Chapitre I: Initiation à PHP

Syntaxe, Variables, operateurs, Structure conditionnel, structure répétitive, PHP et l'orientée objet

MODULE 1 : Langage et Syntaxe

Qu’est-ce que PHP ?

- PHP est un langage de script qui s’exécute côté serveur, le code PHP étant inclus dans une page HTML classique. Il peut donc être comparé à d’autres langages de script qui fonctionnent sur le même principe : ASP (*Active Server Pages*), JSP (*Java Server Pages*) ou PL/SQL Server Pages (PSP).
- À la différence d’un langage comme le JavaScript, où le code est exécuté côté client (dans le navigateur), le code PHP est exécuté côté serveur. Le résultat de cette exécution est intégré dans la page HTML qui est envoyée au navigateur. Ce dernier n’a aucune connaissance de l’existence du traitement qui s’est déroulé sur le serveur.

I: Syntaxe d'un programme basique en PHP

```
<html>
    <head>
    <title>Affichage Date</title>
    </head>
    <body>
        <?php
            /* ……*/
            echo Date("d/m/y");
        ?>
    </body>
</html>
```

le code PHP est inclus dans une page HTML à l'intérieur de balises (aussi appelées tags).
En version 7 ou plus, PHP accepte deux syntaxes pour les balises :

- `<?php ... ?>`
- `<? ... ?>`

II: Structure de données en PHP

PHP n'est pas un langage déclaratif, les variables ne peuvent pas être déclarées par l'utilisateur, cependant suivant les valeurs prises par la variable PHP va lui attribuer un type.

Ce type peut éventuellement être changé par l'utilisateur, comme nous le verrons plus loin.

Les types reconnus et associés aux variables par PHP sont les suivants :

- Le type chaîne de caractères (texte)
- Le type entier (numérique)
- Le type virgule flottante (réel, numérique)
- Le type booléen (Vrai ou Faux)
- Le type tableau
- Le type objet
- Le type ressource associé à des opérations système telles que connexion à une base de données, ouverture d'un flux de fichier.

III: Manipulation des variable en php

1- Nomenclature

Ainsi, pour déclarer une variable en PHP, il faut préciser le signe $ (dollar) avant chaque nom de variable. Le nom ne doit pas contenir d'espace, ni de caractère spécial (-, !, %, etc.) hormis le _ (underscore). De plus, une variable peut seulement commencer par une lettre ou un underscore. Les chiffres sont autorisés mais pas en premier caractère.

```php
<?php
    $test = 1;
    $_test = 2;
    $test_1 = 3;
?>
```

```php
<?php
    $test = 1;
    $TEST = 2;
    $Test = 3;
?>
```

2- Affectation

```php
<?php
        $Institut = "ISIMA ";
        $score = 170;
?>
```

3- Afficher le contenu d'une variable

```php
<?php
    echo $Institut;
?>
```

4- Concaténer deux variables

```php
<?php
        $ch1 = " Hello ";
        $ch2 = " freind ";
        $ch3 = $ch1.$ch2;
        echo $ch3;
?>
```

Résultat

Hello freind

```php
<?php
        $ch1 = "Je suis ";
        $ch2 = " un ";
        $ch3 = "développeur ";
        echo $ch1.$ch2. " future ".$ch3;
?>
```

Résultat

Je suis un future développeur

4- Les opérateurs arithmétiques (+,-,*,/)

```php
<?php
        $a = 1;
        $b = 3;
        $a = $a + 2;
        $a = $a * $b;
        $a = $a - 1;
        $a = $a / 4;
        echo $a;
?>
```

Résultat

2

5- Incrémenter / Décrémenter une variable

++ : Incrémenter
-- : Décrémenter

```php
<?php
$a = 9;
$b = 2;
$a++;
$b--;
echo ' La variable a = ' . $a . ' et b = ' . $b . "\n";
echo " La variable a = $a et b = $b \n" ;
```

Résultat

La variable a = 10 et b = 1
La variable a = 10 et b = 1

IV: Structures conditionnelles

1- L'instruction IF

```php
<?php
$a = 2;
if($a > 1)
echo 'La variable a est plus grand que 1.'; ?>
```

