
- ▶ En plus des boucles habituelles (*for*, *while*, *do...while*), PHP ajoute à partir de la version 4 une autre boucle qui est *foreach*.
- ▶ La boucle *foreach* permet de parcourir l'ensemble des éléments d'un tableau, sans avoir besoin de connaître ni le nombre d'éléments ni les clés. Sa syntaxe est donnée comme suit: *foreach(\$tableau as \$cle=>\$valeur){ }*
- ▶ Des exemples :

```
<?php
//Création d'un tableau associatif
$i=0;
while($i<=4){
    $stab["nombre aleatoire ".$i] = rand(100,1000);
    $i++;
}
//Lecture des clés et des valeurs
foreach($stab as $cle=>$val)
{echo " Le <b>$cle</b> egale à <b>$val</b><br />";}
?>
```

Le nombre aleatoire 0 egale à 408  
Le nombre aleatoire 1 egale à 841  
Le nombre aleatoire 2 egale à 243  
Le nombre aleatoire 3 egale à 362

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

Les puissances de 2 sont : 1 2 4 8 16 32 64 128 256

## 7. Les tableaux

---

- Pour la création des tableaux , PHP fournit deux méthodes:
  - En utilisant la méthode classique : la déclaration commence par dollar suivi de nom de tableau puis par des crochets []. Exemple : \$tab[0] , \$tab[1] = 2004; \$tab[2] = "PHP5";
  - En utilisant la fonction array() : elle permet de créer de manière rapide des tableaux **indiciels** ou **associatifs**. Il s'agit de la méthode le plus souvent utilisée pour la création de tableaux. La syntaxe est: `$tab=array(valeur0,valeur1,...,valeurN)`. Un tableau multidimensionnels peut être défini comme étant un ensemble d'array: `array(array(), array(), ... , array())`.
- La création d'un **tableau associatif** consiste à **associer pour chacun de ses éléments** une **clé et une valeur**. La syntaxe de la fonction array() dans ce cas, est la suivante: `$tabasso=array("clé1"=>valeur1,"clé2"=>valeur2,... "cléN"=>valeurN)`.

- Exemple :

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

*Peter is 35 years old.*

## 8. Fonction en PHP

---

- ▶ Comme d'autres langages, PHP permet bien entendu d'écrire ses propres fonctions. Pour déclarer une fonction en PHP, il suffit d'utiliser **le mot-clef *function***.
- ▶ Comme le langage n'est pas typé, **une fonction peut retourner n'importe quel type de valeur** (chaîne, entier...) **ou ne rien retourner du tout**.
- ▶ Enfin, **ses arguments peuvent avoir des valeurs par défaut**.
- ▶ Exemple :

```
<?php
function foo($x,$y=0){
    print 'La fonction foo('.$x.','.$y.') donne :';
    return $x+$y;
}
echo foo(2); /*La fonction foo('.$x.','.$y.') donne :2 */
echo foo(3,4) /* La fonction foo('.$x.','.$y.') donne :7 */
?>
```

# III. Traitements prédéfinis

## 1. Traitement des chaînes de caractères

---

- ▶ De très nombreuses fonctions de traitement des chaînes de caractères, dont voici les principales:
  - ▶ *int strlen(\$s)*: retourne la taille d'une chaîne de caractère.
  - ▶ *int strcmp(\$s1,\$s2)*: compare deux chaînes, elle retourne 0 si identique, -1 si *\$s1 < \$s2*, 1 sinon (il existe aussi *strcasecmp()* insensible à la casse et *strnatcmp()* qui utilise l'ordre naturel).
  - ▶ *string substr(\$s, int \$indice, int \$n)*: retourne une sous-chaîne de taille *\$n* à partir de *\$indice*.
  - ▶ *int substr\_count(\$s1, \$s2)*: retourne le nombre d'occurrences de *\$s2* dans la chaîne *\$s1*.
  - ▶ *int strpos(\$s1, \$s2, int offset)*: trouve la position d'un caractère dans une chaîne. (*stripos()* version insensible à la casse)
  - ▶ *mixed str\_replace(\$s1,\$s2,\$s, \$var)*: remplace toutes les occurrences de *\$s1* par *\$s2* dans une chaîne *\$s*. Le nombre de remplacement est contenu dans *\$var*. (*str\_ireplace()* version insensible à la casse)
  - ▶ *Array explode(\$sep, \$s)*: retourne un tableau de chaînes, chacune d'elle étant une sous-chaîne du *\$s* extraite en utilisant le séparateur *\$sep*.
  - ▶ *string implode(\$sep, array \$tab)*: fait le contraire de *explode()*. Elle réunit tous les éléments d'un tableau dans une chaîne en les séparant par le caractère *\$sep*.
  - ▶ *string trim(\$s, \$sc) (resp rtrim())*: retire la chaîne *\$sc* de *\$s* (resp de la fin de *\$s*).
  - ▶ *int similar\_text(\$s1, \$s2, \$pourcent)*: calcule la similarité de deux chaînes en nombre de caractères ou en pourcentage retourné dans la variable *\$pourcent*.
  - ▶ *mixed str\_word\_count(\$s)*: retourne le nombre de mots présents dans une chaîne.
  - ▶ *htmlspecialchars()*: permet de convertir des caractères spéciaux (par exemple <, > ) aux entités HTML (& amp; , &#039; , &lt; , &gt; , ...). Une fonction identique est *htmlentities()*.
  - ▶ *string sprintf("format", \$s1, \$s2,... \$sN)*: retourne une chaîne formatée contenant *\$s1* à *\$sN*.
  - ▶ *divers sscanf(\$s, "format")*: décompose une chaîne selon un format donné et retourne ses éléments dans un tableau ou dans les variables *\$s1* à *\$sN*.

.... la suite

## ► Exemples :

```
<?php
$ch1="ENSAH est un établissement public d'enseignement supérieur";
echo substr($ch1 , 0,5); //affiche : ENSAH
echo substr($ch1, 10);
//affiche:un établissement public d'enseignement supérieur
$ch2='p';
$nb=substr_count($ch1, $ch2); //
echo "le caractere $ch2 est présenté $nb fois dans la phrase";
// affiche : le caractere p est présenté 2 fois dans la phrase
$ch3=str_replace('ENSAH', 'FST', $ch1);
echo $ch3;
//affiche: FST est un établissement public d'enseignement supérieur
$ch4= 'FST';
$nb=strpos($ch3,$ch4);
echo "Le mot $ch4 comme à la position $ndans la phrase";
// Le mot FST comme à la position 0 dans la phrase
?>
```

```
<?php
$ch1="Blanc";
$ch2="Bleu";
$ch3="blanc";
echo strcmp ($ch1,$ch2);//Affiche -1
echo strcasecmp ($ch1,$ch3);//Affiche 0
echo strncasecmp ( $ch1, $ch2,2);//Affiche 0
?>
```

```
<?php
$s1="MySQL";
$s2="PgSQL";
echo similar_text($s1,$s2,$pourc), "caractères communs";
//3 caractères communs
echo "Similarité : ", $pourc, "%"; //Similarité : 60%
?>
```

```
<?php
$lien = "ensah.ma/eservices/etudiant/afficher/1";
$query = explode("/", $lien);
echo $query[0]; // ensah.ma
echo $query[1]; // eservices
echo $query[2]; // etudiant
echo $query[3]; // afficher
echo $query[4]; // 1

$lien = implode("/", $query);
echo $lien; // ensah.ma/eservices/etudiant/afficher/1
?>
```

```
<?php
$a = "...Jean " ;
$b = "Dupont__";
echo $a,$b,"<br />"; // Affiche : ...Jean Dupont__
echo trim($a,' .')," ",rtrim($b,' _');
// Affiche : Jean Dupont
?>
```

```
<?php
$personne = "1685-1750 Jean-Sébastien Bach";
$format="%d-%d %s %s";
$nb = sscanf($personne,$format,$ne,$mort,$prenom,$nom);
echo "$prenom $nom né en $ne, mort en $mort <br />";
echo "Nous lisons $nb informations";
?>
```

```
<?php
$ch1 = "Monsieur " ;
$ch2 = "Rasmus" ;
echo sprintf ("Bonjour %s %s, bravo !",$ch1,$ch2);
//Affiche: Bonjour Monsieur Rasmus, bravo !
?>
```



## 2. Traitements sur tableaux

- ▶ PHP fournit plusieurs fonctions prédéfinies pour faire des traitements sur les tableaux :
  - ▶ *int count(array \$tab, mode)*: retourne le nombre d'éléments de *\$tab*. *mode* prend 0 ou 1, *mode=1* pour compter les éléments d'un tableau multidimensionnel.
  - ▶ *mixed array\_sum (array \$tab)* : retourne la somme des éléments d'un tableau.
  - ▶ *\$sous\_tab = array\_slice(array \$tab,int \$debut, int \$fin)* : crée un sous-tableau avec \$debut à \$fin .
  - ▶ *int array\_push(\$tab, valeur1, valeur2,..., valeurN)* : ajoute en une seule opération les N éléments passés en paramètres à la fin du tableau \$tab et retourne la taille.
  - ▶ *int array\_unshift(\$tab, valeur1, valeur2,..., valeurN)* : à l'encontre de *array\_push()* cette fonction ajoute des éléments au début d'un tableau.
  - ▶ *array\_pop(\$tab) resp (array\_shift(\$tab) )*: supprime le dernier (resp le premier ) élément du tableau \$tab.
  - ▶ *array array\_values(\$tab) resp. array\_keys* : retourne un tableau ne contenant que les valeurs du tableau associatif \$tab. Crée des indices numériques de 0 à N.
  - ▶ *void shuffle(\$tab)* : mélange les valeurs de \$tab.
  - ▶ *array array\_unique(array \$tab)* : élimine les doublons et retourne un nouveau tableau.
  - ▶ *array array\_reverse(array \$tab , bool \$cle)* : Inverse l'ordre des éléments du tableau. Préserve les clés si le paramètre \$cle vaut TRUE.
  - ▶ *array\_count\_values(array \$tab)* : compte le nombre d'occurrences des éléments de \$tab et retourne un tableau dont les clés sont les valeurs du tableau \$tab et les valeurs le nombre d'occurrences.
  - ▶ *bool in\_array(divers \$val, array \$tab , bool \$type)* : recherche l'élément \$val dans le tableau \$tab. Si le paramètre type vaut TRUE les types de la valeur recherchée et de l'élément doivent être identiques.
  - ▶ *divers array\_rand (array \$tab , int \$N)* : choisit \$N éléments au hasard dans \$tab et retourne un tableau contenant les clés des éléments choisis. Si \$N vaut 1 la fonction retourne une chaîne contenant sa clé.
  - ▶ *void sort(\$tab)* : tri croissant (décroissant avec rsort) selon les valeurs. Il existe aussi ksort(\$asso), asort(\$asso), void usort(\$tab, moncmp).

