



M107 Développer des sites web dynamiques

B. Programmer en PHP

1

Filière: Développement Digital

Formatrice: Asmae YOUNALA

PLAN DU MODULE

- A. Introduction
- B. Programmer en PHP**
- C. Manipuler les données
- D. Réaliser un site web avec l'architecture MVC

PLAN

- B.1 Maîtriser le langage PHP
- B.2 Traiter les données en PHP
- B.3 Utiliser l'orientée objet en PHP

4

Structure générale d'un script PHP

- Les instructions PHP sont réunies dans un bloc entouré par deux balises spécifiques du fichier HTML : `<?>` et `?>`

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <title>Premier exemple</title>
</head>
<body>
  <h1>Un premier exemple PHP</h1>
  <?php
    echo "Bonjour tout le monde";
  ?>
</body>
</html>
```

Début du script PHP

Fin du script PHP

Manipulation des Variables

➤ *Qu'est ce qu'une variable ?*

Une variable est un “conteneur” qui sert à stocker une ou plusieurs informations.

Elle peut contenir une valeur d'un des types utilisés par PHP (entiers, flottants, chaînes de caractères, tableaux, booléens, objets, ressource ou NULL).

➤ *Comment déclarer des variables PHP ?*

- Chaque variable possède un **identifiant particulier**, qui commence toujours par le caractère dollar (\$) suivi du nom de la variable.
- Exemples: \$prix, \$nom, \$age

6

Manipulation des Variables

- ➔ PHP est sensible à la casse : **var** et **Var** ne sont pas les mêmes variables !
- ➔ Voici les règles à respecter lors du choix d'un nom d'une variable:
 - un nom de variable doit **commencer** par une **lettre** (majuscule ou minuscule) ou un underscore "_" (**pas par un chiffre**)
 - un nom de variable peut **comporter** des **lettres**, des **chiffres** et le caractère _ (**les espaces ne sont pas autorisés!**)

Manipulation des Variables

Exemples de noms de variable corrects

- \$Variable
- \$Nom_De_Variable
- \$nom_de_variable
- \$nom_de_variable_123

Exemple de noms de variable incorrects	Raison
\$Nom de Variable	comporte des espaces
\$123Nom_De_Variable	commence par un chiffre
\$toto@mailcity.com	caractère spécial @
\$Nom-de-variable	signe - interdit
nom_de_variable	ne commence pas par \$

8

Manipulation des constantes

Une **constante** est une variable dont la valeur est inchangeable lors de l'exécution d'un programme.

Avec PHP, les constantes sont définies grâce à la fonction **define()**. la syntaxe de la fonction `define()` est la suivante :

```
define("Nom_de_la_constante", Valeur);
```

Le nom d'une constante ne doit pas commencer par le caractère \$ (de cette façon aucune affectation n'est possible).

Exemples:

```
define("pi", 3.14);
```

```
echo pi;
```


L'Affectation

- Lorsque l'on assigne une valeur à une variable (=), son type change pour correspondre au type de données qui lui est affecté.
- L'affectation se fait toujours de la droite vers la gauche.

`$txt = "Hello world!";` → le type de `$txt` est: **chaîne**

`$x = 5;` → le type de `$x` est: **entier**

`$y = 10.5;` → le type de `$y` est: **double**

`$b = array("red", "green", "blue");` → le type de `$b` est: **array***

L'affectation par référence signifie que les deux variables pointent vers le même conteneur de donnée, rien n'est copié nulle part.

`$x = 5;`

`$y = &$x;`

`$x = 10;`

→ `$x` et `$y` ont la même valeur 10

Manipulation des types de données

- Afin de connaître le type des variables, un certain nombre de fonctions sont disponibles et retournent *true* ou *false* :
 - **is_array()**,
 - **is_double()**, **is_float()**, **is_real()**
 - **is_long()**, **is_int()**, **is_integer()**
 - **is_string()**,
 - **is_object()**,
 - **is_null()**,
 - **is_scalar()** - si la variable est scalaire, c'est à dire si c'est un entier, une chaîne, ou un double.
 - **is_numeric()** - si la variable est un nombre ou une chaîne numérique,

Manipulation des types de données : Exemples

```
$a = array("red", "green", "blue");  
echo "a is " . is_array($a) . "<br>";  
// résultat: a is 1
```

```
$b = true;  
echo "b is " . is_bool($b) . "<br>";  
// résultat: b is 1
```

```
$c = 16;  
echo "c is " . is_bool($c) . "<br>";  
// résultat: c is
```

```
$d = 32.7;  
echo "d is " . is_double($d) . "<br>";  
// résultat: d is 1
```

```
$f = "Hello";  
echo "f is " . is_string($f) . "<br>";  
// résultat: f is 1
```

```
$g = 0;  
echo "g is " . is_string($g) . "<br>";  
// résultat: g is
```

```
$h = 32;  
echo "h is " . is_scalar($h) . "<br>";  
// résultat: h is 1
```

```
$j = array("red", "green", "blue");  
echo "j is " . is_scalar($j) . "<br>";  
// résultat: j is
```

12

Manipulation des types de données

- Enfin, trois fonctions aident à la réalisation du transtypage de variables :
 - *int intval(variable)*,
 - *float floatval(variable)*,
 - *string strval(variable)*.
- Chacune de ces fonctions retourne la valeur transtypée de l'argument passé en paramètre.

```
echo intval(42); ➔ 42
```

```
echo intval(4.2); ➔ 4
```

```
echo intval('42'); ➔ 42
```

```
echo intval('+42'); ➔ 42
```

```
echo intval('-42'); ➔ -42
```

```
echo doubleval("13.78") . "<br>"; ➔ 13.78
```

```
echo doubleval("15.6Kg") . "<br>"; ➔ 15.6
```

```
echo doubleval("hello") . "<br>"; ➔ 0
```

Manipulation des types de données

➡ Il existe également une autre méthode de transtypage :

```
$var1 = (int)"2.34";
```

```
echo "\$var1 = $var1 <br>";
```

➡ **\$var1 = 2**

```
$var2 = (float)"2.34";
```

```
echo "\$var2 = $var2 <br>";
```

➡ ➡ **\$var2 = 2.34**

```
$var3 = (float)"13.5Dhs";
```

```
echo "\$var3 = $var3 <br>";
```

➡ ➡ **\$var3 = 13.5**

Manipulation des types de données

- ▶ **gettype(\$var)** : Retourne le type de la variable \$var

```
$var = 34.7;  
echo gettype($var) . "<br>"; ➔ //double
```

- ▶ **var_dump(\$var)** : Affiche les informations d'une variable

```
$a = 15.5;  
var_dump($a); ➔ float(15.5)  
$b = "-12";  
var_dump($b); ➔ string(3) "-12"
```

- ▶ **settype(\$var, "type")** Affecte un type à une variable et retourne **true** en cas de succès et **false** sinon. Les valeurs possibles pour le paramètre type sont : "boolean" (ou "bool"), "integer" (ou "int"), "float" (ou "double"), "string", "array", "object", "NULL".

```
$var = 2.23;  
settype($var, "int");  
var_dump($var); ➔ int(2)
```

Manipulation des types de données

- Trois fonctions permettent également de connaître l'état de variables :
 - ***isset*** () : retourne *true* si la variable passée en paramètre existe *false* sinon,
 - ***unset*** () : supprime la variable passée en paramètre,
 - ***empty***() : retourne un booléen en vérifiant si la variable est non vide et non nulle.

Les valeurs ci-dessous sont considérées comme étant vides :

- "" (une chaîne vide)
- 0 (0 en tant qu'entier)
- 0.0 (0 en tant que nombre à virgule flottante)
- "0" (0 en tant que chaîne de caractères)
- NULL
- FALSE
- array() (un tableau vide)

Exemples:

```
<?php
$a = 0;
// True car $a est définie
if (isset($a)) {
    echo "Variable 'a' is set.<br>";
}

$b = null;
// False car $b est NULL
if (isset($b)) {
    echo "Variable 'b' is set.";
}
?>
```


16

Instructions de sortie

- **echo** et **print** sont utilisés pour afficher des données à l'écran.
- Les instructions **echo** et **print** peuvent être utilisées avec ou sans parenthèses : **echo** ou **echo()**, **print** ou **print()**

Exemples:

```
$txt1 = "Apprendre PHP";  
$txt2 = "ISTA";  
$x = 5;  
$y = 4;  
echo "<h2>" . $txt1 . "</h2>";  
echo "Etudier PHP à " . $txt2 . "<br>";  
echo $x + $y;
```

