



## Chapitre 4: PHP & MySql



**Avenue Abdou Maachar Al Balki B. P: 1317 Guelmim 81000**

**Année Universitaire: 2022/2023**

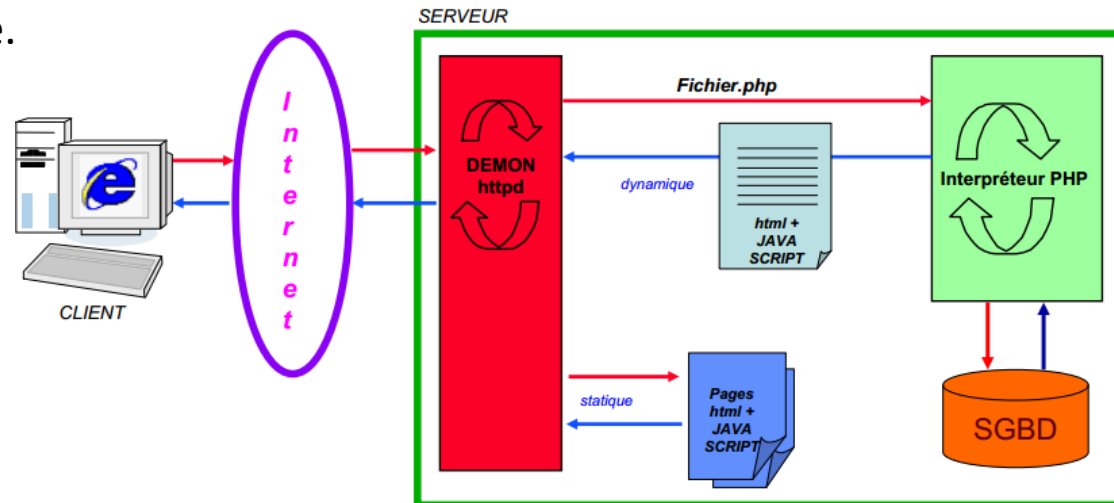
## *Plan de cours*

### **IV. PHP & MySql**

- 1) Introduction.**
- 2) Variables et constantes.**
- 3) Opérateurs.**
- 4) Structures de contrôles.**
- 5) Les boucles.**
- 6) Tableaux et tableaux associatifs.**
- 7) Les Fonctions.**
- 8) Gestion des fichiers.**
- 9) La Programmation Orienté Objet.**
- 10) La Programmation Modulaire.**
- 11) SGBD: Mysql.**
- 12) Exemple: Formulaire.**

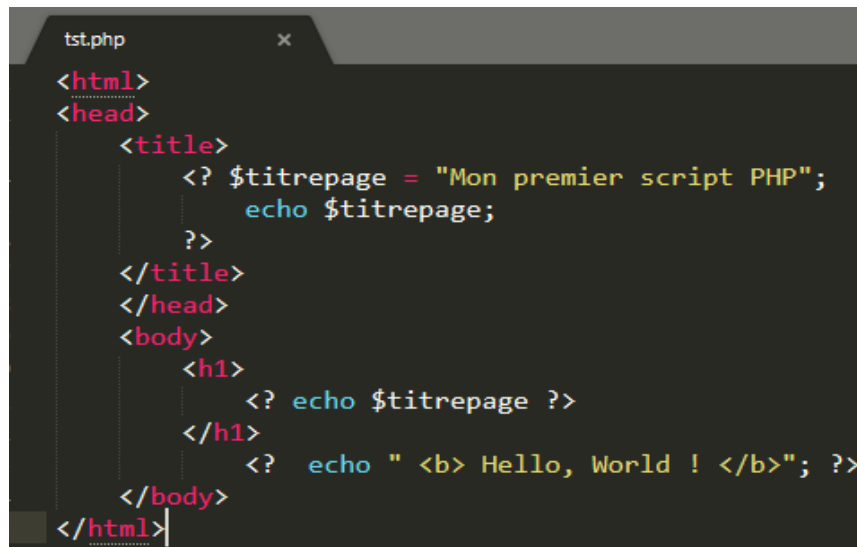
## Introduction

- Lorsqu'une requête HTTP est soumise au serveur Web pour une page dont l'extension est «.php», comme pour un fichier HTML, le serveur commence par rechercher dans son arborescence le fichier d'extension «.php».
- Il va ensuite passer la main à un sous-processus (une dll bien particulière) qui va interpréter le script PHP et produire dynamiquement du code HTML.
- Ce code HTML est alors envoyé à travers le réseau au navigateur client.
- De plus, aucune ligne de code PHP n'apparaît côté client dans la mesure où tout le code a été interprété.