### 3. Traitements sur les variables

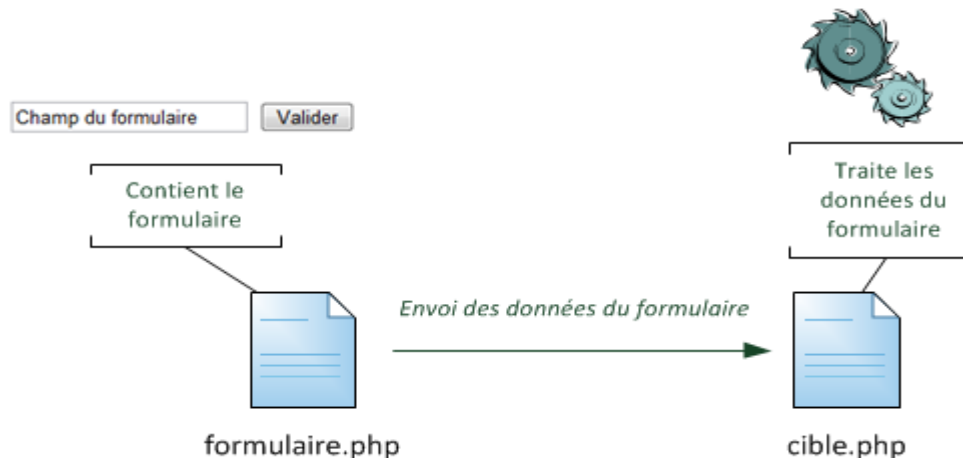
---

- ▶ Déterminer le type: avant de manipuler des variables, en utilisant, par exemple, des opérateurs, il peut être utile de connaître leur type.
  - ▶ La principale fonction permettant de le faire est `gettype()`, dont la syntaxe est la suivante : `string gettype($var)`, elle retourne une chaîne de caractères contenant le type de la variable en clair.
  - ▶ Les fonctions suivantes permettent de vérifier si une variable est d'un type précis : `is_null($var)`, `is_integer($var)` ou `is_int($var)`, `is_double($var)`, `is_string($var)`, `is_bool($var)`, `is_array($var)`, `is_object($var)`, `is_resource($var)`.
- ▶ Convertir le type: parfois, il peut être indispensable de convertir explicitement une variable d'un type dans un autre. Ce qui le cas des variables issues d'un formulaire qui sont toujours de type `string`. Pour convertir une variable d'un type dans un autre, voici comment : `$result=(typeDésiré)$var;`
- ▶ Contrôler l'état: lors de l'envoi de données d'un formulaire vers le serveur, le script qui reçoit les informations doit pouvoir détecter l'existence d'une réponse dans les champs du formulaire. Les fonctions `isset()` et `empty()` permettent ce type de contrôle.

# IV. Récupération des données du formulaire

## 1. Formulaire et PHP

- ▶ Rappelons que pour créer un formulaire en HTML on utilise l'élément `<form>`. Ce dernier possède **deux attributs obligatoires** qui sont *method* et *action*.
  - ▶ Le premier sert à préciser la méthode avec laquelle les données seront envoyées.
  - ▶ Le deuxième (action) sert à préciser le fichier récupérant les données.
- ▶ `<form method=" " action=" " >`



## 2. Fichier de traitement

---

- ▶ Grâce à l'attribut *action*=" ", l'endroit de fichier de traitement est précisé selon les cas suivants :
  - ▶ Au cas où le **fichier de traitement** et le document contenant le **formulaire se trouvent séparés**, il faut donner comme valeur à l'attribut *action* l'adresse où se trouve le fichier. Exemples : *action*="*nom\_de\_fichier.php*" (même endroit), *action*= "*http://www.ensah.ma/dossier/fichier.php*".
  - ▶ Au cas où le **fichier de traitement contient au même temps le script de traitement et le code de formulaire**, la meilleure solution c'est de donner à *action* la variable *\$\_SERVER["PHP\_SELF"]* comme valeur, qui contient le nom du fichier en cours d'exécution comme valeur de l'attribut *action*.
    - ▶ *\$\_SERVER* est un tableau contenant des informations comme les en-têtes, dossiers et chemins du script, etc.... (Pour voir son contenu vous pouvez exécuter la fonction *var\_dump(\$\_SERVER)*)
- ▶ Il est recommandé que ce fichier soit présent dans le même répertoire que celui contenant le formulaire, mais ce n'est pas obligatoire.

### 3. Méthodes d'envoi

---

- ▶ Les deux méthodes (method) d'envoi sont *GET* et *POST*.
  - ▶ Pour la première méthode (*GET*), les données transiteront par l'URL comme on l'a appris précédemment se **récupèrent** grâce à l'array ***\$\_GET***.
  - ▶ Pour la deuxième (*POST*), les informations transiteront de manière masquée par cette méthode se récupèrent grâce à l'array ***\$\_POST***.
- ▶ ***\$\_GET*** et ***\$\_POST*** sont des tableaux de données **associatifs** et **superglobaux**. Voici leurs principales caractéristiques :
  - ▶ Ils sont générés à la volée par PHP avant même que la première ligne du script ne soit exécuté.
  - ▶ Ce sont des **tableaux associatifs** comme ceux que l'on déclare traditionnellement. Leur manipulation est exactement semblable à ces derniers. **Les clés correspondent aux noms des variables** transmises et les valeurs à celles associées à ces variables.
  - ▶ Ils sont des variables **superglobales**, c'est à dire visibles de partout dans le programme (même à l'intérieur d'une fonction utilisateur).
  - ▶ Ils sont **accessibles en lecture et en écriture**. Il est donc possible de les modifier.

## 4. Quelle méthode d'envoi à utiliser ?

---

- ▶ Les deux méthodes ne permettent pas de sécuriser l'envoi des données. Le fait de ne pas afficher les données par la méthode POST ne signifie en rien qu'elles sont cryptées.
- ▶ Le choix d'une méthode alors dépend du contexte:
  - ▶ Pour la méthode GET: il est possible de l'utiliser dans le cas suivant : si par exemple, nous souhaitons mettre en place un moteur de recherche alors nous pourrions nous contenter de la méthode GET qui transmettra les mots-clés dans l'URL. Cela nous permettra aussi de fournir l'URL de recherches à d'autres personnes. C'est typiquement le cas des URLs de Google. Exemple: une URL du moteur de recherche Google : `http://www.google.fr/search?q=php`
  - ▶ Pour la méthode POST: c'est préférée lorsqu'il y'a un nombre important de données à transmettre ou bien lorsqu'il faut envoyer des données sensibles comme des mots de passe. Dans certains cas, seule la méthode POST est requise : un upload de fichier par exemple.

## 5. Comment récupérer les données?

---

- ▶ Le tableau `$_GET` contient tous les couples variable / valeur transmis dans l'url.
- ▶ Le tableau `$_POST` contient tous les couples variable / valeur transmis en POST, c'est à dire les informations qui ne proviennent ni de l'url, ni des cookies et ni des sessions.
- ▶ **Exemple** : Pour récupérer la valeur d'un élément input de type texte dont le nom est *prenom* `<input type = "text" name= "prenom" />`
  - ▶ `<?php echo $_GET['prenom']; ?>`
  - ▶ `<?php echo $_POST['prenom']; ?>`
- ▶ **NB:**
  - ▶ La casse des variables est importante. Il faut bien penser à mettre `$_GET` et `$_POST` en majuscules. Dans le cas contraire, il sera impossible d'obtenir une valeur et une erreur de type undefined variable sera retournée.
  - ▶ A noter aussi qu'il existe d'autres tableaux associatifs *superglobal* `$_REQUEST` qui fonctionne exactement comme tous les autres tableaux.

## 6. Les valeurs uniques et multiples

- ▶ Les valeurs uniques proviennent des champs de formulaire dans lesquels l'utilisateur ne peut **entrer qu'une valeur**, un texte par exemple, ou ne peut faire qu'un seul choix (bouton radio, liste de sélection à choix unique).  
Exemple : `<input type="radio" name="choix" />`
- ▶ Les valeurs multiples proviennent des champs de formulaire qui peuvent permettre aux visiteurs de **saisir plusieurs valeurs** sous un même nom de composant. Cela peut concerner un groupe de cases à cocher ayant le même attribut *name*, dont il est possible de cocher une ou plusieurs cases simultanément. Dans ce cas, il ne s'agit pas d'une valeur scalaire mais un tableau qui est récupéré côté serveur. Il faut pour cela faire **suivre le nom du composant de crochets**, comme pour créer une variable de type array.  
Exemple :
  - ▶ DUT : `<input type="checkbox" name="choix[0]" value="DUT" />`
  - ▶ DEUG : `<input type="checkbox" name="choix[1]" value="DEUG" />`
  - ▶ L'utilisateur peut cocher les deux cases simultanément. Les valeurs sont récupérées comme suit :
    - ▶ `$_POST["choix"][0]` contient la valeur "DUT"
    - ▶ `$_POST["choix"][1]` contient la valeur "DEUG"



## 7. Transfert des fichiers vers le serveur

---

- ▶ Les deux tableaux `$_GET` et `$_POST` permettent le transfert de données simples et ils sont incapable de transférer des documents ou des fichiers de taille importante.
- ▶ Pour permettre le **transfert de fichier** de grande taille, le langage PHP, depuis sa version 4.1, a fournit un autre tableau ***associatif et multidimensionnel*** qui est **`$_FILES`**. Si, par exemple le nom de l'élément est : `<input type="file" name="fichier"/>`, `$_FILES` permet récupérer les informations suivantes :
  - ▶ `$_FILES["fichier"]["name"]`: contient le nom de fichier
  - ▶ `$_FILES["fichier"]["type"]`: le type de fichier (exemple application/pdf)
  - ▶ `$_FILES["fichier"]["size"]`: la taille de fichier en nombre d'octet.
  - ▶ `$_FILES["fichier"]["tmp_name"]`: le nom temporaire sur le serveur.
  - ▶ `$_FILES["fichier"]["error"]` : pour contrôler s'il y avait des erreurs de transfert. La valeur 0 indique la bonne réception.
- ▶ Contrairement aux exemples précédents, l'élément `<form>` **doit avoir** l'attribut ***method*** à la valeur ***post*** et l'attribut ***enctype*** à la valeur ***multipart/form-data***.

- ▶ Consignes de sécurité: pour éviter des problèmes de sécurité voici quelques précautions à faire :
  - ▶ Définir le type MIME de fichiers acceptés, par exemple "image/jpg", "image/gif", ... ceci peut être réalisé grâce à l'attribut *accept* de l'élément `<input type="file" accept="type1, type2, type3">`. Exemple : `<input type="file" accept="image/jpeg, image/gif">`
  - ▶ Limiter la taille des fichiers à télécharger en ajoutant au formulaire un champs caché nommé *MAX\_FILE\_SIZE*. Cette valeur est récupérée dans la variable `$_POST["MAX_FILE_SIZE"]` lorsque le champ caché est défini par le code suivant : `<input type="hidden" name="MAX_FILE_SIZE" value="1000" />`
- ▶ Après avoir transféré le fichier sur le serveur, il est nécessaire de le déplacer dans le dossier cible. Cela peut se faire en utilisant la fonction `move_uploaded_file($tmp_name, $new_name)`.
- ▶ Il est recommandé de renommer le fichier et ne jamais garder son nom de départ, car deux visiteurs différents pourraient avoir un fichier du même nom.

