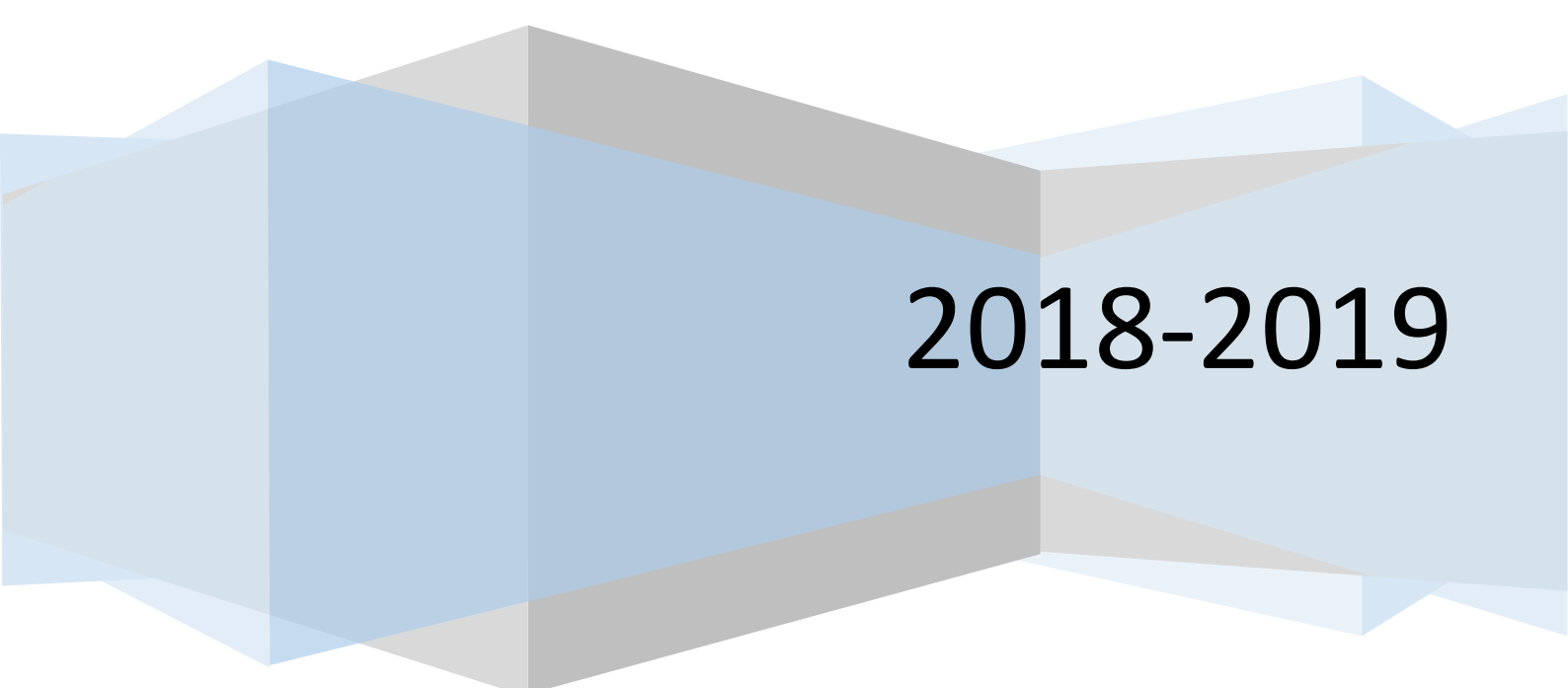


Annick Lacroix

PHP

Notes de Cours AFPA

DWWM



2018-2019

PHP

PHP

Table des matières

1. Introduction au PhP	4
2. Comment lire sa page PhP	5
1- Lire sa page php	5
3. Les variables.....	7
1- Introduction d'une variable	7
2- Les variables booléennes (true or false)	8
3- Les incrémentations :.....	8
4. Les fonctions	8
1- Les variables externes dans les fonctions :.....	9
2- Les modifications de variables :.....	9
3- Les paramètres par défaut :.....	9
5. Les tableaux :	10
1- Parcourir un tableau :	11
2- Parcourir un tableau de façon incursive avec une fonction qui s'appelle elle-même :	11
A. Afficher un tableau lui-même contenu dans un tableau :	11
3- Les implode, explode et str_split :.....	12
4- Les constantes :.....	13
5- Les boucles	14
B. La boucle PHP while.....	14
C. La boucle PHP do...while.....	16
D. La boucle PHP for.....	17
E. La boucle PHP foreach.....	18
F. Quelle boucle utiliser dans quelle circonstance ?.....	18
6- La fonction return	19
A. Le return :	20
B. Tester des tableaux associatifs stockés dans un tableau indexé :	20
6. La collecte de données de formulaires avec GET et POST.....	21
A. Compter la longueur d'une string	23
B. Remplacer un caractère ou une chaîne de caractères dans une string	23
C. Les strings en majuscule ou minuscule	23
D. Obtenir l'adresse ip	24

PHP

E. Différents scripts	24
F. Obtenir le navigateur	30
G. Pour voir le nom du fichier courant	31
H. Créer un nouveau fichier via PHP	31
I. Compter les lignes d'une page	31
J. Enlever les doublons d'un tableau	31
K. Opérer une redirection automatique	32
7. La fonction isset	32
8. Transformer du binaire en décimales	32
9. Les classes	33
10. Les get et set dans les classes	34
1- Les constructeurs	34
2- L'héritage	36
3- Les différents types de propriétés	37
A. Les propriétés publiques :	37
B. Les propriétés privées (private)	37
C. Les propriétés protégées (protected)	37
11. La fonction func_get_arg	37

PHP

1. Introduction au PhP


PHP (officiellement, ce sigle est un acronyme récursif pour *PHP Hypertext Preprocessor*) est un langage de scripts généraliste et Open Source, spécialement conçu pour le développement d'applications web. Il peut être intégré facilement au HTML. Le PhP est proche du Javascript.

Le PhP est introduit dans la page HTML via une balise placée entre les balises *body*

```
<?php  
  
?>
```

Pour afficher quelque chose grâce à PHP, on utilisera le mot clef **echo**.

```
<?php  
echo "Bienvenue dans le monde du PHP";  
?>
```



Echo veut dire : Imprimer à l'écran
Il se place à chaque début de ligne

Le console.log qu'on utilise sur JS pour vérifier une variable est différente sur PHP.

En effet, il faut utiliser *var-dump(contenu à tester)* ;

Contrairement à JS, il n'y a pas de console et le résultat apparaîtra directement sur la page.

Dès lors qu'il y a du PHP dans une page, alors cette page devra avoir l'extension .php et non .html .

```
<!DOCTYPE HTML>  
<html>  
<head>  
<title>Exemple</title>  
</head>  
<body>  
  
<?php  
echo "Bonjour, je suis un script PHP !";  
?>  
  
</body>  
</html>
```

Pour concaténer, en JS on va utiliser +.

En PHP on va utiliser un point.

```
echo "bonjour "." monsieur";
```

PHP

On peut afficher avec des guillemets et des apostrophes.

Toutefois, les apostrophes vont rendre un texte littéral alors que les guillemets vont interpréter les variables ou autres qui sont à l'intérieur :

```
$var1='Hello DWWM';  
echo '$var1';  
echo '<br>';  
echo "$var1";  
// Ici, le premier $var1 va être affiché comme écrit alors que le deuxième entre  
guillemets va appliquer la variable et lire Hello DWWM.
```

On peut mettre un nombre entier (integer) en variable. Pour un float (nombre à virgule), il faut séparer les chiffres par un point :

```
$x = 147;  
$y = 147.8;
```

Pour les entiers, on peut aller de -2 000 000 000 à + 2 000 000 000. Pour les float, on peut aller bien plus loin.

2. Comment lire sa page Php

1- Lire sa page php

Quand on veut ouvrir le fichier avec l'extension .php, il faut passer par le dossier www du serveur, il ne s'ouvrira pas à partir de visual code, ou le php sera visible par l'utilisateur.

Ce qui distingue PHP des langages de script comme le Javascript, est que le code est exécuté sur le serveur, générant ainsi le HTML, qui sera ensuite envoyé au client. Le client ne reçoit que le résultat du script, sans aucun moyen d'avoir accès au code qui a produit ce résultat. Vous pouvez configurer votre serveur web afin qu'il analyse tous vos fichiers HTML comme des fichiers PHP. Ainsi, il n'y a aucun moyen de distinguer les pages qui sont produites dynamiquement des pages statiques.

Le grand avantage de PHP est qu'il est extrêmement simple pour les néophytes, mais offre des fonctionnalités avancées pour les experts. Ne craignez pas de lire la longue liste de fonctionnalités PHP. Vous pouvez vous plonger dans le code, et en quelques instants, écrire des scripts simples.