## Introduction

- Un script PHP peut comprendre à la fois du code PHP et du code HTML, non interprété.
- On doit donc encadrer les parties comportant le code PHP entre 2 balises `<? et ?>`. Le reste de la page n'est pas interprété.
- Note: La balise `<?php` est équivalente à `<?>`. On peut également utiliser les balises `<script language="php">` et `</script>`.



```
tst.php
<html>
<head>
  <title>
    <? $titrepage = "Mon premier script PHP";
      echo $titrepage;
    ?>
  </title>
</head>
<body>
  <h1>
    <? echo $titrepage ?>
  </h1>
  <? echo " <b> Hello, World ! </b>"; ?>
</body>
</html>
```

## Introduction

- Le séparateur d'instructions est le ; est obligatoire, sauf si l'instruction est suivie de la balise ?>.
- La fonction **echo** affiche un (ou plus) argument. Si l'argument est une chaîne entre simple quote ' il est affiché tel quel.

```
echo 'Hello, World';
```

- Avec le quote double " les variables contenues dans cette chaîne sont interprétées.

```
$nom= "Toto";  
echo "Hello, $nom"; // Hello, Toto  
echo 'Hello, $nom'; // Hello, $nom
```

## Introduction

- On peut également inclure le résultat d'une fonction directement dans un echo.

```
echo "Votre Nom en majuscule : ", strtoupper( "Toto" ), "\n";
```

- La fonction `strtoupper` mets tous les caractères de la chaîne en majuscule.
- Pour afficher le caractère `"`, on l'insère à l'aide du caractère d'échappement `\`

```
echo " Escaping de caractères : \" \n";
```

- On peut inclure des caractères spéciaux pour contrôler le flux affiché :
  - `\n` saut de ligne
  - `\r` fin de ligne
  - `\t` tabulation
- Pour terminer l'exécution du script, on utilise la fonction `exit()`;

## Introduction

- Pour commenter le code, on utilise :
  - Commentaire sur une ligne: `//` ou `#`
  - Commentaire sur plusieurs lignes: `/* ... */`
- Utilisation en mode ligne de commande :
  - On peut exécuter un script PHP en ligne de commande, ce qui permet des usages hors du simple cadre "Web".
- L'option `-q` évite l'affichage de la première ligne **Content-type: text/html:**

**C:\WEB\PHP\> php -q monscript.PHP**

## Variables et constantes

### ■ Visibilité et affectation:

- PHP n'est pas un langage fortement structuré, il ne contient donc pas de partie déclarative clairement définie. Pour définir une variable, il suffit de l'initialiser.
- Les variables sont précédées du signe \$, quelque soit leur type. Ainsi pour déclarer une variable var :

**\$var=1;**

- La variable **\$var** est alors définie et vaut 1. Elle devient immédiatement accessible et ce jusqu'à la fin du script.



## Variables et constantes

### ■ Type de variables:

- Les variables PHP sont a typage faible. C'est PHP qui décide de son type lors de l'affectation. Il existe six types de données :
  - ✓ Entier (int, integer);
  - ✓ Décimal (real, float, double);
  - ✓ Chaîne de caractères (string);
  - ✓ Tableau (array);
  - ✓ Objet (object);
  - ✓ Booléen (boolean);
- Il est parfois utile de forcer le type d'une variable. On utilise la fonction **settype** ou bien les opérateurs de casting (**int**), (**string**) settype renvoie vrai si la conversion a fonctionné, faux sinon.

```
$a= 3.1415;
```

```
$result= settype( $a, "integer" ); // => $a = 3 , $result = 1
```

## Variables et constantes

### ■ Les opérateurs de conversion en PHP:

- ✓ (string) conversion en chaîne de caractères
- ✓ (int) conversion en entier, synonyme de (integer)
- ✓ (real) conversion en double, synonyme de (double) et (float)
- ✓ (array) conversion en tableau
- ✓ (object) conversion en objet
- ✓ (bool) conversion en booléen

**`$var= 1; // $var est de type "integer" et vaut 1.`**

**`$chn=(string) $var ; // $var est de type "string" et vaut " 1 ".`**

- On peut également utiliser `strval`, `intval`, `doubleval` qui renvoient la variable convertie en chaîne / entier / réel.

**`$strPI= "3.1415"; $intPI= intval( $strPI ); $PI= doubleval( $strPI );  
echo " $strPI / $intPI / $PI"; // => 3.1415 / 3 / 3.1415`**

- **Remarque** : Ces fonctions ne fonctionnent pas sur les tableaux.

## Variables et constantes

### ■ Règles des conversions implicites :

- Si la chaîne de caractères contient un point, un **e** ou un **E** ainsi que des caractères numériques, elle est convertie en décimal;
- Si la chaîne de caractères ne contient que des caractères numériques, elle est convertie en entier;
- Si la chaîne de caractères contient de chiffres et de lettres ou plusieurs mots, seul le premier est pris en compte et est converti selon les règles ci-dessus.
  - ✓ `$var1 = 1;` // `$var1` est de type "integer" et vaut 1.
  - ✓ `$var2 = 12.0;` // `$var2` est de type "double" et vaut 12.
  - ✓ `$var3 = "3 PHP";` // `$var3` est de type "string" et vaut " 3 PHP ".