.... la suite

## ► Exemple :

```
<form action="FormTraitement.php" method="post" enctype="multipart/form-data">
<fieldset>
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
<legend><b>Transfert de fichier</b></legend>
Fichier : <input type="file" name="fich" accept="image/jpg" size="100"/><br/>
Clic! <input type="submit" value="Envoi" /><br/>
</fieldset>
</form>
```

### Fichier *FormTraitement.php*

```
<?php
if(isset($_FILES['fich']))
{
echo "Taille maximale autorisée :",$_POST["MAX_FILE_SIZE"],"octets<br />";
echo "<b>Clés et valeurs du tableau $_FILES </b><br />";
foreach($_FILES["fich"] as $cle => $valeur)
{
echo "clé : $cle valeur : $valeur <br />";
}
//Enregistrement et renommage du fichier
$result=move_uploaded_file($_FILES["fich"]["tmp_name"],"imagephp.gif");
if($result==TRUE)
{echo "<br /><big>Le transfert est réalisé !</big>";}
else
{echo "<br /> Erreur de transfert n°",$_FILES["fich"]["error"];}
}
?>
```

### Affichage :

Taille maximale autorisée :100000 octets

#### Clés et valeurs du tableau \$\_FILES

clé : name valeur : ensah.jpg

clé : type valeur :

clé : tmp\_name valeur :

clé : error valeur : 2

clé : size valeur : 0

Erreur de transfert n°2

```
<?php
if(isset($_FILES["fich"])){
    $original_name=$_FILES["fich"]["name"];
    $ext=explode(".", $original_name)[1];
    $new_name=rand()).".$ext;
    $done=move_uploaded_file($_FILES["fich"]["tmp_name"], $new_name);
}
?>
```

## 8. Bons pratiques pour les formulaires

- ▶ Contrôler l'état des variables : lors de l'envoi de données d'un formulaire vers le serveur, il est recommandé de vérifier si chaque valeur envoyée depuis est bien reçue et elle est bien récupérée . Ceci est réalisé grâce à la fonction `isset()` .

**Exemple** : `<?php if(isset($_POST["prenom"])) echo $_POST["prenom"];?>`

- ▶ Maintenir l'état de formulaire :

- ▶ Lorsque le script contenant le formulaire est chargé du traitement des données, l'ensemble de la page est réaffiché après traitement, de même que l'ensemble du formulaire. Le formulaire se retrouve alors dans son état initial, et toutes les saisies effectuées sont effacées.
- ▶ En cas d'erreur de saisie sur un seul champ, l'utilisateur est obligé de recommencer l'ensemble de la saisie. Pour éviter cela, il faut affecter le contenu récupéré grâce aux tableaux *superglobaux* (exemple `$_POST`) à la valeur de l'élément de formulaire.
- ▶ **Exemple** : Pour l'élément suivant: `<input type="text" name="prenom"/>`, l'attribut `value` doit être affecté à `"<?php if(isset($_POST["prenom"])) echo $_POST["prenom"];?>"`. Comme ceci : `<input type="text" name="prenom" size="40" value="<?php if(isset($_POST["prenom"])) echo $_POST["prenom"];?>" />`

.... la suite

► Valider les éléments du formulaire:

- La validation des données de formulaire est très nécessaire pour protéger les entrées contre les pirates et les spammeurs!
- Par exemple : la variable \$\_SERVER ["PHP\_SELF"] peut être utilisée par des pirates! Si PHP\_SELF est utilisé dans une page, alors un utilisateur malveillant peut saisir tout un script dans l'URL.
- Pour éviter que \$\_SERVER ["PHP\_SELF"] soit exploité on peut utiliser la fonction *htmlspecialchars()*. Le code de formulaire doit ressembler à ceci:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

- Les zones de saisit d'un formulaire(text, textarea, ..) peuvent être aussi exploités par des pirates, c'est pourquoi il faut bien les contrôler grâce aux différentes fonctions suivantes:

```
function test_input($data) {  
    $data = htmlspecialchars($data);  
    $data = trim($data);  
    $data = stripslashes($data); // Supprime les antislashes d'une chaîne  
    return $data;  
}
```

## ► Un exemple complète :

```
<?php
$nom=isset($_POST["nom"])?$_POST["nom"]:"";
$prenom=isset($_POST["prenom"])?$_POST["prenom"]:"";
$genre=0;
if(isset($_POST["genre"])){$genre= ($_POST["genre"]=="F") ? 1:2;}
for($i=0;$i<4;$i++) $choix[$i]=isset($_POST["d"][$i])?1:0;
?>
<form action="#" method="post">
Nom : <input type="text" name="nom" value="<?php echo $nom;?>"> <br>
Prenom : <input type="text" name="prenom" value="<?php echo $prenom;?>">
<br>
Genre :
<input type="radio" name="genre" <?php if($genre==1) echo "checked";?>> F
<input type="radio" name="genre" <?php if($genre==2) echo "checked";?>> M
<br>
Diplome:
<input type="checkbox" name="d[0]" value="L" <?php if($choix[0]) echo "checked";?>>Licence
<input type="checkbox" name="d[1]" value="M" <?php if($choix[1]) echo "checked";?>>Master
<input type="checkbox" name="d[2]" value="I" <?php if($choix[2]) echo "checked";?>>Ingenieur
<input type="checkbox" name="d[3]" value="D" <?php if($choix[3]) echo "checked";?>>Doctorat
<br>
<input type="submit" value="envoyer" name="submit">
</form>
```

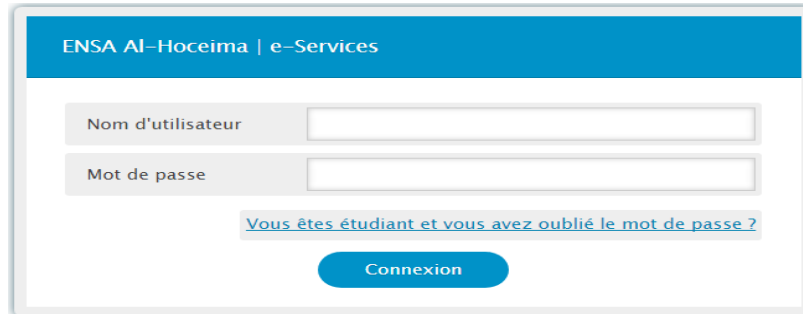
*L'exemple prend en considération le maintien en état de formulaire*

# V. Authentification, redirection et inclusion

## 1. Présentation

---

- ▶ Une application web est un ensemble de pages web créées pour fournir un service ou des services (Exemples: service de facturation, e-commerce, gestion scolarité, ...)
- ▶ Dans la plupart des cas, l'accès aux différentes pages d'une application web exige une authentification pour pouvoir utiliser ses différents services (Exemple de e-services de ENSAH)
- ▶ Une telle application utilise un script qui demande un login et un mot de passe pour qu'un visiteur puisse se « connecter » (s'authentifier) et ensuite rediriger vers ses services.



The image shows a login interface for ENSAH AI-Hoceima e-Services. It features a blue header bar with the text "ENSAH AI-Hoceima | e-Services". Below the header, there are two input fields: "Nom d'utilisateur" and "Mot de passe". A link "Vous êtes étudiant et vous avez oublié le mot de passe ?" is positioned below the password field. At the bottom, there is a blue button labeled "Connexion".

## 2. Authentification: un exemple basique

- Dans cet exemple on va créer un simple script qui va permettre de vérifier si le login et le mot de passe saisis correspondent bien à ceux qui sont prédéfinis à l'aide de *define()* : *define('LOGIN','etudiant');**define('PASSWORD','ensah');*



```
<form action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>" method="post">
  <legend>Identifiez-vous</legend>
  <label for="login">Login :</label>
  <input type="text" name="login" id="login" value="" />
  <label for="password">Password :</label>
  <input type="password" name="password" id="password" value="" />
  <input type="submit" name="submit" value="Se connecter" />
</form>
```

- Le script de vérification est donné comme suit :

```
<?php
$login=isset($_POST['login'])? $_POST['login']: '';
$password=isset($_POST['password'])? $_POST['password']: '';
if($login==LOGIN && $password==PASSWORD)
    echo "vous etes bien connecté $login";
else
    echo "mot de passe et/ou login sont incorrects";
?>
```

- Ce script permet tout simplement d'afficher « vous êtes bien connecté » si l'utilisateur a bien fait entrer comme login 'etudiant' et comme mot passe 'ensah'.



### 3. Redirection: header()

- ▶ Il existe des applications web pour lesquelles on souhaite rediriger le visiteur en fonction de ses paramètres. C'est le cas par exemple pour un script d'identification. Si l'internaute fournit les bons identifiants alors il est redirigé automatiquement vers son espace personnel, sinon il est renvoyé vers le formulaire d'authentification.
- ▶ Pour créer une redirection avec PHP, on utilise la fonction `header('Location: lien_vers_laPage')` permettant d'envoyer des entêtes de type *Location* (adresse).

- ▶ **Exemple :**

```
<?php
define('LOGIN','etud');
define('PASSWORD','test');
$login=isset($_POST['login'])? $_POST['login']: '';
$password=isset($_POST['password'])? $_POST['password']: '';
if($login=='')
    header('location: form.php?error=1');
elseif($password!='test')
    header('location: form.php?error=2&password='.$password);
else header('location: bienvenue.php');
```

- ▶ **NB :** l'appel de cette fonction doit se faire avant tout envoi au navigateur (instruction echo, print, balise html...) et doit être suivi de la fonction `exit();` pour arrêter l'exécution du script qui a déclenché la redirection. *Exemple :*  
`<?php header('Location: pageprotegee.php'); exit(); ?>`

## 4. Inclusion : `include()` et `require()`

---

- ▶ À la différence de la fonction de redirection, il existe des fonctions permettant **d'inclure du contenu sur la même page sans rédiger vers une autre**.
- ▶ Elles peuvent être utilisées pour créer du contenu tel qu'une bannière de page, un bloc d'informations ou un menu dans un fichier de script bien spécifique et de l'inclure en cas de besoin dans n'importe quelle page de l'application.
- ▶ Exemple : Selon l'utilisateur connecté, on peut inclure un contenu qui correspond à un tableau ou bien à une liste.
- ▶ L'avantage de ceci est lorsqu'on souhaite modifier le contenu, il suffit de l'effectuer dans un fichier unique; celle-ci sera ensuite répercutée sur chaque page dans laquelle apparaît le fichier d'inclusion.
- ▶ Il existe deux fonctions d'inclusion de base en PHP : `include()` et `require()`. Les deux se comportent de la même manière mais retournent des erreurs différentes. Si une fonction `include()` n'est pas analysée correctement, elle **continue le traitement du reste de la page** et affiche un avertissement dans la page où le fichier inclus doit apparaître. Si une fonction `require()` fait référence à un fichier manquant, **la fonction arrête le traitement de la page et affiche une page d'erreur dans le navigateur**.