```
$txt1 = "Apprendre PHP";  
$txt2 = "ISTA";  
$x = 5;  
$y = 4;  
  
print "<h2>" . $txt1 . "</h2>";  
print "Etuder PHP à " . $txt2 . "<br>";  
print $x + $y;
```


Les opérateurs de base:

Les opérateurs arithmétiques

Exemple	Nom	Résultat
+\$a	Identité	Conversion de \$a vers int ou float, selon le plus approprié. <code>\$a = "-12";</code> <code>echo gettype(\$a) → string</code> <code>echo gettype(+\$a) → integer</code>
-\$a	Négation	Opposé de \$a.
\$a + \$b	Addition	Somme de \$a et \$b.
\$a - \$b	Soustraction	Différence de \$a et \$b.
\$a * \$b	Multiplication	Produit de \$a et \$b.
\$a / \$b	Division	Quotient de \$a et \$b.
\$a % \$b	Modulus	Reste de \$a divisé par \$b.
\$a ** \$b	Exponentiation	Résultat de l'élévation de \$a à la puissance \$b.

Les opérateurs de base:

Les opérateurs d'affectation

Exemple	Equivalent	Opération
<code>\$a += \$b</code>	<code>\$a = \$a + \$b</code>	Addition
<code>\$a -= \$b</code>	<code>\$a = \$a - \$b</code>	Soustraction
<code>\$a *= \$b</code>	<code>\$a = \$a * \$b</code>	Multiplication
<code>\$a /= \$b</code>	<code>\$a = \$a / \$b</code>	Division
<code>\$a %= \$b</code>	<code>\$a = \$a % \$b</code>	Modulo
<code>\$a **= \$b</code>	<code>\$a = \$a ** \$b</code>	Exponentiation

Exemple	Equivalent	Opération
<code>\$a .= \$b</code>	<code>\$a = \$a . \$b</code>	Concaténation d'une chaîne de caractères
<code>\$a ??= \$b</code>	<code>\$a = \$a ?? \$b</code> (si \$a est null: <code>\$a=\$b</code>)	Opérateur de coalescence nul

Les opérateurs de base:

Les opérateurs de comparaison

Exemple	Nom	Résultat
\$a == \$b	Égal	true si \$a est égal à \$b après le transtypage.
\$a === \$b	Identique	true si \$a est égal à \$b et qu'ils sont de même type.
\$a != \$b	Différent	true si \$a est différent de \$b après le transtypage.
\$a <> \$b	Différent	true si \$a est différent de \$b après le transtypage.
\$a !== \$b	Différent	true si \$a est différent de \$b ou bien s'ils ne sont pas du même type.
\$a < \$b	Plus petit que	true si \$a est strictement plus petit que \$b.
\$a > \$b	Plus grand	true si \$a est strictement plus grand que \$b.
\$a <= \$b	Inférieur ou égal	true si \$a est plus petit ou égal à \$b.
\$a >= \$b	Supérieur ou égal	true si \$a est plus grand ou égal à \$b.

Les opérateurs de base:

Les opérateurs logiques

Exemple	Nom	Résultat
<code>\$a and \$b</code>	And (Et)	true si <code>\$a</code> ET <code>\$b</code> valent true .
<code>\$a or \$b</code>	Or (Ou)	true si <code>\$a</code> OU <code>\$b</code> valent true .
<code>\$a xor \$b</code>	XOR	true si <code>\$a</code> OU <code>\$b</code> est true , mais pas les deux en même temps.
<code>! \$a</code>	Not (Non)	true si <code>\$a</code> n'est pas true .
<code>\$a && \$b</code>	And (Et)	true si <code>\$a</code> ET <code>\$b</code> sont true .
<code>\$a \$b</code>	Or (Ou)	true si <code>\$a</code> OU <code>\$b</code> est true

Contrôles de flux : structures alternatives

► Syntaxes de base:

```
if (condition) {
    ....;
} elseif (condition) {
    ....;
} else {
    .....;
}
```

```
if (condition):
    .....;
    .....;
elseif (condition):
    .....;
    .....;
else:
    .....;
endif;
```

```
<?php
if ($a > $b)
    echo "a est plus grand que b";
?>
```

```
<?php if ($a == 5): ?>
A égal 5
<?php endif; ?>
```

```
<?php
if ($a == 5):
    echo "a égale 5";
    echo "...";
elseif ($a == 6):
    echo "a égale 6";
    echo "!!!";
else:
    echo "a ne vaut ni 5 ni 6";
endif;
?>
```

Contrôles de flux : structures alternatives (Opérateur ternaire)

➡ `$v = 1;`
`$r = (1 == $v) ? 'Yes' : 'No';` // \$r prendra la valeur 'Yes'

Equivalent à :

```
$v = 1;
if ($v == 1)
    $r = 'Yes';
else
    $r = 'No';
```

➡ `echo (1 == $v) ? 'Yes' : 'No';` // 'Yes' will be printed

Equivalent à :

```
$v = 1;
if ($v == 1)
    echo 'Yes';
else
    echo 'No';
```

Contrôles de flux : structures alternatives (Opérateur ternaire)

```
➡ $a = 'valeur a';  
   $b = ($a) ?: 'sans valeur'; // $a est non vide => $b prendra la valeur  
de $a  
   echo "\$b=" . $b . "<BR>";  
  
➡ $c = '';  
   $d = ($c) ?: 'sans valeur'; // $c est vide => $d prendra son alternatif  
'sans valeur'  
   echo "\$d=" . $d . "<BR>";
```

Contrôles de flux : structures alternatives (switch)

```
switch(<expression>){  
case const1: [ <bloc de code> ]  
    [ break; ]  
case const2: [ <bloc de code> ]  
    [ break; ]  
    .  
[default: <bloc de code> ]  
}
```

Exemple1:

```
$nombre = 10;  
    switch ($nombre) {  
        case 9:  
            echo "Le nombre est neuf";  
            break;  
        case 10:  
            echo "Le nombre est dix";  
            break;  
        default:  
            echo "Nous ne connaissons pas le  
nombre";  
    }  
→ Le nombre est dix
```


Contrôles de flux : structures alternatives (switch)

Exemple 2:

```
$count = 35;  
switch (true) {  
    case $count <= 20:  
        echo 'low';  
        break;  
    case $count <= 40:  
        echo 'medium';  
        break;  
    case $count <= 60:  
        echo 'high';  
        break;  
    default:  
        echo 'severe';  
        break;  
}
```

Contrôles de flux : structures répétitives (for)

Une boucle permet de répéter x fois une exécution de code.

Par exemple, si vous voulez afficher dix fois "Bonjour", il suffit d'écrire avec une boucle **for** :

syntaxe :

```
for ($i=nombre de départ ; $i <= nombre d'arrivée ; incrément) {  
    instructions  
}
```

exemple:

```
<?php  
    for ($i = 1; $i <= 10; $i++) {  
        echo 'Bonjour <br />';  
    }  
?>
```

Contrôles de flux : structures répétitives (while)

PHP exécute l'instruction tant que l'expression de la boucle while est évaluée comme true.

Syntaxes:

1.

```
while (expression) {  
    commandes  
}
```

2.

```
while (expression):  
    commandes ...  
endwhile
```

//exemple 1

```
$i = 1;  
while ($i <= 10):  
    echo $i;  
    $i++;  
endwhile;
```

//exemple 2

```
$i = 1;  
while ($i <= 10) {  
    echo $i++; /* La valeur affichée es  
t $i avant l'incréméntation (post-  
incréméntation) */  
}
```

Contrôles de flux : structures répétitives (do-while)

- ➔ Les boucles **do-while** ressemblent beaucoup aux boucles **while**, mais l'expression est testée à la fin de chaque itération plutôt qu'au début.

Exemple:

```
<?php
    $i = 0;
    do {
        echo $i;
    } while ($i > 0);
?>
```

Contrôles de flux : structures répétitives (foreach)

```
➤ foreach ($tableau as $valeur) {  
    insts utilisant $valeur ;  
}
```

Exemple:

```
$list = array('un', 'deux', 'trois', 'quatre', 'stop', 'cinq', 'six',  
'sept');  
  
foreach ($list as $val) {  
    echo $val . ' ';  
}
```

Contrôles de flux : Ruptures de séquence (break)

- L'instruction *break* permet de quitter immédiatement une instruction *switch* ou de *boucle*.
- L'exemple qui suit montre l'utilisation de l'instruction *break* dans une boucle *foreach*.

```
$list = array('un', 'deux', 'trois', 'quatre', 'stop', 'cinq', 'six', 'sept');  
foreach ($list as $val) {  
    if ($val == 'stop') break;  
    echo $val . ' ';  
}
```

➔ Résultat : un deux trois quatre

```
$i = 11;  
while ($i < 20) {  
    echo $i . ' ';  
    if ($i % 5 == 0) break;  
    $i++;  
}
```

➔ Résultat : 11 12 13 14 15

31

Contrôles de flux : Ruptures de séquence (continue)

- L'instruction *continue* est utilisée au sein d'une instruction de *boucle*. Elle permet de passer immédiatement à la prochaine itération de la boucle.
- L'exemple qui suit montre l'utilisation de l'instruction *continue* dans la boucle *foreach* précédente.

Exemple

```
$list =  
array('un', 'deux', 'trois', 'quatre', 'suivant', 'cinq', 'six', 'sept', 'huit');  
foreach($list as $val) {  
    if($val=='suivant')  
        continue;  
    echo $val . ' ';  
}
```

➔ Résultat : un deux trois quatre cinq six sept huit

Contrôles de flux : Ruptures de séquence (**goto**)

L'instruction **goto** permet de passer à un point donné, marqué par un nom dans une fonction. Le point sur lequel nous faisons le branchement est noté de la forme *nomDuPoint* :

Il s'agit du nom du point, suivi de : (deux points).

Exemple

```
<?php
goto a;
echo 'Foo';

a:
echo 'Bar';
?>
```


Les fonctions (définies par l'utilisateur)

► Déclaration:

```
function Nom_De_La_Fonction(argument1,  
argument2, ...)  
{  
    liste d'instructions;  
    [return value;]  
}
```

► Appel de fonction:

```
Nom_De_La_Fonction(argument1, argument2);
```

Exemple:

```
function somme($a, $b)  
{  
    return $a + $b;  
}  
echo somme(15, 18.9);
```

Les fonctions (définies par l'utilisateur)

► Passage par référence:

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

Les fonctions (définies par l'utilisateur)

Fonction avec nombre d'arguments inconnu :

- **func_num_args()** : fournit le nombre d'arguments qui ont été passés lors de l'appel de la fonction
- **func_get_arg(\$i)** : retourne la valeur de la variable située à la position \$i dans la liste des arguments passés en paramètres. *Ces arguments sont numérotés à partir de 0*