Pour pouvoir lire sa page en php, il faut travailler avec son serveur local (**localhost**)

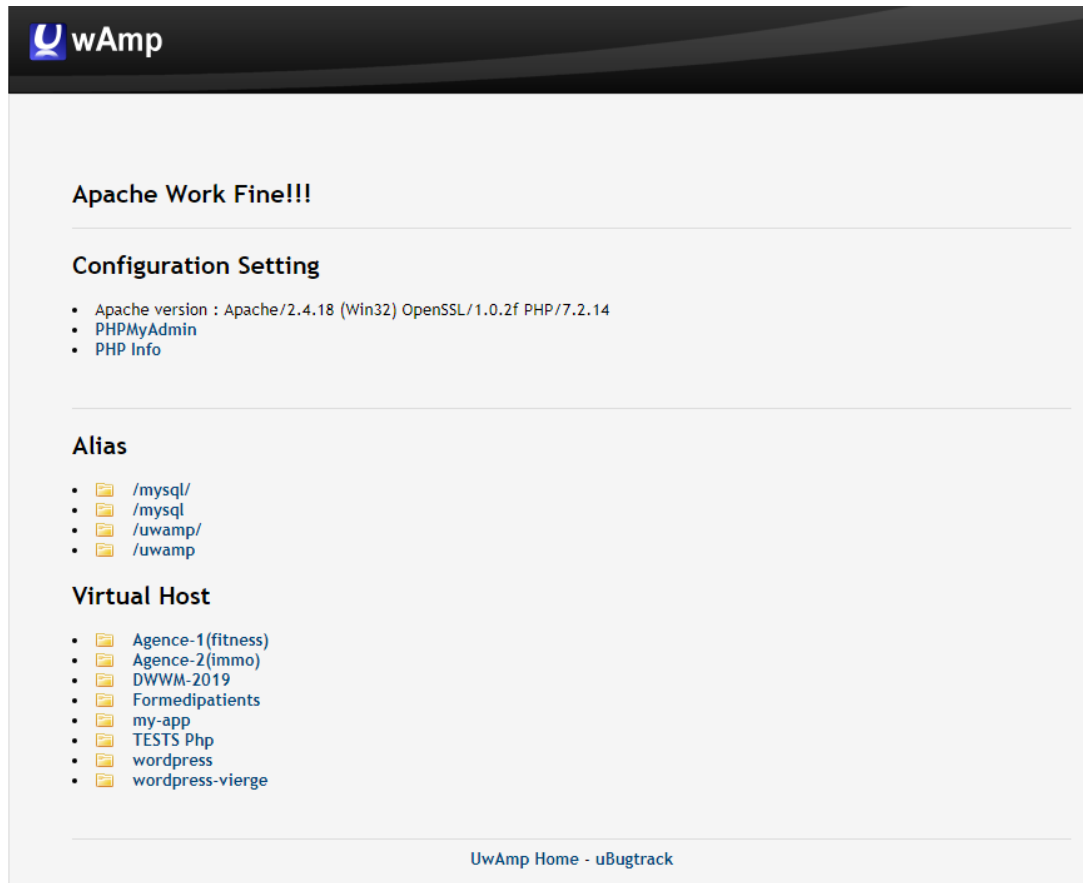
Donc :

- Démarrer son serveur local
- créer un dossier dans le dossier www de son serveur (ici Uwamp)
- créer sa première page nommée **index.php**

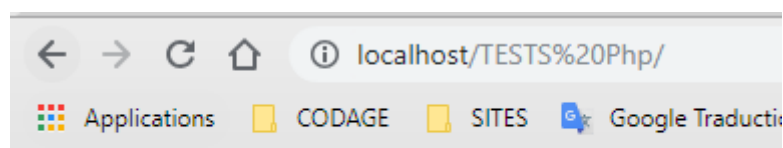
Ensuite retourner dans le serveur et cliquer sur l'onglet navigateur www

PHP

On arrive ensuite sur cette page dans laquelle on clique sur son fichier, ici TEST Php



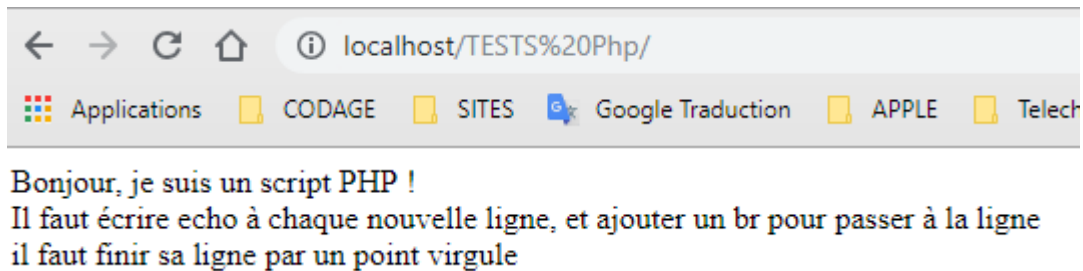
On obtient :



Bonjour, je suis un script PHP!

```
<?php
echo "Bonjour, je suis un script PHP ! <br>";
echo "Il faut écrire echo à chaque nouvelle ligne, et ajouter un br pour passer à la
ligne <br>";
echo "il faut finir sa ligne par un point virgule";
?>
```

PHP



Devant le texte, on peut mettre des « » ou des '' .

3. Les variables

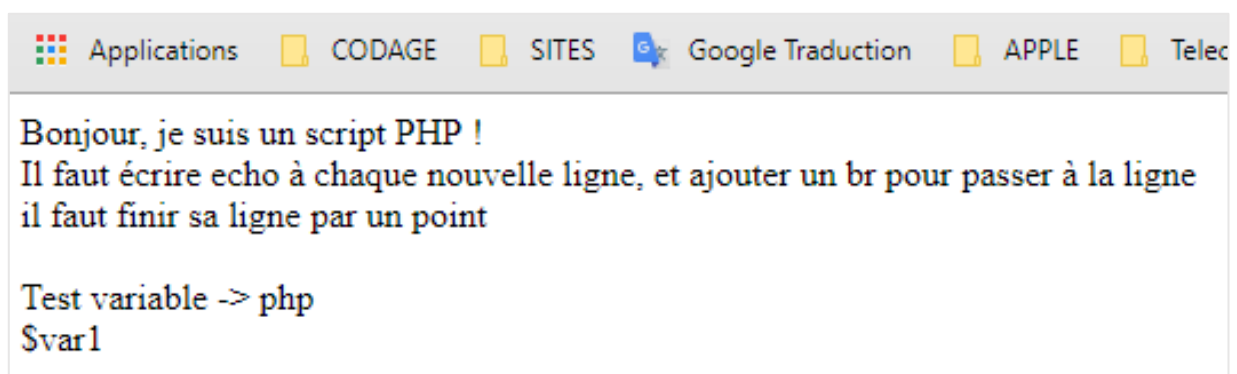
1- Introduction d'une variable

Une variable est introduite avec \$

```
$x=10;
```

Exemple :

```
$var="php";  
echo " Test variable -> ".$var; // le texte affiché est entre guillemets " ", la  
variable sera effectuée à la suite du texte  
echo "<br>"; // une balise ne s'affiche pas même si elle est entre " "  
echo '$var1'; // mettre ici des ' ' sinon il y a erreur  
echo "<br>";
```



PHP

2- Les variables booléennes (true or false)

Le booléen que l'on écrit dans JS marche de la même manière dans PHP.

```
// Ici, on vient dire que 5 est supérieur à 2.
$x = (5>2);
// On vient tester si 5 est supérieur à 2 alors affiche vrai, sinon faux. Comme 5 est
// effectivement supérieur à 2 alors il va afficher true.
if($x){
    echo "true";
}
else {
    echo "false";
}
```

3- Les incrémentations :

Comme sur JS, les incrémentations seront différentes selon l'ordre dans lequel on met les signes.

```
$x=10;
// Ici, le serveur va afficher 10 et ensuite lui ajouter 1.
echo $x++ . "<br>";
$t=10;
// Ici, il va d'abord ajouter 1 et ensuite afficher le résultat, ce qui donnera 11.
echo ++$t;
```

Pour les tests en if/else ou autre (les for, ...), il n'y a aucune différence avec javascript.

4. Les fonctions

Elles s'écrivent comme dans JS :

```
function bienvenue(){
    echo 'bienvenue chez nous!';
}
bienvenue();
```

```
function bienvenue($x){
    echo 'bienvenue '.$x.' !'.<br>';
}
bienvenue('Marc');
bienvenue('Marie');
```

PHP

1- Les variables externes dans les fonctions :

Ici, contrairement à JS, il y a une commande pour utiliser une variable extérieure à la fonction :

```
$e=["marc","sophie","marie","paul"];

function tableau($x){
    // Ici, le global va préciser que la variable e extérieure à la fonction va être
    // utilisée.
    // Si on ne le précise pas, il ne connaîtra pas la variable e.
    global $e;
    echo 'bienvenue '.$e[$x]. '<br>';
}

tableau(0);
tableau(1);
tableau(2);
tableau(3);
```

Il est donc possible d'utiliser un nom de variable propre à une fonction qui n'existera que dans cette fonction. Bien évidemment, il faut toujours avoir des noms différents pour éviter les erreurs.