## Variables et constantes

### ■ Références sur les variables :

- PHP permet d'exploiter les références aux variables.
- Une référence à une variable qu'a un accès à la zone mémoire et qui contient la valeur de cette variable.
- Cette référence est désignée par le caractère & placé devant le nom de la variable.
  - ✓ `$a = 1 ; // $a vaut 1.`
  - ✓ `$b = &$a ; // $b et $a pointent sur la même zone mémoire.`
- Ce sont donc deux noms pour la même variable.
  - ✓ `echo " $a, $b " ; // Affiche 1, 1`
  - ✓ `$a = 2 ;`
  - ✓ `echo " $a, $b " ; // Affiche 2, 2`

## Variables et constantes

### ■ Tests sur les variables :

- La fonction **isset** permet de tester si une variable est définie.
- La fonction **unset** permet de supprimer la variable, et de désallouer la mémoire utilisée.
  - ✓ `echo isset($a); // => 0 (faux)`
  - ✓ `$a= " ";`
  - ✓ `echo isset($a); // => 1 (vrai)`
  - ✓ `unset($a); // => 1 (vrai)`
  - ✓ `echo isset($a); // => 0 (faux)`
- La fonction **gettype** permet de connaître le type de la variable. Elle renvoie une chaîne : "string" ou "integer" ou "double" ou "array" ou "object".
  - ✓ `$a = 12; echo gettype($a) ; // => "integer"`
  - ✓ `$a= $a / 10; echo gettype($a) ; // => "double"`
  - ✓ `unset($a); echo gettype($a) ; // => "string"`

## Variables et constantes

### ■ Tests sur les variables :

- On peut également tester un type particulier à l'aide des fonctions **is\_array**, **is\_string**, **is\_int**, **is\_float**, **is\_object**.
  - ✓ `$a= 123;`
  - ✓ `echo is_int($a); // => (vrai)`
  - ✓ `echo is_double($a) // => (faux)`
  - ✓ `echo is_string($a) // => (faux)`
  - ✓ `$a += 0.5;`
  - ✓ `echo is_float($a) // => (vrai)`
- Remarque :
  - ✓ Les fonctions **is\_double** et **is\_real** sont équivalentes à **is\_float**.
  - ✓ Les fonctions **is\_long** et **is\_integer** sont équivalentes à **is\_int**.

## Opérateurs

- **Les Opérateurs en PHP** : PHP dispose des opérateurs classiques inspirés des langages C et Perl.

Comparaison	Description
==	égalité
>	Inférieur strict
<	Supérieur strict
<=	Inférieur ou égal
>=	Supérieur ou égal
!=	Inégalité

Logique	Description
&&	ET
	OU
xor	OU Exclusif
!	Négation

Binaires	Description
&	ET
	OU
^	XOR
~	NOT

- **Remarque** : les opérateurs **and**, **or** , **not** sont également disponibles et font la même chose.

## Opérateurs

- **Les Opérateurs en PHP** : PHP dispose des opérateurs classiques inspirés des langages C et Perl.

Arithmétiques	Description
+	égalité
-	Inférieur strict
/	Supérieur strict
*	Multiplication
%	Inférieur ou égal
++	Supérieur ou égal
--	Inégalité

Affectation	Description
=	Affectation
+=	Addition puis affectation
-=	Soustraction puis affectation
/=	Division puis affectation
%=	Modulo puis affectation
*=	Multiplication puis affectation

- **Remarque** : l'opérateur `/` renvoie un entier si les 2 opérandes sont des entiers, sinon il renvoie un flottant.



## Structures de contrôles.

### ■ Les tests IF:

Test if “basique”	Test if-else:	Test if-elseif:
<pre>if( [condition] ) { ... }</pre>	<pre>if( [condition] ) { ... } else { ... }</pre>	<pre>if( [condition] ) { ... } elseif( [condition] ) { ... }</pre>

- Dans le cas de plusieurs tests successif portant sur une Même variable, on utilisera plutôt le test **switch**.
- **Remarque** : Si le corps du test ne comporte qu'une instruction, les accolades { } sont optionnels, ( contrairement au Perl).

## Structures de contrôles.

- **Les testes SWITCH:** permet de confronter une variable à plusieurs valeurs prédéfinies.
- Il permet un code plus compact et lisible qu'un test : if-elseif-elseif...
- **Syntaxe :**

```
switch( [variable] ) {  
    case [valeur1] :  
        [bloc d'instructions]  
    break;  
    case [valeur2] :  
        [bloc d'instructions]  
    break;  
    ...  
    default:  
        [bloc d'instructions]  
}
```

## Les boucles

- En PHP, on dispose des structures de boucle similaires au langage C.
- L'instruction **break** permet de sortir d'une boucle à tout moment.
- L'instruction **continue** permet de revenir au début de la boucle.
- **Syntaxe** : la boucle for  
`for( [initialisations] ; [test sortie] ; [faire a chaque fois] ) {}`

```
for( $i=0; $i < sizeof($tableau); $i++ ) {  
    if( $tableau[$i] == 'suivant' ) {  
        continue;  
    }  
    if( $tableau[$i] == 'fin' ) {  
        break;  
    }  
    echo $tableau[$i], "\n";  
}
```

## Les boucles

- **Syntaxe** : la boucle while:

```
While([test sortie] ){ }
```

```
$i=0;
```

```
// parcours du tableau jusqu'au premier élément vide
```

```
while( isset( $tableau[$i] ) ) {
```

```
    echo "tableau[ $i ] = $tableau[$i] \n";
```

```
    ...
```

```
    $i++;
```

```
}
```

- **Syntaxe : la boucle do...while**: La condition de sortie est située en fin de boucle. Ainsi la boucle est parcourue une fois au minimum.

```
Do{ }While([test sortie] );
```

```
$i=1;
```

```
Do{
```

```
    echo $i;
```

```
}While($i<10);
```

## Tableaux et tableaux associatifs

### ■ Déclaration :

```
Nom_Tableau[]=array();
```

**Exemple:** \$fruits=array();

### ■ Affectations :

```
$fruits[0]= "pomme";
```

```
$fruits[1]= "banane";
```

```
$fruits[] .= "orange"; // équivaut a $fruits[2]= "orange"
```

```
$fruits= array( "pomme", "banane", "orange" );
```