```
function produit()  
{  
    $nbarg = func_num_args();  
    $prod = 1;  
    // la fonction produit a ici $nbarg arguments  
    for ($i = 0; $i < $nbarg; $i++) {  
        $prod *= func_get_arg($i);  
    }  
    return $prod;  
}  
echo "le produit est : ", produit(3, 77, 10, 5, 81, 9), "<br />";  
// affiche le produit est 8 419 950
```

La portée des variables : locale/globale

- Variable locale : Visible uniquement à l'intérieur d'un contexte d'utilisation
- Variable globale:
 - Visible dans tout le script
 - Utilisation de l'instruction global dans des contextes locales
 - Ou utilisation du tableau associatif pré-défini \$GLOBALS : `$GLOBALS['a']`

```
$var = 100;  
function test()  
{  
    global $var;  
    return $var;  
}  
$resultat = test();  
if ($resultat) echo $resultat;  
else echo " erreur ";  
→ 100
```

La portée des variables : static

Pour conserver la valeur acquise par une variable entre deux appels de la même fonction, on utilise l'instruction **static**.

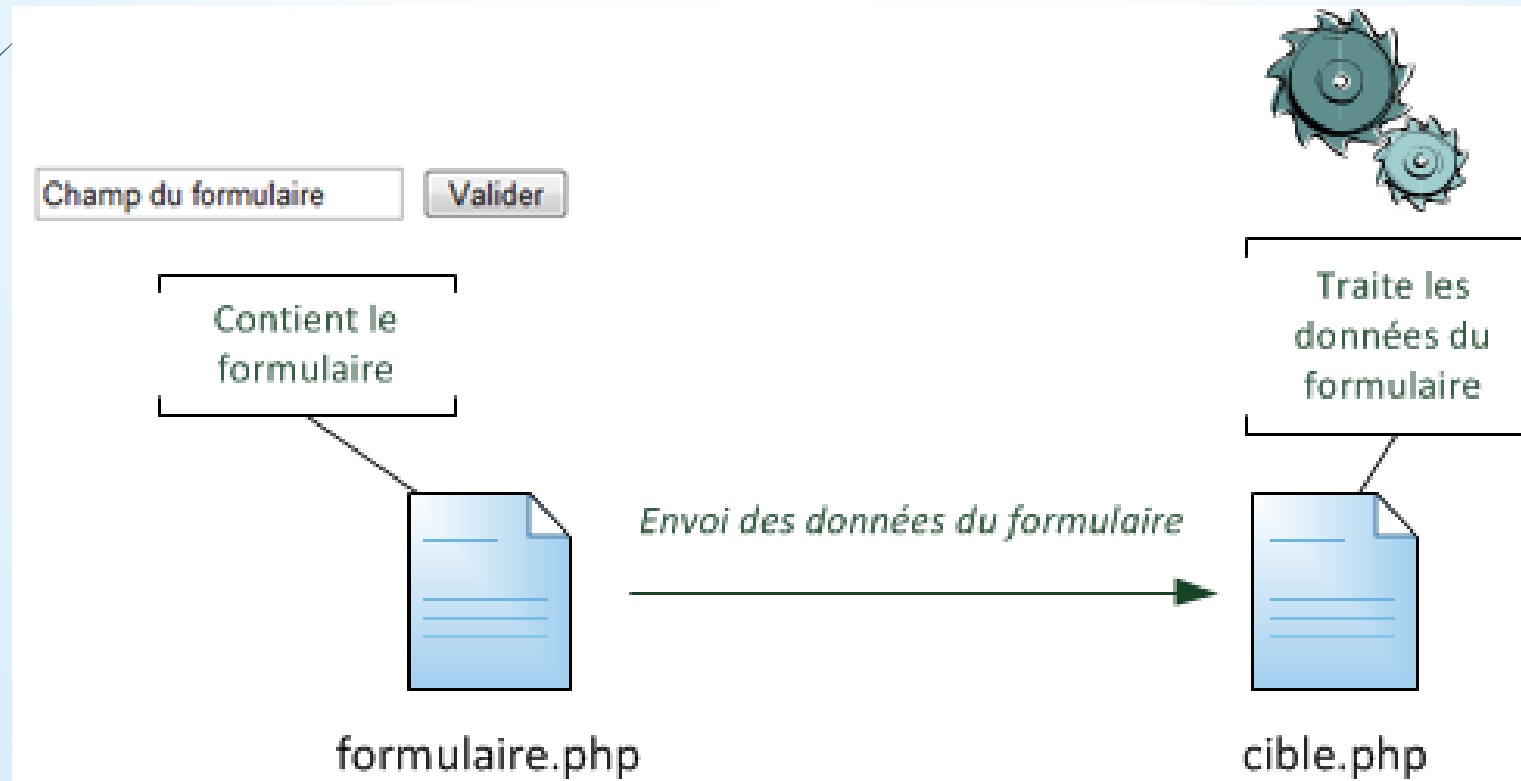
Les variables statiques restent locales à la fonction et ne sont pas réutilisables à l'extérieur.

```
function cumul($prix)
{
    static $cumul = 0;
    static $i = 1;
    echo "Total des achats $i = ";
    $cumul += $prix;
    $i++;
    return $cumul;
}

echo cumul(175), "<br />"; //Total des achats 1 = 175
echo cumul(65), "<br />"; //Total des achats 2 = 240
echo cumul(69), "<br />"; //Total des achats 3 = 309
```

Formulaires simples

```
<form action="cible.php" method="POST">  
  <input type="text" name="champ" placeholder="champ du formulaire"/>  
  <input type="submit" value="Valider">  
</form>
```



Transmission de variables (GET, POST)

- Les informations envoyées au moyen d'un formulaire sont stockées dans les tableaux superglobaux de l'environnement PHP.
- **Voici les tableaux utilisés :**
 - **\$_GET** : Les variables de l'URL (méthode GET)
 - **\$_POST** : Les variables envoyées par la méthode POST
 - **\$_FILES** : Les fichiers envoyés par formulaire
- Il est possible d'utiliser **print_r()** pour afficher le contenu d'un formulaire :

```
if (!empty($_POST)) {  
    echo '<pre>';  
    print_r($_POST);  
    echo '</pre>';  
}
```


Transmission de variables (Zones de texte)

Exemple: Soit la page **formulaire.html** suivante :

```
<form action="inscription.php" method="POST">
    <input type="text" name="nom" placeholder="nom" /> <br>
    <input type="email" name="email" placeholder="email"> <br>
    <input type="submit" value="S'inscrire">
</form>
```

La page **inscription.php**:

```
<html lang="en">
<head></head>
<body>
    <h2>Inscription</h2>
    <?php
if (!empty($_POST["nom"]) && !empty($_POST["email"])) {
    echo "Nom: " . $_POST["nom"] . "<br>";
    echo "Email: " . $_POST["email"] . "<br>";
} ?>
</body></html>
```


Inscription

Nom: Alaoui Ahmed

Email: alaoui.ahmed@gmail.com

41

Transmission de variables (radios boutons)

Exemple: Soit la page **formulaire.html** suivante :

```
<form action="inscription.php" method="POST">
```

```
Niveau en php:<input type="radio" name="niveau" value="Débutant"> Débutant
```

```
<input type="radio" name="niveau" value="Moyen"> Moyen
```

```
<input type="radio" name="niveau" value="Expert"> Expert
```

```
<input type="submit" value="OK">
```

```
</form>
```

La page **inscription.php**:

```
<html lang="en">
```

```
<head></head><body>
```

```
<h2>Inscription</h2>
```

```
<?php
```

```
if (!empty($_POST["niveau"])) {
```

```
    echo "Niveau: " . $_POST["niveau"] . "<br>";
```

```
}?>
```

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

Niveau en php: ☐ Débutant ☒ Moyen ☐ Expert

OK

Inscription

Niveau: Moyen

Transmission de variables (Cases à cocher)

► Soit la page **formulaire.html** suivante :

```
<form action="inscription.php" method="POST">
  Langues:<input type="checkbox" name="langues[]" value="Arabe"> Arabe
         <input type="checkbox" name="langues[]" value="Français"> Français
         <input type="checkbox" name="langues[]" value="Anglais"> Anglais
</form>
```

La page **inscription.php**:

```
<html>
.....
  <?php
  if (!empty($_POST["langues"])) {
    echo "Liste des langues maîtrisées : <ul>";
    foreach ($_POST["langues"] as $lang) {
      echo "<li> " . $lang;
    }
    echo " </ul>";
  }?>
.....
```

Langues: ☒ Arabe ☐ Français ☒ Anglais

OK

Inscription

Liste des langues maîtrisées :

- Arabe
- Anglais

43

Transmission de variables (liste déroulante)

➤ Soit la page **formulaire.html** suivante :

```
<form action="inscription.php" method="POST">
Niveau scolaire: <select name="niveauScolaire">
    <option value="Bac">Bac</option>
    <option value="Deug">Deug</option>
    <option value="Licence">Licence</option>
</select>
</form>
```

La page **inscription.php**:

```
<html>
.....
    <?php
        if (!empty($_POST["niveauScolaire"])) {
            echo "Niveau scolaire : " . $_POST["niveauScolaire"];
        }?>
    .....

```

Niveau:

- Bac
- Deug**
- Licence

Inscription

Niveau scolaire : Deug

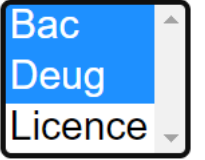
44

Transmission de variables (liste déroulante multi-choix)

```
<form action="inscription.php" method="POST">
Niveau scolaire: <select name="niveauScolaire[]" multiple>
    <option value="Bac">Bac</option>
    <option value="Deug">Deug</option>
    <option value="Licence">Licence</option>
</select>
</form>
```

La page **inscription.php**:

