

ISIG : Langage de programmation PHP (60 H)

- Introduction.
- Utilitaires.
- Syntaxe du php.
- Les Tableaux.
- Les Fonctions.
- Les Formulaire.
- Gestion des fichiers.
- Gestion de base de données.
- Programmation Orientée Objet en PHP

Introduction

Historique

L'apparition du Php a été aux années 90, un certain Rasmus Lerdorf avait placé une grosse macro sur sa page personnelle (Personal Home Page) afin de garder trace de ses visiteurs. À l'époque ce script ne permettait pas encore de réaliser de boucles ni de tests mais déjà exploitait les requête SQL. Le succès immédiat de Php auprès d'une communauté de développeurs a permis l'apparition de la version 3 en 1998.

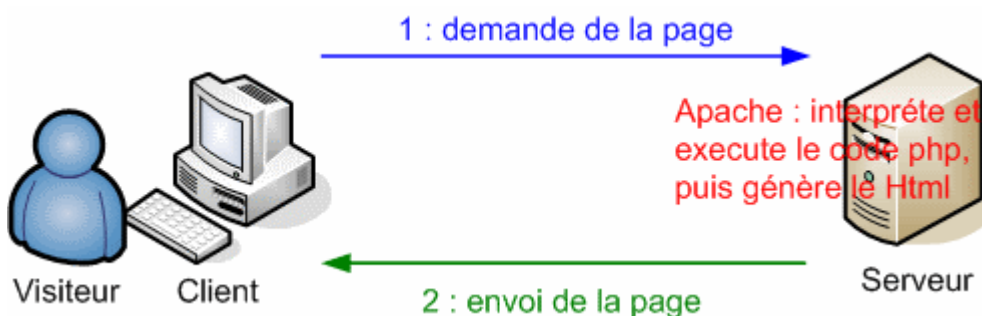
Le php est présenté par la mascotte :



Un script Php peut analyser les données d'un formulaire HTML, communiquer avec une base de données et effectuer des calculs à la volée. Le code source de Php est compilé avec le serveur Apache. Php est gratuit, modifiable (open source), orienté exclusivement Web et très documenté.

Comment ça marche ?

Les balises HTML sont interprétées par le navigateur alors que dans le cas du PHP, le code est interprété puis exécuté par le serveur. Lorsque vous demandez à votre navigateur d'afficher le code source, le serveur ne lui ayant expédié que le résultat au format HTML, vous ne verrez pas le code PHP. Les balises PHP ne peuvent plus être lues. Seul le résultat l'est.



Pour utiliser du PHP dans vos pages, il vous faut :

- Vous assurer que le serveur sur lequel vos pages sont hébergées interprète le PHP.
- Donner à vos pages une extension .php ou selon la configuration du serveur.
- Insérer votre PHP dans le code HTML grâce aux balises réservées. On utilise <?php ?> le plus fréquemment. Voici les balises qui caractérisent une zone de création d'un script :

1. <?.....?> : Balise classique.
2. <?php.....?> : Balise évolué.
3. <%.....%> : balise peu connue.

Utilitaires.

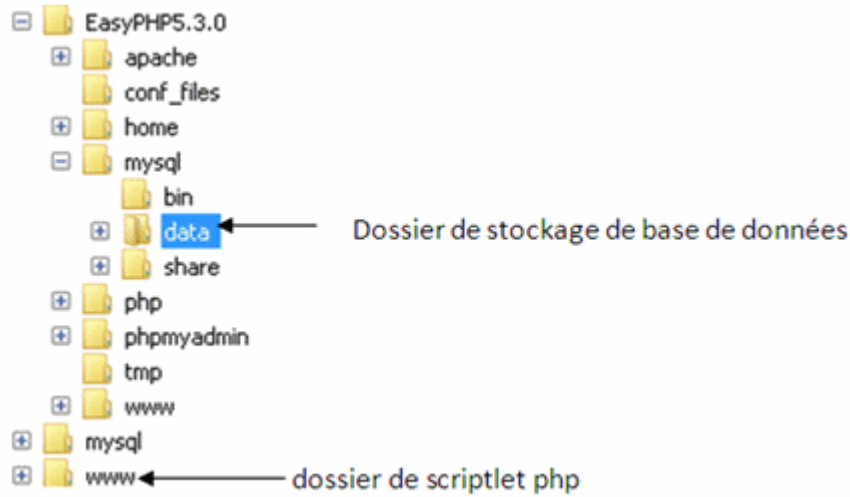
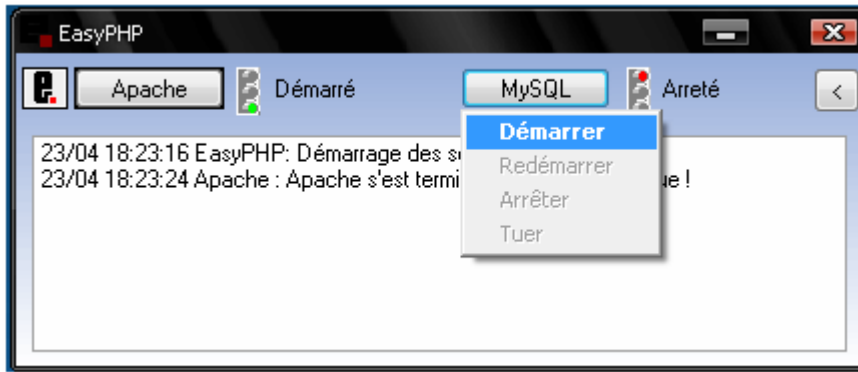
Programmer en PHP nécessite quelques programmes et utilitaires spéciaux pour débiter:

- - **Editeur PHP** : Vous pouvez créer des pages avec Notepad, mais des logiciels gratuits et plus professionnels sont téléchargeables sur Internet. ConText, un freeware (gratuit) utilisable en plusieurs langues. HydraPHP, PHPedit, ... sont aussi possibles.
- - **Serveur** : Une page en PHP doit être décodée par un serveur et ... votre ordinateur sous Windows ne fonctionne pas comme serveur php. Pour tester vos programmes en PHP - MySQL, vous pouvez télécharger logiciel [easyphp](#). Il fonctionne sous Windows et permet non seulement de tester des pages en local, mais également de créer des bases de données locales Mysql. EasyPhp n'est pas seulement un logiciel de test, c'est une réelle application serveur qui peut vous permettre d'héberger votre site Internet sur votre ordinateur en local, mais attention à la sécurité.

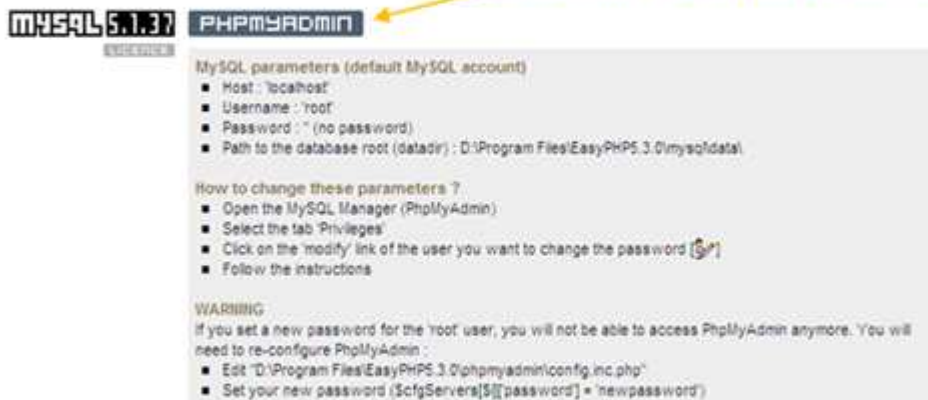
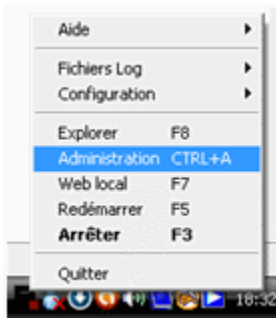
Ces 2 utilitaires vous permettent finalement d'utiliser votre PC sous Windows pour créer et tester des pages programmées en PHP, pour les retransférer vers votre site Internet ensuite.

Installation

Télécharger EasyPHP, l'installation est facile on cliquant sur suivant de l'assistant, spécifier le répertoire où vous voulez garder vos projets, la fin de l'installation une petite icône apparaisse dans la barre des tâches, cliquer sur laquelle par le bouton droite du souris, une fenêtre s'affiche permet de contrôler l'état de deux serveurs.



Le clique du bouton droite de souris sur l'icone de barre de tâche de easyphp, une boite de contexte apparaisse.



Syntaxe du php.

Le PHP est un langage de programmation spécialisé dans la génération de code HTML (notamment). Les balises PHP peuvent être insérées directement dans le code html de la page qui a l'extention (~.php). Toutes les instructions php doivent se terminer par ';' Si vous l'oubliez vous aurez un message d'erreur de type: Parse error: parse error, unexpected T_PRINT in c://prgramFile/easyphp/www/tests.php on line x

Mes premier commandes php.

```
<?php

/*Mes Premiers Commandes Php*/

echo "Salam ! ça va?"; /*La commande echo permet plus particulièrement
d'afficher du texte (entre "" ou '') ou une variable*/

print("Salam ! ça va?"); /*print est similaire a echo, les parenthèses
n'est pas obligatoire .*/

echo 'J'ai réussi ma première commande en PHP'; /*Il faut insérer le
```

```
caractère avant le guillemet*/
echo 'Bonjour ' . "Tout le monde"; /*Le . sert a concaténer 2 chaine de
caractères*/
echo "<h1>Inclure de Code Html dedans</h1>"; /*Le . sert a concaténer 2
chaîne de caractères*/
/*Les Commentaires*/
//C'est un commentaire dans une seule ligne
/*C'est un commentaire
dans une plusieurs lignes*/
# ceci est un commentaire à la Unix.
```

?>

Les variables.

L'un des principaux intérêts du PHP est de mémoriser des informations dans le but de les stocker ou de les traiter, ces informations (données) sont stockées temporairement dans des variable, chaque vaiable possède un nom et une valeur, les variables existent tant que la page est en cours de génération . Le PHP accepte 6 types de variables :

- **Booléen** : True, False (vrai ou faux).
- **Entier** : un nombre entier (sans virgules).
- **Chiffre en virgule flottante** : (c'est automatiquement un nombre décimal de type double), nombre avec des chiffrée derrière la virgule comme 0.23, 45.2369, ... Remarquez le point comme séparateur.
- **Chaîne de caractères** : du texte encadré par ' , exemple: 'Ceci est un texte'
- **Tableaux** : array(2,2,2).
- **Objets** : (image, ...).

PHP ne demande pas de spécifications du type de variable préalable, ni même de les déclarer (sauf les tableaux).

Les noms de variable doivent:

- commencer par \$
- inclure des lettres, des chiffres et le caractère _
- commencer par une lettre ou _

Les noms de variables ne doivent pas:

- inclure les caractères réservés - @ , . ; : /<>
- inclure des espaces

Quelques remarques pour le nom:

- Il peut inclure des caractères accentués mais ce n'est pas souhaitable.

- le nom de la variable est spécifique à la casse (majuscules, minuscules)

Les Variables prédéfinies

En effet, PHP propose toute une série de variables qui sont déjà présentes dans le langage sans que vous n'ayez à les déclarer. Ces variables s'écrivent toujours en majuscules et nous fournissent divers renseignements.