## Tableaux et tableaux associatifs

### ■ Fonction relatives aux tableaux:

- **sizeof()** : Renvoie le nombre d'éléments d'un tableau.  
`$nbelements= sizeof( $tableau );`
- **is\_array()** : renvoie **true** si la variable est de type tableau (ou tableau associatif), **false** sinon.
- **Sort()** & **rsort()**: sont différentes fonctions de tri de tableau. **sort()** trie par valeurs croissantes, **rsort()** par valeurs décroissantes:  
`$tableau_trie = sort( $tableau );`

## Tableaux et tableaux associatifs

### ■ Les tableaux associatifs:

- Un tableau associatif est un tableau dont l'index est une chaîne de caractère au lieu d'un nombre.
- On parle aussi de "**hash array**" ou "**hash**". Il se déclare comme un tableau traditionnel, la distinction se fait lors de l'affectation.

### ■ Déclarations : `Nom_Tableau[]=array();` // comme un tableau

**Exemple:** `$calories=array();`

### ■ Affectations :

```
$calories["pommes"]= 300;  
$calories["banane"]= 130;  
$calories["litchie"]= 30;
```

## Tableaux et tableaux associatifs

### ■ Fonction relatives aux tableaux associatifs:

- **isset()** : pour tester l'existence d'un élément, on utilise la fonction **isset()**.

```
if( isset( $calories["pommes"] ) ) {  
    echo "une pomme contient ", $calories["pommes"] , " calories\n";  
} else {  
    echo "pas de calories définies pour la pomme\n";  
}
```

- **asort()**, **arsort()** : Ces fonctions de tri conservent la relation entre l'index et la valeur, généralement le cas dans un tableau associatif.
  - ✓ **asort()**: trie par valeurs croissantes;
  - ✓ **arsort()**: par valeurs décroissantes;



## Les Fonctions

### ■ Définition:

- A l'image de tout langage structuré, en PHP, une fonction est une suite d'instructions qui peut remplir n'importe quelle tâche.
- Il n'y a pas de distinction fonctions / procédures en PHP.
- Les fonctions PHP prennent de 0 à n paramètres. Ces paramètres peuvent être de type quelconque.

### ■ Syntaxe:

- La syntaxe de déclaration s'appuie sur le mot clé **function**.
- Ce mot clé est immédiatement suivi du nom de la fonction par lequel on va l'appeler depuis n'importe quel endroit du code PHP.
- puis des parenthèses destinées à accueillir les éventuels paramètres.

## Les Fonctions

### ■ Exemple:

```
function bonjour() {  
    echo " Bonjour ";  
}  
.....  
bonjour(); // Affiche " Bonjour " à l'écran
```

- **Retourner une valeur:** Les fonctions peuvent ou non renvoyer un résultat. on utilise l'instruction **return**. La variable retournée peut être de type quelconque. Elle est transmise par copie...

```
function bonjour() {  
    return " Bonjour ";  
}  
.....  
echo bonjour(); // Affiche " Bonjour " à l'écran
```

## Les Fonctions

### ■ Les variables global dans une fonction:

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

```
$debug_mode= 1; // variable globale
...
function mafonction(){
    global $debug_mode;
    if( $debug_mode )
        echo "[DEBUG] in function mafonction()";
    ...}
```

- Une autre solution est d'utiliser le tableau associatif **\$GLOBALS**, qui contient toutes les variables globales déclarées à un instant **T** :

**\$GLOBALS**['debug\_mode'] équivaut à **\$debug\_mode**.

## Les Fonctions

### ■ Le passage des paramètres par valeur:

- Afin de passer des paramètres à la fonction, il suffit de les insérer à l'intérieur des parenthèses prévues à cet effet.

```
function bonjour($prénom, $nom) {  
    $chaîne = " Bonjour $prénom $nom " ;  
    return $chaîne ;}
```

.....

```
echo bonjour("Pierre" , "PAUL") ; // Affiche " Bonjour Pierre PAUL "
```

## Les Fonctions

### ■ Le passage des paramètres par référence:

- Par défaut, les paramètres sont transmis **par copie**, c'est à dire que la fonction possède une copie locale de la variable envoyée.
- Avec la méthode du passage des paramètres par référence, on passe à la fonction l'adresse mémoire d'une variable existante. Cela se fait en précédant de **&** le nom du paramètre. Cela permet de modifier ce paramètre dans la fonction.

```
function bonjour(&$phrase, $prénom, $nom) {  
    $phrase = " Bonjour $prénom $nom " ;  
}  
.....  
$chaîne = " ";  
bonjour($chaîne, "Pierre" , "PAUL") ;  
echo $chaîne ;      // Affiche " Bonjour Pierre PAUL " à l'écran.
```

## Les Fonctions

### ■ Le passage des paramètres par défaut:

- Les paramètres optionnels sont autorisés : il suffit de leur affecter une valeur par défaut:

```
function mafonction( $param1 = "inconnu", $param2="") {  
    echo "param1=$param1 param2=$param2\n";  
}
```

....

```
mafonction( "toto", "titi" ); // => "param1=toto param2=titi"  
mafonction( "toto" ); // => "param1=toto param2="   
mafonction(); // => "param1=inconnu param2="
```