```
.....
<?php
echo "Diplômes obtenues :<br>";
if (!empty($_POST["niveauScolaire"])) {
    foreach ($_POST["niveauScolaire"] as $diplome) {
        echo "- " . $diplome . "<br>";
    }
} ?>
.....
```

Niveau: 

Inscription

Diplômes obtenues :

- Bac
- Deug

Variables de Serveur: `$_SERVER`

- `$_SERVER` est un tableau contenant des informations comme les en-têtes, dossiers et chemins du script. Les entrées de ce tableau sont créées par le serveur web.

Indice	Indication	Exemple
'SERVER_NAME'	Le nom du serveur hôte qui exécute le script suivant.	<code>echo \$_SERVER['SERVER_NAME'] //localhost</code>
'SERVER_ADDR'	L'adresse IP du serveur sous lequel le script courant est en train d'être exécuté.	<code>echo \$_SERVER['SERVER_ADDR'] //:::1</code>
'REQUEST_METHOD'	Méthode de requête utilisée pour accéder à la page ; par exemple 'GET', 'HEAD', 'POST', 'PUT'.	<code>echo \$_SERVER['REQUEST_METHOD'] //GET</code>
'HTTP_HOST'	Contenu de l'en-tête Host: de la requête courante, si elle existe.	<code>echo \$_SERVER['HTTP_HOST'] //localhost:8080</code>
'SERVER_PORT'	Le port de la machine serveur utilisé pour les communications. Par défaut, c'est "80". En utilisant SSL, par exemple, il sera remplacé par le numéro de port HTTP sécurisé.	<code>echo \$_SERVER['SERVER_PORT'] //8080</code>

Variables de Serveur: \$_SERVER['PHP_SELF']

- Le nom du fichier du script en cours d'exécution, par rapport à la racine web. Par exemple, \$_SERVER['PHP_SELF'] dans le script situé à l'adresse <http://example.com/foo/bar.php> sera [/foo/bar.php](#).

Utilisation:

```
<form method="get" action="<?php $_SERVER["PHP_SELF"] ?>">
    <input type="text" name="nom" />
    <br>
    <input type="submit" value="Envoyer" name="envoyer" />
</form>

<?php
if (!empty($_GET["envoyer"])) {
    echo "Bonjour " . $_GET["nom"];
}
?>
```

➔ **Bonne pratique:** `<form method="get" action="<?php htmlentities($_SERVER["PHP_SELF"])?>">`

➔ Pour envoyer les données à la même page contenant le formulaire, on peut également utiliser action sans valeur:

```
<form method="get" action="">
```


Fonctions de math

- **abs(\$val)**: retourne la valeur absolue de \$val
- **acos(\$val), cos(\$val), sin(\$val) ...**
- **exp(\$val), log(\$val)**
- **floor(\$val)**: retourne l'entier inférieur du nombre \$val.
- **round(val, precision)**: Retourne la valeur arrondie de num à la précision *precision* (nombre de chiffres après la virgule). Le paramètre *precision* peut être négatif ou null (sa valeur par défaut).
- **ceil(val)**: Retourne l'entier supérieur du nombre num.
- **max(val1, val2), min(val1, val2)**
- **pow(val, puiss), sqrt(val)**
- **rand(), rand(min, max)**: entier aléatoire entre max et min
- **intdiv** — Division entière
- **is_finite** — Indique si un nombre est fini
- **is_infinite** — Indique si un nombre est infini
- **is_nan** — Indique si une valeur n'est pas un nombre

Fonctions de math : Exemples

48

- `var_dump(abs(-56));`
→ `int(56)`
- `var_dump(floor(12.4));`
→ `float(12)`
- `var_dump(round(3.4));`
→ `float(3)`
- `var_dump(round(3.6));`
→ `float(4)`
- `var_dump(round(5.043, 2));`
→ `float(5.04)`
- `var_dump(ceil(14,05));`
→ `float(15)`

- `var_dump(max(15, 16));`
→ `int(16)`
- `var_dump(min(10, 18));`
→ `int(10)`
- `var_dump(pow(3, 3));`
→ `int(27)`
- `var_dump(sqrt(25));`
→ `float(5)`
- `var_dump(rand());`
→ `int(2134667633)`
- `var_dump(rand(1, 10));`
→ `int(3)`

- `var_dump(is_infinite(log(0)));`
→ `bool(true)`

- `var_dump(intdiv(15, 4));`
→ `int(3)`
- `var_dump(is_finite(0));`
→ `bool(true)`
- `var_dump(is_nan("14"));`
→ `bool(false)`

Les chaînes de caractères

- Une chaîne de caractères est une suite de caractères dont le nombre maximal n'est pas limité. Elle est toujours délimitée par des simples cotes ou des doubles cotes.
- Différence entre simple cote et double cotes:

```
$str="Bonjour";
```

```
echo "$str à tous"; // Affiche: Bonjour à tous
```

```
echo '$str à tous'; // Affiche: $str à tous
```

Fonctions manipulant les chaînes de caractères:

Fonction	Indications	
nl2br	Insère un retour à la ligne HTML à chaque nouvelle ligne	<code>echo nl2br("Ceci\nest\nune\nchaîne");</code> ➔ Ceci est une chaîne
printf	Affiche une chaîne de caractères formatée	<code>printf("resultat= %.2f
", 1 / 3);</code> ➔ 0.33

Les chaînes de caractères

strlen	Calcule la taille d'une chaîne	<code>echo (strlen("Bonjour"));</code> → 7
trim / ltrim / rtrim	trim : Supprime les espaces (ou d'autres caractères) en début et fin de chaîne ltrim : Supprime à gauche rtrim : Supprime à droite	<code>echo (strlen(" Bonjour "));</code> → 19 <code>echo (strlen(trim(" Bonjour ")));</code> → 7
str_word_count	Compte le nombre de mots utilisés dans une chaîne	<code>echo (str_word_count("Hello world"));</code> → 2
strrev	Inverse une chaîne	<code>echo (strrev("Hello world"));</code> → dlrow olleH
strtoupper / strtolower	Renvoie la chaîne en majuscule / en minuscule	<code>echo (strtoupper("Hello world"));</code> → HELLO WORLD
ucfirst / lcfirst	Met le premier caractère en majuscule / en minuscule	<code>echo (ucfirst("hello ").lcfirst("WORLD"));</code> → Hello wORLD
ucwords	Met en majuscule la première lettre de tous les mots	<code>echo (ucwords("il fait beau ce matin!"));</code> → Il Fait Beau Ce Matin!

Les chaînes de caractères

strpos	Cherche la position de la première occurrence dans une chaîne	<code>echo (strpos("Bonjour tout le monde", "tout"));</code> → 8
stripos	Recherche la position de la première occurrence dans une chaîne, sans tenir compte de la casse	<code>echo (stripos("Bonjour tout le monde", "TOUT"));</code> → 8
substr (<i>\$string</i> , <i>\$indiceDebut</i> , <i>?\$taille</i>)	Retourne un segment de chaîne à partir de l'indice de début donné jusqu'à la fin de la chaîne. \$taille: (optionnel) le nombre de caractères à découper	<code>echo (substr("Bonjour tout le monde", 8, 4));</code> → tout
str_replace (<i>\$chaine_recherchée</i> , <i>\$remplacement</i> , <i>\$chaine</i>)	Remplace toutes les occurrences de <i>\$chaine_recherchée</i> dans une chaîne <i>\$chaine</i> par <i>\$remplacement</i>	<code>echo (str_replace("tout le monde", "Sara", "Bonjour tout le monde"));</code> → Bonjour Sara
str_ireplace	Version insensible à la casse de <code>str_replace()</code>	<code>echo (str_ireplace("ER", "ez", "vous vous amuser et manger"));</code> → vous vous amusez et mangez

Les chaînes de caractères

preg_match
(*\$pattern*, *\$subject*)

Effectue une recherche de correspondance avec une expression rationnelle standard

```
$cin = "cd1234";  
echo preg_match("/^[a-z]+\d+$/",  
$cin);  
➔ true
```

preg_match_all
(*\$pattern*,
\$subject, &*\$matches*)

Effectue une recherche de correspondance avec une expression rationnelle et remplit le tableau *matches* par les valeurs trouvées