Voici la liste des variables d'environnement existantes :

Opérateur	Description
<code>\$_SERVER['DOCUMENT_ROOT']</code>	Racine du serveur
<code>\$_SERVER['HTTP_ACCEPT_LANGUAGE']</code>	Langage accepté par le navigateur
<code>\$_SERVER['HTTP_HOST']</code>	Nom de domaine du serveur
<code>\$_SERVER['HTTP_USER_AGENT']</code>	Type de navigateur
<code>\$_SERVER['REMOTE_ADDR']</code>	Adresse IP du client
<code>\$_SERVER['REMOTE_PORT']</code>	Port de la requête HTTP
<code>\$_SERVER['REQUEST_URI']</code>	Chemin du script
<code>\$_SERVER['SERVER_ADDR']</code>	Adresse IP du serveur
<code>\$_SERVER['SERVER_ADMIN']</code>	Adresse de l'administrateur du serveur
<code>\$_SERVER['SERVER_NAME']</code>	Nom local du serveur
<code>\$_SERVER['REQUEST_METHOD']</code>	Méthode d'appel du script

```
<?php
    echo $_SERVER['DOCUMENT_ROOT'] . '<br/>';
    echo $_SERVER['HTTP_ACCEPT_LANGUAGE'] . '<br/>';
    echo $_SERVER['HTTP_HOST'] . '<br/>';
    echo $_SERVER['HTTP_USER_AGENT'] . '<br/>';
    echo $_SERVER['REMOTE_ADDR'] . '<br/>';
    echo $_SERVER['REMOTE_PORT'] . '<br/>';
    echo $_SERVER['REQUEST_URI'] . '<br/>';
    echo $_SERVER['SERVER_ADDR'] . '<br/>';
    echo $_SERVER['SERVER_ADMIN'] . '<br/>';
    echo $_SERVER['SERVER_NAME'] . '<br/>';
    echo $_SERVER['REQUEST_METHOD'] . '<br/>';
?>
```

```
E:/Program Files/vamp5/www
fr-FR;q=0.8,en-US;q=0.6,en;q=0.4
localhost
Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.56 Safari/535.11
127.0.0.1
1239
/test/
127.0.0.1
webmaster@localhost
localhost
GET
```

Les Constantes

Une variable va changer de contenu suivant le programme mais parfois nous allons utiliser la même valeur pour tous le programme. Une constante se définit par la commande DEFINE().

```
<?php
define("VALEUR_FIXE","valeur");
define("VALEUR_FIXE_2","valeur");
Print("La valeur de la constante VALEUR_FIXE est ".VALEUR_FIXE);
ECHO "et VALEUR_FIXE_2 prend la valeur ".VALEUR_FIXE_2;
/*Même si ce n'est pas obligatoire, les constantes sont souvent nommées
en majuscules, ceci facilite la lecture du programme. Pour les afficher
ou les utiliser. Remarquez que nous avons directement utilisé son nom,
sans $ comme pour les variables.*/
?>
```

Les formats de dates

Le langage PHP inclut une fonction gérant les dates et heures. La fonction date(format,optionnel) renvoie une chaîne de caractère suivant le format choisi.

Lettre	Signification	Valeurs possibles	Exemple
s	Secondes	00 à 59	53
i	Minutes	00 à 59	26
H	Heure	00 à 23	00
l	Indique si l'heure d'été est activée (1 = oui, 0 = non)	0 ou 1	1
O	Différence d'heures avec l'heure GMT (Greenwich)	-1200 à +1200	+0200
d	Jour du mois	01 à 31	29
m	Mois de l'année	01 à 12	08
Y	Année, sur 4 chiffres	Beaucoup de possibilités	2005
y	Année, sur 2 chiffres	Beaucoup de possibilités	05
L	Indique si l'année est bissextile (1 = oui, 0 = non)	0 ou 1	0
l	Jour de la semaine écrit en anglais	Sunday à Saturday	Monday
F	Mois écrit en anglais	January à December	August
t	Nombre de jours dans le mois	28 à 31	31
w	Numéro du jour de la semaine	0 (dimanche) à 6 (samedi)	1
W	Numéro de la semaine dans l'année	01 à 52	35
z	Numéro du jour de l'année	0 à 365	240

Quelques exemples de la commande date():

- date('d/m/Y') affiche 26/02/2012
- date('D d F Y h:s') affiche Sun 26 february 2012 17:56

L'utilisation de la partie optionnelle permet par exemple d'afficher la date de modification de la page, comme filemtime.

```
<?php
print('Créé le 26/02/2012');
print(', modifier le ');
print(date('d/m/Y', filemtime('index.php')));
?>
```

Le timestamp

C'est le nombre de secondes écoulées depuis le 1er Janvier 1970 à Minuit. ça représente le début de l'époque où le système d'exploitation Unix a été créé.

```
<?php
    /*le 1er Janvier 1970 à Minuit, le timestamp avait pour valeur 0*/
    echo 'Le timestamp actuel est : ' . time();
    $timestamp=1330324299; /*C'est l'heure qu'il était quand j'écrivais le tuto
    riel !*/
    /*Voici quelque d'infos sur ce timestamp :*/
    echo 'Ce jour était le : ' . date('l', $timestamp);//Sunday
    //Sunday03h 44min 59s
    echo 'Il était exactement : ' . date('Hh imi
    ss', $timestamp);
    echo 'Il y avait : ' . date('t', $timestamp);//29 jours dans ce mois
    echo "C'était le : " . date('z', $timestamp);//56 ème jours de cette année
?>
```

Les Opérateurs

Un opérateur est un symbole qui indique à PHP qu'il doit effectuer une opération.

- Opérateurs d'incrément/décrément

Opérateur	Nom	Résultat
++\$a	Pré-incrément	Incrémente \$a d'un, puis renvoie \$a.
\$a++	Post-incrément	Renvoie \$a, puis incrémente \$a de un.
--\$a	Pré-décrément	Décrémente \$a d'un, puis renvoie \$a.
\$a--	Post-décrément	Renvoie \$a, puis décrémente \$a d'un.

- Opérateurs d'affectation

Opérateur	Signification
=	affectation simple
+=	addition puis affectation
-=	soustraction puis affectation
*=	multiplication puis affectation
/=	division puis affectation
%=	modulo puis affectation
=>	associe une valeur à une clé dans un tableau
->	réalise un appel de méthode

- Opérateurs arithmétiques

Opérateur	Opération
+	addition
-	soustraction
/	division
*	multiplication
%	modulo
++	incrément
	(voir paragraphe "incrément/décrément")

-- décréement
(voir paragraphe "incrémentation/décrémentation")

- Opérateurs de comparaison

Exemple	Nom	Résultat
<code>\$a == \$b</code>	Egal	Vrai si les valeurs de \$a et \$b sont égales Noter les DEUX signes "égale" à la suite : <code>\$a == \$b</code>
<code>\$a === \$b</code>	Identique	Vrai si <code>\$a == \$b</code> et si \$a et \$b sont de même type. Noter les TROIS signes "égale" à la suite : <code>\$a === \$b</code> (PHP 4)
<code>\$a != \$b</code>	Non égal	Vrai si \$a n'est pas égal à \$b.
<code>\$a <> \$b</code>	Non égal	Vrai si \$a n'est pas égal à \$b.
<code>\$a !== \$b</code>	Non identique	Vrai si \$a n'est pas égal à \$b, ou si \$a et \$b sont de types différents (PHP 4).
<code>\$a < \$b</code>	Plus petit que	Vrai si \$a est plus petit que \$b.
<code>\$a > \$b</code>	Plus grand que	Vrai si \$a est plus grand que \$b.
<code>\$a <= \$b</code>	Plus petit ou égal à	Vrai si \$a est plus petit ou égal à \$b
<code>\$a >= \$b</code>	Plus grand ou égal à	Vrai si \$a est plus grand ou égal à \$b.

- Opérateurs logiques

Exemple	Nom	Résultat
<code>\$a and \$b</code>	And	Vrai si \$a et \$b sont vrais S'exécute après le =
<code>\$a or \$b</code>	Or	Vrai si \$a, ou \$b, ou les deux, sont vrais S'exécute après le =
<code>\$a xor \$b</code>	Xor	Vrai si un et un seul parmi \$a ou \$b est vrai
<code>! \$a</code>	Not	Vrai si \$a est faux
<code>\$a && \$b</code>	And	Vrai si \$a et \$b sont vrais. S'exécute avant le =
<code>\$a \$b</code>	Or	Vrai si \$a, ou \$b, ou les deux, sont vrais S'exécute avant le =

```
<?php
/*Les Opérateurs : Incrementation et Decrémentation*/
$x=5;
echo $x;
$x++; //L'incrémentation
echo $x; //6
echo $x++; //Post-incrément, Affichage : 6
echo ++$x; //Pré-incrément, Affichage : 7
$y=5.66; //Double point au lieu de virgule
echo $y++; //Post-incrément, Affichage : 5.66
echo ++$y; //Pré-incrément, Affichage : 6.66
/*Les Opérateurs : Affectation*/
$x=15;
$y=$x; //Affectation simple
$y+=5;
echo $y; //20
$y-=5;
echo $y; //15
$y*=5;
echo $y; //75
```

```

$y/=5;
echo $y;//15
$y%=5;
echo $y;//0
/*Les variables : les chaines de caractères*/
$nom="MOHAMED";//variable nom et sa valeur Mohamed
$prenom="Ahmed";
echo $nom . $prenom;
echo "mon nom est $nom et mon prénom est $prenom";
?>

```

- L'opérateur ?

L'opérateur ? : s'appelle aussi opérateur de test "ternaire". Sa syntaxe est
 [test logique] ? [expression si vrai] : [expression si faux]

```

<?php
    $a=$b=1
    ($a==$b)?$c=10:$c=20;
    /*On peut également l'utiliser pour compacter les séquence de test / affect
ations*/
    $réponse=($a==$b)? "a égal b" : "a différent de b" ;
    echo $réponse;
?>

```

Exercices

Ex1 : Utiliser l'opérateur ternaire pour la comparaison Entre "3.14" et 3.14 .

Ex2 : Affecter votre nom au variable \$nom et afficher dans la page (MON nom est "Mohamed").

Ex3 : Afficher dans la page (la vaible \$nom = "Mohamed").

Ex4 : Affecter cette chaine de caractères "Il y a du \$dollar dans l'air !" a une variable et imprimer la.

Ex5 : Afficher les balise HTML après l'impression.

Instructions de contrôle.

Comme dans tous les langages de programmation, le php dispose de structure de contrôle. Le php étant issue du C, les instructions de contrôle sont presque indentique aux instructions du C.

L'instruction if ().

```

<?php
    /*Les Instructions de Controle : if(condition)*/
    $x=3;
    if($x==1)
    {
        // fait si $i = 1
    }
    else if($x==2)
    {
        // fait si $i = 2
    }

```

```

    }
    else($x==2)
    {
        // bloc fait dans tous les autres cas.
    }
?>

```

L'instruction else if () sert à multiplier le nombre de choix sans pour autant augmenter le if ().

L'instruction switch ().

```

<?php
/*Les Instructions de Controle : switch(condition)*/
$x=3;
switch($x==1)
{
    case 1 :
        // fait si $i = 1
        break;
    case 2 :
        // fait si $i = 2
        break;
    default :
        // fait dans tous les autres cas restant.
        break;
}
?>

```

Dans l'instruction switch () vous devez couvrir tous les cas possible, c'est pour cela qu'à la fin il faut mettre default :

Pour les débutants il fallait mieux utiliser l'instruction if () à la place de switch () car elle procure le même résultat et elle est beaucoup plus facile à utiliser .

La boucle while ().

```

<?php
/*Les Instructions de Controle : while(condition)*/
$x=10;
$i=0;
while($x>$i)
{
    $i++;
}
?>

```

Tant que la condition (\$x>\$i) est vraie \$i sera incrémenté

La boucle while() est très utilisé quand on ne sait pas combien de fois il faut faire la boucle.

l'instruction continue permet de sortir(sans quitter) du boucle sans exécuté les instructions qui la suit.

Vous pouvez mettre l'instruction break; si vous voulez stoper la boucle au milieu de son déroulement, il y a aussi l'instruction exit; qui stoppe toutes les tâches en cours.

La boucle do while ().

```
<?php
/*Les Instructions de Controle : do while(condition)*/
$x=10;
$i=0;
do
{
    $i++;
}while($x>$i);// ; n'est obligatoire
?>
```

La boucle do while () est le même que la boucle while () sauf que l'instruction est faite au moins une fois.

La boucle for(; ;) .

```
<?php
/*Les Instructions de Controle :for( ; ; )*/
for($i=0; $i<40; $i++)
{
    // Instruction faite 40 fois
}
?>
```

Explication des paramètres de boucle :

- \$i=0 : cela est l'initialisation de la boucle.
- \$i<40 : cela est la test d'arrêt de la boucle.
- \$i++ : cela est la condition de continuation de la boucle.

Si il y a une boucle à utiliser, c'est bien la boucle for(;;) car elle est très compacte est en une ligne on fait trois actions.

Exercices.

Ex2 : De 1 à 10, sélectionnez tous les nombres sauf 5.

Ex3 : De 1 à 10, sélectionnez les nombres paire.

Ex4 : De 1 à 10, sélectionnez les nombres de 1 jusqu'à 5.

Ex5 : Montrer la différence entre break exit et continue.