- ▶ Les fonctions `include_once()` et `require_once()` spécifient qu'un fichier d'inclusion ne doit être utilisé qu'une seule fois dans une page. Si deux fonctions **`include()`** font référence au même fichier d'inclusion, seule la première fonction **`include()`** s'affiche dans le navigateur.
- ▶ Une instruction PHP d'inclusion est un bloc de code qui extrait le contenu d'un fichier externe dans une page Web. Voici un exemple d'instruction PHP d'inclusion : `<?php include('pageBanner.php'); ?>`
- ▶ Exemple: Pour l'exemple précédent, au lieu de déclarer des constantes directement dans le script consacré à l'authentification, on peut le faire dans un fichier séparé (exemple `definition.php`) et puis de l'inclure après.

```
<?php  
define('LOGIN','etud');  
define('PASSWORD','test');  
?>
```

```
<?php  
include 'definition.php';  
$login=isset($_POST['login'])? $_POST['login']:'';  
$password=isset($_POST['password'])? $_POST['password']:'';  
if($login=='')  
    header('location: form.php?error=1');  
elseif($password!='test')  
    header('location: form.php?error=2&password='.$password);  
else header('location: bienvenue.php');
```

# VI. Cookies et Sessions

## 1. Présentation

---

- ▶ Rappelons que le protocole HTTP est dit "stateless", ou autrement dit un protocole sans état. Pour un serveur web donné, **chaque requête** qu'il reçoit **est indépendante de la précédente**, ainsi que de la suivante.
- ▶ Lorsqu'un utilisateur demande une page, puis une autre, HTTP n'offre aucun moyen de préciser que ces deux requêtes émanent du même utilisateur.
- ▶ Ce processus est néanmoins gênant pour **une application Web**, car celle-ci a souvent besoin de **"se souvenir"** d'un (des) paramètre(s) d'un utilisateur **lors de passage d'une page à l'autre**.
- ▶ Pour surmonter ce problème **les solutions offertes par PHP** sont: **les cookies** et **les sessions**.
- ▶ Leur utilisation a pour but de pouvoir **garder en mémoire les informations d'un visiteur** afin de pouvoir **les utiliser tout au long de sa connexion**.
- ▶ Comme exemple d'informations : pseudonyme, mot de passe, Nom, Prénom, etc...

## 2. Cookie

---

- ▶ Un cookie est **un petit fichier** de taille limitée que **le serveur peut intégrer sur la machine client si ce dernier l'accepte**.
- ▶ Son utilisation a pour but de pouvoir **garder en mémoire les informations d'un visiteur afin de pouvoir les réutiliser à chacune de ses visites**. Par exemple on y stocke son Nom, son Prénom, son pseudo, etc...
- ▶ En rajoutant l'en-tête ***set-cookie***, dans sa réponse, le serveur indique au client qu'il souhaite y stocker un cookie. Le client peut décider, ou non, de créer le cookie demandé.
- ▶ S'il l'accepte, le navigateur client va ensuite **joindre le cookie à l'en-tête de toutes ses requêtes vers le même domaine**.
- ▶ Utiliser/ne pas utiliser:
  - ▶ Les cookies sont stockés sur le navigateur de l'utilisateur et envoyés au serveur à chaque requête. Ils peuvent être modifiés et consultés par l'utilisateur donc il est déconseillé de sauvegarder des informations sensibles dedans.
  - ▶ Les Cookies permettent de stocker les informations de manière plus durable même après la fermeture du navigateur.
  - ▶ Un cookie est cependant limité en taille, 4Ko maximum; et en nombre de cookies; mais ce chiffre est généralement très grand. Pour stocker une quantité importante d'informations, le cookie n'est pas la solution. Il sert dans une grande majorité des cas à identifier un utilisateur.

.... la suite

- Pour écrire un cookie, on utilise la fonction PHP `setcookie()`, dont la syntaxe est comme suit :

```
<?php setcookie(nom_cookie, valeur, temps, chemin, nom_domaine,
secure, true); ?>
```

On lui donne en général trois paramètres, dans l'ordre suivant :

- le nom du cookie (ex. : pseudo) ;
  - la valeur du cookie (ex. : M@teo21) ;
  - la date d'expiration du cookie, sous forme de timestamp (ex. : 1491933186) qui désigne le nombre de secondes écoulées depuis le 1<sup>er</sup> janvier 1970 à minuit UTC précise. Pour l'obtenir on utilise la fonction `time()`.
  - Le paramètre `secure` prend (true/false), true si le cookie doit être envoyé en utilisant https
- Cette fonction doit être appelée au début de script; avant tout code HTML (donc avant la balise `<!DOCTYPE>`).
  - Pour accéder au contenu d'un cookie, on utilise la variable `$_COOKIE`.
  - *Exemple :*

```
<?php
setcookie('login','ensah',time()+20); //sa durée est 20s
setcookie('statut','on',time()+3600); //1 heure
setcookie('role','admin',time()+365*24*3600); //une année
?>
```

```
<?php
echo $_COOKIE["login"];
echo $_COOKIE["statut"];
echo $_COOKIE["role"];
?>
```

### 3. Session

---

- ▶ Une session est un mécanisme technique permettant de **sauvegarder temporairement sur le serveur** des informations relatives à un internaute.
- ▶ Les informations seront quant à elle transférées de page en page **par le serveur** et non par le client comme c'est le cas des cookies.
- ▶ Contrairement aux Cookies, la session conserve les informations pendant quelques minutes. Cette durée dépend de la configuration du serveur mais est généralement fixée à 24 minutes par défaut. Le serveur crée des fichiers stockés dans un répertoire particulier.
- ▶ Les sessions sont particulièrement utilisées pour ce type d'applications :
  - ▶ Les espaces membres et accès sécurisés avec authentification.
  - ▶ Formulaires éclatés sur plusieurs pages.
  - ▶ Stockage d'informations relatives à la navigation de l'utilisateur (thème préféré, langues...).

## 4. Session: fonctionnement

---

- ▶ En PHP, chaque **session** est **caractérisée** par un **identifiant unique** (PHPSESSID) qui est un nombre aléatoire chiffré. Cet ID est généré par PHP lors de la création d'une session et enregistré sur la machine du client pendant toute la durée de la session. Selon la configuration du serveur, cet identifiant **peut être conservé sur la machine du client dans un cookie** (comportement par défaut) ou **être transmis via les URL**.
- ▶ Pour **créer une session**, PHP introduit la fonction ***session\_start()***. Une fois la session générée, une infinité de **variables de session peuvent être créées** grâce à **`$_SESSION['var'] = 'valeur';`**. Par exemple, on peut créer une variable `$_SESSION['role']` qui stockera le rôle du visiteur.
- ▶ Le serveur conserve les variables même lorsque la page PHP a fini d'être générée. Cela que, sur n'importe quelle page de l'application, **on peut les récupérer grâce à la variable superglobale `$_SESSION`**.
- ▶ **NB:** pour accéder à nouveau aux variables de session, la fonction ***session\_start()*** doit être appelée de nouveau.
- ▶ Lorsque le visiteur se déconnecte, les variables de la session doivent être vidées en utilisant par exemple la fonction ***unset()***. Aussi, il est possible de **détruire une session** en utilisant la fonction ***session\_destroy()***.



## 5. Session : exemple

```
<?php
session_start();

$_SESSION['Prenom']='Peter';
$_SESSION['Nom']='Griffin';
$_SESSION['Profession']='Professeur';
?>

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Page 1</title>
</head>
<body>
<p>
Tu es sur la page 1. Ton nom et prenom est :
<?php
echo $_SESSION['Nom'].' '. $_SESSION['Prenom'];
?>
<br>
<a href="page2.php"> Aller vers page 2</a> <br>
<a href="page3.php"> Aller vers page 3</a><br>
<a href="page4.php"> Aller vers page 4</a>
</p>
</body>
</html>
```

Tu es sur la page 1.  
Ton nom et prenom est : Griffin Peter  
[Aller vers page 2](#)  
[Aller vers page 3](#)  
[Aller vers page 4](#)

Tu es sur la page 2.  
Je souviens toujours de toi, ton nom et prenom est : Griffin Peter

Tu es sur la page 3.  
Je souviens toujours de toi, ton nom et prenom est : Griffin Peter

Tu es sur la page 4.  
Je souviens toujours de toi, ton nom et prenom est : Griffin Peter

```
<?php
session_start();

echo "Tu es sur la page 2.<br>";
echo "Je souviens toujours de toi, ton nom et prenom est: " . $_SESSION['Prenom'] . " " . $_SESSION['Nom'];
```

## 6. Session : des cas d'utilisation

---

- ▶ Cas d'une application web qui exige une authentification pour pouvoir utiliser ses différents services (Exemple de e-services de ENSAH)
- ▶ Une telle application utilise un script qui demande un login et un mot de passe pour qu'un visiteur puisse se « connecter » (s'authentifier). Une session doit être créée pour sauvegarder ces informations pour se souvenir de l'utilisateur sur toutes les pages du site.
- ▶ Puisque les informations (login, motpasse, ...) ne sont retenues qu'après avoir réussi l'authentification, la session sert à restreindre l'accès aux pages de l'application qu'aux visiteurs authentifiés. Cela permet de créer toute une zone d'administration sécurisée : si la variable de session login existe, on affiche le contenu, sinon on affiche une erreur.
- ▶ On se sert activement des sessions sur les sites de vente en ligne. Cela permet de gérer un « panier » : on retient les produits que commande le client quelle que soit la page où il est. Lorsqu'il valide sa commande, on récupère ces informations et... on le fait payer.

# VII. Programmation orientée objet en PHP

## 1. Introduction

---

- ▶ La programmation orientée objet (POO) a fait son apparition en PHP depuis sa version 4.
- ▶ Avec l'apparition de PHP5 beaucoup de fonctionnalités ont été ajoutées au langage pour la prise en charge de la POO comme elle se doit.
- ▶ La POO a pris sa place dans PHP surtout avec l'intégration de ses véritables concepts, comme l'héritage, les interfaces, les classes abstraites etc...
- ▶ Si la POO gagne du terrain c'est principalement grâce à ses nombreux avantages qu'on peut résumer dans ces points :
  - ▶ **Plus organisé:** Les objets étant indépendants les uns des autres (même s'ils peuvent interagir entre eux), le code source devient plus organisé et plus claire.
  - ▶ **Plus évolutif:** Un code organisé est plus facile à maintenir.
  - ▶ **Code réutilisable:** En POO le code peut être divisé en modules plus faciles à réutiliser, non pas seulement dans le projet courant, mais dans d'autres projets.
  - ▶ **Favorise le travail en équipe:** A l'inverse de la programmation procédurale, la POO favorise le travail en équipe en découpant les parties du projet faciles à regrouper après.
  - ▶ **Plus intuitive:** La POO permet de manipuler des objets qui peuvent interagir entre eux. Ce concept est plus proche du monde physique où nous vivons.