```
$texte = "Bonjour, aujourd'hui c'est le  
01/03/2022, j'ai 25 ans ";  
$resultat = [];  
preg_match_all("/\d{2}/", $texte,  
$resultat);  
print_r($resultat);  
➔ Array  
(  
    [0] => Array  
        (  
            [0] => 01  
            [1] => 03  
            [2] => 20  
            [3] => 22  
            [4] => 25  
        )  
)
```

Les chaînes de caractères

preg_replace	Recherche une expression régulière et la remplace par une chaîne de caractères.	<pre>\$texte = "Bonjour, aujourd'hui c'est le 01/03/2022, j'ai 25 ans "; echo preg_replace("/\d/", "*", \$texte); → Bonjour, aujourd'hui c'est le **/**/****, j'ai ** ans</pre>
ctype_digit	Vérifie qu'une chaîne est un entier	
str_repeat	Répète une chaîne	<pre>echo (str_repeat("*", 10)); → **********</pre>
str_pad	Complète une chaîne jusqu'à une taille donnée	<pre>echo (str_pad("Bonjour", 12, "*")); → Bonjour***** echo str_pad("100", 6, "0"); → 100000</pre>
htmlspecialchars	Convertit les caractères spéciaux en entités HTML	<pre>echo (htmlspecialchars("Bonjour Sara
Bonne journée")); → Bonjour Sara
Bonne journée</pre>

Format de date

Aujourd'hui est le **14/03/2022 13:22:25**

- Formats de date:

Caractère	Signification	Exemple
d	Représente le jour du mois (01 à 31)	<code>echo date("d");</code> => 14
m	Représente le mois (01 à 12)	<code>echo date("m");</code> => 03
Y	Représente l'année (en 4 chiffres)	<code>echo date("Y");</code> => 2022
l (lowercase 'L')	Représente le jour de la semaine	<code>echo date("l");</code> => Monday

- Formats de Temps :

Caractère	Signification	Exemple
H	24 heure : format de l'heure(00 to 23)	<code>echo date("H");</code> => 13
h	12 heure : format de l'heure en deux chiffres (01 à 12)	<code>echo date("h");</code> => 01
i	Minutes en deux chiffres (00 to 59)	<code>echo date("i");</code> => 22
s	Secondes en deux chiffres (00 to 59)	<code>echo date("s");</code> => 25
a	L'indication (am ou pm) en miniscule	<code>echo date("a");</code> => pm

Fonctions sur les dates

Aujourd'hui est le **14/03/2022 14:01:49**

Fonction	Signification	Exemples
date(format, ?timestamp)	formate une valeur timestamp en une date et une heure plus lisibles	<pre>echo date("d.m.Y H:i:s");</pre> → 14.03.2022 14:01:49
mktime(hour, minute, second, month, day, year)	Retourne la Valeur timestamp de la date en paramètre	<pre>\$d = mktime(11, 14, 54, 8, 12, 2022);</pre> <pre>echo "La date créée : " . date("d-m-Y h:i:sa", \$d);</pre> → La date créée : 12-08-2022 11:14:54am
strtotime	Créer une date à partir d'une chaîne	<pre>\$d = strtotime("15 April 2022 18:30:00");</pre> <pre>echo "Date créée : " . date("Y-m-d h:i:sa", \$d);</pre> → Date créée : 2022-04-15 06:30:00pm <pre>echo date("d/m/Y", strtotime("tomorrow"));</pre> → 15/03/2022 <pre>\$date1 = strtotime("Saturday");</pre> <pre>echo date("d/m/Y", strtotime("+6 months", \$date1));</pre> → 19/09/2022

Les exceptions

Si on exécute ce code:

```
<?php
$a = "test " * "7";
echo $a;
?>
```

L'erreur suivante est affichée à l'utilisateur:

```
Fatal error: Uncaught TypeError: Unsupported operand types: string * string
in C:\xampp\htdocs\exempleException\exception.php:28 Stack trace: #0
{main} thrown in C:\xampp\htdocs\exempleException\exception.php on
line 28
```

Pour éviter cette erreur, PHP permet de gérer ces exceptions de la façon suivante:

```
try {
    $a = "test " * "7";
    echo $a;
} catch (Error $e) {
    echo "Erreur : " . $e->getMessage();
}
```

```
Erreur : Unsupported operand types: string * string
```


Les exceptions - Exemple

```
function inverse($x)
{
    if (!$x)
        throw new Exception('Division par zéro.');
```

return 1 / \$x;

```
}
try {
    echo inverse(5) . "<br>";
    echo inverse(0) . "<br>";
} catch (Exception $e) {
    echo "Exception reçue : ". $e->getMessage(). "<br>";
} finally{
    echo "Toujours affiché";
}
echo "Suite de l'execution <br>";
```

Redirection entre pages -Exemple

- header : Envoie un en-tête HTTP brut
- Elle permet de spécifier l'en-tête HTTP string lors de l'envoi des fichiers HTML
- N'oubliez jamais que header() doit être appelée avant que le moindre contenu ne soit envoyé, soit par des lignes HTML habituelles dans le fichier, soit par des affichages PHP.

```
function redirect($chaine)
{
    header("Location:$chaine");
    exit();
}
```

Soient deux pages php: matin.php et soir.php:

```
$heure = date('H');
if ($heure <= 12)
    redirect("matin.php");
else
    redirect("soir.php");
```

Le champ « hidden »

- Un champ *input hidden* sert à stocker un résultat calculé du côté client et transférer sa valeur au serveur au moment du *submit* du formulaire.

- Exemple:

```
<?php
    if (!isset($_GET["compteur"]))
        $compt = 0;
    else {
        $compt = $_GET["compteur"];
        $compt++;
    }
    echo $compt;
?>

<form action="" method="get">
    <input type="hidden" name="compteur" value="<?php echo $compt; ?>">
    <input type="submit" value="ok" name="ok" />
</form>
```

Les tableaux

- Création à l'aide de la fonction **array()**
- Uniquement des tableaux à une dimension
 - Les éléments d'un tableau peuvent pointer vers d'autres tableaux
- Les éléments d'un tableau peuvent appartenir à des types distincts
- L'index d'un tableau en PHP commence par 0
- Pas de limites supérieures pour les tableaux
- La fonction **count()** pour avoir le nombre d'éléments d'un tableau

Les tableaux simples

Accéder aux éléments par l'intermédiaire de numéros

```
$tableau = array(valeur0, valeur1,..., valeurN);  
$tableau[indice] = valeur; // l'accès:
```

```
$note = array(10, 15, 12.6, 17, 10, 20, 11, 18, 19);  
$note[0] = 20;  
echo $note[1] + $note[5];
```

```
$jour = ["Dimanche", "Lundi", "Mardi", "", "Jeudi",  
"Vendredi", "Samedi"]; // on peut remplacer array par []  
$jour[3] = "Mercredi"; // Affecte mercredi à la 4ème case :  
$JJ = $jour[6]; // affecte "Samedi" à $JJ
```

Les tableaux associatifs

Les éléments sont référencés par des chaînes de caractères associatives en guise de nom: la clé d'index

```
$tableau = array(ind0 => val0, ind1 => val1,..., indN => valN);  
$tableau["clé"] = valeur; //accès  
$variable = $tableau["clé"]; //lecture
```

```
$jour = array("Dimanche" => 1, "Lundi" => 2, "Mardi" => 3,  
"Mercredi" => 4, "Jeudi" => 5, "Vendredi" => 6, "Samedi" => 7);  
$jour["Dimanche"] = 7;  
$jour["Mercredi"] = "Le jour des enfants";  
$JJ = $jour["Vendredi"]; //affecte 6 à $JJ  
echo $jour["Lundi"]; //retourne la valeur 2
```

Parcours d'un tableau associatif

Lecture des éléments d'un tableau

```
$tableau = array(clé1 => val1, clé2 => val2, ..., cléN => valN);  
foreach($tableau as $clé => $valeur) {  
    echo "Valeur ($clé): $valeur";  
}
```

```
$jour = array("Dimanche" => 7, "Lundi" => 1, "Mardi" => 2, "Mercredi" =>  
3, "Jeudi" => 4, "Vendredi" => 5, "Samedi" => 6);  
foreach($jour as $cle => $valeur) {  
    echo "Le jour de la semaine n° ". $valeur . " : " . $cle. "<br>";  
}
```


Les tableaux associatifs : Fonctions de tri

- La fonction **ksort()** trie les clés du tableau selon le critère des codes ASCII, et préserve les associations clé /valeur
- La fonction **krsort()** effectue la même action mais en ordre inverse des codes ASCII

```
$tab2 = array("1802" => "Hugo", "1920" => "Vian", "1622" => "Molière");  
ksort($tab2);  
echo "<h3 > Tri sur les clés de \$tab2 </h3>";  
foreach ($tab2 as $cle => $valeur) {  
    echo " l'élément a pour clé <b>: $cle:</b> et pour valeur :<b> $valeur</b>  
<br>";  
}
```

//Résultat:

l'élément a pour clé : **1622:** et pour valeur : **Molière**

l'élément a pour clé : **1802:** et pour valeur : **Hugo**

l'élément a pour clé : **1920:** et pour valeur : **Vian**

Les tableaux et les chaînes de caractères

Explode(\$separator, \$string)	Retourne un tableau de sous chaînes en découpant une chaîne de caractères en segments à l'aide d'un séparateur	<pre>\$tab = explode(" ", "Bonjour tout le monde"); var_dump(\$tab); ➔ array(4) { [0]=> string(7) "Bonjour" [1]=> string(4) "tout" [2]=> string(2) "le" [3]=> string(5) "monde" }</pre>
implode	Joindre les éléments d'un tableau avec une chaîne	<pre>\$arr = array('Hello', 'World!', 'Beautiful', 'Day!'); var_dump(implode(" ", \$arr)); ➔ Hello World! Beautiful Day!</pre>

66

Les fonctions de tableaux

Soient les deux tableaux suivants:

```
$mois1 = ["Janvier" => 31, "Février" => 28, "Mars" => 31];
$mois2 = ["Mars" => 31, "Avril" => 30, "Mai" => 31];
```

array_diff_key	Calcule la différence de deux tableaux en utilisant les clés pour comparaison	<pre>print_r(array_diff_key(\$mois1, \$mois2)); ➔ Array([Janvier] => 31 [Février]= 28) print_r(array_diff_key(\$mois2, \$mois1)); ➔ Array ([Avril] => 30 [Mai] => 31)</pre>
array_diff	Calcule la différence entre des tableaux	<pre>print_r(array_diff(\$mois1, \$mois2)); ➔ Array ([Février] => 28) print_r(array_diff(\$mois2, \$mois1)); ➔ Array ([Avril] => 30)</pre>
array_fill_keys	Remplit un tableau avec des valeurs, en spécifiant les clés	<pre>print_r(array_fill_keys(["a", "b", "c"], 0)); ➔ Array ([a] => 0 [b] => 0 [c] => 0)</pre>
array_key_first	Récupère la première clé d'un tableau	<pre>echo array_key_first(\$mois1); ➔ Janvier</pre>

Les fonctions de tableaux