Une variable appelée en global, si elle est modifiée dans une fonction, restera modifiée après être passée dans la fonction. Il faut prendre ça en considération pour éviter les erreurs.

2- Les modifications de variables :

Quand on appelle une variable, on ne fait qu'afficher son résultat ou s'en servir pour arriver à un résultat. La variable en elle-même ne change pas. Il est possible de changer l'adresse mémoire d'une variable et donc de modifier son résultat avec le signe & :

```
function foo(&$var) {
    $var++;
}
$a=5;
foo ($a);
// $a vaut 6 maintenant
```

3- Les paramètres par défaut :

On peut rentrer des variables qui ont un paramètre par défaut dans une fonction. Cependant,

Les paramètres par défaut doivent toujours être les derniers de la liste.

```
// Ici, l'âge par défaut est donc 11 ans.
function agg($nom,$age=11){
echo "nom: ".$nom." , âge =".$age." ans. <br>";
}
agg("marie");
agg("max");
agg("solange");
// Ici, on veut modifier ce qui va être affiché après âge = donc on vient le préciser.
agg("pierre",47);
```

5. Les tableaux :

Il existe deux types de tableaux : les tableaux indexés et les tableaux associatifs.

Les tableaux indexés sont pareil que sur JS. On va appeler tel élément grâce à [x].

Ici, au lieu de **.length** on utilisera le mot **count**.

Les tableaux associatifs sont différents. On va associer une valeur à un mot, afin que l'on puisse appeler la valeur en mettant le mot entre crochets.

```
// Créer un tableau:
$x= array();
// ou
$x=[]; // Pour le créer.

// LES TABLEAUX INDEXES:
// Ils se créent comme dans JS, il faut ajouter les valeurs dans un ordre précis et
appeler un numéro précis pour avoir la valeur correspondante. Comme partout, la
première valeur du tableau est 0.

// Pour ajouter une valeur à un tableau à la suite:
$x[count($x)]="toto";
// Ici, le count remplace le .length et vient directement compter le tableau et ajouter
à la suite.
// Pour ajouter à une place en particulier ou remplacer une valeur:
$x[2]="Jean";
// Ici, on cible la donnée que l'on veut remplacer.

// LES TABLEAUX ASSOCIATIFS:
// Ici, on va associer une valeur à un mot afin de ressortir la valeur en appelant le
mot: La flèche => va définir que telle mot renvoie à telle valeur.
$z=["nom" => "Maureau", "prenom" => "Maurice", "age" => 38];

// Si on veut sortir le prénom:
echo $z["prenom"];
```

PHP

Les tableaux associatifs renvoient à une personne en particulier. Il est possible de stocker plusieurs tableaux associatifs dans un tableau indexé par exemple.

1- Parcourir un tableau :

Tableau associatif :

```
$x=["nom"=>"marc","prenom"=>"pierre","age"=>21];

// ici, on vient dire que pour chaque valeur du tableau $x
// as $k(nom de valeur au pif qui fait référence au mot key) => $v(nom de valeur au
// pif qui fait référence au mot value)
// on renvoie le nom de la clé et la valeur qu'elle représente.
foreach ($x as $k=>$v){
    echo"key=".$k." valeur =".$v."<br>";
}
```

On utilisera le mot clef **foreach** pour aller chercher chaque key et la value à laquelle elle rapporte.

2- Parcourir un tableau de façon incursive avec une fonction qui s'appelle elle-même :

```
//essai de tableau

$arr=array(1,2,3,array(10,20,30,array(100,200,300)));
function affiche_arr($tableau){ // nomme la fonction affiche_arr
    foreach ($tableau as $valeur){ // parcourir le tableau avec la valeur
        if (is_array($valeur)){
            affiche_arr($valeur);
        }else{
            echo $valeur. '<br>';
        }
    }
}
affiche_arr($arr); // appel de la fonction arr elle-même
$tableau = $tab et $valeur=$val
```

```
1
2
3
10
20
30
100
200
300
```

A. Afficher un tableau lui-même contenu dans un tableau :

```
// Ici, on a un tableau. Dans ce tableau est stocké un autre tableau qui lui-même
// contient un autre tableau.
$x=array(1,2,3,array(10,20,30,array(100,200,300)));

// Pour afficher les données des tableaux internes:

// On va créer une fonction qui va lire les tableaux et les afficher: On met un
// paramètre pour ensuite l'utiliser avec un
// tableau en paramètre.
function returnArray($tab){
```

PHP

```
// Pour chaque élément ($v) contenu dans le tableau ($tab):
foreach($tab as $v){
    // Si l'élément v est un tableau:
    if(is_array($v)){
        // Ici, la fonction s'appelle elle-même pour dire: "affiche les éléments
        // contenus dans le tableau". On met $v en
        // paramètre pour en afficher les éléments si c'est un tableau.
        // Comme on est rentré dans le tableau interne (grâce au if), alors tout
        // ce qui est contenu dans le tableau
        // va s'afficher.
        returnArray($v);
    }
    // Si l'élément v n'est pas un tableau, alors on va l'afficher.
    else{
        echo $v."<br>";
    }
}
}
// Ici, on va appeler la fonction et l'utiliser sur le tableau $x. Les éléments du
// tableau qui sont eux-mêmes des tableaux
// vont s'afficher à la suite puisque la fonction est une boucle qui va traiter le
// cas où les valeurs sont un tableau.
returnArray($x);
```

La fonction sans les commentaires pour la réutiliser :

```
function returnArray($tab){
    if(is_array($v)){
        returnArray($v);
    }
    else{
        echo $v."<br>";
    }
}
}
returnArray($x);
```

3- Les implode, explode et str_split :

Il existe des variables pour agir sur les tableaux :

```
// IMplode:
// Implode permet de créer un string à partir d'un tableau et les séparer par le
// premier paramètre que l'on rentre:
```

PHP

```
$x=["marc","sophie","jean","louise","hector"];
// var_dump est l'équivalent php du console.log.
var_dump(implode(",",$x));
echo "<br>";

// EXPLODE:
// Explode permet de créer un array à partir d'une string. Le premier paramètre va
// permettre de dire quel caractère va permettre de séparer la string:

$str="un,deux,trois,quatre";
// Ici, il va chercher les virgules dans la string et séparer ce qu'il y a entre les
// virgules.
var_dump(explode(",",$str));
echo "<br>";

// STR_SPLIT:
// str_split va prendre une string et la diviser tous les x éléments, définis en
// second paramètre pour la transformer en array:

$str2="abcdefghijklmnopqrstuvwxy";
// ici, l'alphabet sera divisé toutes les 5 lettres et chaque portion équivaldra à
// une valeur du tableau.
var_dump(str_split($str2,5));
```

4- Les constantes :

Ce sont des valeurs que l'on utilisera dans tout le programme et que l'on ne pourra jamais modifier.

Ce sont les seules variables qui ne prennent pas de \$. Par convention, on les écrit en majuscules pour les différencier.

```
// Pour définir une constante:
define("SOCIETE","Microsoft");
// ici, à chaque fois qu'on appellera SOCIETE alors Microsoft ressortira. Il faut
// impérativement respecter les majuscules.

// Il est possible de créer une variable qui n'est pas case sensitive:
define("SOCIETE","Microsoft",true);
// avec le true, même si on écrit societe sans majuscules alors ça marchera.
```

5- Les boucles

Pour faire des loops :

Les boucles, tout comme les conditions, représentent l'une des fondations du PHP et il est donc essentiel de comprendre comment elles fonctionnent et de savoir les utiliser.

Les boucles vont nous permettre d'exécuter plusieurs fois un bloc de code tant qu'une condition donnée est vérifiée.

En effet, lorsque l'on code, on va souvent devoir exécuter plusieurs fois un même code. Utiliser une boucle nous permet de n'écrire le code qu'on doit exécuter plusieurs fois qu'une seule fois.

Nous disposons de quatre boucles différentes en PHP :

- La boucle **while** (« tant que ») ;
- La boucle **do... while** (« faire... tant que ») ;
- La boucle **for** (« pour ») ;
- La boucle **foreach** (« pour chaque ») ;

Une nouvelle fois, nous allons principalement utiliser les boucles avec les variables.

B. La boucle PHP while

La **boucle while** (« tant que » en français) est la boucle PHP la plus simple à appréhender.