Correction.

```
<?php
/*Rp : 1*/
$var_1="3.14";
$var_2=3.14;
($var_1 === $var_2) ? print "Condition vraie" : print "Condition fausse" ;
// "Condition fausse"
($var_1 == $var_2) ? print "Condition vraie" : print "Condition fausse" ;
// "Condition vraie"
for($i=0; $i<40; $i++)
/*Rp : 2*/
for($i=1; $i<=10; $i++)
{
if($i!=5){
echo $i;
}
}
for($i=1; $i<=10; $i++)
if($i!=5)
echo $i;
for($i=1; $i<=10; $i++)
{
if($i==5)
continue;
echo $i;//1234678910
}
/*Rp : 3*/
for($i=1; $i<=10; $i++)
if($i%2 == 0)
echo $i;//246810
/*Rp : 4*/
for($i=1; $i<=10; $i++)
{
echo $i;//12345
if($i==5)
break;
}
$i=1;
while($i<=5){
echo $i;//12345
$i++;
}
/*Rp : 5*/
/*
*break; : permet de sortir d'un bloc;
*continue; : permet de sortir d'un bloc d'instruction sans le quitter
*exit : permet de quitter le programme.*/
?>
```

Les Tableaux.

Un tableau PHP est une variable pouvant contenir d'autres variables. C'est en fait un système d'associations ordonnées (map) qui fait correspondre des valeurs à des clés. Une clé est une valeur, unique dans une liste donnée, permettant de retrouver une valeur ou une ensemble de valeurs dans une liste (Exemple : le N° de Sécurité sociale est une clé permettant de retrouver votre situation familiale, date et lieu de naissance...)

Initialisation des tableaux

```
<?php

/*Il y a deux façons d'initialiser un tableau, qui donnent le même résultat
:*/

/* 1 - Le mot clé array est utilisé :*/
/*On peut ne spécifier aucune clé :*/
$monthName=array("Janvier", "Février", "Mars", "Avril", "Mai", "Juin", "Jui
llet",
                "Aout", "Septembre", "Octobre", "Novembre", "Decembre
");

print("Le 5eme mois est $monthName[5]
");//Mois 5 est Juin
/*On spécifie seulement la valeur de la première clé :*/
$monthName=array("Janvier", "Février", "Mars", "Avril", "Mai", "Juin", "Jui
llet",
                "Aout", "Septembre", "Octobre", "Novembre", "Decembre
");

print("Moi 5 est $monthName[5]
");//Mois 5 est Mai
/*On utilise l'opérateur d'affectation pour chaque valeur*/
$music=array("pop" => "Britney Spears", "rock" => "Aerosmith",
            "jazz" => "Louis Armstrong");

print("Top de 'pop' ".$music["pop"] . "
");//Britney Spears

/* 2 - Le mot clé array n'est pas utilisé mais les crochets pour indiquer
qu'il s'agit d'un tableau:*/
$monthName[1]="Janvier";
$monthName[2]="Février";
/*Nb : les clés ne sont pas forcément numériques :*/
$utilisateur["nom"]="Mohamed Ali";
$utilisateur["ville"]="Rabat";
$utilisateur["tel"]="03655648221";
/*On ne spécifie pas la clé pour chaque valeur, PHP affecte d'office
```

```

        des valeurs en partant de 0 :*/
$ville[]="Rabat";
$ville[]="Tanger";
$ville[]="Casablanca";
$ville[]="Agadir";
$ville[]="Marakech";
$ville[]="Fes";
/* Tableaux multidimensionnels */
$events = array("1914"=> array ("Juin" => array ("23" => "Innauguration can
al
        de Kiev par Hitler", "28" => "Archiduc Ferdinand assassiné à Sarajevo"),
        "July" => array ("26" => "Mobilisation en Autriche", "28" =>
        "L'Autriche déclare la guerre à la Serbie" ))
        , "1918" => array( "November" => array ("11" => "Armistice si
gnée ",
        "16" => "retrait des troupes allemandes")));
?>

```

1914	<table> <tr> <td>Juin</td><td> <table> <tr> <td>23</td><td>Innauguration canal de Kiev par Hitler</td></tr> <tr> <td>28</td><td>Archiduc Ferdinand assassiné à Sarajevo</td></tr> </table> </td></tr> <tr> <td>Juillet</td><td> <table> <tr> <td>26</td><td>Mobilisation en Autriche</td></tr> <tr> <td>28</td><td>L'Autriche déclare la guerre à la Serbie</td></tr> </table> </td></tr> </table>	Juin	<table> <tr> <td>23</td><td>Innauguration canal de Kiev par Hitler</td></tr> <tr> <td>28</td><td>Archiduc Ferdinand assassiné à Sarajevo</td></tr> </table>	23	Innauguration canal de Kiev par Hitler	28	Archiduc Ferdinand assassiné à Sarajevo	Juillet	<table> <tr> <td>26</td><td>Mobilisation en Autriche</td></tr> <tr> <td>28</td><td>L'Autriche déclare la guerre à la Serbie</td></tr> </table>	26	Mobilisation en Autriche	28	L'Autriche déclare la guerre à la Serbie
Juin	<table> <tr> <td>23</td><td>Innauguration canal de Kiev par Hitler</td></tr> <tr> <td>28</td><td>Archiduc Ferdinand assassiné à Sarajevo</td></tr> </table>	23	Innauguration canal de Kiev par Hitler	28	Archiduc Ferdinand assassiné à Sarajevo								
23	Innauguration canal de Kiev par Hitler												
28	Archiduc Ferdinand assassiné à Sarajevo												
Juillet	<table> <tr> <td>26</td><td>Mobilisation en Autriche</td></tr> <tr> <td>28</td><td>L'Autriche déclare la guerre à la Serbie</td></tr> </table>	26	Mobilisation en Autriche	28	L'Autriche déclare la guerre à la Serbie								
26	Mobilisation en Autriche												
28	L'Autriche déclare la guerre à la Serbie												
1918	<table> <tr> <td>November</td><td> <table> <tr> <td>11</td><td>Armistice signée</td></tr> <tr> <td>16</td><td>retrait des troupes allemandes</td></tr> </table> </td></tr> </table>	November	<table> <tr> <td>11</td><td>Armistice signée</td></tr> <tr> <td>16</td><td>retrait des troupes allemandes</td></tr> </table>	11	Armistice signée	16	retrait des troupes allemandes						
November	<table> <tr> <td>11</td><td>Armistice signée</td></tr> <tr> <td>16</td><td>retrait des troupes allemandes</td></tr> </table>	11	Armistice signée	16	retrait des troupes allemandes								
11	Armistice signée												
16	retrait des troupes allemandes												

Remarque importante :

- Il ne peut y avoir deux clés identiques, mais deux clés différentes peuvent pointer vers deux valeurs identiques
- En php les tableaux multidimensionnels n'existent pas, par contre on a la possibilité de mettre un tableau dans une cellule d'un tableau.
- Depuis la version PHP 4 une structure de contrôle foreach a été inclut, comme en Perl ou d'autres langages. C'est un moyen simple de passer en revue les valeurs (et éventuellement les clés) d'un tableau.

Exemple de foreach

```
<?php
    $tab=array ( "un" => 1,"deux" => 2,"trois" => 3,"dix-sept" => 17 );
    foreach($tab as $i => $v)
        print "$tab[$i]=>$v.
";
?>

<?php
    $lettre[0][0]="a";
    $lettre[0][1]="b";
    $lettre[1][0]="c";
    $lettre[1][1]="d";
    foreach($lettre as $v1)
        foreach($v1 as $v2)
            print "$v2
";
?>
```

Fonctions spécifiques utilisables avec les tableaux

Voici un tableau qui présente quelque fonctions spécifique au tableaux .

*Fonction	Description
array_count_values	Compte le nombre de valeurs dans un tableau
array_filter	Filtre les éléments d'un tableau
array_flip	Remplace les clés par les valeurs, et les valeurs par les clés
array_key_exists	Cherche si une valeur ou une clé existe dans le tableau
array_keys	Retourne toutes les clés d'un tableau
xarray_push	Empile un ou plusieurs éléments à la fin d'un tableau
array_reverse	Renverse l'ordre des éléments d'un tableau
array_search	Recherche dans un tableau la clé associée à une valeur
xarray_shift	Dépille (enlève) un ou plusieurs élément au début d'un tableau
array_sum	Calcule la somme des valeurs du tableau
xarray_unshift	Empile (rajoute) un ou plusieurs éléments au début d'un tableau
array_values	Retourne les valeurs d'un tableau réindexées avec des entiers
__xarray	Crée un tableau

xarsort	Trie un tableau en ordre inverse
xsart	Trie un tableau en ordre
count	Compte le nombre d'éléments d'un tableau
xcurrent	Transforme une variable en tableau
each	Retourne chaque paire clé/valeur d'un tableau
end	Positionne le pointeur de tableau en fin de tableau
in_array	Indique si une valeur appartient à un tableau
key	Retourne une clé d'un tableau associatif
xkrsort	Trie un tableau en sens inverse et suivant les clés
xksort	Trie un tableau suivant les clés
next	Avance le pointeur interne d'un tableau
pos	Retourne l'élément courant d'un tableau
prev	Reculé le pointeur courant de tableau
reset	Remet le pointeur interne de tableau au début
xshuffle	Mélange les éléments d'un tableau
sizeof	Retourne le nombre d'élément d'un tableau
xsrt	Trie le tableau
is_array()	Pour savoir si une variable est un tableau
xexplode()	array explode (string separator, string string [, int limit]) Retourne un tableau qui contient les éléments d'une chaîne contenant des délimiteurs
ximplode()	string implode (string glue, array pieces) Retourne une chaîne constituée de tous les éléments du tableau, pris dans l'ordre, transformés en chaîne, et séparés par le séparateur
xsplt()	array split (string pattern, string string [, int limit]) Scinde une chaîne en un tableau, grâce à une expression régulière
xjoin()	string join (string glue, array pieces) Regroupe tous les éléments d'un tableau dans une chaîne, avec une chaîne de jointure.
* : le x signifie que la fonction modifie le(s) tableau(x) spécifié en argument, sinon en principe c'est un nouveau tableau qui est créé	

Les Fonctions.

Une fonction, dans la majorité des langages de programmation permet de regrouper un ensemble d'instructions qui doivent être exécutés plusieurs fois dans un programme; des paramètres peuvent être envoyés à cette fonction qui pourra ou non renvoyer des valeurs.

Deux sortes de fonctions en php :

- Les fonctions intégrées à PHP : fonctions pour gérer les tableaux, les conversions, les chaînes de caractères ... (plus de 1000 fonctions en standard)
- Les fonctions dites "utilisateur", que l'on construit soi-même

Les fonctions utilisateurs

Une fonction peut être définie en utilisant la syntaxe suivante :

```
<?php
    function nom_de_fonction ($arg_1, $arg_2, ..., $arg_n) {
        echo "Exemple de fonction.";
    };
    return $return_value;
```

```
}
```

```
?>
```

Exemple

```
<?php

/* Fonction qui ne retourne aucune valeur*/
function bonjour($nom){
    echo"Bonjour $nom !";
}

/* Fonction qui retourne une valeur*/
function bonsoir($nom){
    return"Bonsoir $nom !";
}

/* L'appelation des fonctions*/
bonjour("Mohamed");
bonjour("Aba baker");
echo bonsoir("Omar");
echo bonsoir("Otmame");
echo bonsoir("Ali");

?>
```

Syntaxe et principaux principes de fonctionnement :

- Les parenthèses sont obligatoires, même s'il n'y a pas d'arguments
- Pour activer une fonction, il suffit d'indiquer son nom, suivi des arguments éventuels entre parenthèses (parenthèses toujours obligatoires, à la fois dans la définition et dans l'appel, ainsi un nom de fonction est facilement identifiable (sauf pour les deux fonctions intégrées print et echo).
- Tout code PHP, correct syntaxiquement, peut apparaître dans une fonction et dans une définition de classe.
- En PHP 3, les fonctions doivent être définies avant qu'elles ne soient utilisées. Ce n'est plus le cas en PHP 4 ou 5.
- PHP 4 ne supporte pas le surchargement de fonction, ni la destruction ou la redéfinition de fonctions déjà déclarées.

