

# LES BASES DE PHP

## 4 – Test et boucle

### Table des matières

Les instructions conditionnelles.....	2
Qu'est ce qu'une condition ?.....	3
L'instruction if.....	3
L'instruction if...else.....	4
L'instruction if...elseif ...else.....	5
L'instruction switch.....	5
Les boucles.....	8
While.....	8
do...while.....	9
L'instruction for.....	11
L'instruction foreach.....	13

## Les instructions conditionnelles

Dans la plupart des cas, dès l'instant où vous aurez à coder un algorithme, vous aurez besoin de réaliser des tests. Par exemple, si telle variable vaut « vrai » alors passe à la suite, sinon stop tout. Ce mécanisme est à la base des instructions dites conditionnelles. Ces instructions permettent de définir précisément ce que votre programme devra effectuer dans un cas de figure et ce qui se passera dans un autre cas.

PHP gère plusieurs types d'instruction conditionnelles :

- 1) **if(condition)** : Le code entre crochet sera exécuté uniquement si le résultat du test de la condition est vrai.

```
if (condition) {  
    code a exécuter si la condition est vrai;  
}
```

- 2) **if(condition)...else**: Si le test de la condition retourne vrai une portion de code sera exécuté, mais cette fois si le test retourne « faux » alors une autre portion de code sera exécutée.

```
if (condition) {  
    code a exécuter si la condition est vrai;  
} else {  
    code a exécuter si la condition est fausse;  
}
```

- 3) **if(condition)...elseif(condition)...else**: Une première condition sera testée, si elle est retournée vrai, la portion du code concerné sera exécutée, sinon, la seconde condition sera testée, si celle-ci retourne vrai elle sera exécutée, et si aucune des deux conditions est vraie alors la portion du code « else » sera exécutée

```
if (condition1) {  
    code a exécuter si la condition1 est vrai;  
}  
elseif (condition2) {  
    code a exécuter si la condition est vrai;  
} else {  
    code a exécuter si aucune des 2 conditions est vrai;  
}
```

- 4) **switch** : sélectionne un bloc de code à exécuter

## Qu'est ce qu'une condition ?

Une condition permet d'évaluer si d'une condition booléenne, à savoir si c'est vrai ou faux est vérifiée ou non. En autre terme on cherche a savoir si une condition est vraie ou fausse.

Par exemple, « Est qu'il porte des basket ? » Oui ... ben y rentre pas !

Plus sérieusement, pour évaluer une condition on va devoir utiliser des opérateurs de comparaison. Par exemple on peut vérifier si une variable est vide. La réponse ne peut être que oui ou non. Notez qu'un opérateur de comparaison PHP est toujours utilisé pour comparer 2 valeurs.

Opérateur	Nom	Exemple	Résultat
==	égale	<code>\$x == \$y</code>	Renvoie <code>true</code> si <code>\$x</code> est égale à <code>\$y</code>
===	Identique	<code>\$x === \$y</code>	Renvoie <code>true</code> si <code>\$x</code> est égale à <code>\$y</code> et si ils sont de même type
!=	différent	<code>\$x != \$y</code>	Renvoie <code>true</code> si <code>\$x</code> n'est pas égale à <code>\$y</code>
<>	Non égale	<code>\$x &lt;&gt; \$y</code>	Renvoie <code>true</code> si <code>\$x</code> n'est pas égale à <code>\$y</code>
!==	Non identique	<code>\$x !== \$y</code>	Renvoie <code>true</code> si <code>\$x</code> n'est pas égale <code>\$y</code> , ou s'ils ne sont pas du même type
>	Plus grand que	<code>\$x &gt; \$y</code>	Renvoie <code>true</code> si <code>\$x</code> est plus grand que <code>\$y</code>
<	Plus petit que	<code>\$x &lt; \$y</code>	Renvoie <code>true</code> si <code>\$x</code> est plus petit que <code>\$y</code>
>=	Plus grand ou égale	<code>\$x &gt;= \$y</code>	Renvoie <code>true</code> si <code>\$x</code> est plus grand ou égale à <code>\$y</code>
<=	Plus petit ou égale	<code>\$x &lt;= \$y</code>	Renvoie <code>true</code> si <code>\$x</code> est plus petit ou égale à <code>\$y</code>

Pour résumer je peux comparer :

- Si `a` est supérieur à `b`, sera noté `$a > $b`

### ATTENTION

Soyez conscient que la comparaison de données de types différents n'est pas toujours évidente. En particulier si vous souhaitez comparer un entier à un booléen ou un entier à une chaîne de caractères. Il est recommandé d'utiliser les opérateurs de comparaison `===` et `!==` au lieu de `==` et `!=` dans la plupart de ces cas.