Résultat

```
La variable a est plus grand que 1.
```

```php
<?php
$a = 2;
if ($a > 1) {
echo "La variable a est plus grand que 1 \n";
$b = 0;
if ($b < 2) {
echo "La variable b est plus petite que 2 \n";
}
}
?>
```

Résultat

```
La variable a est plus grand que 1
La variable b est plus petite que 2
```

2- L'instruction ELSE

```
<?php
$a = 2;
if ($a > 1) {
   echo ' La variable $a est plus grand que 1. ';
} else {
   echo ' La variable $a est plus petit ou égale à 1. ';
}
 ?>
```

Résultat

La variable $a est plus grand que 1.

3- L'instruction ELSEIF

```
<?php
$a = 5;
if ($a < 1) {
echo 'La variable $a est plus petit que 1. ';
} elseif ($a > 5) {
echo ' La variable $a est plus grand que 5. ';
} else {
echo ' La variable $a est entre 1 et 5. ';
}
?>
```

Résultat

La variable $a est entre 1 et 5.

V: Les opérateurs de comparaison

Opérateur	Description	Exemple
==	`true` si la valeur est égale à	`$a == $b`
!=	`true` si la valeur est différente de	`$a != $b`
>	`true` si la valeur est strictement supérieure à	`$a > $b`
<	`true` si la valeur est strictement inférieure à	`$a < $b`
>=	`true` si la valeur est supérieure ou égale à	`$a >= $b`
<=	`true` si la valeur est inférieure ou égale à	`$a <= $b`

VI: Les boucles

1- Boucle WHILE

```
<?php
$z = 0;
while ($z < 5) {
echo 'La variable z a ' . $z . ' comme valeur.' . "\n";
$z++;
}
?>
```

Résultat

```
La variable z a 0 comme valeur.
La variable z a 1 comme valeur.
La variable z a 2 comme valeur.
La variable z a 3 comme valeur.
La variable z a 4 comme valeur.
```

2- Boucle DO..WHILE

```
<?php
$z = 4;
do {
echo 'La variable $z a ' . $z . ' comme valeur. '."\n";
$z--;
}while ($z >0);
?>
```

La différence est que la boucle do...while commence par exécuter sa portion de code avant de vérifier sa condition alors que la boucle while fait l'inverse.

3- Boucle FOR

```php
<?php
for ($z = 0; $z < 5; $z++) {
   echo 'La variable $z a ' . $z . ' comme valeur' . "\n";
}
?>
```

Résultat

```
La variable $z a 0 comme valeur.
La variable $z a 1 comme valeur.
La variable $z a 2 comme valeur.
La variable $z a 3 comme valeur.
La variable $z a 4 comme valeur.
```

4- Boucle FOREACH

```php
<?php
$villes = array('Tunis', ‘Gafsa', 'Mahdia');
foreach ($villes as $key => $city) {
   echo $city . ' a la clé ' . $key . "\n";
}
?>
```

Résultat

```
Tunis a la clé 0
Gafsa a la clé 1
Mahdia a la clé 2
```

Pour la boucle foreach, pas besoin de condition qui permet de stopper la boucle. En effet, cette dernière parcourt seulement tous les éléments du tableau. Elle commence donc au premier et s'arrête automatiquement après la lecture du dernier. On précise donc en premier lieu le tableau que l'on souhaite parcourir. Dans notre exemple, la variable $villes. À chaque itération de la boucle, la clé de l'élément parcouru est assignée dans la variable $key et la valeur de l'élément est assignée à la variable $ville.