## 2. Syntaxe de déclaration d'une classe

---

- Le code suivant présente une manière de déclarer et de structurer une classe.

```
<?php
class NomDeClasse
{
    type_accessibilité $nom_variable; // Attributs
    const NOM_CONST; // Constantes
    // Constructeur
    public function __construct( ) {

    }

    // Méthodes
    type_accessibilité function nom_méthode1 ($arg1,...,$argN) {
    }

    // affectation
    type_accessibilité function nom_méthode2 ($arg) {
        $this-> nom_variable= $arg;
    }
}
```

### 3. Éléments de la classe

---

- ▶ **Propriétés de la classe** : une propriété d'une classe **se déclare comme étant une variable PHP**. Pour contrôler l'accès, elle peut être précédée de l'une des options d'accessibilité suivantes : *public*, *protected* et *private*.
- ▶ **Constructeur /destructeur** :
  - ▶ Le constructeur est une méthode particulière. PHP5 permet désormais de créer des constructeurs unifiés avec la méthode `__construct()` dont la syntaxe est la suivante: *public function \_\_construct(**divers \$arg1,...,argN**)*. Cette dernière est **appelée implicitement** à la création de l'objet (instanciation).
  - ▶ De même, PHP 5 donne la possibilité d'utiliser des destructeurs unifiés à l'aide de la fonction `__destruct()`, dont la syntaxe est la suivante : *void \_\_destruct()*
  - ▶ **NB** : en PHP, **la surcharge** de méthode/constructeur **n'est pas permet**. On ne peut définir qu'une seule et unique fois la même méthode.  
*Question* : quoi faire dans ce cas?

- ▶ **Création de méthodes** : pour la déclaration de méthodes, il suffit de faire précéder le mot-clé *function* à la visibilité de la méthode. Les types de visibilité des méthodes sont les mêmes que les attributs. **Exemple** : `public function nom_méthode($arg1,...,$argN) { }`
- ▶ **Pseudo-variable *\$this*** : pour pouvoir accéder aux méthodes et propriétés d'une classe, la POO en PHP **exige** de préfixer la variable d'instance par la pseudo-variable *\$this*, qui peut s'interpréter comme "moi-même", ou bien "l'objet courant". Exemple `$this->nom= $nom; .`
- ▶ **Propriétés et méthodes statiques** : il possible de définir des propriétés et des méthodes statiques grâce au mot-clé *static*. Pour y faire référence on utilise à la place de *\$this* une des syntaxes suivantes : *self::\$propriété* / *self::\$méthode* ou encore si la méthode est celle d'une autre classe : *nomclasse::\$propriété* / *nomclasse::\$méthode*. Exemple :

```
<?php
class Test{
    public static $var_test="php";
    static function fct(){
        return self::$var_test;
    }
}
echo Test::fct();
```

.... la suite

## ► Un exemple : la classe CompteBancaire

```
<?php
class CompteBancaire
{
    private $numero;
    private $solde;
    private $titulaire;
    public function __construct($numero, $solde, $titulaire)
    {
        $this->numero = $numero;
        $this->solde = $solde;
        $this->titulaire = $titulaire;
    }
    public function getNumero()
    {
        return $this->numero;
    }
    public function getSolde()
    {
        return $this->solde;
    }
    public function setSolde($solde)
    {
        $this->solde = $solde;
    }
    public function getTitulaire()
    {
        return $this->titulaire;
    }
    public function crediter($montant) {
        $this->solde += $montant;
    }
    public function __toString()
    {
        return "Le solde du compte de $this->titulaire,
        numero $this->numero est de $this->solde DH";
    }
}
?>
```

## 4. Utilisation de la classe

---

- ▶ Une classe est généralement stockée dans un fichier dédié, qui peut porter son nom. Elle peut être incluse donc dans un programme de la même manière qu'une bibliothèque en utilisant les fonctions suivantes :  
**include**('ma\_classe.php'); //ou **require**('ma\_classe.php'); //ou  
**require\_once**('ma\_classe.php');
- ▶ **Instanciation d'une classe** : on utilise le mot-clé *new* suivant du nom de la classe. Cette instruction appelle implicitement la méthode constructeur ( `__construct()` ) qui construit l'objet et le place en mémoire. Voici un exemple qui illustre une instance de la classe *CompteBancaire* et son utilisation.

```
<?php
    require_once('CompteBancaire.php');
    $compte1 = new CompteBancaire(20072333,1452, 'Peter');
    echo $compte1 ;
    $compte1->Crediter(12.5);
    echo $compte1;

?>
```



## 5. Héritage

---

### ► Rappels :

- L'héritage est l'un des grands principes de l'approche orientée objet, et PHP l'implémente dans son modèle objet.
- Lorsqu'une classe hérite d'une autre classe, elle peut utiliser les propriétés et les méthodes de la classe parente et ajouter ses propres propriétés et méthodes spécifiques.
- L'héritage permet de **réutiliser du code déjà existant** pour l'adapter à ses besoins. En d'autres termes, il permet, à partir d'une classe déjà existante, d'en créer une nouvelle qui reprendra ses caractéristiques ce qui permettra de les **adapter à d'autres besoins sans modifier la classe de base**.
- Il permet également de **spécialiser les classes**. Une classe enfant peut ajouter de nouvelles propriétés et méthodes spécifiques à son fonctionnement, tout en conservant les propriétés et les méthodes de la classe parente.
- En PHP, pour qu'une classe X puisse hériter d'une classe Y on utilise le mot-clé **extends**.

## ► Exemple : *CompteEpargne* hérite de *CompteBancaire*

```
<?php
require_once 'CompteBancaire.php';
class CompteEpargne extends CompteBancaire
{
    private $tauxInteret;

    public function __construct($numero, $solde, $titulaire, $tauxInteret)
    {
        parent::__construct($numero, $solde, $titulaire);
        $this->tauxInteret = $tauxInteret;
    }

    public function getTauxInteret()
    {
        return $this->tauxInteret;
    }

    public function calculerInterets($ajouterAuSolde = false)
    {
        $interets = $this->getSolde() * $this->tauxInteret;
        if ($ajouterAuSolde == true)
            $this->setSolde($this->getSolde() + $interets);
        return $interets;
    }

    public function __toString()
    {
        return parent::__toString() .
            '. Son taux d\'interet est de '. $this->tauxInteret * 100 . '%.';
    }
}
?>
```

### Son utilisation

```
<?php
require 'CompteBancaire.php';
require 'CompteEpargne.php';
$compteJean = new CompteBancaire(2700172, 150, "Jean");
echo $compteJean . '<br />';
$compteJean->crediter(100);
echo $compteJean . '<br />';

$comptePaul = new CompteEpargne(2700173, 200, "Paul", 0.05);
echo $comptePaul . '<br />';
echo 'Interets pour ce compte :'.
$comptePaul->calculerInterets(). ' DH <br/>';
$comptePaul->calculerInterets(true);
echo $comptePaul . '<br />';
?>
```

### Résultats :

Le solde du compte de Jean, numero 2700172 est de 150DH  
Le solde du compte de Jean, numero 2700172 est de 250DH  
Le solde du compte de Paul, numero 2700173 est de 200DH.  
Son taux d'interet est de 5%.  
Interets pour ce compte :10 DH  
Le solde du compte de Paul, numero 2700173 est de 210DH.  
Son taux d'interet est de 5%.

## 6. Interfaces

---

- POO en PHP donne la possibilité de créer des interfaces. Rappelons qu'une interface se comporte comme une **classe abstraite dont toutes les méthodes sont abstraites**.

***Question:** quelles sont les caractéristiques d'une classe abstraite?*

- Pour définir une interface, on utilise le mot-clé **interface** suivi du nom de celle-ci (avec une première lettre majuscule de préférence, tout comme pour une classe) suivi des accolades qui renfermeront les membres de l'interface comme ceci:

```
<?php
    interface Convertisseur {
        public function toDollar($c);
    }
```

- L'utilisation (l'<sup>?</sup>implémentation) d'une interface par une classe se fait grâce au mot-clé **implements**.

## 7. Traits

- ▶ Les **traits** ont été intégrés au PHP à partir de sa version 5.4. Ils permettent de minimiser le code en réutilisant des méthodes déjà déclarées sans être obligé d'hériter d'une classe entière.
- ▶ Un trait ne peut être ni instancié comme une classe ni implémenté comme une interface. Il s'agit d'un bloc de code qui sera réutilisé par une classe.
- ▶ On déclare un trait grâce au mot-clé **trait** suivi de son nom (capitalisé de préférence) puis des accolades qui contiennent le corps. Pour s'en servir par une classe on utilise le mot-clé **use** suivi de nom du trait. Exemple :

```
<?php
trait Notification{
    public function afficher(){
        echo "Alerte signalée!";
    }
}
```

```
<?php
class Operation{
    use Notification;
}
$op=new Operation();
$op->afficher(); // Affiche "Alerte Signalée!"
?>
```

- ▶ Le code du trait *Notification* a été réutilisé comme s'il était déclaré directement dans la classe *Operation*.

## 8. Auto chargement des classes

---

- ▶ Il est préférable de séparer les classes dans différents fichiers. Le problème c'est que l'on est obligé ensuite de faire beaucoup de *require* pour les charger.
- ▶ L'autoloading permet de remédier à ce problème en incluant les classes de façon automatique.
- ▶ Pour faire de loading automatique, la méthode `__autoload($nom_class)` peut être utilisée. Un exemple d'utilisation:

```
function __autoload($class_name){  
    require('myclasses/'. $class_name . '.php'); /* myclasses est un dossier où les classes sont stockées.*/  
}
```

- ▶ La méthode `__autoload($nom_class)` est officiellement **obsolète** depuis **PHP 7.2.0**.
- ▶ PHP propose une autre solution au développeur qui consiste à **définir ses propres méthodes** d'autoloading grâce à `spl_autoload_register()`.