## L'instruction if

L'instruction `if` est l'une des plus importante en programmation. Elle permet l'exécution d'un fragment de code en fonction du résultat d'un test. Par exemple testez ceci :

```
<?php
$a = 10 ;
$b = 2 ;
if ($a > $b){
    echo "a est plus grand que b";
}
```

```
}  
?>
```

Notez que si vous modifiez la valeur de \$a pour qu'elle soit plus inférieure à \$b, votre condition \$a>\$b sera fausse. Vu que nous avons prévu aucun comportement pour ce cas de figure, la portion du code concerné a été ignorée, vous obtenez une page vide.

Autre exemple :

La fonction date() permet de récupérer l'heure. Si l'on mentionne entre guillemets H, PHP affichera uniquement les heures. Du coup il est possible d'afficher un message si la journée n'est pas fini.

Faite le test.

```
<?php  
$t = date("H");  
if ($t < "18") {  
    echo "Passez une bonne journée!";  
}  
?>
```

Même chose, modifiez ce code pour que votre condition soit fausse ; vous obtenez une page vide. La portion de code entre {...} qui délimite les instructions à exécuter dans le cas où votre condition est vraie a été ignorée.

## L'instruction if...else

Imaginons maintenant que nous souhaitons afficher tout de même quelque chose même si la condition n'est pas vérifiée (c-à-d vrai). Pour cela il nous suffit d'ajouter l'instruction else. Elle sera exécutée au cas où la condition du if est fausse.

```
<?php  
$a = 10;  
$b = 2;  
if ($a > $b) {  
    echo "a est plus grand que b";  
} else {  
    echo "a est plus petit que b";  
}  
?>
```

Modifiez les valeurs des \$a et \$b et vérifiez que quelque chose s'affiche dans un cas ou dans l'autre.

## L'instruction if...elseif ...else

Il peut arriver que nous ayons à gérer des tests différents que c'est vrai ou c'est faux. Par exemple si l'on souhaite savoir si un nombre est supérieur, inférieur ou égale à un autre. Cette troisième option permet d'affiner et d'intercepter des valeurs plus précises que de savoir si le résultat d'un test est vrai ou faux.

Modifiez les valeurs de \$a et \$b afin de valider les différents tests.

```
<?php
$a = 10;
$b = 2;
if ($a > $b) {
    echo "a est plus grand que b";
}
elseif ($a < $b) {
    echo "a est plus petit que b";
}
else {
    echo "a est égale à b";
}
?>
```

Notez que vous pouvez utiliser autant de elseif que vous avez besoin. Considérez cet opérateur comme permettant de réaliser des tests de l'ordre :

- si ceci est vrai
- ou bien si cela est vrai
- ou bien si cela est vrai
- ou bien si cela est vrai
- sinon

## L'instruction switch

Au lieu de d'accumuler les elseif, php permet d'utiliser l'instruction switch. C'est une manière différente de réaliser des actions en fonction du résultat d'un test de condition (exactement comme plusieurs elseif les uns à la suite des autres).

La syntaxe est la suivante :

```
switch (n) {
    case label1:
        code à exécuter si n=valeur1;
        break;
    case label2:
        code à exécuter si n=valeur2;
        break;
    case label3:
        code à exécuter si n=valeur3;
        break;
    ...
    default:
        code à exécuter si n n'est pas une valeur reconnue ;
}
```

Voici un exemple

```
<?php
$i = 1 ;
switch ($i) {
    case 0:
        echo "i est égale à 0";
        break;
    case 1:
        echo "i est égale 1";
        break;
    case 2:
        echo "i est égale 2";
        break;
    default:
        echo "i n'a pas été reconnu";
        break;
}
?>
```

Ce code est équivalent à ceci

```
<?php
$i = 1 ;
if ($i == 0) {
    echo "i est égale à 0";
}
elseif ($i == 1) {
    echo "i est égale à 1";
}
elseif ($i == 2) {
    echo "i est égale à 2";
}
else {
    echo "i n'a pas été reconnu";
}
?>
```