VIII: Les Tableaux

```
<?php
$villes = array('Tunis', 'Sousse', 'Mahdia');

?>
```

Key	Value
0	Tunis
1	Sousse
2	Mahdia

1- Modifier un élément

```
<?php
$villes = array('Tunis', 'Sousse', 'Beja');
$villes[0] = 'Bizerte';
var_dump($villes);
?>
```

Résultat

```
array(3) { [0]=> string(7) "Bizerte" [1]=> string(6) "Sousse" [2]=>
string(4) "Beja" }
```

2- Ajouter un élément

```
<?php
$villes = array('Bizerte', 'Sousse', 'Mahdia’);
$villes[9] = 'Gabes';
var_dump($villes);
?>
```

Résultat

```
array(4) { [0]=> string(7) ‘’ Bizerte" [1]=> string(6) ‘’ Sousse"
[2]=> string(6) ‘’ Mahdia" [9]=> string(5) ‘’ Gabes" }
```

VI: Les Tableaux

```php
<?php
$villes = array('Bizerte', 'Sousse', 'Mahdia’);
 $villes[] = 'Gabes';
var_dump($villes);
?>
```

Résultat

array(4) { [0]=> string(7) ‘’ Bizerte" [1]=> string(6) ‘’ Sousse" [2]=> string(6) ‘’ Mahdia" [3]=> string(5) ‘’ Gabes" }

3- Supprimer un élément

```php
<?php
$villes = array('Bizerte', 'Sousse', 'Mahdia');
unset($villes['Sousse']);
var_dump($villes);
?>
```

Résultat

array(2) { [0]=> string(7) ‘’ Bizerte" [2]=> string(6) ‘’ Mahdia" }

4- Compter les éléments d’un tableau

```php
<?php
$villes = array('Beja', 'Monastir', 'Gafsa', 'Tozeur', 'Kasserine');
$n_villes = count($villes);
 echo 'Il y a '.$n_villes.' éléments dans le tableau $villes.';
?>
```

Résultat

Il y a 5 éléments dans le tableau $villes.

5- Rechercher une valeur dans un tableau

```
<?php
<?php
$villes = array('Beja', 'Monastir', 'Gafsa', 'Tozeur', 'Kasserine');
$ville_recherche = 'Gafsa';
if (in_array($ville_recherche, $villes)) {
    echo 'la ville ' . $ville_recherche . ' existe';
} else {
    echo 'la ville ' . $ville_recherche . ' n\'existe pas';
}
?>
```

Résultat

```
La ville Gafsa existe.
```

6- Rechercher une clé dans un tableau

```
<?php
<?php
$villes = array('Tunis' => 1001, 'Sousse' => 4000, 'Mahdia' => 5100);
$key_recherche = 'Sfax';
if (array_key_exists($key_recherche, $villes)) {
    echo 'Oui la clé ' . $key_recherche . 'existe dans le tableau $villes.';
} else {
    echo 'non la clé ' . $key_recherche . ' n\'existe dans le tableau $villes.';
}
?>
```

Résultat

```
Non, la clé Sfax n'existe pas dans le tableau $villes.
```

MODULE 2 : POO et PHP (Programmation orientée objet)

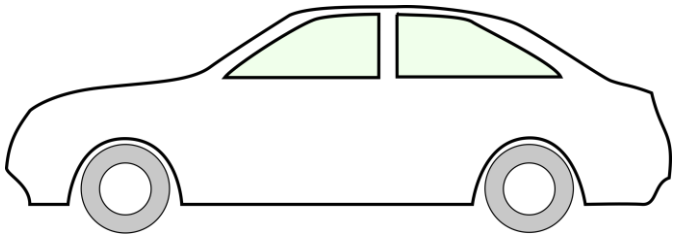
Qu'est ce qu'un objet ?

- La programmation orientée objet est plus naturelle donc plus intuitive. Si c'est le cas, c'est parce qu'elle utilise des entités appelées **objets**. Un objet possède sa propre structure interne qui définit ses propriétés et son comportement. Si on compare avec le monde réel, les objets sont partout autour de nous. Une personne est un objet, une voiture en est un autre, une maison, une école, un animal de compagnie... tous sont des objets.

  Ces points représentent les caractéristiques (ou propriétés) de l'objet voiture. Dans le jargon de la POO, on les appelle **des attributs**.

  - couleur
  - options
  - puissance du moteur
  - vitesse
  - source d'énergie
  - ...