## Gestion des fichiers

- PHP fournit plusieurs fonctions qui permettent de prendre en charge l'accès au système de fichiers du système d'exploitation du serveur.
- Opérations élémentaires sur les fichiers en PHP :
  - **copy**(\$source, \$destination) Copie d'un fichier;
  - **\$fp=fopen**("filemane", \$mode) Ouvre un fichier et retourne un "id" de fichier;
  - **fclose**(\$fp) Ferme un fichier ouvert;
  - **rename**("ancien", "nouveau") Renomme un fichier;
  - **fwrite**(\$fp, \$str) Ecrit la chaîne de caractères \$str,
  - **fputs**(\$fp, \$str) Correspond à fwrite(),
  - **readfile**( "filename") Lit un fichier et retourne son contenu,
  - **fgets**(\$fp, \$maxlength) Lit une ligne d'un fichier,
  - **fread**(\$fp, \$length) Lit un nombre donné d'octets à partir d'un fichier.

## Gestion des fichiers

- L'accès à un fichier se fait toujours par un identificateur "**id**" de fichier.
- Cet "**id**" est créé avec la fonction **fopen()** et, est requis comme paramètre par la plupart des autres fonctions de fichiers en PHP.
- **Exemple:**

```
<?
$path="/usr/local/apache/htdocs/donnees.txt";
$mode="w";
if ($fp= fopen($path, $mode) ) {
    echo "Le fichier a été ouvert";
}
else
    echo "Fichier impossible à ouvrir";
if ( fclose($fp) )
    echo " et a été refermé";
?>
```



## La Programmation Orienté Objet

- PHP dispose des concepts de la Programmation Orientée Objet au travers des classes.
- Rappelons d'abord qu'un objet possède des attributs et des méthodes, et doit utiliser les mécanismes d'héritage et de polymorphisme.
  - **Attribut**: caractéristique d'un objet.
  - **Méthode**: action qui s'applique à un objet.
  - **Héritage**: définition d'un objet comme appartenant à la même famille qu'un autre objet plus général, dont il hérite des attributs et des méthodes.
  - **Polymorphisme**: capacité d'un ensemble d'objet à exécuter des méthodes de même nom, mais dont le comportement est propre à chacune des différentes versions.

## **La Programmation Orienté Objet**

### ■ **Les classes:**

- Une classe est la description sémantique et complète d'un objet.
- Elle comprend la déclaration des attributs ainsi que l'implémentation des méthodes de cet objet.
- La création d'un objet est déclenchée par celle d'une instance de la classe qui le décrit.

## La Programmation Orienté Objet

### ■ Les classes:

- **Déclaration:** La déclaration d'une classe s'appuie sur le mot clé `class`. La syntaxe est comparable à celle de la déclaration des fonctions.

```
class NOM_CLASSE {  
    //les attribut;  
    // les constructeurs;  
    // les méthodes;  
}
```

- **Les attributs:** les attributs présentes les caractéristique d'un objet. Ils sont déclarés avec le mot clé **var**.

```
class Véhicule{  
    var $nb_roues;  
    .....}
```

- Il n'y a pas de méthodes et attributs privés. Tout est public et accessible de l'extérieur.

## La Programmation Orienté Objet

### ■ Les classes:

- **Constructeur:** Le constructeur se déclare comme une méthode. Il doit porter le nom de la classe comme en C++ . Il est appelé automatiquement lors de l'instanciation de la classe.

```
class Véhicule{  
    var $nb_roues;  
    function Véhicule( $nb_roues ){  
        $this-> nb_roues= $nb_roues;  
    }  
}
```

- **Destructeur:** Il n'y a pas notion de destructeur d'objet en PHP.
- **Opérateur this->:** est l'opérateur de self-référence. Il point sur l'objet lui-même.

## La Programmation Orienté Objet

### ■ Les classes:

- **Affectation:** Pour exploiter les méthodes et les propriétés d'un objet, on utilise un accesseur dont la syntaxe est "->".

`$objet_test -> ma_méthode() ; // appelle la méthode`

`$objet_test -> ma_propriété ; // accède à la propriété`

- **Instanciation:** C'est le processus de création d'un objet à partir d'une classe en utilisant le mot clé **new**.

```
class Véhicule{  
    var $nb_roues;  
    function Véhicule( $nb_roues ){  
        $this-> nb_roues= $nb_roues; }  
    function NbRoues() {  
        return $this-> nb_roues;}  
    ...  
}  
$moto= new Véhicule( 2 );
```

## La Programmation Orienté Objet

### ■ Héritage:

- L'héritage simple est possible en utilisant **extends**.
- Le constructeur de la classe mère n'est pas appelé automatiquement. Il convient donc de le faire si nécessaire.
- L'héritage multiple n'existe pas !