Les fonctions php

```
<?php

/*Les fonctions PHP principales spécifiques au traitement des chaînes de
caractères et nombres*/

/*Fonction strlen() : renvoie la longueur de la chaîne de caractères */
```

```

echo strlen("bonjour");//6
/*Fonctions Trim, Rtrim, ltrim : Ces fonctions permettent de supprimer
les blancs dans une chaîne de caractères.*/
echo trim(" bonjour ");/*efface les blancs devant et derrière dans la chaîn
e
de caractères, pas les espaces au milieu.*/
echo rtrim("bonjour ");/*supprime les blancs à droite de la chaîne
de caractères.*/
echo rtrim("bonjour", "jour");//A partir de PHP 4.0.1, une opérante
optionnelle a été rajoutée. =>bon*/
echo ltrim(" bonjour");//supprime les blancs à gauche de la chaîne
de caractères.*/
echo ltrim("bonjour", "bon");//A partir de PHP 4.0.1, une opérante
optionnelle a été rajoutée. =>jour*/
/*Fonctions MAJUSCULES et minuscules.*/
echo STRTOLOWER("BONJOUR");//transforme tous les caractères de la
chaîne en minuscule .*/
echo strtoupper("bonjour");//transforme tous les caractères de la chaîne
en MAJUSCULES .*/
echo UCWORDS("bonjour tout le monde");//transforme la première lettre chaqu
e
mot en majuscule .*/
echo ucfirst("bonjour tout le monde");//transforme la première lettre de
la chaîne en majuscule .*/
/*Manipulation et modification de chaînes : Ces fonctions PHP vont modifier
le contenu d'une chaîne de caractères..*/
echo addslashes("l'entreprise");//ajoute les anti-slashes devant les carac
tères
spéciaux.=>l'entreprise*/
/* Cette fonction est utilisée pour les fonctions Print('') et ECHO'' et
lors
d'introduction de valeurs via un formulaire.=>l'entreprise*/
echo stripslashes("l'entreprise");//supprime les anti-slashes, notamment ut
ilisée
lors de version imprimable.=>l'entreprise*/
echo CHUNK_SPLIT("bonbon","3","-");//permet de scinder une chaîne de
caractère.=>bon-bon-*/

```

```

    echo strpos("programmation en partique","e");/*recherche le caractère et af
fiche
    le reste de la chaîne, y compris le caractère.=>en partique*/
    echo STR_replace(";", " "; "programmation;en;partique");/*remplace dans la
chaîne les
    caractères à remplacer(;) par le caractère de remplacement(' ') et l'assigne
à une
    variable.=>programmation en partique*/
    echo htmlentities("< >");/*remplace le caractère par son équivalent HTML si
possible.=> & lt;& nbsp;& gt;*/
    echo ereg('MA', 'MA3625141414');/*recherche si la chaîne1 est contenue dans
la
    chaîne 2, renvoie une valeur logique.=> true*/
    echo nl2br("Bonjour
tout
le monde");/*affiche les sauts de lignes en remplaçant
    les /n par <br>*/
    /*Fonctions spécifique au nombre.*/
    echo DECHEX(5555);/*renvoie la valeur hexadécimale d'un nombre. =>15b3*/
    echo CEIL(5.55);/*renvoie le nombre entier supérieur. =>6*/
    echo FLOOR(5.55);/*renvoie le nombre entier inférieur. =>5*/
    echo is_int(5.55);/*renvoie true si le contenu est un entier, false sinon.*
/
    echo rand(0, 9);/*ou 0 est min, 9 est max, Cette fonction renvoie une varia
ble
    entière entre 0 et 9 aléatoirement.*/
    echo intval("525");/*convertit une chaîne en variable entière.*/
    /*Fonctions sur les dates.*/
    echo checkdate ( int $mois, int $jour, int $année);/*vérifie si une date es
t valide.
    mois doit être compris entre 1 et 12, jour entre 1 et 31 et année entre 1 e
t 32767.
    La fonction tient compte des années bissextiles. Remarquez que le format es
t
    anglophone. Le résultat est une valeur logique.*/
?>

```

Exercices

Ex 1 : Nous avons des cercles avec diamètre défférentes et nous voulons calculant leurs surfaces.

Ex 2 : Faites un programme qui donne les mentions a partir des points.

Ex 3 : Créer la fonction de jeux love matter.

Ex 4 : Créer la fonction qui nous permet de récupérer la date en français.

Les formulaires

Passage de parametre par l'url

On peut passer des informations lors de la construction du lien d'une page vers une autre

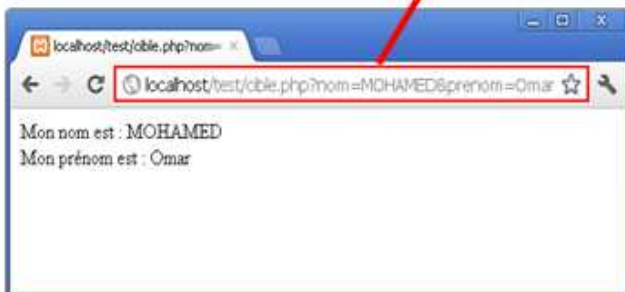
page : index.html

Source : index.html



```
<html>
<head>
  <title>Envoi des Données</title>
</head>
<body>
  <a href="cible.php?nom=MOHAMED&prenom=Omar">Envoyer</a>
</body>
</html>
```

http://localhost/test/cible.php?nom=MOHAMED&prenom=Omar



```
<?php
$nom = $_GET['nom'];
$prenom = $_GET['prenom'];

echo "Mon nom est : $nom <br>";
echo "Mon prénom est : $prenom <br>";
?>
```

page : cible.php

Source : cible.php

- **http://** :HyperText Transfer Protocole est un protocole de communication Client/Serveur.
- **localhost** : Nom de server équivalent(127.0.0.1) son adresse IP.
- **test/** : Nom de site.
- **cible.php** : Nom de fichier.
- **?** : Il sépare le nom de fichier de paramètre de paramètre envoyés dans l'url.
- **nom=MAHAMED et prenom=Omar** : Noms des paramètres et leurs valeurs.
- **&** : Permet de séparer les paramètres.
- **\$_GET['nom_variable']** C'est un tableau qui regroupe les valeurs des variables et leurs nom sont les indices.

Zones de saisie standard POST

C'est la méthode à utiliser par défaut. Les données sont transmises de façon non visibles du navigateur au serveur. La syntaxe au niveau du formulaire est :

page : index.html



Source : index.html

```
<html>
<head>
  <title>Envoi des Données - POST</title>
</head>
<body>
  <form name="form1" id="form1" action="cible.php" method="POST">
    Nom : <input type="text" id="nom" name="nom"/><br/>
    Prénom : <input type="text" id="prenom" name="prenom"/><br/>
    <input type="submit" value="Envoyer" />
  </form>
</body>
</html>
```

<http://localhost/test/cible.php>



page : cible.php

```
<?php
$nom = $_POST['nom'];
$prenom = $_POST['prenom'];

echo "Mon nom est : $nom <br>";
echo "Mon prénom est : $prenom <br>";
?>
```

Source : cible.php

Dans le fichier cible.php, les données sont récupérées dans le tableau \$_POST : \$_POST["nom_de_variable"]

Zones de saisie standard GET

Cette méthode est à proscrire pour les passages des mots de passe et autres informations 'sensibles', par contre elle est pratique pour la mise au point puisque l'on voit dans l'URL les valeurs envoyées au serveur. La syntaxe au niveau du formulaire est :

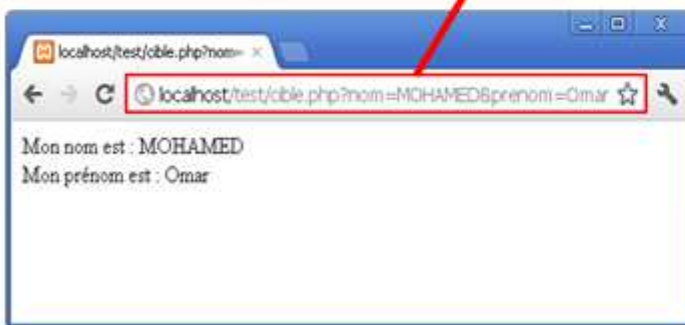
page : index.html



Source : index.html

```
<html>
<head>
  <title>Envoi des Données - GET</title>
</head>
<body>
  <form name="form1" id="form1" action="cible.php" method="get">
    Nom : <input type="text" id="nom" name="nom"/><br/>
    Prénom : <input type="text" id="prenom" name="prenom"/><br/>
    <input type="submit" value="Envoyer" />
  </form>
</body>
</html>
```

<http://localhost/test/cible.php?nom=MOHAMED&prenom=Omar>



<?php

```
$nom = $_GET['nom'];
$prenom = $_GET['prenom'];
```

```
echo "Mon nom est : $nom <br>";
echo "Mon prénom est : $prenom <br>";
?>
```

page : cible.php

Source : cible.php

Les autres Contrôles HTML

Ces Contrôles peuvent fonctionner avec un formulaire GET ou POST.

- **Boutons radio** : Ce type de controle est à utiliser lorsque l'on veut que l'internaute ne puisse choisir qu'une option parmi plusieurs.
- **Case à cocher** : Ce type de controle est à utiliser lorsque l'on veut que l'internaute puisse choisir plusieurs cases à cocher à la fois.
- **Liste de choix** : Liste déroulante à choix unique. On ne peut sélectionner qu'un seul élément de la liste.
- **Liste à choix multiple** : Liste déroulante à choix multiple : l'internaute peut sélectionner plusieurs options dans la liste.
- **Boîte de dialogue** : Zone de texte multiligne permet d'entrer des grandes textes.

Envoi des Données - Les Déffé x

localhost/test/

Radio button

Civilité : ☒ Mr ☐ Mme ☐ Mlle

Case à cocher

Pour créer un site web il faut savoir :

☒ le Php
☐ cuisinier
☒ le Html
☒ le Javascript

Liste de choix

Tunisie

Liste à choix multiple

Tunisie
Egypte
Lybie
Yamen
Sorya

Boite de dialogue

Posez vos questions :
Qui est le déclencheur
du printemps ARABE ?

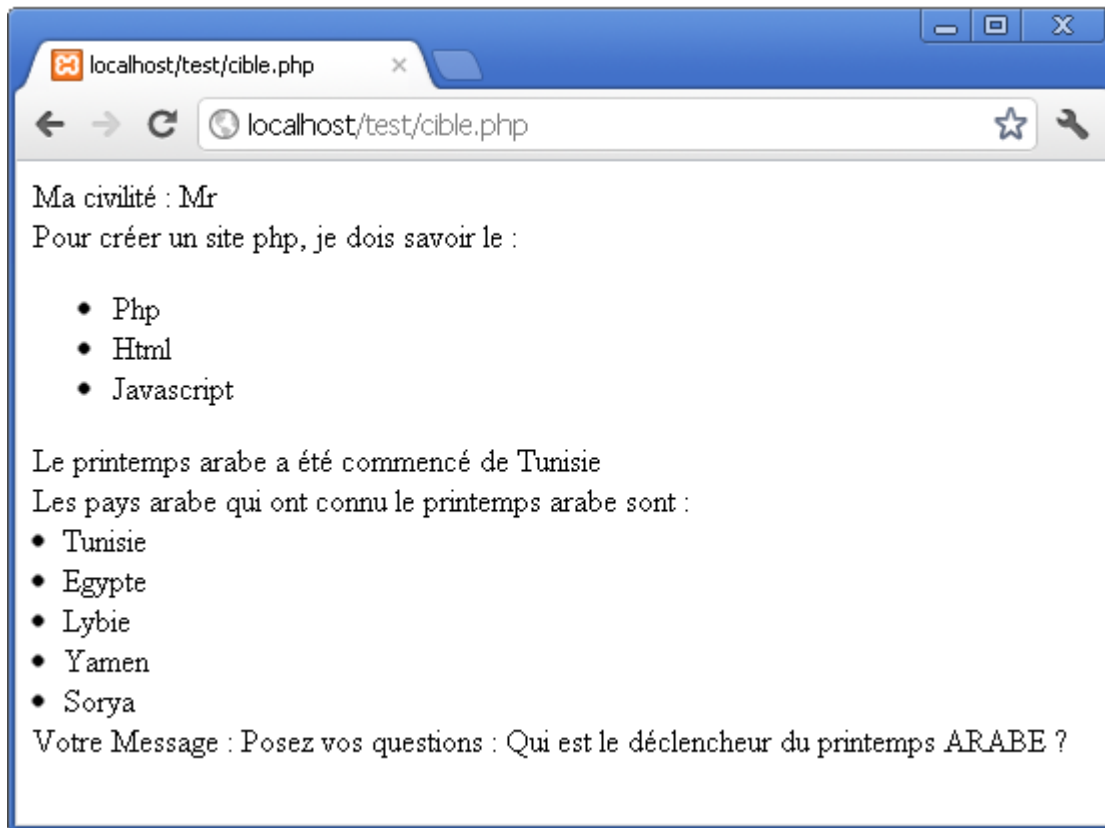
Envoyer

Le code source de ce formulaire index.html


```

<html>
<head>
  <title>Envoi des Données - Les Différentes Contrôle HTML</title>
</head>
<body>
  <form name="form1" id="form1" action="cible.php" method="post">
    <h1>Radio button</h1>
    Civilité : <input type="radio" id="civilite" name="civilite" value="Mr" checked /> Mr
               <input type="radio" id="civilite" name="civilite" value="Mme" /> Mme
               <input type="radio" id="civilite" name="civilite" value="Mlle" /> Mlle
    <h1>Case à cocher</h1>
    Pour créer un site web il faut savoir :<br/>
    <input type="checkbox" id="siteWeb" name="siteWeb1" /> le Php<br/>
    <input type="checkbox" id="siteWeb" name="siteWeb2" /> cuisinier<br/>
    <input type="checkbox" id="siteWeb" name="siteWeb3" /> le Html<br/>
    <input type="checkbox" id="siteWeb" name="siteWeb4" /> le Javascript<br/>
    <h1>Liste de choix</h1>
    <select id="listPays" name="listPays">
      <option value="Tunisie">Tunisie</option>
      <option value="Egypte">Egypte</option>
      <option value="Lybie">Lybie</option>
      <option value="Yamen">Yamen</option>
      <option value="Sorya">Sorya</option>
    </select>
    <h1>Liste à choix multiple</h1>
    <select id="listPaysMultiple" name="listPaysMultiple[]" size=7 multiple>
      <option value="Tunisie">Tunisie</option>
      <option value="Egypte">Egypte</option>
      <option value="Lybie">Lybie</option>
      <option value="Yamen">Yamen</option>
      <option value="Sorya">Sorya</option>
    </select>
    <h1>Boite de dialogue</h1>
    <textarea name="message" id="message" col=45 rows=5>Posez vos questions : </textare
    <br/><br/>
    <input type="submit" value="Envoyer" />
  </form>
</body>
</html>