.... la suite

- Un exemple de *spl\_autoload\_register*.

```
<?php
function loader($className){
    $path="mesclasses/".$className.".php";
    if(file_exists($path)){
        require_once $path;
    }
}
spl_autoload_register("loader");
```

### Utilisation

```
<?php
require_once 'autoloader.php';

$c=new CompteBancaire(1234,150,"Bob");
$cep=new CompteEpargne(1234,150,"Bob", 0.02);
```

# VIII.Accès à la BD avec PHP

## 1. Introduction

---

- ▶ Les sites Web dynamiques reposent généralement sur des bases de données qui sont interrogées par les scripts utilisés pour leur développement (comme PHP).
- ▶ Le contenu ainsi visible sur la page provient partiellement (voir entièrement) de la base de données. Donc, le fait de modifier les données stockées dans celle-ci revient à modifier le contenu du site Web.
- ▶ Le SGBD le plus connu pour accompagner le langage PHP est MySQL. Cependant, PHP peut aussi dialoguer avec plusieurs autres SGBD comme PostgreSQL, Oracle, SQL Server...
- ▶ MySQL est utilisable par d'autres langages que PHP, Java par exemple. Le couple PHP-MySQL est cependant le plus répandu sur le Web.
- ▶ Pour que la base de données soit accessible à partir des pages d'un site, il faut pouvoir l'utiliser par l'intermédiaire d'un script.
- ▶ Dans un tel script, il faut mettre d'abord les fonctions permettant de connecter à la BD et ensuite les différentes fonctions permettant de réaliser les opérations CRUD(Create, Read, Update, Delete)

## 2. Les étapes de connexion

---

- ▶ L'accès à une base MySQL et son utilisation, qu'il s'agisse d'insérer, de modifier ou de lire des données, suit les étapes ci-dessous :
  - ▶ Connexion au serveur MySQL.
  - ▶ Envoi de diverses requêtes SQL au serveur (insertion, lecture, suppression ou mise à jour des données).
  - ▶ Récupération du résultat d'une requête.
  - ▶ Fermeture de la connexion au serveur.
- ▶ Toute opération à réaliser sur une base nécessite d'envoyer au serveur une requête SQL rédigée à l'aide des commandes SQL.
- ▶ Les requêtes étant souvent longues, il est recommandé, à des fins de lisibilité du code, de les écrire dans une variable chaîne *\$requete* passée ensuite à la fonction.



### 3. API de connexion à la BD

---

- ▶ PHP propose plusieurs moyens de se connecter à une base de données MySQL.
  - ▶ L'extension ***mysql\_*** : ce sont des fonctions qui permettent d'accéder à une base de données MySQL et donc de communiquer avec MySQL. Leur nom commence toujours par ***mysql\_***. Toutefois, ces fonctions sont vieilles et il est recommandé de ne plus les utiliser aujourd'hui. Cette extension a été supprimée depuis PHP 7.
  - ▶ L'extension ***mysqli\_*** : ce sont des fonctions améliorées (improved) d'accès à MySQL. Elles proposent plus de fonctionnalités et sont plus à jour. Cette extension peut être utilisé en *mode procédural* ou en *mode objet*.
  - ▶ L'extension ***PDO*** : c'est un outil complet qui permet d'accéder à n'importe quel type de base de données. On peut donc l'utiliser pour se connecter aussi bien à MySQL que PostgreSQL ou Oracle.

## 4. L'extension mysqli: mode procédural

- ▶ La fonction essentielle permettant de se connecter à une BD est `mysqli_connect()`, dont la syntaxe est la suivante : *resource mysqli\_connect (string \$host, string \$user, string \$passwd, string \$ db)* : Cette fonction retourne un identifiant de connexion.
- ▶ Pour mettre fin à la connexion, on fait appel la fonction `mysqli_close()`, dont la syntaxe est la suivante : `mysqli_close($conn)`.
- ▶ Un exemple :

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$db="gestionpersonnes";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $db);

if (!$conn) die("Connection failed: ");
else echo "Connected successfully $username";

$request="SELECT * FROM personnes";

$resultat = mysqli_query($conn, $request);
$ligne=mysqli_fetch_assoc($resultat);

var_dump($ligne);
?>
```

## 5. L'extension mysqli: mode objet

---

- ▶ Cette possibilité est liée à l'utilisation de l'extension *mysqli* en mode objet. Ce dernier comprend les trois classes suivantes :
  - ▶ La classe *mysqli* qui permet de créer des objets de type `mysqli_object`.
  - ▶ La classe *mysqli\_result*, permettant de gérer les résultats d'une requête SQL effectuée par l'objet précédent.
  - ▶ La classe *mysqli\_stmt* qui représente une requête préparée. Il s'agit là d'une nouveauté par rapport à l'extension `mysql`.
- ▶ Pour se connecter à la BD, il faut créer un objet de classe `mysqli`. Un exemple :

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$db="gestionpersonnes";
// Create connection
$conn = new mysqli($servername, $username, $password,$db);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
```

## 6. L'extension PDO (PHP Data Object)

---

- ▶ PDO est **une extension** qui offre une couche d'abstraction de données introduite dans PHP5, ce qui signifie qu'elle n'est pas liée, comme les extensions mysql ou mysqli, mais **indépendante de la BD utilisée**. Grâce à elle, **le code devient portable très facilement**, c'est-à-dire qu'elle modifie uniquement la ligne de connexion, d'une base de données MySQL à SQLite, PostgreSQL, Oracle et d'autres encore par exemple.
- ▶ PDO offre donc aussi bien l'avantage de la portabilité, de la facilité d'utilisation que de la rapidité. L'extension PDO comprend les trois classes suivantes :
  - ▶ La classe **PDO**, qui permet de créer des objets représentant la connexion à la base et qui dispose des méthodes permettant de réaliser les fonctions essentielles, à savoir **l'envoi de requête, la création de requêtes préparées** et la **gestion des transactions**.
  - ▶ La classe **PDOStatement**, qui représente, par exemple, une requête préparée ou un résultat de requête SELECT. Ses méthodes permettent de gérer les requêtes préparées et de lire les résultats des requêtes.
  - ▶ La classe **PDOException** qui permet de gérer et **d'afficher des informations sur les erreurs à l'aide d'objets**.

.... la suite

- L'étape de connexion consiste à créer un objet de la classe PDO en utilisant le constructeur de la classe PDO dont les paramètres changent selon le serveur auquel on veut connecter. Par exemple :

- Pour MySQL : `$conn= new PDO("mysql:host=$servername;dbname=$db",$username,$password) ;`
- Pour SQLite : `$conn= new PDO("sqlite2:/chemin/rep/$db") ;`
- Pour PostgreSQL : `$conn= new PDO("pgsql:host=$servername port=5432 dbname=$db user=$username password=$password").`

- Exemple :

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$db="gestionpersonnes";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$db", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
}
catch(PDOException $e){ echo "Connection failed: " . $e->getMessage();}
?>
```

## 7. Séparation du fichier de connexion

- Pour éviter de réécrire les parties de code concernant la connexion et pour des raisons de sécurité, il est commandé de :

- Placer les valeurs des paramètres de connexion dans un fichier séparé, dont son nom prend comme extension « .inc.php ». Les différents paramètres sont à définir sous forme de constantes grâce à la fonction *define(paramètre, valeur)*. Un exemple : un fichier « myparam.inc.php »

```
<?php
define("MYHOST","localhost");
define("MYUSER","root");
define("MYPASS","");
?>
```

- Créer une fonction spécialisée, dans laquelle, on fait appel à la méthode permettant d'établir la connexion avec la BD. Une telle fonction va prendre deux paramètres, \$base, qui est le nom de la BD, et \$param, qui est le nom du fichier contenant les paramètres de connexion. Ce dernier doit être inclus au début du fichier de connexion grâce à *require\_once(\$param.".inc.php")*. De même le fichier de connexion prend comme extension « .inc.php ». Exemple : connexion.inc.php

```
<?php
function connexionMySQLi($param){
    include_once($param.".inc.php");
    $conn = new mysqli($servername, $username, $password, $db);
    return $conn;
}
?>
```

```
<?php
function connexionPDO($param){
    include_once($param.".inc.php");
    $mysql="mysql:host=$servername;dbname=$db";
    try{
        $conn=new PDO($mysql,$username,$password);
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        return $conn;
    }catch(PDOException $e){
        echo "Connection failed: " . $e->getMessage();
    }
}
```

## 8. Envoi de requêtes SQL au serveur

---

- ▶ Les différentes opérations à réaliser sur la BD impliquent l'envoi de requêtes SQL au serveur comme **INSERT**, **UPDATE** ou **DELETE**. Les méthodes permettant de le faire sont :
  - ▶ Pour l'extension *mysqli*:
    - ▶ Pour une seule requête, la méthode à utiliser est `query()` dont la syntaxe est : `divers $conn->query(string $requete )`. Elle retourne un jeu de résultats sous forme d'un objet.
    - ▶ Pour des requêtes multiples, la méthode à utiliser est : `$conn->multi_query($sql)`
    - ▶ Dans les deux cas, pour fermer la connexion on utilise la méthode `$conn->close()`;
  - ▶ Pour l'extension *PDO*, de même que l'extension *mysqli*, il est possible d'utiliser la méthode `query()`. Mais en plus, il possible d'utiliser une autre méthode qui est `exec()` des objets PDO dont la syntaxe est la suivante : `integer $conn->exec(string requete)`. Cette méthode permet de retourner uniquement le nombre de lignes affectées, elle peut être utilisée pour des requêtes d'insertion (INSERT), de modification (UPDATE) ou de suppression (DELETE). Pour mettre fin à la connexion voilà comment : `$conn=null`.

## ► Exemple (extension Mysqli) :

```
<?php
```

```
require 'connMysqli.inc.php';  
$conn=connexionMYSQLi('parametres');
```

← Connexion à la BD  
« gestionpersonnes »

```
// sql to create table  
$sqlCreate = "CREATE TABLE Personne (  
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
Nom VARCHAR(30) NOT NULL,  
Prenom VARCHAR(30) NOT NULL,  
Profession VARCHAR(50)  
)";  
$conn->query($sqlCreate);
```

← Création d'une  
table sur la BD

```
$sqlInsert = "INSERT INTO Personne (Nom, Prenom, Profession)  
VALUES ('Swanson', 'Joe', 'Journaliste')";  
$conn->query($sqlInsert);
```

← Insertion d'une  
personne dans la BD

```
$sqlInsertMulti = "INSERT INTO Personne (Nom, Prenom, Profession)  
VALUES ('Griffin', 'Peter', 'Professeur');";  
$sqlInsertMulti .= "INSERT INTO Personne (Nom, Prenom, Profession)  
VALUES ('Browen', 'Cleveland', 'Avocate');";  
$conn->multi_query($sqlInsertMulti);
```

← Insertion multiples  
de deux personnes  
dans la BD

```
$conn->close();  
?>
```



## ► Exemple (extension PDO)

```
<?php
require 'connPDO.inc.php';
$conn=connexionPDO('parametres');

$sqlCreate = "CREATE TABLE Personnes (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
Nom VARCHAR(30) NOT NULL,
Prenom VARCHAR(30) NOT NULL,
Profession VARCHAR(50)
)";
echo $conn->exec($sqlCreate);

$sqlInsert = "INSERT INTO Personnes (Nom, Prenom, Profession)
VALUES ('Swanson', 'Joe', 'Journaliste')";
$conn->exec($sqlInsert);

$sqlInsertMulti = "INSERT INTO Personnes (Nom, Prenom, Profession)
VALUES ('Griffin', 'Peter', 'Professeur');";
$sqlInsertMulti .= "INSERT INTO Personnes (Nom, Prenom, Profession)
VALUES ('Browen', 'Cleveland', 'Avocate');";
$conn->exec($sqlInsertMulti);