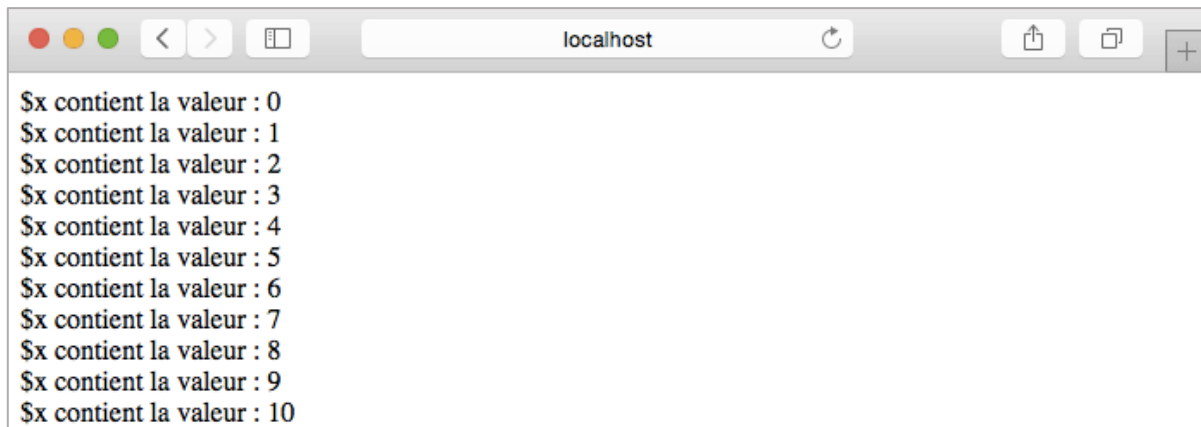
Cette boucle va nous permettre exécuter un certain bloc de code « tant qu'une » condition donnée est vérifiée.

Voyons à travers un exemple la syntaxe de cette boucle.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Démarrer avec PHP</title>
    <meta charset="utf-8">
  </head>
  <body>
    <?php
      $x = 0;

      while ($x <= 10){
        echo '$x contient la valeur : ' . $x . '<br>';
        $x++;
      }
    ?>
  </body>
</html>
```

PHP



```
$x contient la valeur : 0
$x contient la valeur : 1
$x contient la valeur : 2
$x contient la valeur : 3
$x contient la valeur : 4
$x contient la valeur : 5
$x contient la valeur : 6
$x contient la valeur : 7
$x contient la valeur : 8
$x contient la valeur : 9
$x contient la valeur : 10
```

Dans l'exemple ci-dessus, on commence par stocker la valeur 0 dans notre variable \$x. On dit également qu'on initialise notre variable.

Ensuite, on crée notre boucle **while**, qui va bien entendu réutiliser \$x.

Dans le cas présent, on demande à notre boucle d'afficher la phrase \$x contient la valeur : suivie de la valeur de \$x **tant que** la valeur stockée dans \$x ne dépasse pas 10.

On incrémente également notre variable à chaque nouveau passage dans la boucle, c'est-à-dire qu'on ajoute 1 à la dernière valeur connue de notre variable.

Cette étape d'incrémentation est indispensable. Sans celle-ci, \$x contiendrait toujours la valeur 0 (sa valeur de départ) et on ne pourrait jamais sortir de la boucle.

Voici ce que va faire PHP : il va rentrer dans la boucle et vérifier la valeur de \$x. Lors du premier passage dans la boucle, \$x contient 0. La condition est donc vérifiée. On affiche donc la phrase \$x contient la valeur 0 et on ajoute 1 à la valeur de \$x.

\$x contient donc maintenant 1. Comme 1 est toujours inférieur à 10, la condition de notre boucle est toujours vérifiée. On ne sort donc pas de la boucle et on effectue un deuxième passage.

Lors de ce deuxième passage, on affiche cette fois-ci la phrase \$x contient la valeur 1 puisque \$x contient désormais 1, puis on incrémente à nouveau \$x. \$x contient donc désormais la valeur 2.

Le PHP va continuer à effectuer des passages dans la boucle jusqu'à ce que la valeur contenue dans \$x soit strictement supérieure à 10, c'est-à-dire jusqu'à ce que la condition ne soit plus vérifiée. Il sort alors de la boucle.

Notez qu'il faut toujours bien penser à initialiser la variable utilisée dans la boucle en dehors de la boucle. En effet, ici, si j'avais initialisé \$x dans ma boucle, sa valeur aurait été réinitialisée à chaque nouveau passage de boucle !

Une nouvelle fois, pensez bien qu'il faudra toujours s'arranger pour qu'à un moment ou un autre la condition de la boucle ne soit plus vérifiée, afin de pouvoir en sortir. Dans le cas contraire, le PHP ne peut jamais sortir de la boucle et le script plante.

PHP

Evidemment, selon les cas, vous n'êtes pas obligés d'incrémenter la valeur de votre variable. Vous pouvez tout aussi bien décrémenter, multiplier par 2, etc.

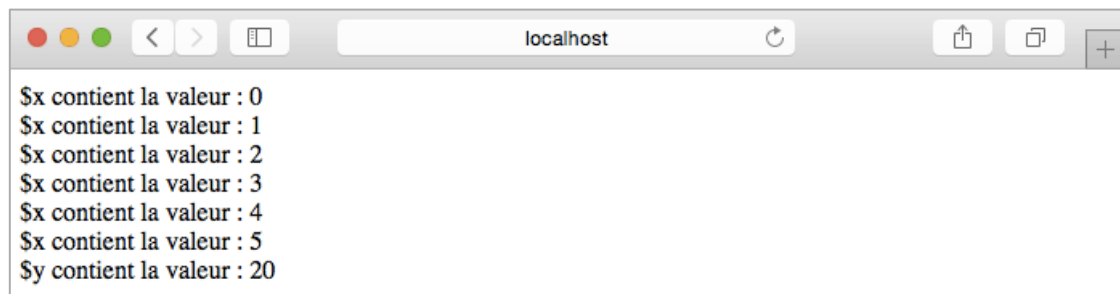
C. La boucle PHP do...while

La boucle PHP **do... while** (« faire... tant que » en français) ressemble à priori à la boucle while mais va fonctionner en sens inverse.

Cette boucle va également nous permettre d'exécuter un code tant qu'une condition donnée est vraie, mais cette fois-ci la condition ne va être vérifiée qu'après avoir exécuté le code en question.

Ainsi, même si une condition est fausse dès le départ, on effectuera toujours au moins un passage au sein d'une boucle **do...while**, ce qui n'est pas le cas avec une boucle while.

Pour obtenir ce résultat :



A screenshot of a web browser window with the address bar showing 'localhost'. The page content displays the output of a PHP script using a do...while loop. It shows seven lines of text: '\$x contient la valeur : 0' through '\$x contient la valeur : 5', followed by '\$y contient la valeur : 20'. The text is left-aligned and uses a monospaced font.

On a utilisé le code suivant :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Démarrer avec PHP</title>
    <meta charset="utf-8">
  </head>
  <body>
    <?php
      $x = 0;
      $y = 20;

      do{
        echo '$x contient la valeur : ' . $x. '<br>';
        $x++;
      }
      while($x <= 5);

      do{
        echo '$y contient la valeur : ' . $y. '<br>';
        $y++;
      }
      while($y <= 5);
    ?>
  </body>
</html>
```

Faites bien attention : le **while** est bien en dehors des accolades.

Dans l'exemple ci-dessus, nous avons créé deux boucles **do...while** afin de bien illustrer ce que j'ai dit précédemment.

Dans la première boucle, \$x contient la valeur 0 au départ. La phrase est affichée et la valeur de \$x est incrémentée puis la condition est vérifiée.

On sort ensuite de la boucle lorsque \$x contient 6. En effet, regardez bien notre boucle : on affiche \$x contient la valeur 5 lors du dernier passage puis on incrémente à nouveau cette valeur. Lorsqu'on teste \$x, la variable contient donc bien 6.

La deuxième boucle est strictement la même que la première à part qu'on utilise cette fois-ci une variable \$y.

Notre variable \$y contient la valeur 20 au départ et ainsi la condition de la boucle est fausse dès le départ. Cependant, dans le cadre d'une boucle **do...while**, la condition n'est vérifiée qu'en fin de boucle, c'est-à-dire après le passage dans la boucle.

Ainsi, même si la condition est fausse dès le départ, on effectue tout de même un premier passage dans la boucle (et on exécute donc le code à l'intérieur).

D. La boucle PHP for

La boucle PHP **for** (« pour » en français) est plus complexe à appréhender à priori que les boucles précédentes.

Cependant, cette boucle est très puissante et c'est celle qui sera majoritairement utilisée dans nos scripts PHP, faites donc l'effort de comprendre comment elle fonctionne.

Nous allons devoir préciser trois phases au sein d'une boucle **for** :

- Une phase d'initialisation ;
- Une phase de test ;
- Une phase d'incrément.