```



Le code source de cible.php

```
<?php
```

```
/*Récupérer a partir de Radio button*/
```

```
$civillite= $_POST['civillite'];
```

```
echo 'Ma civilité : ' . $civillite . '<br/>';
```

```
/*Récupérer a partir de Case à cocher*/
```

```
if($_POST['siteWeb1'])
```

```
| $siteWeb[] = 'Php';
```

```
if($_POST['siteWeb2'])
```

```
| $siteWeb[] = 'cuisinier';
```

```
if($_POST['siteWeb3'])
```

```
| $siteWeb[] = 'Html';
```

```
if($_POST['siteWeb4'])
```

```
| $siteWeb[] = 'Javascript';
```

```
echo 'Pour créer un site php, je dois savoir le : <ul>';
```

```
for($i=0;$i<count($siteWeb); $i++)
```

```
| echo '<li>'. $siteWeb[$i]. '</li>';
```

```
echo '</ul>';
```

```
/*Récupérer a partir de Liste de choix*/
```

```
echo 'Le printemps arabe a été commencé de ' . $_POST['listPays']. '<b
```

```
/*Récupérer a partir de Liste à choix multiple*/
```

```
$TableauPays = $_POST['listPaysMultiple'];
```

```
echo 'Les pays arabe qui ont connu le printemps arabe sont : ';
```

```
for($i=0;$i<count($TableauPays); $i++)
```

```
| echo '<li>'. $TableauPays[$i]. '</li>';
```

```
echo '</ul>';
```

```
/*Récupérer a partir de Boite de dialogue*/
```

```
echo 'Votre Message : ' . $_POST['message'];
```

```
?>
```

Gestion de fichiers.

Les fichiers textes sont une bonne alternative à la BDD. PHP possède toute une série de fonctions qui permet de créer, écrire, modifier et supprimer dans un fichier texte.

Ouvrir un fichier

```
<?php
```

```
/*Ouvrir un fichier text*/
```

```
$f = fopen("fichier.txt", "r");/*Le premier paramètre de la fonction est le  
nom du
```

```
fichier et le second paramètre est le mode dans lequel on peut ouvrir nos f  
ichiers*/
```

```
?>
```

Les différents modes

modeDescription

- 'r' Ouvre en lecture seule : Le pointeur est placé au début du fichier.
 - 'r+' Ouvre en lecture/écriture : Le pointeur est placé au début du fichier.
 - 'w' Ouvre en écriture seule : Le pointeur est placé au début du fichier.
Réduit la taille du fichier à 0. Tentative de création si celui-ci n'existe pas.
 - 'w+' Ouvre en lecture/écriture : Le pointeur est placé au début du fichier.
Réduit la taille du fichier à 0. Tentative de création si celui-ci n'existe pas.
 - 'a' Ouvre en écriture seule : Le pointeur est placé au début du fichier.
Tentative de création si celui-ci n'existe pas.
 - 'a+' Ouvre en lecture/écriture : Le pointeur est placé à la fin du fichier. Tentative de création si celui-ci n'existe pas.
 - 'x' Crée et ouvre le fichier en lecture seule : Le pointeur est placé au début du fichier. Si le fichier existe déjà, fopen va échouer.
 - 'x+' Crée et ouvre le fichier en lecture/écriture : Le pointeur est placé au début du fichier. Si le fichier existe déjà, fopen va échouer.
- Si la fonction fopen() échoue lors de l'ouverture du fichier, celle-ci retourne 0.
 - La fonction fclose() est utilisée pour fermer un fichier.

Exemple

```
<?php
    /*Overture de fichier en mode "r"*/
    /*Si le fichier n'existe pas un message d'erreur généré par php sera afficher avec
    le message au dessous 'Fichier introuvable !'*/
    if(!($f = fopen("data.txt", "r")))
        exit("Fichier introuvable !");
    fclose($f);
?>
```

Écrire dans un fichier.

L'écriture dans un fichier texte se fait avec la fonction fwrite(). Pour faire des retour à la ligne vous devez utiliser : " "

```
<?php
    $f = 'data.txt';
    $text = "ma chaîne de caractères";
    $handle = fopen($f,"w");
    // regarde si le fichier est accessible en écriture
    if (is_writable($f)) {
```

```

        // Ecriture
        if (fwrite($handle, $text) === FALSE) {
            echo 'Impossible d\'écrire dans le fichier ' . $f . ' ';
            exit;
        }
        echo 'Ecriture terminé';
        fclose($handle);
    }
    else {
        echo 'Impossible d\'écrire dans le fichier ' . $f . ' ';
    }
}
?>

```

Compter le nombre de lignes.

A l'aide de 2 fonctions. L'une place le fichier dans un tableau et l'autre pour compter les lignes du tableau.

```

<?php
    $f = 'data.txt';
    if(file_exists($f)){
        $tab = file($f); // place le fichier dans un tableau
        $nb = count($tab); // compte le nombre de ligne
        echo $nb; // Affiche le résultat
    }
?>

```

Récupérer le contenu.

La fonction `file_get_contents` permet de récupérer le contenu du fichier ou utiliser `fopen` avec `fread` !

```

<?php
    $contents = file_get_contents("data.txt");
    echo $contents;
    // Ou Avec fopen, fread
    $filename = "data.txt";
    $handle = fopen($filename, "r");
    $contents = fread($handle, filesize($filename));
    echo $contents;

```

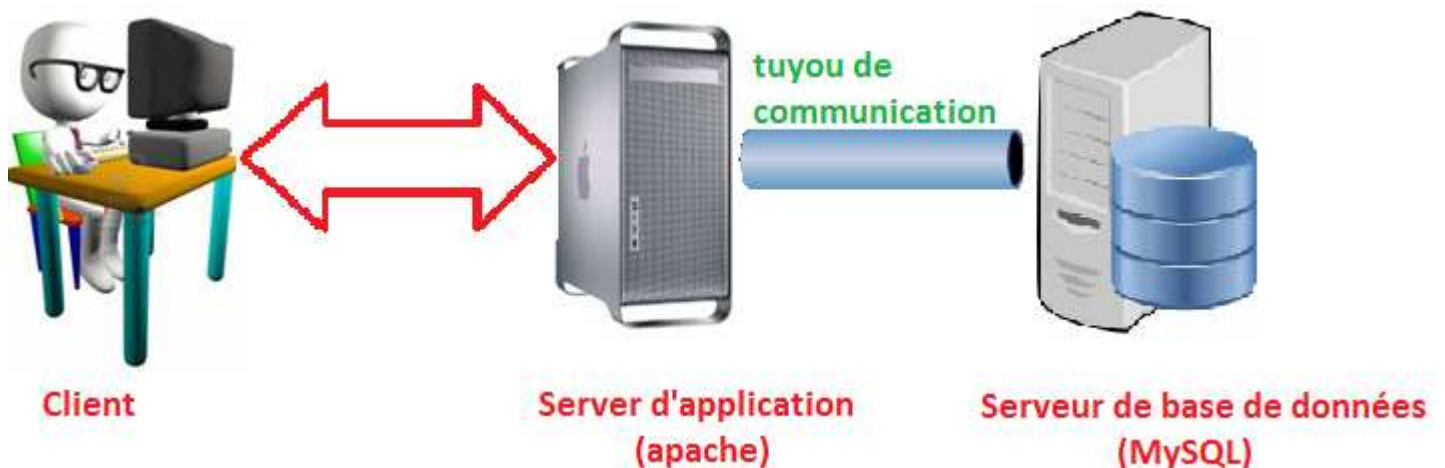


```
fclose($handle);
```

```
?>
```

TP : Gestion de base de données

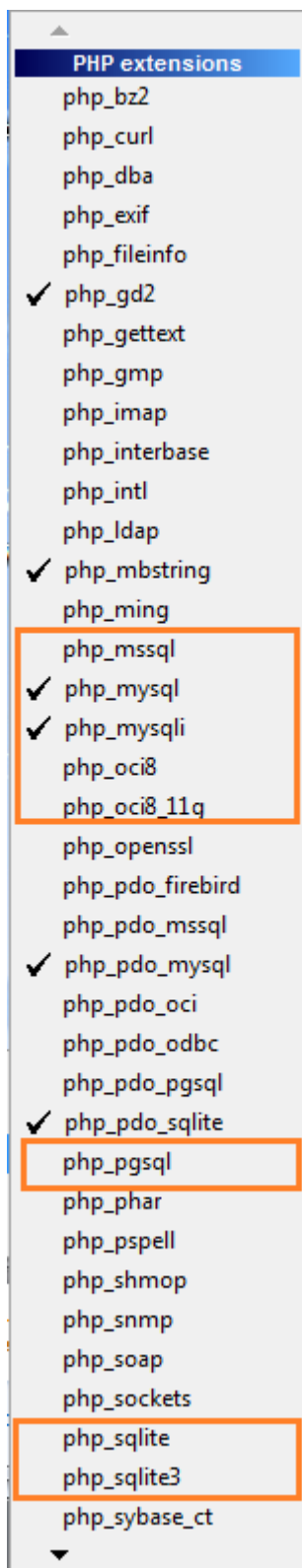
Pour que le php soit capable de gérer une base de données située dans un SGBDR, il doit communiquer avec ce dernier. Cette communication est présentée sous forme d'un tuyau qui permet d'établir une connexion entre le serveur d'application et le serveur de base de données.



Apache peut communiquer avec plusieurs type de SGBDR (Oracle, SQLServer, PostgreSQL, ...), et chacun d'eux il le fallait un tuyau spécifique. Ces tuyous se sont des extensions, pour activer une des extensions on doit se situer dans le fichier `php.ini` et décommenter l'extension qu'on voulait activer par l'enlèvement de ';' au début la ligne;

```
;extension=php_bz2.dll
;extension=php_curl.dll
;extension=php_dba.dll
;extension=php_exif.dll
;extension=php_fileinfo.dll
extension=php_gd2.dll
;extension=php_gettext.dll
;extension=php_gmp.dll
;extension=php_intl.dll
;extension=php_imap.dll
;extension=php_interbase.dll
;extension=php_ldap.dll
extension=php_mbstring.dll
;extension=php_ming.dll
;extension=php_mssql.dll
extension=php_mysql.dll
extension=php_mysqli.dll
;extension=php_oci8.dll ; Use with Oracle 10gR2 Instant Client
;extension=php_oci8_11g.dll ; Use with Oracle 11g Instant Client
;extension=php_openssl.dll
;extension=php_pdo_firebird.dll
;extension=php_pdo_mssql.dll
extension=php_pdo_mysql.dll
;extension=php_pdo_oci.dll
;extension=php_pdo_odbc.dll
;extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
;extension=php_pgsql.dll
;extension=php_phar.dll
;extension=php_pspell.dll
;extension=php_shmop.dll
;extension=php_snmp.dll
;extension=php_soap.dll
```

Ou on sélectionnant une de la liste, on cliquant sur la barre tâche=>Le gestionnaire s'affiche=>PHP=>PHP extensions



L'activation d'une extension permet l'utilisation d'ensemble des fonctions spécifiant un SGBD précis. exemple `mysql_query("Une request");` qui permet de retourner un résultat après l'exécuter une requête côté serveur de base de données **MySQL**, la fonction `mssql_query("Une request");` fait la même chose avec autre SGBDR **SQLServer**.