**Exercice 1**

1. Afficher « *Bonjour tout le monde* » si la variable `$a` **est supérieur** à la variable `$b`
2. Afficher « *Bonjour tout le monde* » si la variable `$a` **n'est pas égale** à la variable `$b`
3. Affichez « *oui* » si la variable `$a` **est égale** à la variable `$b`, **sinon** affichez « *Non* »
4. Affichez « *1* » si la variable `$a` est égale à la variable `$b`, Affichez « *2* » si la variable `$a` **est supérieur** à `$b`, **sinon** affichez « *3* »
5. A partir de l'instruction **switch**, affichez "*Bonjour*" si `$couleur` est "rouge" et "*bienvenue*" si `$couleur` est "vert" et pour tous les autres cas « *Les goûts et mes couleurs, ça ne se discute pas* ».
6. A partir des instructions **if ... elseif ...**, affichez "*Bonjour*" si `$couleur` est "rouge" et "*bienvenue*" si `$couleur` est "vert", **sinon** « *Les goûts et mes couleurs, ça ne se discute pas* ».

## Les boucles

Souvent, lorsque vous écrivez du code, vous aurez besoin que le même bloc de code s'exécute un certain nombre de fois tant qu'une condition n'est pas vérifiée.

Par exemple :

- « Tant que la note de l'élève n'est pas supérieur à 15, relance le QCM »
- « Pour tout les éléments d'un tableau, affiche chaque valeur sur une ligne »

PHP gère ce type d'actions avec les instructions suivantes :

- **while** : réalise une boucle sur un bloc de code tant que la condition spécifiée n'est pas vraie
- **do...while** : Réalise une boucle sur un bloc de code tant que la condition spécifiée est vraie
- **for** : Répète un bloc de code un nombre de fois spécifié
- **foreach** : Répète un bloc de code pour chaque élément d'un tableau

## While

Cette instruction réalise une boucle sur un bloc de code tant que la condition spécifiée n'est pas vraie. Ceci implique que le bloc de code exécuté soit en mesure de modifier la valeur de la condition testée. Cette boucle sera interrompu dès que la condition testée est vraie.

Par exemple

```
<?php
$x = 1;

while($x < 5) {
    echo "la variable x vaut: $x <br>";
    $x++;
}
?>
```

Que fait ce code ?

1. La variable \$x est défini à 1.
2. Lorsque la boucle while est lancée, la condition \$x < 5 est testée
3. \$x valant 1, elle n'est pas inférieure à 5. La boucle se lance puisque l'opérateur de comparaison retourne comme valeur faux. Elle ne s'arrêtera que si la condition retourne vrai.



4. L'ouverture de crochet { indique à PHP le début du bloc de code de notre boucle
5. « La variable x vaut : 1 » s'affiche au premier tour de boucle.
6. \$x++, comme on l'a vu est équivalent à l'instruction \$x = \$x+1. La valeur de \$x passe à 2.
7. La fermeture de crochet } indique à php la fin du bloc à exécuter en boucle.
8. La condition est testée à nouveau.
9. Puisqu'elle est toujours fausse, PHP relance la boucle
10. « La variable x vaut : 2 » s'affiche au second tour de boucle.
11. \$x++ incrémente la variable \$x qui vaut maintenant 3
12. Etc ...

Cette boucle tournera jusqu'à ce que la valeur de \$x inférieure à 5. Dès l'instant où la condition est vérifiée, PHP sortira de boucle.

### Exercice 2

1. Faites une boucle **while** où \$a est initialisé à 0. Tant que \$a n'est pas supérieur ou égale à 10 affichez \$a sur une ligne.
2. Faites une boucle **while** où \$b est initialisé à 20. Tant que \$b n'est pas inférieur à 6 affichez \$b sur une ligne.
3. Faites une boucle **while** où \$c est initialisé à 100. Tant que \$c n'est pas inférieur à 20 affichez \$c sur une ligne et divisez \$c par 10.
4. Faites une boucle **while** où \$d est initialisé à 1. Tant que \$d n'est pas supérieur à 1000 affichez \$d sur une ligne et multipliez \$d par 10.

## do...while

Contrairement à while, la boucle do...while sera exécutée tant que la condition est fausse.

Essayez ce code :

```
<?php
$x = 1;

do {
    echo "$x est inférieur à 5<br>";
    $x++;
}
while ($x <= 5);
?>
```

*Explication sur ce code ?*

1. La variable \$x est initialisée à 1.
2. l'instruction do { ... } définit la portion de code à exécuter tant que la condition du while () n'est pas fausse.