?>
```

## 9. Lecture du résultat de la requête *SELECT*

---

- ▶ Lorsqu'il s'agit de lire le résultat d'une requête contenant la commande *SELECT* :
  - ▶ dans le cas où l'extension utilisé est *mysqli*, la méthode *query()* retourne un objet de type *mysqli\_result*,
  - ▶ dans le cas où l'extension utilisé est *PDO*, la méthode *query()* retourne un objet de type *PDOStatement*,
- ▶ Les deux classes offrent une grande variété de méthodes permettant de récupérer des données sous des formes diverses, la plus courante étant un tableau.
  - ▶ Pour le cas de la classe *mysqli\_result*, la méthode la plus perfectionnée pour lire des données dans un tableau est *fetch\_array()*. Sa syntaxe est : *array \$result->fetch\_array (int type)*.
  - ▶ Pour le cas de la classe *PDOStatement*, la méthode la plus utilisée pour lire des données dans un tableau est *fetch()*, dont la syntaxe est : *array \$result->fetch(int type)*.
- ▶ Chacune des deux méthodes ne récupèrent qu'une ligne du tableau à la fois.

.... la suite

- ▶ Pour lire toutes les lignes du résultat, il faut écrire une boucle (while par exemple) qui effectue un nouvel appel de la méthode *fetch\_array()/fetch()* pour chaque ligne. Cette boucle teste s'il y a encore des lignes à lire.
- ▶ Les deux méthodes retournent un tableau:
  - ▶ **indiqué** pour *fetch\_array(MYSQLI\_NUM)* et *fetch(PDO::FETCH\_NUM)*.
  - ▶ **associatif** pour *fetch\_array(MYSQLI\_ASSOC)* et *fetch(PDO::FETCH\_ASSOC)*.
  - ▶ **mixte** pour *fetch\_array(MYSQLI\_BOTH)* et *fetch(PDO::FETCH\_BOTH)*.
- ▶ D'autres traitements peuvent être utilisés pour l'opération de récupération de données :
  - ▶ La méthode *fetch\_fields()* est utilisée pour récupérer les noms des colonnes de la table interrogée. Sa syntaxe est : *array \$result->fetch\_fields(void)*
  - ▶ Pour récupérer le nombre de lignes de tableau.
    - ▶ Dans le cas de l'extension mysqli, on utilise la propriété *num\_rows* de la façon suivante : *\$nbart=\$result->num\_rows;*
    - ▶ Dans le cas de l'extension PDO, on utilise la méthode *rowCount()* de la façon suivante : *\$nbart=\$result->rowCount();*
  - ▶ La méthode *fetch\_object()*: retourne la ligne courante d'un jeu de résultat sous forme d'objet. Sa syntaxe est : *object \$result->fetch\_object();*

.... la suite

## ► Exemple (extension mysqli) :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Lecture de la table Personne</title>
<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
</head>
<body>
<div class="container">
<?php
require 'connMysqli.inc.php';
$conn=connexionMYSQLi('parametres');

$requete="SELECT * FROM personnes";
$result=$conn->query($requete);
if(!$result)
{ echo "Lecture impossible"; }
else
{
    $nbart=$result->num_rows;
    echo "<h4> Il y a $nbart personnes dans la BD </h4>";
    echo '<table class="table table-condensed">';
    echo "<tr><th>ID </th> <th>Nom</th>";
    echo "<th>Prénom</th> <th>Profession</th></tr>";
    while($ligne=$result->fetch_array(MYSQLI_NUM))
    {
        echo "<tr>";
        foreach($ligne as $valeur)
        { echo "<td> $valeur </td>"; }
        echo "</tr>";
    }
    echo "</table>";
}
$result->close();
$conn->close();
?>
</div>
</body>
</html>
```

Il y a 3 personnes dans la BD

ID	Nom	Prénom	Profession
1	Griffin	Peter	Professeur
2	Swanson	Joe	Journaliste
3	Brown	Cleveland	Avocate

.... la suite

## ► Exemple (extension PDO) :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Lecture de la table Personne</title>
<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
</head>
<body>
<div class="container">
<?php
require 'connPDO.inc.php';
$conn=connexionPDO('parametres');
$requete="SELECT * FROM personnes";
$result=$conn->query($requete);
if(!$result)
{ echo "Lecture impossible"; }
else
{
    $nbart=$result->rowCount();
    echo "<h4> Il y a $nbart personnes dans la BD </h4>";
    echo '<table class="table table-condensed">';
    echo "<tr><th>ID </th> <th>Nom</th>";
    echo "<th>Prénom</th> <th>Profession</th></tr>";
    while($ligne=$result->fetch(PDO::FETCH_NUM))
    {
        echo "<tr>";
        foreach($ligne as $valeur)
        { echo "<td> $valeur </td>"; }
        echo "</tr>";
    }
    echo "</table>";
}
?>
</div>
</body>
</html>
```

Il y a 3 personnes dans la BD

ID	Nom	Prénom	Profession
1	Griffin	Peter	Professeur
2	Swanson	Joe	Journaliste
3	Brown	Cleveland	Avocate

## 10. Requête préparée

- ▶ Le principe des requêtes préparées est, comme son nom l'indique, de préparer les requêtes pour ensuite les utiliser.
- ▶ Il permet de préparer une requête SQL avant son exécution en remplaçant les valeurs des paramètres par des marqueurs de position. Cette technique permet d'améliorer la sécurité et les performances des requêtes SQL dans une application.
- ▶ **Des exemples :**

```
<?php
$requete = 'INSERT INTO contacts (contact_prenom, contact_nom)
          VALUES (:prenom, :nom)';
$requete_preparee = $pdo->prepare($requete);
$requete_preparee->execute($_POST);
```

```
$stmt = $pdo->prepare("INSERT INTO users (username, email) VALUES (:username, :email)");
$stmt->bindParam(':username', $username);
$stmt->bindParam(':email', $email);
$stmt->execute();
```

```
$PreQuery=$conn->prepare("SELECT * from user Where username = ?");
$result=$PreQuery->execute(array($_POST["login"]));
```

## 11. POO et BD: Un exemple

### ► classe *Personne*

```
<?php
class Personne
{
    private $Id;
    private $Nom;
    private $Prenom;
    private $Profession;
    public function __construct(array $donnees){
        $this->hydrate($donnees);
    }
    public function hydrate(array $donnees)
    { foreach ($donnees as $key => $value)
      {
          $method = 'set'.ucfirst($key);
          if(method_exists($this, $method)){ $this->$method($value); }
      }
    }
    public function getId() { return $this->Id; }
    public function getNom() { return $this->Nom; }
    public function getPrenom() { return $this->Prenom; }
    public function getProfession() { return $this->Profession; }

    public function setId($Id){ $this->Id = (int) $Id; }
    public function setNom($Nom)
    { if (is_string($Nom) && strlen($Nom) <= 30)
      { $this->Nom = $Nom; }
    }
    public function setPrenom($Prenom)
    { if (is_string($Prenom) && strlen($Prenom) <= 30)
      { $this->Prenom = $Prenom; }
    }
    public function setProfession($Profession)
    {
        if (is_string($Profession) && strlen($Profession) <= 100)
        { $this->Profession = $Profession; }
    }
}
```

.... la suite

## ► Classe *PersonneManger*

```
<?php
require_once('Personne.php');
class PersonneManager
{
    private $conn;
    public function __construct($conn)
    { $this->setDb($conn); }
    public function setDb(PDO $conn)
    { $this->conn = $conn; }
    public function add(Personne $perso)
    {
        $sqlInsert = "INSERT INTO Personne (Nom, Prenom, Profession)
        VALUES ('".$perso->getNom()."', '".$perso->getPrenom()."', '".$perso->getProfession()."')";
        return $this->conn->exec($sqlInsert);
    }
    public function deletePersonne($Id)
    {
        return $this->conn->exec('DELETE FROM Personne WHERE id = '.$Id);
    }
    public function getPersonne_ByID($Id)
    {
        $result = $this->conn->query('SELECT id, Nom, Prenom, Profession FROM Personne WHERE id = '.$Id);
        $donnees = $result->fetch(PDO::FETCH_ASSOC);
        return new Personne($donnees);
    }
    public function getPersonnes()
    {
        $persos = [];
        $result = $this->conn->query('SELECT Id, Nom, Prenom, Profession FROM Personne ORDER BY Nom');
        while ($donnees = $result->fetch(PDO::FETCH_ASSOC))
        {
            $persos[] = new Personne($donnees);
        }
        return $persos;
    }
    public function updatePersonne(Personne $perso)
    {
        $sqlupdate = "UPDATE Personne SET Nom='".$perso->getNom()."', Prenom='".$perso->getPrenom()."',
        Profession='".$perso->getProfession()." WHERE Id = '".$perso->getId()."';";
        return $this->conn->exec($sqlupdate);
    }
}
```



## ► Utilisation :

```
<?php
require('Autoloader.php');
Autoloader::register();

include("connexPDO.inc.php");
$conn=connPDO("gestionpersonnes","myparam");

$manager = new PersonneManager($conn);

$perso = new Personne([
    'id' => '4',
    'Nom' => 'Griffin',
    'Prenom' => 'Peter',
    'Profession' => 'Commercant'
]);
$manager->addPersonne($perso);

echo $manager->getPersonne_ByID(4);