Dans la phase d'initialisation, nous allons tout simplement initialiser la variable que nous allons utiliser dans la boucle.

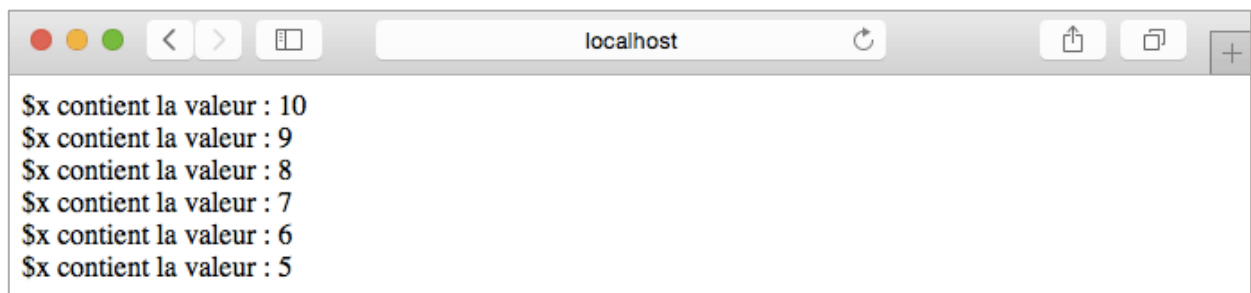
Dans la phase de test, nous allons préciser la condition qui doit être vérifiée pour rester dans la boucle.

Dans la phase d'incrément, nous allons pouvoir incrémenter ou décrémenter notre variable afin que notre condition soit fausse à un moment donné.

Voyons immédiatement à travers un exemple ce que cela signifie.

PHP

```
<!DOCTYPE html>
<html>
  <head>
    <title>Démarrer avec PHP</title>
    <meta charset="utf-8">
  </head>
  <body>
    <?php
      for($x = 10; $x >= 5; $x--){
        echo '$x contient la valeur : ' . $x . '<br>';
      }
    ?>
  </body>
</html>
```



Comme vous pouvez le voir, on initialise cette fois-ci directement notre variable à l'intérieur de notre boucle. Ensuite, nous sommes en territoire connu, seule la syntaxe change par rapport aux boucles vues précédemment.

Notez qu'on a cette fois-ci décrémenté notre variable, pour changer par rapport aux exemples précédents.

Notez également que vous devez séparer chaque phase par un point virgule.

E. [La boucle PHP foreach](#)

La boucle PHP **foreach** est un peu particulière puisqu'elle a été créée pour fonctionner avec des variables tableaux.

Nous étudierons donc le fonctionnement de cette boucle en même temps que le type de valeur PHP array (« tableau » en français).

F. [Quelle boucle utiliser dans quelle circonstance ?](#)

A ce stade, vous pourriez vous demander quand utiliser une boucle PHP plutôt qu'un autre.

Il va être très simple de déterminer quelle boucle PHP utiliser pour chaque situation.

Tout d'abord, si vous travaillez avec des tableaux, vous utiliserez une boucle PHP for ou **foreach** selon le type de **tableau** sur lequel vous travaillez.

PHP

Si vous savez à l'avance combien de passages vous allez effectuer dans la boucle, alors vous utiliserez la boucle PHP **for**. En effet, cette boucle a été **conçue pour un nombre de passage précis**.

Si vous ne savez pas à priori combien de passages vous allez effectuer dans votre boucle, mais que votre code a besoin d'effectuer toujours **au moins un passage dans la boucle**, vous utiliserez une boucle PHP **do...while**.

Finalement, si vous ne savez pas combien de passages vous allez effectuer dans votre boucle et que vous ne souhaitez **entrer dans la boucle que si la condition est vraie à un moment donné**, vous utiliserez une boucle PHP **while**.

6- La fonction return

Si appelée depuis une fonction, la commande **return** termine immédiatement la fonction, et retourne l'argument qui lui est passé.

Si appelée depuis l'environnement global, l'exécution du script est interrompue.

```
// Fonction
function nombre_aleatoire()
{
    $nombre_aleatoire = rand(1,100); // rand signifie « aléatoire »
    return $nombre_aleatoire;
}
// Tirage aléatoire de 10 nombres entre 1 et 100
echo 'Tirage aléatoire de 10 nombres entre 1 et 100 :';
for ($i = 1; $i < 11; $i++) echo ' '.nombre_aleatoire();
```

Tirage aléatoire de 10 nombres entre 1 et 100 : 1 64 40 97 61 79 11 19 51 9

Exemple de tirage du loto :

```
// Fonction
function nombre_aleatoire()
{
    $nombre_aleatoire = rand(1,49);
    return $nombre_aleatoire;
}
// Tirage aléatoire de 6 nombres entre 1 et 49 pour le tirage du loto
echo 'Tirage aléatoire de 6 nombres entre 1 et 49 :';
for ($i = 1; $i < 7; $i++) echo ' '.nombre_aleatoire();
```

Tirage aléatoire de 6 nombres entre 1 et 49 : 46 25 11 5 30 28

PHP

A. Le return :

Comme dans JS, on utilise return pour retourner un résultat après un test de fonction. Le return permet de sortir de la fonction. Il est donc inutile d'écrire quelque chose dans une fonction après un return, puisque s'il est appliqué alors on sort de la fonction et ce qui vient ensuite ne sera jamais lu par l'ordinateur.

```
// Ici, on vient dire que si le premier paramètre est plus grand que le deuxième alors
on retourne le premier,
// sinon le deuxième. Si le premier est plus grand alors le return fait directement
sortir de la fonction.
function maxab($a,$b){
    if($a>$b){
        return $a;
    }
    else{
        return $b;
    }
}
$c= maxab(2,7);
echo $c;
// Ici, le viewport va donc afficher 7.
```

B. Tester des tableaux associatifs stockés dans un tableau indexé :

```
function returnArray($tab)
{
    // Ici, pour chaque tableau construit de manière associative (donc construit de
    manière $k=>$v):
    foreach ($tab as $k => $v) {
        // Si $v est un tableau:
        if (is_array($v)) {
            // On utilise la fonction avec $v en paramètre pour afficher ce qu'il y a
            dans ce tableau:
            returnArray($v);
        } else {
            // Montre de quelle manière afficher les éléments qui ne sont pas des
            tableaux:
            echo $k . " " . $v;
        }
    }
}
returnArray($x);
```

Sans les commentaires :

```
function returnArray($tab)
{
    foreach ($tab as $k => $v) {
        if (is_array($v)) {
            returnArray($v);
        } else {
            echo $k . " " . $v;
        }
    }
}

returnArray($x);
```

6. La collecte de données de formulaires avec GET et POST

La variable `$_GET` est un tableau qui récupère les données d'un formulaire ou d'une url. Les données transiteront par l'URL, on pourra les récupérer grâce à l'array `$_GET`. Cette méthode est assez peu utilisée car on ne peut pas envoyer beaucoup d'informations dans l'URL (il est préférable de ne pas dépasser 256 caractères).

Avec la variable `$_POST` les données ne transiteront pas par l'URL, l'utilisateur ne les verra donc pas passer dans la barre d'adresse. Cette méthode permet d'envoyer autant de données que l'on veut, ce qui fait qu'on la privilégie le plus souvent.

La différence entre post et get :

Le get va transférer les infos via l'URL (donc les mdp également aussi, il faut faire attention). Alors que le post va envoyer les informations via un header caché.

Néanmoins, les données ne sont pas plus sécurisées qu'avec la méthode GET et il faudra toujours vérifier si tous les paramètres sont bien présents et valides. On ne doit pas plus faire confiance aux formulaires qu'aux URL.

C'est à vous de choisir par quelle méthode vous souhaitez que les données du formulaire soient envoyées. Si vous hésitez, sachez que dans 99 % des cas la méthode que l'on utilise est **post**, vous écrirez donc `method="post"`.