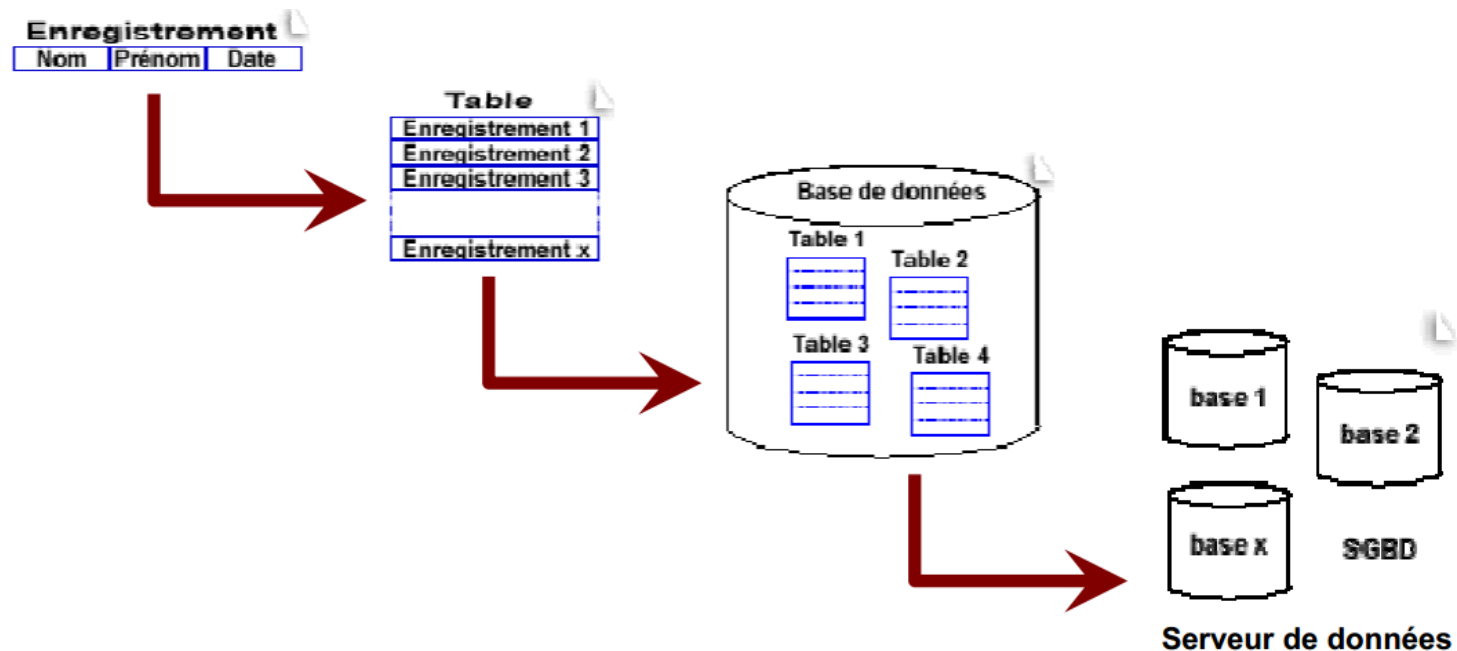
```
class Automobile extends Véhicule {  
    var $marque= "";  
        function Automobile( $marque, $nb_roues ) {  
            $this-> Véhicule( $nb_roues );//appel constructeur classe parente  
            $this-> marque= $marque; // set de la marque  
        }  
    }
```

## La Programmation Modulaire

- La programmation modulaire permet de la réutilisation de code, notamment par l'écriture de librairies.
- De ce fait, PHP permet cette modularité par la programmation de librairies classiques et de classes.
- On inclus un fichier en utilisant les deux instructions **include** ou **require**.
- Il existe une différence importante entre les deux :
  - Un fichier inclus par **include 'Nom\_Fichier'** est inclus dynamiquement, lors de l'exécution du code, c'est-à-dire qu'il est lu puis interprété.
  - Un fichier inclus par **require 'Nom\_Fichier'** est inclus avant l'interprétation du code. Il est équivalent à la directive **#include** du langage C.

## SGBD: Mysql

- Un **SGBD** est un ensemble d'applications permettant de manipuler les données (ajout, suppression, modification et lecture), mais aussi de contrôler l'accès. Les données sont structurées de la manière suivante :





## SGBD: Mysql

- En général, la communication entre un programme et une base de données suit le schéma suivant:

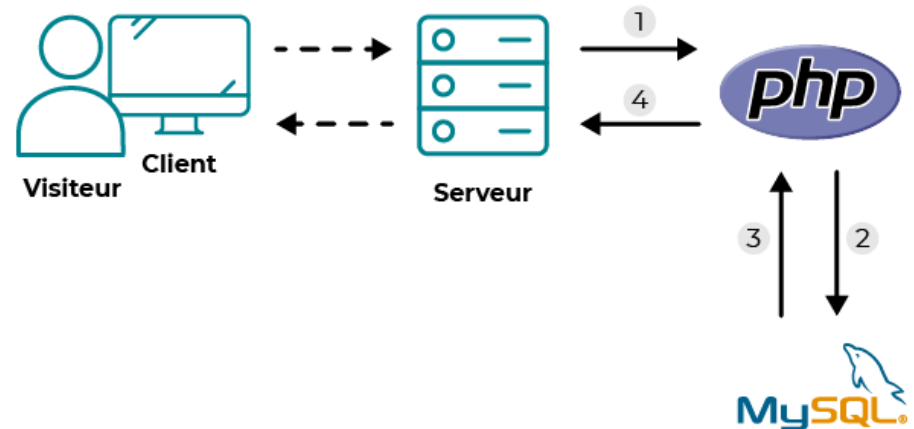


- En programmation PHP, il existe deux méthodes pour mettre en place cette architecture :
  - Accéder nativement à la base par l'intermédiaire de l'**API** de son middleware associé.
  - Passer par **ODBC (Open DataBase Connectivity)** qui est un protocole qui permet de se connecter à une base de données. L'avantage de **ODBC** est de proposer une **API** unifiée quelque soit le **SGBD** utilisé.
  - PHP gère en accès natifs de nombreux **SGBD** grâce à **ODBC** : **Oracle, MySQL,...**