```
$mois1 = ["Janvier" => 31, "Février" => 28, "Mars" => 31];
$mois2 = ["Mars" => 31, "Avril" => 30, "Mai" => 31];
```

array_chunk	Sépare un tableau en tableaux de taille inférieure	<pre>\$arr = ["a", "b", "c", "d", "e"]; print_r(array_chunk(\$arr, 2));</pre> <p>→</p> <pre>Array([0] => Array([0] => a [1] => b) [1] => Array([0] => c [1] => d) [2] => Array([0] => e))</pre>
array_merge	Fusionne plusieurs tableaux en un seul	<pre>print_r(array_merge(\$mois1, \$mois2));</pre> <p>→</p> <pre>Array ([Janvier] => 31 [Février] => 28 [Mars] => 31 [Avril] => 30 [Mai] => 31)</pre>
array_search	Recherche dans un tableau la première clé associée à la valeur	<pre>echo array_search(31, \$mois2);</pre> <p>→</p> <pre>Mars</pre>

Opérateurs de tableaux

Exemple	Nom	Résultat
<code>\$a + \$b</code>	Union	Union de \$a et \$b.
<code>\$a == \$b</code>	Égalité	true si \$a et \$b contiennent les mêmes paires clés/valeurs.
<code>\$a === \$b</code>	Identique	true si \$a et \$b contiennent les mêmes paires clés/valeurs dans le même ordre et du même type.
<code>\$a != \$b</code>	Inégalité	true si \$a n'est pas égal à \$b.
<code>\$a <> \$b</code>	Inégalité	true si \$a n'est pas égal à \$b.
<code>\$a !== \$b</code>	Non-identique	true si \$a n'est pas identique à \$b.

Manipulation des fichiers: Ouverture d'un fichier

Lors de l'ouverture du fichier il faut préciser pour quelle opération celui-ci sera utilisé, Entrée, Sortie ou Ajout.

```
$ressource=fopen ("filename", "mode");
```

Exemple: `$idFile = fopen("test.txt", "w");`

mode	Description
'r'	Ouvre en lecture seule et place le pointeur de fichier au début du fichier.
'r+'	Ouvre en lecture et écriture et place le pointeur de fichier au début du fichier.
'w'	Ouvre en écriture seule ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
'w+'	Ouvre en lecture et écriture ; le comportement est le même que pour 'w'.
'a'	Ouvre en écriture seule ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer. Dans ce mode, la fonction fseek() n'a aucun effet, les écritures surviennent toujours.
'a+'	Ouvre en lecture et écriture ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer. Dans ce mode, la fonction fseek() n'affecte que la position de lecture, les écritures surviennent toujours.

Manipulation des fichiers: Ecriture

Exemple1: Création d'un nouveau fichier avec le mode « w »

```
$idFile = fopen("test.txt", "w");  
fwrite($idFile, "Bonjour c'est le " . date("d/m/Y H:i:s"));  
fclose($idFile);
```

Exemple2: Modification d'un fichier avec le mode « a »

```
if (file_exists("test.txt")) {  
    $idFile = fopen("test.txt", "a");  
    fwrite($idFile, "\nNouvelle ligne");  
    fclose($idFile);  
}
```

//A chaque exécution de ce code, la chaine « Nouvelle ligne » est ajoutée à la fin de ce fichier.

Manipulation des fichiers : Lecture

► Lecture avec fread:

```
if (file_exists("test.txt")) {  
    $idFile = fopen("test.txt", "r");  
    echo fread($idFile, filesize("test.txt")); //filesize est une fonction retournant la  
    taille du fichier  
    fclose($idFile);  
}
```

► Lecture ligne par ligne avec fgets:

```
if (file_exists("test.txt")) {  
    $idFile = fopen("test.txt", "r");  
    while(!feof($idFile)){  
        echo fgets($idFile);  
    }  
    fclose($idFile);  
}
```

//Le programme ne lit que la première ligne du fichier "test.txt"

Manipulation des fichiers : Fonctions

fread()

Lecture du fichier en mode binaire

file()

Lit le fichier et renvoie le résultat dans un tableau

file_get_contents

Lit tout un fichier dans une chaîne

fstat

Lit les informations sur un fichier à partir d'un pointeur de fichier

fgets()

Récupère la ligne courante à partir de l'emplacement du pointeur sur fichier

fgetc

Lit un caractère dans un fichier

readfile

Affiche un fichier

Autre opérations sur un fichier

Liste : <https://www.php.net/manual/fr/ref.filesystem.php>

fclose

Ferme un fichier

fwrite

Écrit un fichier en mode binaire

fputs

Alias de fwrite

file_put_contents

Écrit des données dans un fichier

Chargement des fichiers

Un formulaire de téléchargement de fichiers peut être construit en créant un formulaire spécifique comme ceci :

```
<!-- Le type d'encodage des données, enctype, DOIT être spécifié comme ce qui suit -->
<form action="" method="post" enctype="multipart/form-data">
  <!-- MAX_FILE_SIZE doit précéder le champ input de type file -->
  <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
  <!-- Le nom de l'élément input détermine le nom dans le tableau $_FILES -->
  Choisissez un fichier <input type="file" name="myFile" />
  <br>
  <input type="submit" value="ENVOI" />
</form>
```

Le champ caché `MAX_FILE_SIZE` (mesuré en octets) doit précéder le champ `input` de type `file` et sa valeur représente la taille maximale acceptée du fichier par PHP.

Cet élément de formulaire doit toujours être utilisé, car il permet d'informer l'utilisateur que le transfert désiré est trop lourd avant d'atteindre la fin du téléchargement.

Notes de configuration

Voir aussi les directives [file uploads](#), [upload max filesize](#), [upload tmp dir](#), [post max size](#) et [max input time](#) dans `php.ini`

Chargement des fichiers

Après avoir soumis le formulaire, la variable globale `$_FILES` contiendra toutes les informations sur le fichier téléchargé.

<code>\$_FILES["myFile"]["name"]</code>	Le nom original du fichier, tel que sur la machine du client web.
<code>\$_FILES["myFile"]["type"]</code>	Le type MIME du fichier, si le navigateur a fourni cette information. Par exemple, cela pourra être "image/gif". Ce type mime n'est cependant pas vérifié du côté de PHP et, donc, ne prend pas sa valeur pour se synchroniser.
<code>\$_FILES["myFile"]["size"]</code>	La taille, en octets, du fichier téléchargé.
<code>\$_FILES["myFile"]["tmp_name"]</code>	Le nom temporaire du fichier qui sera chargé sur la machine serveur.
<code>\$_FILES["myFile"]["error"]</code>	Le <u>code d'erreur</u> associé au téléchargement de fichier.

Le fichier téléchargé sera stocké temporairement dans le dossier temporaire du système, à moins qu'un autre dossier soit fourni avec la directive upload_tmp_dir du `php.ini`.

Chargement des fichiers

move_uploaded_file(string *\$source*, string *\$destination*): bool

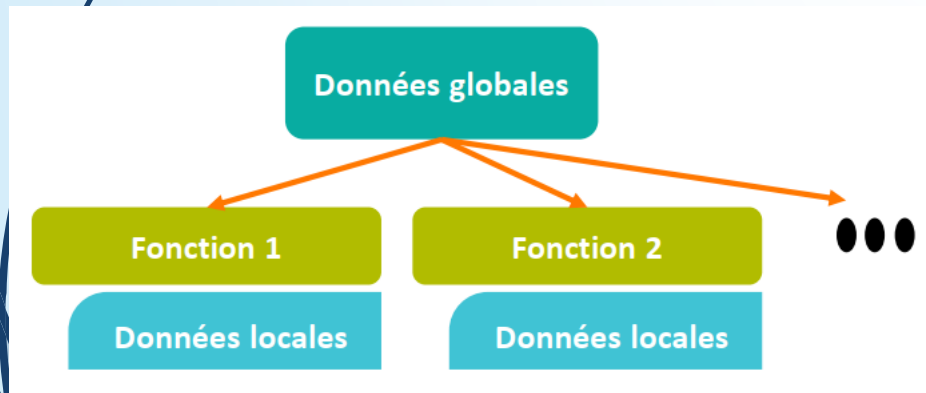
S'assure que le fichier *\$source* est un fichier téléchargé par HTTP POST. Si le fichier est valide, il est déplacé jusqu'à *\$destination*.