La variable `$_GET` (pour récupérer les infos dans le formulaire)

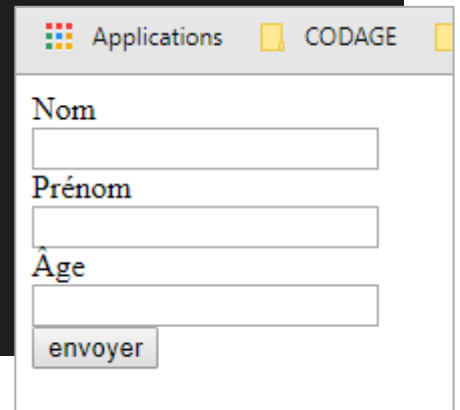
```
form action="Test_get.php" method="GET">
    <label>Nom </label><br><input type="text" name="nom"><br>
    <label>Prénom </label><br><input type="text" name="prenom"><br>
    <label>Age </label><br><input type="text" name="age"><br>
    <input type="submit" value="envoyer">
</form>
```

PHP

Dans la page php :

```
<?php
// Le $_GET est une variable super globale. Elle est toujours la même et générée via
PHP. Il faut donc l'écrire
// telle qu'elle.
$nom= $_GET['nom'];
$preénom= $_GET['prenom'];
$age= $_GET['age'];

echo 'Votre Nom est : <br><u>' . $nom . '</b></u>';
echo 'Votre Prénom est : <br><u>' . $prenom . '</b></u>';
echo 'Votre age est : <br><u>' . $age . '</b></u>';
?>
```



La variable **\$_POST**

```
<form action="Test_post.php" method="POST">
    <label>Nom </label><br><input type="text" name="nom"><br>
    <label>Prénom </label><br><input type="text" name="prenom"><br>
    <label>Age </label><br><input type="text" name="age"><br>
    <input type="submit" value="envoyer">
</form>
```

Dans la page php :

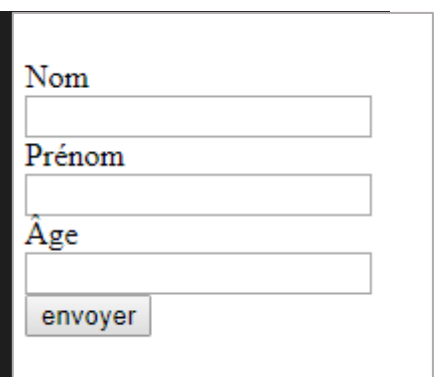
Si on veut reprendre les infos en post :

```
<!-- On met évidemment le method="post" en HTML. Le action = « » fait référence au
fichier PHP qui va créer les variables. -->
<form action="bienvenue.php" method="post">
<label >Nom</label> <input type="text" name="nom"> <br>
<label >Prenom</label> <input type="text" name="prenom"> <br>
<label >Age</label> <input type="text" name="age"> <br>
<input type="submit" value="envoyer"><br>
</form>
```

\$_POST est également une super globale.

```
<?php
// formulaire
$nom= $_POST['nom'];
$preénom= $_POST['prenom'];
$age= $_POST['age'];

echo 'Votre Nom est : <br><u>' . $nom . '</b></u>';
echo 'Votre Prénom est : <br><u>' . $prenom . '</b></u>';
echo 'Votre age est : <br><u>' . $age . '</b></u>';
?>
```



PHP

Un autre exemple avec un text-area ;

```
<form action="bienvenue.php" method="get">
<textarea name="text" id="" cols="30" rows="10"></textarea><br>
<input type="submit" value="envoyer"><br>
</form>
```

Le fichier de sortie « bienvenue.php » va afficher le texte entré dans le text area. On le verra également dans l'URL parce qu'on utilise la méthode GET.

```
$text=$_GET["text"];

echo $text;
```

A. Compter la longueur d'une string

```
function longueurChaine($a){
    // strlen permet de compter la longueur d'une string.
    echo strlen( $a);
}
$alphabet="abcdefghijklmnopqrstuvwxyz";
longueurChaine($alphabet);
```

B. Remplacer un caractère ou une chaîne de caractères dans une string

Il est possible de remplacer le contenu d'une string grâce à la fonction str_replace :

```
$alphabet="abcdefghijklmnopqrstuvwxyz";
function replaceCaractere($a){
    // 1/ Le premier paramètre va définir ce que l'on veut remplacer.
    // 2/ Le second ce par quoi on veut le remplacer.
    // 3/ Enfin le dernier va servir à définir dans quelle string ou array on veut le
    remplacer.
    echo str_replace("e","_",$a);
}

replaceCaractere($alphabet);
```

C. Les strings en majuscule ou minuscule

```
function listVal($tab)
{
    foreach ($tab as $k => $v) {
        // strtoupper va afficher le paramètre en majuscules. strtolower les affichera
        en minuscules.
        echo strtoupper($k) . " ". strtolower($v) . "(" . strlen($v) . ") <br>";
    }
}
listVal($tableau);
```

PHP

D. Obtenir l'adresse ip

Pour obtenir une adresse IP, il faut utiliser la super globale \$_SERVER :

```
echo $_SERVER['REMOTE_ADDR'] ;
```

E. Différents scripts

1) Ouvrir le fichier source et afficher le nombre de lignes contenues dans le fichier

```
<?php
/////////exo 06
$f = basename($_SERVER['PHP_SELF']);
echo 'il y a ' . count(file($f)) . ' lignes dans ' . $f;
echo '<hr>';
?>
```

2) Afficher une table de multiplication (par exemple la table de 7)

```
<?php
/////////exo 10
for ($i=1; $i <= 7; $i++) {
    for ($j=1; $j <= 7; $j++) {
        //echo ($i*$j) . '&nbsp;';
        echo sprintf('%1$02d ', ($i*$j))."&nbsp;";
    }
    echo '<br>';
}
echo '<hr>';
?>
```

10 – Ecrire un script qui affiche la table de multiplication de

1	2	3	4	5	6	7
2	4	6	8	10	12	14
3	6	9	12	15	18	21
4	8	12	16	20	24	28
5	10	15	20	25	30	35
6	12	18	24	30	36	42
7	14	21	28	35	42	49

3) Script qui part d'un tableau \$b=array('a','b','c','d'), et qui retourne un tableau sous la forme ('aa','bb','cc','dd')

```
$b=array('a','b','c','d');
$mycount=count($b);
for ($i=0; $i < $mycount; $i++) {
    $b[$i] .= $b[$i];
}
var_dump($b);
echo '<hr>';
```

Ecrire un formulaire qui demande 3 valeurs : a,b,c.

En cliquant sur Envoyer la page de destination donne les solutions de l'équation : $ax^2+bx+c=0$

PHP

```
////////// exo 12
$a=$_GET['a'];
$b=$_GET['b'];
$c=$_GET['c'];

$d = $b*$b -4 * $a * $c;
if ($a == 0 ) {
    die("La valeur de 'a' ne doit pas etre nulle .... BYE");
}
if ($d <0) {
    echo "Pas de solutions dans le monde réel !!";
} elseif ($d == 0) {
    echo "On a une solution double : ". -$b/(2*$a);
} else {
    echo "on a deux solutions :<br>";
    echo "x1 = ".((- $b - sqrt($d))/(2*$a)).'<br>';
    echo "x2 = ".((- $b + sqrt($d))/(2*$a)).'<br>';
}
```

4) fonction qui prend un nombre n et qui retourne un triangle d'Etoiles :

```
////////// exo 13
$n = 5;

if (isset($_GET['n'])) {
    $n = $_GET['n'];
}
for ($i = 1; $i <= $n; $i++) {
    for ($j = 1; $j <= $i; $j++) {
        echo "&nbsp;";
    }
    echo '<br>';
}
echo '<hr>';
```

```
n= 2  *
      *  *

N= 5  *
      *  *
      *  *  *
      *  *  *  *
      *  *  *  *  *
```

14 – Ecrire un script qui affiche le schéma suivant :

```
  *      *      *      *      *      *      *      *
 *      *      *              *              *
 *      *      *      *      *      *      *
 *      *              *      *
 *      *      *      *      *
```

```
<title>e14</title>
</head>
<body>
<?php

function clean_phrase($str)
{
    $tmp = $str;
    $tmp = str_replace("é", "e", $tmp);
    $tmp = str_replace("è", "e", $tmp);
    $tmp = str_replace("à", "a", $tmp);
    $tmp = strtoupper($tmp);

    $a_tmp = str_split($tmp, 1);
    $tmplen = strlen($str);
    for ($i = 0; $i < $tmplen; $i++) {
        $x = ord($tmp[$i]);
        if (($x >= 65 && $x <= 90) || ($x == 32)) {
            ; //rien
        } else {
            $tmp[$i] = ' ';
        }
    }
    return $tmp;
}

////////// exo 14 bis

$allchar = array(
    array(bindec("11111"), bindec("10001"), bindec("11111"), bindec("10001"),
    bindec("10001")), //A
    array(bindec("11110"), bindec("10001"), bindec("11110"), bindec("10001"),
    bindec("11110")), //B
    array(bindec("01110"), bindec("10001"), bindec("10000"), bindec("10001"),
    bindec("01110")), //C
    array(bindec("11110"), bindec("10001"), bindec("10001"), bindec("10001"),
    bindec("11110")), //D
    array(bindec("11111"), bindec("10000"), bindec("11111"), bindec("10000"),
    bindec("11111")), //E
    array(bindec("11111"), bindec("10000"), bindec("11110"), bindec("10000"),
    bindec("10000")), //F
    array(bindec("01110"), bindec("10000"), bindec("10111"), bindec("10001"),
    bindec("01110")), //G
    array(bindec("10001"), bindec("10001"), bindec("11111"), bindec("10001"),
    bindec("10001")), //H
```