Le code php qui permet d'établir la connexion entre l'application et le serveur de base de données **MySQL**(extension de mysql est activé par défaut).

```
<?php
mysql_connect('server', 'user', 'password');
```

```
//Après l'établissement de la connexion, on selecte la base de données
mysql_db_select('database_name');
```

```
?>
```

Le code optimisé pour établir la connexion et la gestion d'éventuelles erreurs

```
<?php
//Connexion avec server en local
$host = 'localhost';
$base = 'mabase';
$login = 'root';
$pwd = '';
$con = mysql_connect($host, $login, $pwd); //Etablir la connexion
//Si il y a une erreur lors d'établissement de la connexion avec le server
if(!$con){
    die("Désolé, connexion impossible à [$host] : ".mysql_error());
}
//si il y a une erreur lors de la sélection de la base de données (au cas d'erreur n
'existe pas)
$db = mysql_select_db($base, $con);
if(!$db){
    print "Désolé, la base de données [$base] n'a pas été trouvée sur le
    serveur [$host]<br>".mysql_error();
    exit();
}

//die(message) joue le même rôle de print(message) avec exit()

?>
```

Après qu'on a établi la connexion avec le serveur, et sélection la base de données, on peut maintenant interroger cette dernière avec des requêtes SQL.

La fonction php qui permet d'envoyer une requête SQL au serveur de base de données et récupérer le résultat est **mysql_query('requête SQL');** (mysql_query cela dépend au serveur à qui adresser la requête, le MySQL, mssql_query() au cas d'un serveur SQLServer), le résultat :

- **Vrai (1) :** La requête s'est déroulée normalement.
- **Faux(0) :** Une erreur s'est produite lors de l'exécution de la requête MySQL.

```
<?php
//Connexion avec server en local
:
:
//Création d'une nouvelle table
$result = mysql_query('CREATE TABLE persons(id int(3) PRIMARY KEY AUTO_INCREMENT, nom varchar(25), prenom varchar(25))');
if($result)
    echo 'La requête a été bien exécutée';
else
    echo 'Erreur : '.mysql_error();
```


?>

Cette fonction **mysql_query('requêtes')** (requêtes avec **s**) peut envoyer une ou plusieurs requêtes à la fois au serveur, même qu'il est déconseillé, il faut envoyer une par une en récupérant à chaque fois le résultat d'exécution de la requête.

Les requêtes envoyées au serveur sont soit des requêtes LDD (Langage de Définition de données) qui permet la description de la structure de la base (tables, vues, index, attributs, ...) ou des requêtes LMD (Langage de Manipulation de Données) permet la manipulation des tables et des vues.

Pour ne pas monopoliser la communication avec la base de données, il faut libérer la connexion, pour qu'un autre puisse l'utiliser.

```
<?php
    //Connexion avec server en local
    //Traitement
    //Fermeture de la connexion
    mysql_close() or die('Problème lors de la fermeture de la connexion : '.mysql_error());
?>
```

II. Programmez-en orienté objet en PHP

Programmez-en orienté objet en PHP

Bienvenue dans ce tutoriel sur la programmation orientée objet (souvent abrégé par ses initiales « POO ») en PHP.

Ici, vous allez découvrir un nouveau moyen de penser votre code, un nouveau moyen de le concevoir. Vous allez le représenter de façon *orienté objet*, un moyen de conception inventé dans les années 1970 et qui prend de plus en plus de place aujourd'hui. La principale raison de ce succès est due à de nombreux avantages apportés par ce paradigme, comme une organisation plus cohérente de vos projets, une maintenance plus facile et une distribution de votre code plus aisée.

Cependant, avant de vous lancer dans ce (très) vaste domaine, vous devez avoir quelques connaissances au préalable.

Ce qui doit être acquis

Afin de suivre au mieux ce tutoriel, il est indispensable voire obligatoire :

- d'être à l'aise avec PHP et sa syntaxe. Si ce n'est pas le cas, le Site du Zéro propose [un tutoriel](#) ;
- d'avoir bien pratiqué ;
- d'être patient ;
- d'avoir PHP 5 sur son serveur. Je ne parlerai pas de POO en PHP 4 car sous cette version de PHP, certaines fonctions indispensables de la POO ne sont pas présentes (on ne peut donc pas vraiment parler de POO).

Si vous avez déjà pratiqué d'autres langages apportant la possibilité de programmer orienté objet, c'est un gros plus, surtout si vous savez programmer en Java (PHP a principalement tiré son modèle objet de ce langage).

Introduction à la POO

Alors ça y est, vous avez décidé de vous lancer dans la POO en PHP ? Sage décision ! Nous allons donc plonger dans ce vaste domaine par une introduction à cette nouvelle façon de penser : qu'est-ce que la POO ? En quoi ça consiste ? En quoi est-ce si différent de la méthode que vous employez pour développer votre site web ? Tant de questions auxquelles je vais répondre.

Cependant, puisque je sais que vous avez hâte de commencer, nous allons entamer sérieusement les choses en créant notre première **classe** dès la fin de ce chapitre. Vous commencerez ainsi vos premiers pas dans la POO en PHP !

Qu'est-ce que la POO ?

Il était une fois le procédural

Commençons ce cours en vous posant une question : comment est représenté votre code ? La réponse est unique : vous avez utilisé la « représentation procédurale » qui consiste à séparer le traitement des données des données elles-mêmes. Par exemple, vous avez un système de news sur votre site. D'un côté, vous avez les données (les news, une liste d'erreurs, une connexion à la BDD, etc.) et de l'autre côté vous avez une suite d'instructions qui viennent modifier ces données. Si je ne me trompe pas, c'est de cette manière que vous codez.

Cette façon de se représenter votre application vous semble sans doute la meilleure puisque c'est la seule que vous connaissez. D'ailleurs, vous ne voyez pas trop comment votre code pourrait être représenté de manière différente. Eh bien cette époque d'ignorance est révolue : voici maintenant la programmation orientée objet !

Puis naquit la programmation orientée objet

Alors, qu'est-ce donc que cette façon de représenter son code ? La POO, c'est tout simplement faire de son site un ensemble d'objets qui interagissent entre eux. En d'autres termes : tout est objet.

Définition d'un objet

Je suis sûr que vous savez ce que c'est. D'ailleurs, vous en avez pas mal à côté de vous : je suis sûr que vous avez un ordinateur, une lampe, une chaise, un bureau, ou que sais-je encore. Ce sont tous des objets. En programmation, les objets sont sensiblement la même chose.

L'exemple le plus pertinent quand on fait un cours sur la POO est d'utiliser l'exemple du personnage dans un jeu de combat. Ainsi, imaginons que nous ayons un objet **Personnage** dans notre application. Un personnage a des caractéristiques :

- une force ;
- une localisation ;
- une certaine expérience ;
- et enfin des dégâts.

Toutes ses caractéristiques correspondent à des valeurs. Comme vous le savez sûrement, les valeurs sont stockées dans des variables. C'est toujours le cas en POO. Ce sont des variables un peu spéciales, mais nous y reviendrons plus tard.

Mis à part ces caractéristiques, un personnage a aussi des capacités. Il peut :

- frapper un autre personnage ;
- gagner de l'expérience ;
- se déplacer.

Ces capacités correspondent à des fonctions. Comme pour les variables, ce sont des fonctions un peu spéciales et on y reviendra en temps voulu. En tout cas, le principe est là.

Vous savez désormais qu'on peut avoir des objets dans une application. Mais d'où sortent-ils ? Dans la vie réelle, un objet ne sort pas de nulle part. En effet, chaque objet est défini selon des caractéristiques et un plan bien précis. En POO, ces informations sont contenues dans ce qu'on appelle des **classes**.

Définition d'une classe

Comme je viens de le dire, les classes contiennent la définition des objets que l'on va créer par la suite. Prenons l'exemple le plus simple du monde : les gâteaux et leur moule. Le moule est unique. Il peut produire une quantité infinie de gâteaux. Dans ce cas-là, les gâteaux sont les *objets* et le moule est la *classe* : le moule va définir la forme du gâteau. La classe contient donc le plan de fabrication d'un objet et on peut s'en servir autant qu'on veut afin d'obtenir une infinité d'objets.

Concrètement, une classe, c'est quoi ?

Une classe est une entité regroupant des variables et des fonctions. Chacune de ces fonctions aura accès aux variables de cette entité. Dans le cas du personnage, nous aurons une fonction `frapper()`. Cette fonction devra simplement modifier la variable `$degats` du personnage en fonction de la variable `$force`. Une classe est donc un regroupement logique de variables et fonctions que tout objet issu de cette classe possédera.

Définition d'une instance

Une instance, c'est tout simplement le résultat d'une *instanciation*. Une *instanciation*, c'est le fait d'*instancier* une classe. *Instancier* une classe, c'est se servir d'une classe afin qu'elle nous crée un objet. En gros, une instance est un objet.

Exemple : création d'une classe

Nous allons créer une classe `Personnage` (sous forme de schéma bien entendu). Celle-ci doit contenir la liste des variables et des fonctions que l'on a citées plus haut : c'est la base de tout objet `Personnage`. Chaque instance de cette classe possédera ainsi toutes ces variables et fonctions. Voici donc cette fameuse classe à la figure suivante.

Classe Personnage



Le schéma de notre classe

Vous voyez donc les variables et fonctions stockées dans la classe `Personnage`. Sachez qu'en réalité, on ne les appelle pas comme ça : il s'agit d'**attributs** (ou propriétés) et de **méthodes**. Un attribut désigne une variable et une méthode désigne une fonction.

Ainsi, tout objet `Personnage` aura ces attributs et méthodes. On pourra modifier ces attributs et invoquer ces méthodes sur notre objet afin de modifier ses caractéristiques ou son comportement.

Le principe d'encapsulation

L'un des gros avantages de la POO est que l'on peut masquer le code à l'utilisateur (l'utilisateur est ici celui qui se servira de la classe, pas celui qui chargera la page depuis son navigateur). Le concepteur de la classe a englobé dans celle-ci un code qui peut être assez complexe et il est donc inutile voire dangereux de laisser l'utilisateur manipuler ces objets sans aucune restriction. Ainsi, il est important d'interdire à l'utilisateur de modifier directement les attributs d'un objet.

Prenons l'exemple d'un avion où sont disponibles des centaines de boutons. Chacun de ces boutons constituent des actions que l'on peut effectuer sur l'avion. C'est l'*interface* de l'avion. Le pilote se moque de quoi est composé l'avion : son rôle est de le piloter. Pour cela, il va se servir des boutons afin de manipuler les composants de l'avion. Le pilote ne doit pas se charger de modifier manuellement ces composants : il pourrait faire de grosses bêtises.

Le principe est exactement le même pour la POO : l'utilisateur de la classe doit se contenter d'invoquer les méthodes en ignorant les attributs. Comme le pilote de l'avion, il n'a pas à les trituer. Pour instaurer une telle contrainte, on dit que les attributs sont **privés**. Pour l'instant, ceci peut sans doute vous paraître abstrait, mais nous y reviendrons. ;)

Bon, je pense que j'ai assez parlé, commençons par créer notre première classe !

Créer une classe

Syntaxe de base

Le but de cette section va être de traduire la figure précédente en code PHP. Avant cela, je vais vous donner la syntaxe de base de toute classe en PHP :

```
<?php

class Personnage // Présence du mot-clé class suivi du nom de la classe.

{

    // Déclaration des attributs et méthodes ici.

}

?>
```

Cette syntaxe est à retenir absolument. Heureusement, elle est simple.

Ce qu'on vient de faire est donc de créer le moule, le plan qui définira nos objets. On verra dans le prochain chapitre comment utiliser ce plan afin de créer un objet. Pour l'instant, contentons-nous de construire ce plan et de lui ajouter des fonctionnalités.

La déclaration d'attributs dans une classe se fait en écrivant le nom de l'attribut à créer, précédé de sa **visibilité**.

Visibilité d'un attribut ou d'une méthode

La visibilité d'un attribut ou d'une méthode indique à partir d'où on peut y avoir accès. Nous allons voir ici deux types de visibilité : `public` et `private`.