$perso = new Personne([
    'id' => '4',
    'Nom' => 'Griffin',
    'Prenom' => 'Peter',
    'Profession' => 'Professeur'
]);
$manager->updatePersonne($perso);

echo '<br/>'. $manager->getPersonne_ByID(4);
$manager->deletePersonne(4);
```

# IX. L'architecture MVC

## 1. Principe de modèle MVC

---

- ▶ Le **MVC** (Modèle-Vue-Contrôleur) c'est un modèle d'architecture qui cherche à **séparer** nettement **les couches** **présentation**, **traitement** et **accès aux données**.
- ▶ Une application web respectant ce modèle sera architecturée de la façon suivante :
  - ▶ **Modèle** : Gère les données
  - ▶ **Vue** : Gère la présentation (UI)
  - ▶ **Contrôleur** : Agit
- ▶ Le pattern MVC permet de bien organiser le code source de l'application.
- ▶ Il existe de **nombreux Framework PHP basés sur cette architecture**.



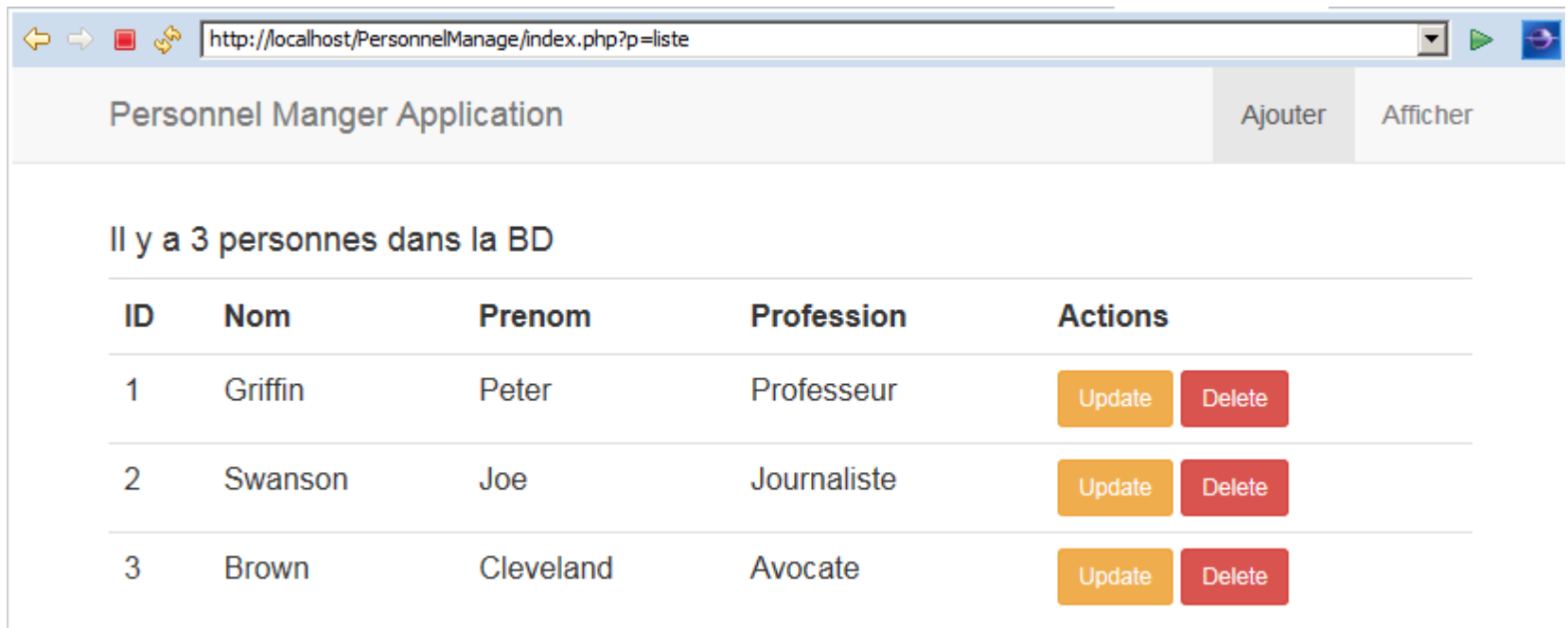
## 2. Description de MVC

---

- ▶ **Modèle** : cette partie *gère les données de site*. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc les requêtes SQL. Il est utilisé pour faire l'abstraction sous forme de classes les entités (les tables) de la bd. On peut utiliser un ORM (object-relational mapping) : doctrine, pdoMap, Propel, etc ...
- ▶ **Vue** : cette partie *se concentre sur l'affichage*. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples.
- ▶ **Contrôleur** : cette partie *gère la logique du code* qui prend des *décisions*. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).

### 3. MVC : concrétiser avec un exemple

- Une mini-application de gestion de personnes: Le but est de concevoir une toute petite application web permettant de réaliser les opérations CRUD(Create Read Update Delete) sur une liste de personnes constitués d'un nom, prénom et d'une profession.



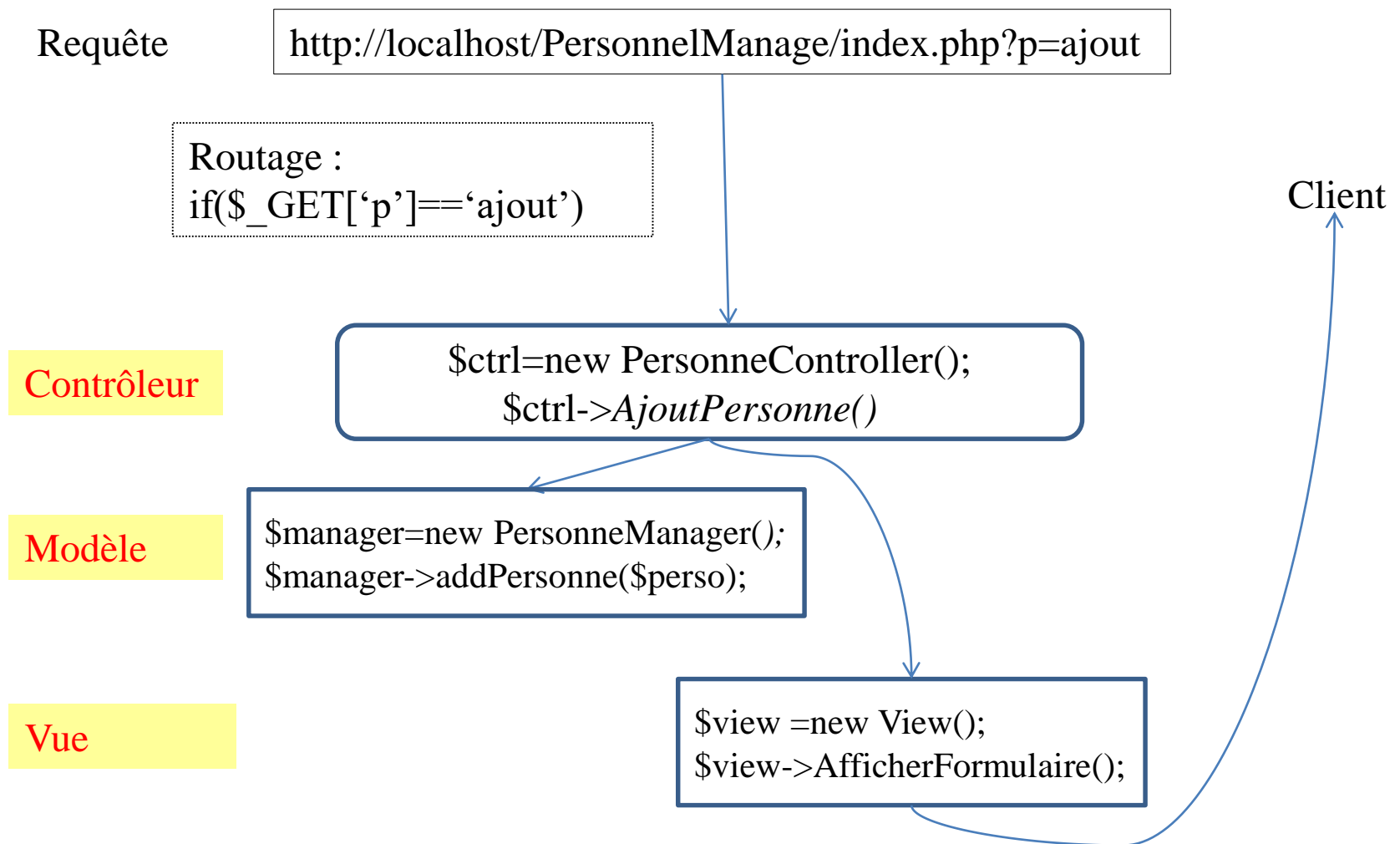
Personnel Manger Application

Ajouter Afficher

Il y a 3 personnes dans la BD

ID	Nom	Prenom	Profession	Actions
1	Griffin	Peter	Professeur	<button>Update</button> <button>Delete</button>
2	Swanson	Joe	Journaliste	<button>Update</button> <button>Delete</button>
3	Brown	Cleveland	Avocate	<button>Update</button> <button>Delete</button>

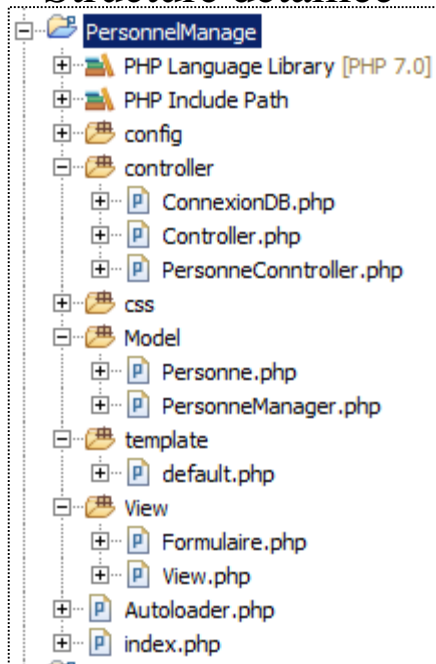
## ► Une requête et son cheminement :



.... la suite

## ► Quelques détails :

### Structure détaillée



### Script de routage

```
<?php

if (isset($_GET['p']))
{ $p=$_GET['p']; }
else{ $p='ajout'; }

switch ($p){
    case 'ajout':
        // ...
        break;
    case 'liste':
        //...
        break;
}
```

### Extrait de script de Controller

```
<?php

class PersonneController extends Controller{
    private $idconn;
    public function __construct($idconn){
        $this->idconn=$idconn;
    }
    public function AjoutPersonne(){
        View::DisplayFormulaire(null);
        $perso = new Personne($_POST);
        $manager=new PersonneManager($this->idconn);
        $manager->addPersonne($perso);
    }
    public function ShowPersonne(){
        //...
    }
    public function DeletePersonne($id){
        $manager=new PersonneManager($this->idconn);
        $manager->deletePersonne($id);
    }
    public function UpdatePersonne($id){
        //...
    }
}
```

## 4. Fichier .htaccess

---

- ▶ La réécriture d'URL est, comme son nom l'indique, un système permettant de changer l'apparence d'une URL en vue d'une utilisation optimale pour les moteurs de recherches ou bien tout simplement pour des raisons esthétiques.
- ▶ Au lieu de fournir une URL originale avec des paramètres sous forme de chaînes de requête comme "<http://.../index.php?rub=lireArticle&id=8>", on peut présenter aux visiteurs un lien avec une structure plus conviviale et compréhensible telle que "<http://.../articles/lire-8-titre-de-l-article.html>".
- ▶ Les fichiers *.htaccess* sont des fichiers de configuration d'Apache, permettant de définir des règles dans un répertoire et dans tous ses sous-répertoires (qui n'ont pas de tel fichier à l'intérieur). On peut les utiliser pour protéger un répertoire par mot de passe, ou pour changer le nom ou l'extension de la page index, ou encore pour interdire l'accès au répertoire.

## ► Un exemple (réécriture de l'url)

```
# On suit les liens symboliques
Options +FollowSymlinks
# Activation du mod rewrite d'Apache
RewriteEngine On
# Réécriture de index.html vers index.php
RewriteRule ^index\.html$ /index.php
# Réécriture des URL des articles
RewriteRule ^/articles/lire-([0-9]+)-([a-z0-9\-]+)\.html$
/index.php?rub=lireArticle&id=$1 [L]
```