PHP

```
    array(bindec("11111"), bindec("00100"), bindec("00100"), bindec("00100"),
bindec("11111")), //I
    array(bindec("11111"), bindec("00100"), bindec("00100"), bindec("10100"),
bindec("01100")), //J
    array(bindec("10001"), bindec("10010"), bindec("11100"), bindec("10010"),
bindec("10001")), //K
    array(bindec("10000"), bindec("10000"), bindec("10000"), bindec("10000"),
bindec("11111")), //L
    array(bindec("10001"), bindec("11011"), bindec("10101"), bindec("10001"),
bindec("10001")), //M
    array(bindec("10001"), bindec("11001"), bindec("10101"), bindec("10011"),
bindec("10001")), //N
    array(bindec("01110"), bindec("10001"), bindec("10001"), bindec("10001"),
bindec("01110")), //O
    array(bindec("11110"), bindec("10001"), bindec("11110"), bindec("10000"),
bindec("10000")), //P
    array(bindec("01110"), bindec("10001"), bindec("10001"), bindec("10101"),
bindec("01110")), //Q
    array(bindec("11110"), bindec("10001"), bindec("11110"), bindec("10010"),
bindec("10001")), //R
    array(bindec("01111"), bindec("10000"), bindec("01110"), bindec("00001"),
bindec("11110")), //S
    array(bindec("11111"), bindec("00100"), bindec("00100"), bindec("00100"),
bindec("00100")), //T
    array(bindec("10001"), bindec("10001"), bindec("10001"), bindec("10001"),
bindec("01110")), //U
    array(bindec("10001"), bindec("10001"), bindec("01010"), bindec("01010"),
bindec("00100")), //V
    array(bindec("10001"), bindec("10001"), bindec("10101"), bindec("10101"),
bindec("01010")), //W
    array(bindec("10001"), bindec("01010"), bindec("00100"), bindec("01010"),
bindec("10001")), //X
    array(bindec("10001"), bindec("10001"), bindec("01010"), bindec("00100"),
bindec("00100")), //Y
    array(bindec("11111"), bindec("00010"), bindec("00100"), bindec("01000"),
bindec("11111")), //Z
    array("0", "0", "0", "0", "0"), //aspace
);

////////////////////////////////////

if (isset($_GET['phrase']) and strlen($_GET['phrase']) > 0) {
    $phraseT = clean_phrase($_GET['phrase']);
    $a_tableau = str_split($phraseT, 1);
    $phrase_len = count($a_tableau);
} else {
```

PHP

```
        die("merci de saisir une phrase");
    }

    for ($i = 0; $i < 5; $i++) { // on imprime ligne par ligne

        for ($k = 0; $k < $phrase_len; $k++) { // on boucle sur chaque lettre
            $c = $a_tableau[$k];
            $index = ord($c);
            if ($index == 32){
                $index = 26;
            }else{
                $index = $index -65;
            }

            $a = $allchar[$index];
            for ($j = 4; $j >= 0; $j--) { // on imprime le dessin de la ieme ligne d'une
lettres
                if (($a[$i] & pow(2, $j)) == pow(2, $j)) {
                    echo "*";
                } else {
                    echo "&nbsp;";
                }
                echo '&nbsp;';
            }
            echo '&nbsp;&nbsp;&nbsp;&nbsp;';

        }
        echo '<br>';
    }
?>
```

Ou de la façon suivante :

```
<title>e14</title>
</head>
<body>
<?php
////////// exo 14
$a = array(
    bindec("101111111"),
    bindec("101100101"),
    bindec("101111111"),
    bindec("101001101"),
    bindec("111111101"),
);
```

PHP

```
// $a=array(383,357,383,333,509);
for ($i = 0; $i < 5; $i++) {
    for ($j = 8; $j >= 0; $j--) {
        if (($a[$i] & pow(2, $j)) == pow(2, $j)) {
            echo "*";
        } else {
            echo "&nbsp;";
        }
        if ($j % 3 == 0) {
            echo '&nbsp;';
        }
    }
    echo '<br>';
}
echo '<hr>';

////////////////////////
?>
```

- 5) [Prendre une valeur N, en url et qui dessine un damier de taille 2N avec un tableau HTML et du CSS](#)

```
<style>

    html{

        font-family: 'Courier New', Courier, monospace;

    }

    .B{

        background-color:black;

        width: 20px;

        height:20px;

    }

    .W{

        background-color:white;

        width: 20px;

        height:20px;

    }

</style>
```

```
<title>Document</title>
</head>
<body>
<?php
////////// exo 15

$n = 4;
if (isset($_GET['n'])) {
    $n = $_GET['n'];
}

echo '<table border="1">';
for ($i = 0; $i < 2 * $n; $i++) {
    echo '<tr>';
    for ($j = 0; $j < 2 * $n; $j++) {
        if (($i + $j) % 2 == 0) {
            echo '<td class="B"></td>';
        } else {
            echo '<td class="W"></td>';
        }
    }
    echo '</tr>';
}
echo '</table>';
//////////
?>
</body>
```

F. Obtenir le navigateur

Il faut également utiliser la super globale \$_SERVER :

```
// Permet de voir le navigateur utilisé par la personne.
```

PHP

```
$nav = $_SERVER['HTTP_USER_AGENT'];  
echo $nav."<br>";
```

G. Pour voir le nom du fichier courant

```
// Permet de voir le nom du fichier courant et son dossier parent.  
$path = $_SERVER['PHP_SELF'];  
// Cette commande permet de n'afficher que le nom du fichier courant.  
$file = basename ($path);  
echo $path;
```

H. Créer un nouveau fichier via PHP

```
// Cette variable renvoie au nom de fichier et le type (ici .txt):  
$file = 'php_ex2.txt';  
// Cette variable définit le contenu à ajouter.  
$content = "Vive le PHP";  
// Ecrit le contenu dans le fichier, en utilisant le drapeau  
// FILE_APPEND pour rajouter à la suite du fichier et  
// LOCK_EX pour empêcher quiconque d'autre d'écrire dans le fichier  
// en même temps  
file_put_contents($file, $content, FILE_APPEND | LOCK_EX);
```

file_put_contents permet de créer n'importe quel type de fichier de manière concise et efficace. Cette fonction est l'ensemble de plusieurs fonctions qui permettent à elles toutes de faire la même chose.

I. Compter les lignes d'une page

```
// la variable homepage cible la page du fichier courant:  
$homepage = basename($_SERVER['PHP_SELF']);  
// Cette variable permet de compter le nombre de lignes de la page.  
$noOfLines = count(file($homepage));  
echo "Il y a <b>".$noOfLines."</b> lignes dans la page ".$homepage;
```

J. Enlever les doublons d'un tableau

```
$tabex7 = [1, 1, 1, 2, 3, 5, 5, 6, 7, 9];  
// Pour enlever les doublons d'un array, on utilise array_unique. Cette commande ne  
// fonctionne pas sur  
// Les tableaux multi-directionnels.  
$tabSansDoublon=array_unique($tabex7);  
print_r($tabSansDoublon);
```


K. Opérer une redirection automatique

```
// Le header(location:lien) permet de rediriger automatiquement vers le lien ciblé.  
Au lancement du fichier, on sera  
// directement sur la page du lien.  
header("location:http://www.jeuxvideo.com/forums/0-51-0-1-0-1-0-blabla-18-25-  
ans.htm");
```

7. La fonction isset

Pour éviter les erreurs en cas de formulaire non rempli où la valeur est nécessaire pour qu'un site fonctionne par exemple on peut mettre une valeur par défaut.

On utilise la fonction **isset** pour définir que si la valeur a été rentrée, alors on va l'utiliser, sinon on aura une valeur par défaut.

```
// On dit que si rien n'a été ajouté dans l'URL (via un form par exemple) alors la  
valeur n sera de 5  
// par défaut.  
$n=5;  
// En revanche, si l'URL propose une valeur pour n, dans ce cas on utilisera cette  
valeur.  
if(isset($_GET['n'])){  
    $n=$_GET['n'];  
}
```

Le **isset** marche aussi en POST. Il marche dès lors qu'il faut utiliser une valeur par défaut si l'utilisateur n'en a pas ajoutée.

8. Transformer du binaire en décimales

Le binaire est uniquement composé de valeurs en 0 et 1 (et est compté en interval 2).