Le premier, `public`, est le plus simple. Si un attribut ou une méthode est `public`, alors on pourra y avoir accès depuis n'importe où, depuis l'intérieur de l'objet (dans les méthodes qu'on a créées), comme depuis l'extérieur. Je m'explique. Quand on crée un objet, c'est principalement pour pouvoir exploiter ses attributs et méthodes. L'extérieur de l'objet, c'est tout le code qui n'est pas *dans* votre classe. En effet, quand vous créerez un objet, cet objet sera représenté par une variable, et c'est à partir d'elle qu'on pourra modifier l'objet, appeler des méthodes, etc. Vous allez donc dire à PHP « dans cet objet, donne-moi cet attribut » ou « dans cet objet, appelle cette méthode » : c'est ça, appeler des attributs ou méthodes depuis l'extérieur de l'objet.

Le second, `private`, impose quelques restrictions. On n'aura accès aux attributs et méthodes *seulement* depuis l'intérieur de la classe, c'est-à-dire que seul le code voulant accéder à un attribut privé ou une méthode privée écrit(e) à l'intérieur de la classe fonctionnera. Sinon, une jolie erreur fatale s'affichera disant que vous ne pouvez pas accéder à telle méthode ou tel attribut parce qu'il ou elle est privé(e).

Là, ça devrait faire *tilt* dans votre tête : le principe d'encapsulation ! C'est de cette manière qu'on peut interdire l'accès à nos attributs.

Création d'attributs

Pour déclarer des attributs, on va donc les écrire entre les accolades, les uns à la suite des autres, en faisant précéder leurs noms du mot-clé `private`, comme ça :

```
<?php
class Personnage
{
    private $_force;    // La force du personnage
    private $_localisation; // Sa localisation
    private $_experience; // Son expérience
    private $_degats;    // Ses dégâts
}
?>
```

Vous pouvez constater que chaque attribut est précédé d'un underscore (« _ »). Ceci est une notation qu'il est préférable de respecter (il s'agit de la notation PEAR) qui dit que chaque nom d'élément privé (ici il s'agit d'attributs, mais nous verrons plus tard qu'il peut aussi s'agir de méthodes) doit être précédé d'un underscore.

Vous pouvez initialiser les attributs lorsque vous les déclarez (par exemple, leur mettre une valeur de 0 ou autre). Exemple :

```
<?php
class Personnage
{
    private $_force = 50;    // La force du personnage, par défaut à 50.
    private $_localisation = 'Lyon'; // Sa localisation, par défaut à Lyon.
    private $_experience = 1;    // Son expérience, par défaut à 1.
    private $_degats = 0;    // Ses dégâts, par défaut à 0.
}
?>
```

La valeur que vous leur donnez par défaut doit être une expression constante. Par conséquent, leur valeur ne peut être issue d'un appel à une fonction (`private $_attribut = intval('azerty')`), d'une opération (`private $_attribut = 1 + 1`), d'une concaténation (`private $_attribut = 'Mon ' . 'super ' . 'attribut'`) ou d'une variable, superglobale ou non (`private $_attribut = $_SERVER['REQUEST_URI']`).

Création de méthodes

Pour la déclaration de méthodes, il suffit de faire précéder le mot-clé `function` à la visibilité de la méthode. Les types de visibilité des méthodes sont les mêmes que les attributs. Les méthodes n'ont en général pas besoin d'être masquées à l'utilisateur, vous les mettez souvent en `public` (à moins que vous teniez absolument à ce que l'utilisateur ne puisse pas appeler cette méthode, par exemple s'il s'agit d'une fonction qui simplifie certaines tâches sur l'objet mais qui ne doit pas être appelée n'importe comment).

```
<?php
class Personnage
{
```

```

private $_force;    // La force du personnage
private $_localisation; // Sa localisation
private $_experience; // Son expérience
private $_degats;    // Ses dégâts

public function deplacer() // Une méthode qui déplacera le personnage (modifiera sa localisation).
{

}

public function frapper() // Une méthode qui frappera un personnage (suivant la force qu'il a).
{

}

public function gagnerExperience() // Une méthode augmentant l'attribut $experience du personnage.
{

}
}
?>

```

Et voilà. :)

De même que l'underscore précédant les noms d'éléments privés, vous pouvez remarquer que le nom des classes commence par une majuscule. Il s'agit aussi du respect de la notation PEAR.

En résumé

- Une classe, c'est un ensemble de variables et de fonctions (attributs et méthodes).
- Un objet, c'est une *instance* de la classe pour pouvoir l'utiliser.
- Tous vos attributs doivent être privés. Pour les méthodes, peu importe leur visibilité. C'est ce qu'on appelle le principe d'encapsulation.
- On déclare une classe avec le mot-clé `class` suivi du nom de la classe, et enfin deux accolades ouvrantes et fermantes qui encercleront la liste des attributs et méthodes.

Utiliser la classe

Une classe, c'est bien beau mais, au même titre qu'un plan de construction pour une maison, si on ne sait pas comment se servir de notre plan pour construire notre maison, cela ne sert à rien ! Nous verrons donc ainsi comment se servir de notre **classe**, notre modèle de base, afin de créer des **objets** et pouvoir s'en servir.

Ce chapitre sera aussi l'occasion d'approfondir un peu le développement de nos classes : actuellement, la classe que l'on a créée contient des méthodes vides, chose un peu inutile vous avouerez. Pas de panique, on va s'occuper de tout ça !

Créer et manipuler un objet

Créer un objet

Nous allons voir comment créer un objet, c'est-à-dire que nous allons utiliser notre classe afin qu'elle nous fournisse un objet. Pour créer un nouvel objet, vous devez faire précéder le nom de la classe à instancier du mot-clé `new`, comme ceci :

```
<?php
$perso = new Personnage();
?>
```

Ainsi, `$perso` sera un objet de type `Personnage`. On dit que l'on *instancie* la classe `Personnage`, que l'on crée une instance de la classe `Personnage`.

Appeler les méthodes de l'objet

Pour appeler une méthode d'un objet, il va falloir utiliser un opérateur : il s'agit de l'opérateur `->` (une flèche composée d'un tiret suivi d'un chevron fermant). Celui-ci s'utilise de la manière suivante. À gauche de cet opérateur, on place l'**objet** que l'on veut utiliser. Dans l'exemple pris juste au-dessus, cet objet aurait été `$perso`. À droite de l'opérateur, on spécifie le nom de la méthode que l'on veut invoquer.

Et puisqu'un exemple vaut mieux qu'un long discours...

```
<?php
// Nous créons une classe « Personnage ».
class Personnage
{
    private $_force;
    private $_localisation;
    private $_experience;
    private $_degats;

    // Nous déclarons une méthode dont le seul but est d'afficher un texte.
    public function parler()
    {
        echo 'Je suis un personnage !';
    }
}

$perso = new Personnage();
$perso->parler();
```

La ligne 18 signifie donc « va chercher l'objet `$perso`, et invoque la méthode `parler()` sur cet objet ». Notez donc bien quelque part l'utilisation de cet opérateur : de manière générale, il sert à accéder à un élément de la classe. Ici, on s'en est servi pour atteindre une méthode, mais nous verrons plus tard qu'il nous permettra aussi d'atteindre un attribut.

Accéder à un élément depuis la classe

Je viens de vous faire créer un objet, et vous êtes maintenant capables d'appeler une méthode. Cependant, le contenu de cette dernière était assez simpliste : elle ne faisait qu'afficher un message. Ici, nous allons voir comment une méthode peut accéder aux attributs de l'objet.

Lorsque vous avez un objet, vous savez que vous pouvez invoquer des méthodes grâce à l'opérateur « `->` ». Je vous ai par ailleurs dit que c'était également grâce à cet opérateur qu'on accédait aux attributs de la classe. Cependant, rappelez-vous : nos attributs sont privés. Par conséquent, ce code lèvera une erreur fatale :

```
<?php
```



```

class Personnage
{
    private $_force;
    private $_experience;
    private $_degats;
}

$perso = new Personnage();
$perso->_experience = $perso->_experience + 1; // Une erreur fatale est levée suite à cette instruction.

```

Ici, on essaye d'accéder à un attribut privé hors de la classe. Ceci est interdit, donc PHP lève une erreur. Dans notre exemple (qui essaye en vain d'augmenter de 1 l'expérience du personnage), il faudra demander à la classe d'augmenter l'expérience. Pour cela, nous allons écrire une méthode `gagnerExperience()` :

```

<?php
class Personnage
{
    private $_experience;

    public function gagnerExperience()
    {
        // Cette méthode doit ajouter 1 à l'expérience du personnage.
    }
}

$perso = new Personnage();
$perso->gagnerExperience();

```

Oui mais voilà : comment accéder à l'attribut `$_experience` dans notre méthode ? C'est là qu'intervient la pseudo-variable `$this`.

Dans notre script, `$perso` représente l'objet. Il est donc facile d'appeler une méthode à partir de cette variable. Mais dans notre méthode, nous n'avons pas cette variable pour accéder à notre attribut `$_experience` pour le modifier ! Du moins, c'est ce que vous croyez. En fait, un paramètre représentant l'objet est passé implicitement à chaque méthode de la classe. Regardez plutôt cet exemple :

```

<?php
class Personnage
{
    private $_experience = 50;

    public function afficherExperience()
    {
        echo $this->_experience;
    }
}

$perso = new Personnage();
$perso->afficherExperience();

```

Commentons la ligne 8. Vous avez sans doute reconnu la structure `echo` ayant pour rôle d'afficher une valeur. Ensuite, vous reconnaissez la variable `$this` dont je vous ai parlé : elle représente l'objet que nous sommes en train d'utiliser. Ainsi, dans ce script, les variables `$this` et `$perso` représentent le même objet. L'instruction

surlignée veut donc dire : « Affiche-moi cette valeur : dans l'objet utilisé (donc `$perso`), donne-moi la valeur de l'attribut `$_experience`. »

Ainsi, je vais vous demander d'écrire la méthode `gagnerExperience()`. Cette méthode devra ajouter 1 à l'attribut `$_experience`. Je suis sûr que vous pouvez y arriver ! :)

Voici la correction :

```
<?php
class Personnage
{
    private $_experience = 50;

    public function afficherExperience()
    {
        echo $this->_experience;
    }

    public function gagnerExperience()
    {
        // On ajoute 1 à notre attribut $_experience.
        $this->_experience = $this->_experience + 1;
    }
}

$perso = new Personnage();
$perso->gagnerExperience(); // On gagne de l'expérience.
$perso->afficherExperience(); // On affiche la nouvelle valeur de l'attribut.
```

Implémenter d'autres méthodes

Nous avons créé notre classe Personnage et déjà une méthode, `gagnerExperience()`. J'aimerais qu'on en implémente une autre : la méthode `frapper()`, qui devra infliger des dégâts à un personnage.

Pour partir sur une base commune, nous allons travailler sur cette classe :

```
<?php
class Personnage
{
    private $_degats = 0; // Les dégâts du personnage.
    private $_experience = 0; // L'expérience du personnage.
    private $_force = 20; // La force du personnage (plus elle est grande, plus l'attaque est puissante).

    public function gagnerExperience()
    {
        // On ajoute 1 à notre attribut $_experience.
        $this->_experience = $this->_experience + 1;
    }
}
```

Commençons par écrire notre méthode `frapper()`. Cette méthode doit accepter un argument : le personnage à frapper. La méthode aura juste pour rôle d'augmenter les dégâts du personnage passé en paramètre.

Pour vous aider à visualiser le contenu de la méthode, imaginez votre code manipulant des objets. Il doit ressembler à ceci :

```
<?php
// On crée deux personnages
$perso1 = new Personnage();
$perso2 = new Personnage();

// Ensuite, on veut que le personnage n°1 frappe le personnage n°2.
$perso1->frapper($perso2);
```

Pour résumer :

- la méthode `frapper()` demande un argument : le personnage à frapper ;
- cette méthode augmente les dégâts du personnage à frapper en fonction de la force du personnage qui frappe.

On pourrait donc imaginer une classe ressemblant à ceci :

```
<?php
class Personnage
{
    private $_degats; // Les dégâts du personnage.
    private $_experience; // L'expérience du personnage.
    private $_force; // La force du personnage (plus elle est grande, plus l'attaque est puissante).

    public function frapper($persoAFrapper)
    {
        $persoAFrapper->_degats += $this->_force;
    }

    public function gagnerExperience()
    {
        // On ajoute 1 à notre attribut $_experience.
        $this->_experience = $this->_experience + 1;
    }
}
```

Commentons ensemble le contenu de cette méthode `frapper()`. Celle-ci comporte une instruction composée de deux parties :

1. La première consiste à dire à PHP que l'on veut assigner une nouvelle valeur à l'attribut `$_degats` du personnage à frapper.
2. La seconde partie consiste à donner à PHP la valeur que l'on veut assigner. Ici, nous voyons que cette valeur est atteinte par `$this->_force`.