3. « 1 est inférieur à 5 » s'affiche au premier tour de boucle.
4. `$x++`, (équivalent à `$x = $x+1`) incrémente `$x` qui vaut maintenant 2.
5. La fermeture de crochet `}` indique à php la fin du bloc à exécuter en boucle.
6. La condition (`$x <= 5`) est testée.
7. Puisqu'elle est toujours vraie, PHP relance la boucle
8. « 2 est inférieur à 5 » s'affiche au second tour de boucle.
9. `$x++` incrémente la variable `$x` qui vaut maintenant 3
10. Etc ...

Cette boucle tournera jusqu'à ce que la valeur de `$x` soit supérieur ou égale à 5. Dès l'instant où la condition ne sera plus vérifiée (c-à-d fausse), PHP sortira de boucle.

### Exercice 3

1. Faites une boucle **do...while** où `$e` est initialisé à 0. Tant que `$e` est supérieur à 10 affichez `$e` sur une ligne.
2. Faites une boucle **do...while** où `$f` est initialisé à 20. Tant que `$f` est inférieur à 6 affichez `$f` sur une ligne.
3. Faites une boucle **do...while** où `$g` est initialisé à 100. Tant que `$g` est inférieur ou égale à 20 affichez `$g` sur une ligne et divisez `$g` par 10.
4. Faites une boucle **do...while** où `$h` est initialisé à 1. Tant que `$h` est supérieur à 1000 affichez `$h` sur une ligne et multipliez `$h` par 100.

## Boucle for

La boucle `for` est un peu plus complexe. Elle permet d'exécuter un certain nombre de fois un bloc d'instructions. Par exemple répéter 10 fois le bloc d'instructions. Cette répétition en programmation s'appelle une itération.

Voici la syntaxe:

```
for (expr1; expr2; expr3)
    commandes
```

1. La première expression (`expr1`) est **exécuté** une seule fois avant de démarrer la boucle. C'est ici que l'on initialisera notre variable pour débiter un décompte.
2. La seconde expression (`expr2`) sera **évaluée au début de chaque itération** avec un opérateur de comparaison. C'est ici que l'on test si le décompte est terminé ou pas.
  - a) Si l'évaluation retourne `true`, le décompte n'est pas terminé. La boucle continue et les commandes sont exécutées.
  - b) Si l'évaluation vaut `false`, le décompte est terminé, la boucle s'arrête et PHP passe à la suite du code.
3. La 3ème expression (`expr3`) est **exécutée à la fin de chaque itération**. C'est ici que l'on pourra incrémenter par exemple la variable que l'on initialisé avant le début de la boucle pour faire avancer notre compteur.

Voyez cet exemple

Le compteur `$i`  
début à 1

Si `$i` est  $\leq$  à 10  
La boucle sera exécutée

A chaque fin de boucle  
`$i` augmentera de 1

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

Code exécuté à chaque itération

Testez ce code

Il affiche 12345678910, soit le résultat de nos 10 itérations.

Modifiez votre code pour faire apparaître plus clairement le résultat de vos itérations.

```
<?php
for ($i = 1; $i <= 10; $i++) {
    echo "Itération N° $i <BR>";
}
?>
```

On voit plus clairement que l'on a bien exécuté 10 fois notre bloc d'instructions. Notez également que PHP a reconnu votre variable \$i au sein de la chaîne de caractères. Vous n'avez pas eu besoin de concaténer une chaîne de caractères pour afficher \$i.

Faisons un peu plus fort. Utilisons \$i pour faire un petit calcul a chaque itération. Faisons +10 par exemple et affichons le résultat sur une seule ligne.

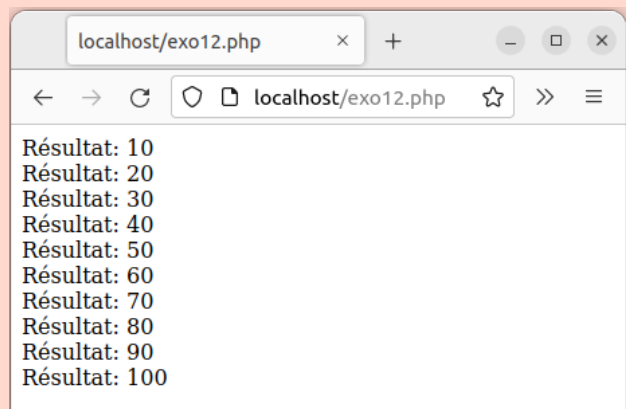
```
<?php
for ($i = 1; $i <= 10; $i++) {
    echo $i + 10 . "<BR>";
}
?>
```