Cependant, une voiture peut aussi entamer **des actions**, par exemple:



- Accélérer
- Ralentir
- Tourner à droite
- Tourner à gauche
- Reculer
- ...

Une classe c'est quoi?

Les objets de la POO doivent être créés d'abord pour pouvoir être manipulés après. C'est la **classe** qui se charge de donner vie aux objets.

Une classe est une structure cohérente de propriétés (attributs) et de comportements (méthodes). C'est elle qui contient la définition des objets qui vont être créés après. En général on considère une classe comme un moule à objets. Avec un seul moule on peut créer autant d'objets que l'on souhaite.

L'instanciation d'une classe

L'instanciation est le fait de créer une instance. Pour être précis, on parle d'une instance de classe. La classe étant le moule qui sert à fabriquer les objets, alors chaque objet créé correspond à une instance de la classe qui lui a donné vie.

Déclaration de la classe

La classe renferme l'ensemble des propriétés et de méthodes qui servent à définir l'identité de l'objet qui en découlera (l'instance de classe).

```
<?php
   class Voiture{
      // Déclaration des membres
   }
?>
```

Déclaration des attributs - Principe d'encapsulation

Les attributs sont les variables membres de la classe. Ils constituent les propriétés ou les caractéristiques de l'objet (l'instance de classe) qui en sera né.

Pour déclarer un attribut il faut le précéder par sa **visibilité**

- **public:** dans ce cas, l'attribut est accessible de partout (de l'intérieur de la classe dont il est membre comme de l'extérieur).
- **private**: dans ce cas, l'attribut est accessible seulement de l'intérieur de la classe dont il est membre.
- **protected**: dans ce cas, l'attribut est accessible seulement de l'intérieur de la classe dont il est membre ainsi que de l'intérieur des classes fille qui héritent de cette classe (Nous verrons l'héritage plus loin dans ce cous).

```
<?php
class Voiture{
private $couleur;
private $puissance;
private $vitesse;}
?>
```

Déclaration des méthodes

Les méthodes sont des fonctions membres. Ce sont elles qui se chargent de manipuler les attributs et dotent ainsi la classe de son comportement.

Pour déclarer une méthode il suffit de procéder comme si on déclarait une fonction en PHP en la précédant par la visibilité qui peut être soit **public** ou **private**. Une méthode publique est accessible de partout (de l'intérieur comme de l'extérieur de la classe). Une méthode privée est accessible seulement de l'intérieur de la classe. Elle est généralement sollicitée par une autre méthode publique.

```
<?php
class Voiture{
    private $couleur;
    private $puissance;
    private $vitesse;
    public function accelerer(){
                }
        public function ralentir(){
                }
}

?>
```

Instancier une classe

Supposons qu'on veut créer un objet du nom de **$mavoiture** à partir de la classe **Voiture**. L'instanciation de la classe ressemblerait à ceci:

```
<?php
    $mavoiture = new Voiture();
?>
```

Appel d'attributs et méthodes

Après avoir créé l'objet, on peut désormais accéder à ses membres (attributs et méthodes) par le biais de l'opérateur **->** (un tiret suivi d'un chevron fermant).

```
<?php
   class Voiture{
      private $couleur="Rouge";
      private $puissance;
      private $vitesse;
      public function accelerer(){
         echo "Appel de la méthode accelerer()";
      }
      public function ralentir(){
         echo "Appel de la méthode ralentir()";
      }
   }
   $mavoiture = new Voiture();
   $mavoiture -> accelerer();
   echo $mavoiture -> couleur;
?>
```

Instancier une classe

Supposons qu'on veut créer un objet du nom de **$mavoiture** à partir de la classe **Voiture**. L'instanciation de la classe ressemblerait à ceci:

```
<?php
    $mavoiture = new Voiture();
?>
```