```
<?php
if (isset($_FILES['myFile']))
{
    //Enregistrement et renommage du fichier
    $nom = "uploaded_" . $_FILES["myFile"]["name"];
    $result = move_uploaded_file($_FILES["myFile"]["tmp_name"], $nom);
    if ($result == TRUE) {
        echo "<b>Vous avez bien transféré le fichier</b><br>";
        echo "Le nom du fichier est : ", $_FILES["myFile"]["name"], "<br>";
        echo "Votre fichier a une taille de ", $_FILES["myFile"]["size"], "<br>";
    } else {
        echo "<hr /> Erreur de transfert n°", $_FILES["myFile"]["error"];
    }
}
?>
```

Intérêt de programmer en Orienté Objet

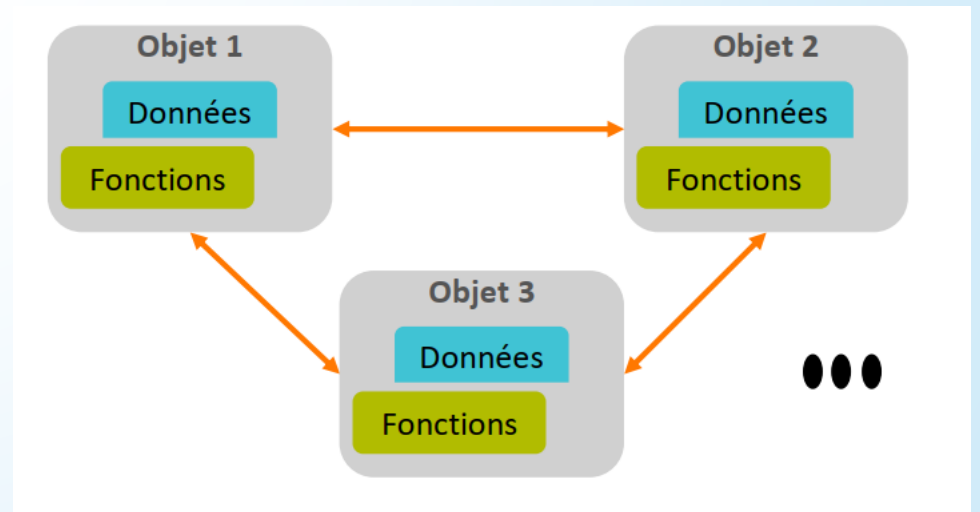
Programmation procédurale

- Le programme est divisé en procédures ou fonctions.
- Chaque fonction contient des données différentes.
- Une procédure permet d'effectuer des opérations sur les données généralement contenues dans des variables.
- Les opérations sont exécutées selon leur ordre d'écriture dans le script.



Programmation orientée objet

- Le programme est divisé en objets.
- Les objets agissent comme une seule unité.
- Un objet est une entité qui va pouvoir contenir un ensemble de fonctions et de variables.
- Regrouper des tâches précises au sein d'objets pour obtenir une nouvelle organisation du code



Intérêt de programmer en Orienté Objet

Programmation procédurale

- **Entretien du programme complexe** : puisque le programme s'exécute séquentiellement du début à la fin, le fait de vouloir modifier son comportement (même légèrement) revient à faire des modifications (parfois majeures) dans plusieurs endroits.
- **Avancement lent dans les grandes applications**
- **Travail en groupe non favorable**

Programmation orientée objet

- **Plus intuitif** : La POO permet de manipuler des objets qui peuvent interagir entre eux. Ce concept est plus proche du monde physique où nous vivons.
- **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 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 qui sont faciles à regrouper après.

Concepts de base de la POO en PHP : Définition de classe

- Les **objets** possèdent des propriétés et des comportements propres.
- Une même famille d'objets, c'est-à-dire avec des caractéristiques communes, peut être regroupée dans un ensemble appelé **classe**.
- Une **classe** permet de regrouper au sein d'une même structure **le format des données** qui définissent l'objet et **les fonctions** destinées à manipuler ces objets.

```
<?php
class voiture
{
    //Attributs
    private $couleur;
    private $kilometrage;
    //Méthodes
    public function getKilometrage()
    {
        return $this->kilometrage;
    }
    public function setCouleur($couleur)
    {
        $this->couleur = $couleur;
    }
    public function rouler($distance)
    {
        $this->kilometrage+=$distance;
    }
}??>
```


Concepts de base de la POO en PHP : Déclaration d'un objet

Un **objet** : un élément particulier d'une classe, est appelé **instance d'une classe**.

La déclaration d'un objet revient à instancier une classe avec **new** :

```
//Déclaration d'une instance  
$renault = new voiture();
```

```
//Utilisation  
$renault->rouler(10);  
echo "Kilométrage : " . $renault->getKilometrage() . "<br>";
```

```
//Vérification  
if (is_a($renault, "voiture")) {  
    echo "renault est un objet de la classe voiture";  
}
```

Résultat:

Kilométrage : 10

renault est un objet de la classe voiture

Concepts de base de la POO en PHP : L'encapsulation

L'encapsulation est un principe fondamental de la POO. Il vise à masquer les attributs aux utilisateurs du code.

C'est la visibilité `private` ou `protected` qui est recommandée.

Types de déclaration:

- **public** : l'attribut ou la méthode sera accessible de l'intérieur de la classe dont il est membre comme de l'extérieur.
- **private** : dans ce cas, l'attribut ou la méthode est accessible seulement de l'intérieur de la classe dont il est membre.
- **protected** : dans ce cas, l'attribut ou la méthode est accessible seulement de l'intérieur de la classe dont il est membre ainsi que des classes qui héritent de cette classe.

Concepts de base de la POO en PHP : Membres statiques

En PHP il est possible de créer des propriétés de classe. Leur valeur est la même pour toutes les instances d'une même classe.

Pour déclarer une telle propriété, on utilise le mot-clé **static**.

```
class Bourse
{
    //attribut statique
    public static $lieu = "Fès";

    .....
    //méthode statique
    public static function infosGenerales()
    {
        $heure = date("H:i:s");
        $texte = self::$lieu . ", il est " . $heure;
        return $texte;
    }
}

echo Bourse::$lieu . "<br>"; // Fès
echo Bourse::infosGenerales(); // Fès, il est 17:17:40
```

Concepts de base de la POO en PHP : Le Constructeur

Quand une instance d'une classe d'objet est créée au moment de **l'instanciation** d'une variable avec **new**, une fonction particulière est exécutée. Cette fonction s'appelle le **constructeur**. Elle permet d'initialiser chaque instance pour que ses propriétés aient un contenu cohérent.

En PHP, cette méthode spéciale est appelé **__construct**, et est appelée automatiquement au moment de la création d'un objet.

```
public function __construct(args, ...)  
{  
    //code  
}
```

Concepts de base de la POO en PHP : Le Constructeur

Exemple:

```
//Définition de la classe
class voiture
{
    //Attributs
    private string $couleur;
    private int $kilometrage;
    //constructeur
    public function __construct(string $couleur,
int $kilometrage)
    {
        $this->couleur = $couleur;
        $this->kilometrage = $kilometrage;
    }
    public function afficher()
    {
        return "Voiture => Couleur : " . $this-
>couleur . ", Kilométrage : " . $this->kilometrage;
    }
}
```

//Utilisation:

```
$v1 = new voiture("Rouge", 1000);
echo $v1->afficher();
```

Résultat :

Voiture => Couleur : Rouge, Kilométrage : 1000

```
$v2 = new voiture();
echo $v2->afficher();
```

Résultat :

Fatal error: Uncaught ArgumentCountError:
Too few arguments to function
voiture::__construct(),

Concepts de base de la POO en PHP : Le Constructeur

Exemple: Initialiser tous ou une partie des attributs

```
//Définition de la classe
class voiture
{
    //Attributs
    private string $couleur;
    private int $kilometrage;
    //constructeur
    public function __construct(string $couleur =
    "Rouge", int $kilometrage = 0)
    {
        $this->couleur = $couleur;
        $this->kilometrage = $kilometrage;
    }
    public function afficher()
    {
        return "Voiture => Couleur : " . $this-
        >couleur . ", Kilométrage : " . $this->kilometrage;
    }
}
```

```
//Utilisation:
$v1 = new voiture("Bleu", 1000);
echo $v1->afficher();
```

Résultat :

Voiture => Couleur : Bleu, Kilométrage : 1000

```
$v2 = new voiture();
echo $v2->afficher();
```

Résultat :

Voiture => Couleur : Rouge, Kilométrage : 0

```
$v3 = new voiture(kilometrage: 5000);
echo $v3->afficher();
```

Résultat :

Voiture => Couleur : Rouge, Kilométrage : 5000

Concepts de base de la POO en PHP : Le Destructeur

- Le destructeur efface un objet lorsqu'il n'est plus nécessaire.
- Cette méthode est utilisée pour nettoyer les ressources et libérer la mémoire pour en accueillir davantage.
- Un et un seul destructeur peut être exister dans la même classe.
- On peut forcer la destruction en utilisant la fonction `unset($MonObjet)`.

```
class voiture
{
    private $marque;
    public function __construct()
    {
        $this->marque = "Renault";
    }
    public function __destruct()
    {
        echo "Destruction ... " . $this->marque . " de la classe " . __CLASS__;
    }
}
$v1 = new voiture();
```

Résultat :

Destruction ... Renault de la classe voiture

Concepts de base de la POO en PHP : L'héritage

L'héritage consiste en la création d'une nouvelle classe dite ***classe dérivée*** à partir d'une classe existante dite ***classe de base*** ou ***classe parente***.

L'héritage permet de :

- **recupérer** le comportement standard d'une classe d'objet (classe parente) à partir de propriétés et des méthodes définies dans celles-ci,
- **ajouter** des fonctionnalités supplémentaires en créant de nouvelles propriétés et méthodes dans la classe dérivée,
- **modifier** le comportement standard de la classe parente en surchargeant certaines méthodes de la classe parente dans la classe dérivée.

Concepts de base de la POO en PHP : L'héritage

```
class animal
{
    protected $poids;
    protected $age;

    public function respirer()
    {
        echo "Je respire!!<br>";
    }
}
```

```
class chien extends animal
{
    public function aboyer()
    {
        echo "Wooaf!!<br>";
    }
}
```

```
$a = new animal();
$a->respirer();
```

Résultat :

Je respire!!

```
$c = new chien();
$c->respirer();
$c->aboyer();
```

Résultat :

Je respire!!

Wooaf!!

Concepts de base de la POO en PHP : La redéfinition

Exemple 1:

```
class animal
{
    protected $poids;
    protected $age;
    public function __construct($poids, $age)
    {
        $this->poids = $poids;
        $this->age = $age;
    }
    public function sePresenter()
    {
        echo "Je suis un animal, j'ai $this->age mois
et je pèse $this->poids kgs.<br>";
    }
}
```

```
class chien extends animal
{
    public function sePresenter()
    {
        parent::sePresenter();
        echo "Je suis un chien";
    }
}
```

```
$a = new animal(6, 10);
$a->sePresenter();
```

Résultat:

Je suis un animal, j'ai 10 mois et je pèse 6 kgs .