## SGBD: Mysql

■ L'utilisation en général d'un SGBD tel que MySQL avec PHP s'effectue en 5 temps:

- 1) Connexion au serveur de données;
- 2) Sélection de la base de données;
- 3) Requête;
- 4) Exploitation des requêtes;
- 5) Fermeture de la connexion;



## SGBD: Mysql

1. **Connexion au serveur de données:** Pour se connecter au serveur de données, il existe deux méthodes:
  - Ouverture d'une connexion simple avec la fonction **mysql\_connect**
  - Ouverture d'une connexion persistante avec la fonction **mysql\_pconnect**
  - la deuxième méthode diffère de la première par le fait que la connexion reste active après la fin du script.

```
<?
```

```
if( mysql_connect("ma_base" , $login , $password ) >0 )
```

```
    echo "Connexion réussie ! " ;
```

```
else
```

```
    echo "Connexion impossible ! " ;
```

```
?>
```

## SGBD: Mysql

2. **Sélection de la base de données:** Pour faire cette sélection, utilisez la fonction **mysql\_select\_db** et vous lui passez en paramètre, le nom de votre base.

```
<?
```

```
if( mysql_select_db("ma_base" ) == True )
```

```
    echo "Sélection de la base réussie" ;
```

```
else
```

```
    echo "Sélection de la base impossible" ;
```

```
?>
```

- Les étapes sélection et requête peuvent être faites en même temps.
- Il est plus simple surtout pour une seule base, de sélectionner la table avant de commencer les requêtes.
- Ainsi, toutes les requêtes à venir utiliseront cette base par défaut.

## SGBD: Mysql

3. **Envoi d'une requête:** Pour envoyer ces requêtes, on peut utiliser deux fonctions :

- **mysql\_query** dans le cas où la base de données serait déjà sélectionnée.
- **mysql\_db\_query** dans le cas où l'on voudrait sélectionner la base en même temps.

```
<?
$reqête = "SELECT * FROM membres WHERE pseudo = 'président' ";
$résultat = mysql_query( $reqête );
?>
```

```
<?
$nameBd="ESTG-BD";
$reqête = "SELECT * FROM membres WHERE pseudo = 'président' ";
$résultat = mysql_query($nameBd , $reqête );
?>
```

## SGBD: Mysql

### 4. Exploitation des requêtes:

- Après l'exécution d'une requête de sélection, les données ne sont pas "affichées", elles sont simplement mises en mémoire.
- Il faut les chercher, enregistrement par enregistrement, et les afficher avec un minimum de traitement.
- PHP gère un pointeur de résultat, et l'enregistrement pointé qui sera retourné.
- Lorsque vous utilisez une fonction de lecture, le pointeur est déplacé sur l'enregistrement suivant et ainsi de suite jusqu'à ce qu'il n'y en ait plus.
- Les fonctions qui retournent un enregistrement sont : **mysql\_fetch\_row**, **mysql\_fetch\_array** et **mysql\_fetch\_object** et prennent comme paramètre l'identifiant de la requête.

## SGBD: Mysql

## 4. Exploitation des requêtes:

**Exemple:** "SELECT nom, prénom, date FROM membres"

- **mysql\_fetch\_row** : Cette fonction retourne un enregistrement sous la forme d'un tableau simple.

<?

```
$enregistrement = mysql_fetch_row ($résultat);
```

```
// Affiche le champ - nom -
```

```
echo $enregistrement[0] . "<br>";
```

```
// Affiche le champ - prénom -
```

```
echo $enregistrement[1] . "<br> ";
```

```
// Affiche le champ - date -
```

```
echo $enregistrement[2] . "<br> ";
```

?>

## SGBD: Mysql

## 4. Exploitation des requêtes:

**Exemple:** "SELECT nom, prénom, date FROM membres."

- **mysql\_fetch\_array** : Cette fonction retourne un enregistrement sous la forme d'un tableau associatif.

<?

```
$enregistrement = mysql_fetch_array ($résultat);
```

```
// Affiche le champ - nom -
```

```
echo $enregistrement["prénom"]. "<br>";
```

```
// Affiche le champ - prénom -
```

```
echo $enregistrement["nom"]. "<br> ";
```

```
// Affiche le champ - date -
```

```
echo $enregistrement["date"]. "<br> ";
```

?>



## SGBD: Mysql

## 4. Exploitation des requêtes:

**Exemple:** "SELECT nom, prénom, date FROM membres."

- **mysql\_fetch\_object:** Cette fonction retourne un enregistrement sous forme d'une structure (objet).

<?

```
$enregistrement = mysql_fetch_object ($résultat);
```

```
// Affiche le champ - nom -
```

```
echo $enregistrement->date."<br>";
```

```
// Affiche le champ - prénom -
```

```
echo $enregistrement->nom."<br> ";
```

```
// Affiche le champ - date -
```

```
echo $enregistrement->prénom."<br> ";
```

?>

## SGBD: Mysql

## 4. Exploitation des requêtes:

Exemple: "SELECT nom, prénom, date FROM membres."