Maintenant, grosse question : la variable `$this` fait-elle référence au personnage qui frappe ou au personnage frappé ? Pour répondre à cette question, il faut savoir sur quel objet est appelée la méthode. En effet, souvenez-vous que `$this` est une variable représentant l'objet à partir duquel on a appelé la méthode. Dans notre cas, on a appelé la méthode `frapper()` à partir du personnage qui frappe, donc `$this` représente le personnage qui frappe.

L'instruction contenue dans la méthode signifie donc : « Ajoute la valeur de la force du personnage qui frappe à l'attribut `$_degats` du personnage frappé. »

Maintenant, nous pouvons créer une sorte de petite mise en scène qui fait interagir nos personnages. Par exemple, nous pouvons créer un script qui fait combattre les personnages. Le personnage 1 frapperait le personnage 2 puis gagnerait de l'expérience, puis le personnage 2 frapperait le personnage 1 et gagnerait de l'expérience. Procédez étape par étape :

- créez deux personnages ;
- faites frapper le personnage 1 ;
- faites gagner de l'expérience au personnage 1 ;
- faites frapper le personnage 2 ;
- faites gagner de l'expérience au personnage 2.

Ce script n'est qu'une suite d'appels de méthodes. Chaque puce (sauf la première) correspond à l'appel d'une méthode, ce que vous savez faire. En conclusion, vous êtes aptes à créer ce petit script ! Voici la correction :

```
<?php
$perso1 = new Personnage(); // Un premier personnage
$perso2 = new Personnage(); // Un second personnage

$perso1->frapper($perso2); // $perso1 frappe $perso2
$perso1->gagnerExperience(); // $perso1 gagne de l'expérience

$perso2->frapper($perso1); // $perso2 frappe $perso1
$perso2->gagnerExperience(); // $perso2 gagne de l'expérience
?>
```

Cependant, un petit problème se pose. Puisque, à la base, les deux personnages ont le même niveau de dégâts, la même expérience et la même force, ils seront à la fin toujours égaux. Pour pallier ce problème, il faudrait pouvoir assigner des valeurs spécifiques aux deux personnages, afin que le combat puisse les différencier. Or, vous ne pouvez pas accéder aux attributs en-dehors de la classe ! Pour savoir comment résoudre ce problème, je vais vous apprendre deux nouveaux mots : **accesseur** et **mutateur**. Mais avant, j'aimerais faire une petite parenthèse.

Exiger des objets en paramètre

Reprenons l'exemple du code auquel nous sommes arrivés et concentrons-nous sur la méthode `frapper()`. Celle-ci accepte un argument : un personnage à frapper. Cependant, qu'est-ce qui vous garantit qu'on passe effectivement un personnage à frapper ? On pourrait très bien passer un argument complètement différent, comme un nombre par exemple :

```
<?php
$perso = new Personnage();
$perso->frapper(42);
?>
```

Et là, qu'est-ce qui se passe ? Une erreur est générée car, à l'intérieur de la méthode `frapper()`, nous essayons d'appeler une méthode sur le paramètre qui n'est pas un objet. C'est comme si on avait fait ça :

```
<?php
$persoAFrapper = 42;
$persoAFrapper->_degats += 50; // Le nombre 50 est arbitraire, il est censé représenter une force.
?>
```

Ce qui n'a aucun sens. Il faut donc s'assurer que le paramètre passé est bien un personnage, sinon PHP arrête tout et n'exécute pas la méthode. Pour cela, il suffit d'ajouter un seul mot : le nom de la classe dont le paramètre

doit être un objet. Dans notre cas, si le paramètre doit être un objet de type `Personnage`, alors il faudra ajouter le mot-clé `Personnage`, juste avant le nom du paramètre, comme ceci :

```
<?php
class Personnage
{
    // ...

    public function frapper(Personnage $persoAFrapper)
    {
        // ...
    }
}
```

Grâce à ça, vous êtes sûrs que la méthode `frapper()` ne sera exécutée *que* si le paramètre passé est de type `Personnage`, sinon PHP interrompt tout le script. Vous pouvez donc appeler les méthodes de l'objet sans crainte qu'un autre type de variable soit passé en paramètre.

Le type de la variable à spécifier doit obligatoirement être un nom de classe ou alors un tableau. Si vous voulez exiger un tableau, faites précéder le nom du paramètre devant être un tableau par le mot-clé `array` comme ceci : `public function frapper(array $coups)`. Vous ne pouvez pas exiger autre chose : par exemple, il est impossible d'exiger un nombre entier ou une chaîne de caractères de cette façon.

Les accesseurs et mutateurs

Comme vous le savez, le principe d'encapsulation nous empêche d'accéder directement aux attributs de notre objet puisqu'ils sont privés : seule la classe peut les lire et les modifier. Par conséquent, si vous voulez récupérer un attribut, il va falloir le demander à la classe, de même si vous voulez les modifier.

Accéder à un attribut : l'accesseur

À votre avis, comment peut-on faire pour récupérer la valeur d'un attribut ? La solution est simple : nous allons implémenter des méthodes dont le seul rôle sera de nous donner l'attribut qu'on leur demande ! Ces méthodes ont un nom bien spécial : ce sont des **accesseurs** (ou *getters*). Par convention, ces méthodes portent le même nom que l'attribut dont elles renvoient la valeur. Par exemple, voici la liste des accesseurs de notre classe `Personnage` :

```
<?php
class Personnage
{
    private $_force;
    private $_experience;
    private $_degats;

    public function frapper(Personnage $persoAFrapper)
    {
        $persoAFrapper->_degats += $this->_force;
    }

    public function gagnerExperience()
    {
        // Ceci est un raccourci qui équivaut à écrire « $this->_experience = $this->_experience + 1 »
    }
}
```

```

// On aurait aussi pu écrire « $this->_experience += 1 »
$this->_experience++;
}

// Ceci est la méthode degats() : elle se charge de renvoyer le contenu de l'attribut $_degats.
public function degats()
{
    return $this->_degats;
}

// Ceci est la méthode force() : elle se charge de renvoyer le contenu de l'attribut $_force.
public function force()
{
    return $this->_force;
}

// Ceci est la méthode experience() : elle se charge de renvoyer le contenu de l'attribut $_experience.
public function experience()
{
    return $this->_experience;
}
}

```

Modifier la valeur d'un attribut : les mutateurs

Maintenant, comment cela se passe-t-il si vous voulez modifier un attribut ? Encore une fois, il va falloir que vous demandiez à la classe de le faire pour vous. Je vous rappelle que le principe d'encapsulation est là pour vous empêcher d'assigner un mauvais type de valeur à un attribut : si vous demandez à votre classe de le faire, ce risque est supprimé car la classe « contrôle » la valeur des attributs. Comme vous l'aurez peut-être deviné, ce sera par le biais de méthodes que l'on demandera à notre classe de modifier tel attribut.

La classe doit impérativement contrôler la valeur afin d'assurer son intégrité car, si elle ne le fait pas, on pourra passer n'importe quelle valeur à la classe et le principe d'encapsulation n'est plus respecté ! Ces méthodes ont aussi un nom spécial : il s'agit de **mutateurs** (ou *setters*). Ces méthodes sont de la forme `setNomDeLAttribut()`.

Voici la liste des mutateurs (ajoutée à la liste des accesseurs) de notre classe `Personnage` :

```

<?php
class Personnage
{
    private $_force;
    private $_experience;
    private $_degats;

    public function frapper(Personnage $persoAFrapper)
    {
        $persoAFrapper->_degats += $this->_force;
    }

    public function gagnerExperience()
    {
        $this->_experience++;
    }
}

```

```

// Mutateur chargé de modifier l'attribut $_force.
public function setForce($force)
{
    if (!is_int($force)) // S'il ne s'agit pas d'un nombre entier.
    {
        trigger_error('La force d\'un personnage doit être un nombre entier', E_USER_WARNING);
        return;
    }

    if ($force > 100) // On vérifie bien qu'on ne souhaite pas assigner une valeur supérieure à 100.
    {
        trigger_error('La force d\'un personnage ne peut dépasser 100', E_USER_WARNING);
        return;
    }

    $this->_force = $force;
}

// Mutateur chargé de modifier l'attribut $_experience.
public function setExperience($experience)
{
    if (!is_int($experience)) // S'il ne s'agit pas d'un nombre entier.
    {
        trigger_error('L\'expérience d\'un personnage doit être un nombre entier', E_USER_WARNING);
        return;
    }

    if ($experience > 100) // On vérifie bien qu'on ne souhaite pas assigner une valeur supérieure à 100.
    {
        trigger_error('L\'expérience d\'un personnage ne peut dépasser 100', E_USER_WARNING);
        return;
    }

    $this->_experience = $experience;
}

// Ceci est la méthode degats() : elle se charge de renvoyer le contenu de l'attribut $_degats.
public function degats()
{
    return $this->_degats;
}

// Ceci est la méthode force() : elle se charge de renvoyer le contenu de l'attribut $_force.
public function force()
{
    return $this->_force;
}

// Ceci est la méthode experience() : elle se charge de renvoyer le contenu de l'attribut $_experience.
public function experience()
{
    return $this->_experience;
}

```

```
}  
}
```

Voilà ce que j'avais à dire concernant ces accesseurs et mutateurs. Retenez bien ces définitions, vous les trouverez dans la plupart des classes !

Retour sur notre script de combat

Maintenant que nous avons vu ce qu'étaient des accesseurs et des mutateurs, nous pouvons améliorer notre script de combat. Pour commencer, je vais vous demander d'afficher, à la fin du script, la force, l'expérience et le niveau de dégâts de chaque personnage.

Voici la correction :

```
<?php  
  
$perso1 = new Personnage(); // Un premier personnage  
  
$perso2 = new Personnage(); // Un second personnage  
  
  
$perso1->frapper($perso2); // $perso1 frappe $perso2  
  
$perso1->gagnerExperience(); // $perso1 gagne de l'expérience  
  
  
$perso2->frapper($perso1); // $perso2 frappe $perso1  
  
$perso2->gagnerExperience(); // $perso2 gagne de l'expérience  
  
  
echo 'Le personnage 1 a ', $perso1->force(), ' de force, contrairement au personnage 2 qui a ', $perso2->  
force(), ' de force.<br />';  
  
echo 'Le personnage 1 a ', $perso1->experience(), ' d\'expérience, contrairement au personnage 2 qui a ',  
$perso2->experience(), ' d\'expérience.<br />';  
  
echo 'Le personnage 1 a ', $perso1->degats(), ' de dégâts, contrairement au personnage 2 qui a ', $perso2  
->degats(), ' de dégâts.<br />';
```

Comme nous l'avions dit, les valeurs finales des deux personnages sont identiques. Pour pallier ce problème, nous allons modifier, juste après la création des personnages, la valeur de la force et de l'expérience des deux personnages. Vous pouvez par exemple favoriser un personnage en lui donnant une plus grande force et une plus grande expérience par rapport au deuxième.

Voici la correction que je vous propose (peu importe les valeurs que vous avez choisies, l'essentiel est que vous ayez appelé les bonnes méthodes) :

```
<?php  
$perso1 = new Personnage(); // Un premier personnage  
$perso2 = new Personnage(); // Un second personnage  
  
$perso1->setForce(10);  
$perso1->setExperience(2);  
  
$perso2->setForce(90);  
$perso2->setExperience(58);
```

```
$perso1->frapper($perso2); // $perso1 frappe $perso2
$perso1->gagnerExperience(); // $perso1 gagne de l'expérience

$perso2->frapper($perso1); // $perso2 frappe $perso1
$perso2->gagnerExperience(); // $perso2 gagne de l'expérience

echo 'Le personnage 1 a ', $perso1->force(), ' de force, contrairement au personnage 2 qui a ', $perso2->force(),
' de force.<br />';
echo 'Le personnage 1 a ', $perso1->experience(), ' d\'expérience, contrairement au personnage 2 qui a ', $perso2->experience(),
' d\'expérience.<br />';
echo 'Le personnage 1 a ', $perso1->degats(), ' de dégâts, contrairement au personnage 2 qui a ', $perso2->degats(),
' de dégâts.<br />';
```

Ce qui affichera :



Résultat affiché par le script

Comme vous le voyez, à la fin, les deux personnages n'ont plus les mêmes caractéristiques !

Pour bien être sûr que vous me suiviez toujours, je vous ai fait un schéma résumant le déroulement du script.