```
$c = new chien(6, 10);
$c->sePresenter();
```

Résultat:

Je suis un animal, j'ai 10 mois et je pèse 6 kgs .
Je suis un chien

Concepts de base de la POO en PHP : La redéfinition

Exemple 2:

```
class animal
{
    protected $poids;
    protected $age;
    public function __construct($age, $poids)
    {
        $this->age = $age;
        $this->poids = $poids;
    }
}
```

```
class chien extends animal
{
    private $race;
    public function __construct($age, $poids, $race)
    {
        parent::__construct($age, $poids);
        $this->race = $race;
    }
}
```

Concepts de base de la POO en PHP : Organisation des fichiers

animal.class.php

```
<?php
class animal
{
    protected $poids;
    protected $age;
    public function __construct($poids,
    $age)
    {
        $this->poids = $poids;
        $this->age = $age;
    }
}
?>
```

main.php

```
//Ici on appelle le fichier
"animal.class.php" ou la classe animal
est définie

require_once("animal.class.php");
$a = new animal(6, 10);
```

Les instructions **include** et **require** sont identiques, sauf en cas d'échec (fichier non existant) :

- **require** produira une erreur fatale (E_COMPILE_ERROR) et arrêtera le script
- **include** ne produira qu'un avertissement (E_WARNING) et le script continuera

Concepts de base de la POO en PHP : Auto chargement de classes

Soit la fonction suivante qui permet d'inclure les fichiers de classes (*il faut placer toutes les pages qui contiennent des définitions des classes au même emplacement*).

```
<?php
function appel($classe)
{
    include($classe . ".class.php");
}
?>
```

Fonction `spl_autoload_register()`

La fonction `spl_autoload_register()` enregistre une fonction de notre choix dans la pile d'autoload. Cette pile est destinée à enregistrer des fonctions et les appeler, d'une manière implicite, quand on en aura besoin. De cette manière, si on fait appel à une classe (instanciation, héritage ou appel statique d'un membre) l'autoload se charge d'appeler la fonction qui permet d'inclure la classe souhaitée.

Pour enregistrer la fonction « **appel** » dans la pile d'autoload on fait comme ceci:

```
<?php
    spl_autoload_register("appel");
?>
```

Concepts de base de la POO en PHP : Les méthodes magiques

Les méthodes magiques sont des méthodes prédéfinies et toutes préfixées par double sous-tirets (__) dans une classe PHP. Elles sont appelées automatiquement suite à un événement spécial qui peut survenir lors de l'exécution.

- **__get()** :

Est appelée pour lire des données depuis des propriétés inaccessibles (protégées ou privées) ou non existante.

Syntaxe : `public __get(string $name): mixed`

```
class voiture
{
    //Attributs
    private int $kilometrage;
}
$v = new voiture();
echo $v->kilometrage;
```

Résultat:

Fatal error: Uncaught Error: Cannot access private property voiture::\$kilometrage

```
class voiture
{
    //Attributs
    private int $kilometrage;
    public function __get($name)
    {
        echo "L'attribut $name n'est pas accessible";
    }
}
$v = new voiture();
echo $v->kilometrage;
```

Résultat:

L'attribut kilometrage n'est pas accessible

Concepts de base de la POO en PHP : Les méthodes magiques

- **__set()** :

Est sollicitée lors de l'écriture de données vers des propriétés inaccessibles (protégées ou privées) ou non existante.

Syntaxe : `public __set(string $name, mixed $value): void`

```
class voiture
{
    private int $kilometrage;
}
$v = new voiture();
$v->kilometrage = 1000;
```

Résultat:

Fatal error: Uncaught Error: Cannot access private property voiture::\$kilometrage

```
class voiture
{
    private int $kilometrage;
    public function __set($name, $value)
    {
        echo "Impossible d'affecter la valeur
$value à l'attribut $name : propriété
inaccessible!";
    }
}
$v = new voiture();
$v->kilometrage = 1000;
```

Résultat:

Impossible d'affecter la valeur 1000 à l'attribut kilometrage : propriété inaccessible!

Concepts de base de la POO en PHP : Les méthodes magiques

__isset() :

- Est sollicitée lorsque `isset()` ou `empty()` sont appelées sur des propriétés inaccessibles (protégées ou privées) ou non existante.

__unset() :

- Est invoquée lorsque `unset()` est appelée sur des propriétés inaccessibles (protégées ou privées) ou non

```
class voiture
{
    private int $kilometrage;
    public function setKilometrage($kilometrage)
    {
        $this->kilometrage = $kilometrage;
    }
    public function __isset($name)
    {
        return isset($this->$name);
    }
    public function __unset($name)
    {
        unset($this->$name);
    }
}
```

```
$v = new voiture();
$v->setKilometrage(1000);
var_dump(isset($v->kilometrage));
//=> bool(true)
unset($v->kilometrage);
//suppression de l'attribut
var_dump(isset($v->kilometrage));
//=> bool(false)
```

Concepts de base de la POO en PHP : Les méthodes magiques

__toString() :

- Elle est appelée automatiquement quand on tente de traiter un objet en tant que chaîne de caractères (à l'aide de la structure echo par exemple). Dans ce cas, cette méthode retourne la chaîne de caractères de notre choix.

```
class voiture
{
    private string $marque;
    private int $kilometrage;
    public function __construct(int $kilometrage, string
$marque)
    {
        $this->kilometrage = $kilometrage;
        $this->marque = $marque;
    }
    public function __toString()
    {
        return "La voiture est de la marque $this-
>marque et elle a roulé $this->kilometrage kilomètres.";
    }
}
```

```
$v = new voiture(10000, "Renault");
echo $v;
```

Résultat:

La voiture est de la marque Renault et elle a roulé 10000 kilomètres.

Concepts de base de la POO en PHP : Les méthodes magiques

__serialize() :

Retourne les noms de propriétés privées des classes parentes.

Construit et retourne un tableau associatif de paire clé/valeur qui représente la forme linéarisée de l'objet

```
class voiture
{
    private string $marque;
    private int $kilometrage;
    public function __construct(int $kilometrage, string $marque)
    {
        $this->kilometrage = $kilometrage;
        $this->marque = $marque;
    }
    public function __serialize(): array
    {
        return array("marque" => $this->marque,
            "kilometrage" => $this->kilometrage);
    }
}
```

```
$v = new voiture(10000, "Renault");

$file = fopen("voiture.txt", "w");
fwrite($file, serialize($v));
fclose($file);
```

Le fichier voiture.txt

```
O:7:"voiture":2:{s:6:"marque";s:7:"Renault";s:11:"kilometrage";i:10000;}
```

Concepts de base de la POO en PHP : Les méthodes magiques

__unserialize() :

unserialize() vérifie la présence d'une fonction avec le nom magique __unserialize(). Si c'est le cas, cette fonction prendra en paramètre le tableau restauré qui a été retournée depuis serialize(). Il peut alors restaurer les propriétés de l'objet depuis ce tableau comme approprié.

```
class voiture
{
    private string $marque;
    private int $kilometrage;
    .....
    public function __toString()
    {
        return "Voiture: Marque : <b>$this->marque </b>, Kilométrage
        :<b>$this->kilometrage</b> kilomètres.";
    }
    public function __serialize(): array
    {
        return array("marque" => $this->marque, "kilometrage" =>
        $this->kilometrage);
    }
    public function __unserialize(array $data): void
    {
        $this->marque = $data["marque"];
        $this->kilometrage = $data["kilometrage"];
    }
}
```

```
$file = fopen("voiture.txt", "r");
$lecture = fgets($file);
$nvVoiture = unserialize($lecture);
echo $nvVoiture;
fclose($file);
```

Résultat:

Voiture: Marque : **Renault** , Kilométrage
:10000 kilomètres.

Concepts de base de la POO en PHP : Les méthodes magiques

__clone(): Une **copie** d'objet est créée en utilisant le mot-clé clone (qui fait appel à la méthode **__clone()** de l'objet, si elle a été définie).

```
class voiture
{
    private string $marque;
    private int $kilometrage;
    public function __construct(int $kilometrage, string $marque)
    {
        $this->kilometrage = $kilometrage;
        $this->marque = $marque;
    }
    public function setKilometrage($kilometrage)
    {
        $this->kilometrage = $kilometrage;
    }
    public function __toString()
    {
        return "Voiture: Marque : <b>$this->marque </b>,
        Kilométrage :<b>$this->kilometrage</b> kilomètres.";
    }
    public function __clone()
    {
        $this->kilometrage = $this->kilometrage;
        $this->marque = "copie " . $this->marque;
    }
}
```

```
$v1 = new voiture(10000, "Renault");
$v2 = clone $v1;
echo "v2 : " . $v2 . "<br>";
```

Résultat:

v2 : Voiture: Marque : **copie Renault** , Kilométrage :**10000** kilomètres.

```
$v2->setKilometrage(5000);
echo "v2 : " . $v2 . "<br>";
echo "v1 : " . $v1 . "<br>";
```

Résultat:

- v2 : Voiture: Marque : **copie Renault** , Kilométrage :**5000** kilomètres.

- v1 : Voiture: Marque : **Renault** , Kilométrage :**10000** kilomètres.

Concepts de base de la POO en PHP : Les méthodes magiques

__sleep() :

Elle peut nettoyer l'objet, et elle est supposée retourner un tableau avec les noms de toutes les variables de l'objet qui doivent être linéarisées.

Syntaxe : `public __sleep(): array`

Si `__serialize()` et `__sleep()` sont tous les deux définies dans le même objet, alors seulement `__serialize()` sera appelé.

__wakeup() :

Cette fonction peut reconstruire toute ressource que l'objet pourrait posséder.

Son but est de rétablir toute connexion de base de données qui aurait été perdue durant la linéarisation et d'effectuer des tâches de réinitialisation.

Syntaxe : `public __wakeup(): void`

Si `__unserialize()` et `__wakeup()` sont tous les deux définies dans le même objet, alors seulement `__unserialize()` sera appelée.

__call() :

- Est appelée lorsque l'on invoque des méthodes inaccessibles dans un contexte objet.

Syntaxe: `public __call(string $name, array $arguments): mixed`