➤ **mysql\_num\_rows:**

- ✓ Si il n'y a pas ou plus d'enregistrement à lire, ces fonctions retournent "false."
- ✓ Pour savoir combien d'enregistrements ont été retournés par la sélection, la commande **mysql\_num\_rows** prend comme paramètre l'identifiant de la requête.

&lt;?

```
echo "Il y a " . mysql_num_rows( $résultat ) . " membre(s) ";  
while( $enregistrement = mysql_fetch_array( $résultat )){  
    echo $enregistrement['nom'] . " " . $enregistrement['prénom'];  
    echo " – " . $enregistrement['date'] . "<br>" ;  
}
```

?&gt;

## SGBD: Mysql

## 5. Fermeture de la connexion:

- Vous pouvez fermer la connexion au moyen de la fonction **mysql\_close**.
- Il est bon de savoir que cette opération sera faite lorsque le script se terminera. C'est donc une opération facultative.

## ■ Gestion des erreurs:

- S'il y a une erreur dans la syntaxe de la requête, on utilise la fonction **mysql\_error** qui ne prend pas de paramètres.

```
<?
    $résultat = mysql_query( $requête )
    echo ("Erreur dans la requête : " . $requête . "<br>Avec l'erreur : ".
        mysql_error());
?>
```

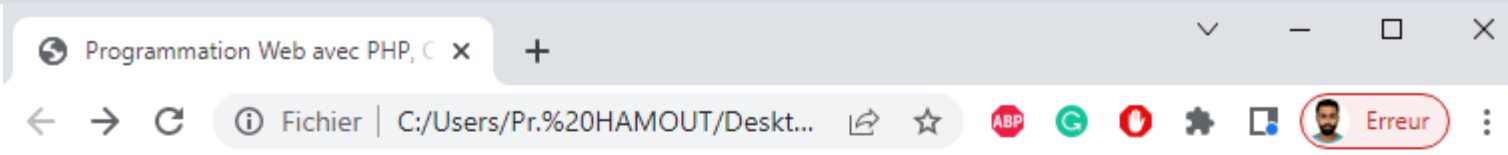
## Exemple: Formulaire

```
Formulaire_HTML_PHP.html x
<!DOCTYPE HTML>
<HTML>
  <HEAD>
    <TITLE>
      Programmation Web avec PHP, Code V-1: Mise en oeuvre d'un test (IF)
    </TITLE>
  </HEAD>
  <BODY>
    <H1>Exemple de test (IF)</H1>
    <i>Saisir les nombres entiers à tester :</i><br>
    <FORM action="formifres.php" method=GET>
      <b>
        <input type=text size=3 name="a"> est-il compris entre
        <input type=text size=3 name="b"> et <input type=text size=3 name="c">?
      </b>
      <br><br>
      <input type=submit value="Test">
    </FORM>
    <BR><HR><P><A href="menu.php">Retourner au menu principal</A></P>
  </BODY>
</HTML>
```

## Exemple: Formulaire

```
reception.php x
<HTML>
  <HEAD>
    <TITLE>
      Programmation Web avec PHP, CodeV-2: Mise en oeuvre d'un test (IF), résultat.
    </TITLE>
  </HEAD>
  <BODY>
    <H1>Résultat de l'exemple de test (IF)</H1>
    <?
      $ai = intval($a);
      $bi = intval($b);
      $ci = intval($c);
      if ($ci < $bi) {
        $tmp = $ci;
        $ci = $bi;
        $bi = $tmp;
      }
      echo "<font color=\"008800\"><b><i>$ai est-il compris entre $bi et $ci ?<br></i>";
      echo "</font><font color=\"ff0000\">";
      if ($ai < $bi) {
        echo "Non, $ai est inférieur à $bi " ;
      } elseif ($ai > $ci) {
        echo "Non, $ai est supérieur à $ci " ;
      } else { echo "Oui, $ai est compris entre $bi et $ci " ; }
    ?>
    <BR><HR><P><A href="menu.php">Retourner au menu principal</A></P>
  </BODY>
</HTML>
```

## Exemple: Formulaire



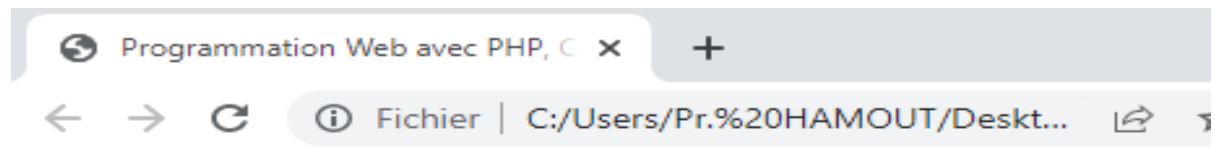
### Exemple de test (IF)

Saisir les nombres entiers à tester :

est-il compris entre  et

Test

[Retourner au menu principal](#)



### Résultat de l'exemple de test (IF)

6 est-il compris entre 4 et 8 ?

6 est-il compris entre 4 et 8 ?

[Retourner au menu principal](#)

**FIN.**

**MERCI DE VOTRE ATTENTION**

**DES QUESTIONS ?**