Pour transformer le binaire en décimales, il existe une fonction appelée **bindec** :

```
$a = array(  
    // Ce que l'on met à l'intérieur d'un bindec est une suite binaire que l'on veut  
transformer en décimales.  
    bindec("10111111"),  
    bindec("101100101"),  
    bindec("10111111"),  
    bindec("101001101"),  
    bindec("111111101"),  
);  
// Chaque bindec du tableau donne les nombres suivants:
```

```
// $a=array(383,357,383,333,509);

for ($i = 0; $i < 5; $i++) {
    for ($j = 8; $j >= 0; $j--) {
        // La fonction pow() sert à définir la puissance. Le premier chiffre équivaut
        // au chiffre, le deuxième
        // a la puissance que l'on veut lui attribuer. Par exemple, ici ça sera 2^$j.
        if (($a[$i] & pow(2, $j)) == pow(2, $j)) {
            echo "*";
        } else {
            echo "&nbsp;";
        }
        if ($j % 3 == 0) {
            echo '&nbsp;';
        }
    }
    echo '<br>';
}
echo '<hr>';
```

9. Les classes

```
// Par convenance, on met TOUJOURS la première lettre de la classe en majuscule.
class Personne{
    // On déclare les propriétés de la classe
    public $nom;
    public $prenom;

    // On définit les méthodes qui vont utiliser les propriétés de la classe. Une méthode
    // est toujours une fonction.
    function affiche(){
        // Le $this est comme dans JS. Il servira à appeler une valeur propre à une
        // variable.
        echo 'je suis '.$this->$prenom.' '.$this->$nom.'<br>';
    }
}

// Pour ajouter un objet de type Personne (la classe)
$marDel= new Personne();
// On utilise la flèche pour dire telle variable aura pour valeur prenom Marc. Comme
// la variable est au début,
// Il ne faut pas la remettre a la propriété.
$marDel->prenom='Marc';
```

```
$marDel->nom='Delpot';  
// On dit que pour telle variable on applique la méthode de la classe Personne (d'où  
la flèche).  
$marDel->affiche();
```

10. Les get et set dans les classes

Les getters et setters sont basés sous le même principe que ceux utilisés en JavaScript :

```
class Person2  
{  
    public $nom;  
    public $prenom;  
    function setNom($str){  
        // La variable précisera le prénom à entrer  
        $this->nom=$str;  
    }  
    function getNom(){  
        // Cette fonction viendra spécifier la valeur de la variable x.  
        return $this->nom;  
    }  
    function setPrenom($str){  
        $this->prenom=$str;  
    }  
    function getPrenom(){  
        return $this->prenom;  
    }  
    function affiche(){  
        echo 'Je suis ' . $this->prenom . ' ' . $this->nom . '<br>';  
    }  
}  
  
$marcDel = new Person2();  
// On précise ici ce que l'on veut ajouter comme objet avec les propriétés de la  
classe Person2  
$marcDel->setNom('Delpot');  
$marcDel->setPrenom('marc');  
$marcDel->affiche();
```

1- Les constructeurs

En PHP, il est possible de créer une fonction qui vient directement créer des objets avec les propriétés et les méthodes souhaitées :

PHP

Cette fonction se lance automatiquement dès que l'on tape le mot « new ». Il faut mettre deux underscores construct (`__construct`) avant le nom de notre fonction pour en créer un.

Cette fonction reste une fonction comme une autre et peut tout faire comme une fonction classique.

On peut par exemple créer un constructeur qui vérifie dès que l'on ajoute des données si le lien entre nous et la base de données en question est possible.

Bien que le nom soit fixe, il est possible de créer plusieurs `__construct`. Ce qui va les différencier est le nombre de paramètres qu'ils auront.

```
class Person3
{
    public $nom;
    public $prenom;

    // Cette fonction se lancera dès que l'on utilisera le mot new.
    function __construct($prenom,$nom){
        echo 'je suis entrain de naitre...';
        $this->prenom = $prenom;
        $this->nom=$nom;
    }
    function setNom($str){
        $this->nom=$str;
    }
    function getNom(){
        return $this->nom;
    }
    function setPrenom($str){
        $this->prenom=$str;
    }
    function getPrenom(){
        return $this->prenom;
    }
    function affiche(){
        echo 'Je suis '.$this->prenom.' '.$this->nom.'<br>';
    }
}

// Ici la fonction se lance et il suffit de mettre les paramètres (dans l'ordre de la
fonction évidemment)
$alexD = new Person3('alex','Dupont');
$alexD->affiche();
$alexD->setNom('Delpot');
$alexD->setPrenom('marc');
```

2- L'héritage

On peut « étendre » une classe déjà existante en en créant une nouvelle qui va utiliser les données de la classe existante en ajoutant des nouvelles propriétés grâce au mot extends.

Plusieurs classes peuvent hériter d'une classe. Par contre, une classe enfant peut avoir un seul parent.

Il faut être vigilant : C'est toujours la dernière valeur qui sera utilisée. Par exemple, si un parent a une méthode qui affiche en string classique alors que la fille héritière a une fonction qui affiche sous forme de tableau, alors la string sera écrasée et c'est l'affichage en tableau qui sera effectué.

```
// Le require sert à appeler un autre fichier PHP. Person3 est stockée sur un autre
fichier.
require 'person3.php';
// Cette fonction signifie que la nouvelle classe Person5 vient compléter Person3 avec
de nouveaux paramètres
// et de nouvelles méthodes.
class Person5 extends Person3
{
    public $age;

    function __construct($p,$n,$a){
        // parent::__construct() est une super globale. Elle permet de faire appel à une fonction
        // de la classe parent.
        // La fonction __construct va demander à la fonction __construct parent (qui
        // a deux paramètres) de gérer
        // les deux paramètres qui la concernent et par la suite elle viendra ajouter
        // l'âge.
        parent::__construct($p,$n);
        $this->age=$a;
    }
    function setAge($a){
        $this->age=$a;
    }
    function getAge(){
        return $this->age;
    }

    function affiche(){
        // Ici, on demande d'utiliser la fonction affiche du parent qui fera la phrase
        // avec le nom et le prénom
        // et on vient y ajouter la fin de la phrase avec l'âge.
        parent::affiche();
        echo 'et j\'ai '.$this->age.'<br>';
    }
}
```

```
}  
  
$alexD = new Person5('alex','Dupont',25);  
$alexD->affiche();  
echo 'je vais changer d\'identité <br>';  
$alexD->setNom('Delpot');  
$alexD->setPrenom('marc');  
$alexD->setAge(28);  
$alexD->affiche();
```

3- Les différents types de propriétés

A. Les propriétés publiques :

Elles sont accessibles par les classes enfants et tout le reste du programme.

```
public $nom;
```

B. Les propriétés privées (private)

Elles ne sont accessibles qu'à la classe en question. Les enfants ne peuvent y accéder que si une fonction a été créée dans la classe parent pour les modifier.

```
private $prenom;
```

C. Les propriétés protégées (protected)

C'est un mélange des deux. Elles sont accessibles aux classes héritières mais sont protégées par un programme externe.

```
protected $prenom;
```

11. La fonction func_get_arg

Il est possible d'avoir une fonction **construct** a trois arguments mais ne pas avoir les données pour tous les remplir.

Il existe une fonction qui permet de définir que si tel argument n'est pas ici, alors on donnera un paramètre par défaut :

```
class Maison  
{  
    private $rue;  
    private $num;  
    private $ville;
```

```
public function __construct()
{
    // Cette fonction permet de savoir combien de paramètres ont été passés dans
    la fonction.
    $nbreargs = func_num_args();

    // Le switch permet de gérer les cas qui nous intéressent
    switch ($nbreargs) {
        // On met case x: pour chaque cas que l'on veut traiter.
        case 1:
            // La fonction func_get_args est une fonction intégrée à PHP qui
            permet de cibler le paramètre d'une fonction.
            // Les paramètres se comptent comme un tableau, le premier étant 0.
            $this->mettreLesChamps(func_get_arg(0), '10', 'LIEVIN');
            break;
        case 2:
            $this->mettreLesChamps(func_get_arg(0), func_get_arg(1), 'LIEVIN');
            break;
        case 3:
            $this->mettreLesChamps(func_get_arg(0), func_get_arg(1),
            func_get_arg(2));
            break;
        default:
            // On met ce qui se passe si aucun des cas que l'on a prévu n'est atteint.
            echo 'Erreur lors de l\'instanciation de l\'objet (' . $nbreargs . '
            arguments): ' . $nbreargs;
            $this->mettreLesChamps('INCONNU', -999, 'SANS PAYS');
            //die();
            break;
    }
}

public function mettreLesChamps($rue, $num, $ville)
{
    $this->rue = $rue;
    $this->num = $num;
    $this->ville = $ville;
}

public function afficher()
{
    echo 'Rue : ' . $this->rue . ', Numéro = ' . $this->num . ' et Ville : ' .
    $this->ville . '<br>';
}
}
```

PHP

```
// On test en entrant un seul paramètre:
$x1 = new Maison('Pierre Legrand');
$x1->afficher();
// Avec deux:
$x2 = new Maison('Pierre Legrand', 25);
$x2->afficher();
// Avec 3:
$x3 = new Maison('Pierre Legrand', 50, 'TOULOUSE');
$x3->afficher();
// Avec 4 (le message d'erreur en default sera affiché)
$x4 = new Maison('Pierre Legrand', 50, 'TOULOUSE', "32 498");
$x4->afficher();
```