Appel d'attributs et méthodes

Après avoir créé l'objet, on peut désormais accéder à ses membres (attributs et méthodes) par le biais de l'opérateur **->** (un tiret suivi d'un chevron fermant).

```
<?php
   class Voiture{
      private $couleur="Rouge";
      private $puissance;
      private $vitesse;
      public function accelerer(){
         echo "Appel de la méthode accelerer()";
      }
      public function ralentir(){
         echo "Appel de la méthode ralentir()";
      }
   }
   $mavoiture = new Voiture();
   $mavoiture -> accelerer();
   echo $mavoiture -> couleur;
?>
```

Ce qui donne:
Appel de la méthode accelerer()
Fatal error**: Cannot access private property Voiture::$couleur in **index.php** on line **17

La première ligne de l'exécution est logique. On a pu appeler la méthode accelerer() depuis l'extérieur de la classe car elle est publique. Ce qui affiche le message "Appel de la méthode accelerer()« et même chose pour l'attribut $couleur était en privé.

Constantes de classe

Une constante de classe est un élément statique par défaut. Son rôle est le même que celui d'une constante classique déclarée à l'aide de la fonction define(). Sa valeur est inchangée et elle appartient à la classe dans laquelle est elle évoquée et non à l'objet qui constitue l'instance de classe.

```php
<?php
   class Voiture{
      const ROUES=4;
      private $couleur="Rouge";
      private $puissance;
      private $vitesse;
      public function accelerer(){
         echo "Appel de la méthode accelerer()";
      }
      public function ralentir(){
         echo "Appel de la méthode ralentir()";
      }
   }
?>
```

La constante de classe peut être appelée de l'intérieur comme de l'extérieur de la classe grâce à l'opérateur de résolution de portée ***::***

Si par exemple on exécute le code suivant après la définition de la classe Voiture:

```php
<?php
   echo "Nombre de roues: ".Voiture::ROUES;
?>
```

on obtient:

Nombre de roues: 4

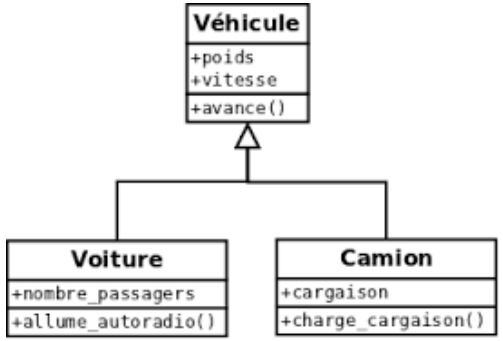
Attributs et méthodes statiques

```php
<?php
   class Voiture{
      private static $roues=4;
      public static function message(){
         echo "Il s'agit d'une voiture.<br />";
         echo "Elle a ".self::$roues." roues.";
      }
   }
   Voiture::message();
?>
```

Concept de l'héritage

L'héritage est un concept fondamental de la POO. C'est d'ailleurs l'un des plus importants puisqu'il permet de réutiliser le code d'une classe autant de fois que l'on souhaite tout en ayant la liberté d'en modifier certaines parties. Supposons que nous disposons d'une classe du nom de **classe1**. Si on crée une classe du nom de **classe2** qui hérite de **classe1**, alors **classe2** hérite de tous les membres (attributs et méthodes) qui constituent **classe1**. Autrement dit, si on instancie **classe2**, alors tous les attributs et méthodes de **classe1** peuvent être appelés à partir de l'objet créé (l'instance de **classe2**). Bien entendu, il faut que les membres appelés soient publiques. Dans ce cas on dit que **class1** est la classe **mère** et **class2** est la classe **fille**.

Pour procéder à l'héritage, on fait appel au mot clé **extends** comme ceci:




```php
<?php
   class Mere{
      public $attribut="Bonjour.";
      public function methode1(){
         $str=$this->attribut;
         $str.=" Je suis la classe Mère.";
         return $str;
      }
   }
   class Fille extends Mere{
      public function methode2(){
         $str=$this->attribut;
         $str.=" Je suis la classe Fille.";
         return $str;
      }
   }
   $objet=new Fille();
   echo $objet->methode1();
   echo "<br />";
   echo $objet->methode2();
?>
```

On obtient alors:

Bonjour. Je suis la classe Mère.
Bonjour. Je suis la classe Fille.