#### Notez le "."<BR>"

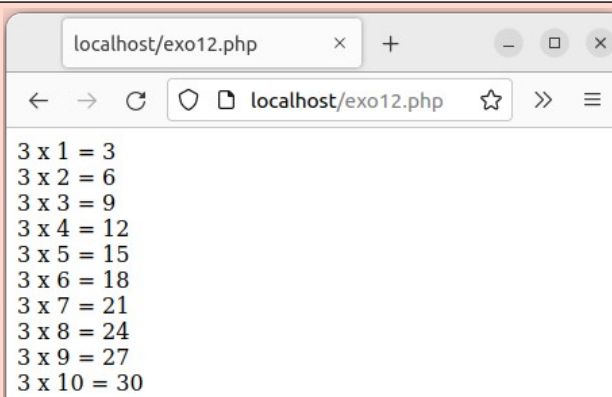
On a utilisé un opérateur de concaténation (le point) pour accoler une chaîne de caractères (entre guillemets) à notre petit calcul. Cette méthode est très utile pour générer des chaînes de caractères. Utilisez là sans modération.

### Exercice 4

1. Modifiez le code ci-dessus pour obtenir ceci



2. Faire une boucle for pour afficher la table de 3 sous cette forme.



```
localhost/exo12.php
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

## Boucle foreach

La boucle foreach est utilisée pour réaliser des itérations sur les tableaux.

foreach peut donc être utilisée pour parcourir

- Les valeurs des valeurs d'un tableau
- les paires clé/valeur d'un tableau associatif.

### Pour rappel,

1) Un tableau peut être constitué de données de type différents. Chaque élément peut être affiché par son index.

```
<?php
$colors = array("rouge", "vert", "bleu", "jaune");
echo $colors[2];
?>
```

2) Un tableau associatif stocke les données sous la forme de clé/valeur. A chaque valeur est associée une clé. Pour accéder à une valeur, il faut appeler sa clé.

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo $age["Ben"];
?>
```

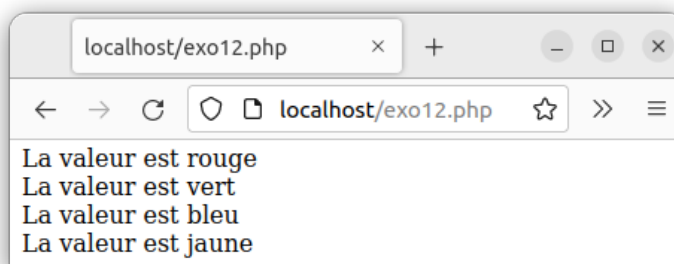
L'utilisation de tableaux est un moyen très efficace pour stocker des données dès l'instant que le volume d'information que l'on a à gérer augmente. L'inconvénient est que pour réaliser un traitement sur un tableau, nous devons fournir soit un index, soit une clé.

La boucle foreach nous fera gagner un temps précieux car cette instruction est conçu pour réaliser une itération sur chaque entrée d'un tableau.

```
<?php
$couleurs = array("rouge", "vert", "bleu", "jaune");

foreach ($couleurs as $valeur) {
    echo "La valeur est $valeur <br>";
}
?>
```

Dans cet exemple, foreach parcourt le tableau \$couleurs et réalise une itération pour chaque entrée. De cette manière nous pouvons rendre plus lisible le contenu d'un tableau.

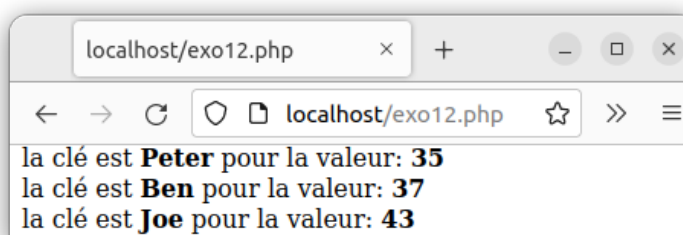


Pour les tableau\*x associatifs, la syntaxe est un peu différente. En effet, nous devons récupérer la clé en plus de la valeur. Foreach fonctionnera de la même manière. Nous récupéreront la clé/valeur à chaque itération.

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $cle => $valeur) {
    echo "la clé est <b>$cle</b> pour la valeur:
    <b>$valeur</b><br>";
}
?>
```

Notez que le syntaxe de foreach pour récupérer les clé et les valeurs est la même que celle utilisée pour créer un tableau associatif (clé => valeur).



Il suffit de mentionner à foreach :

- Entre parenthèses, le nom du tableau, ici \$age
- l'instruction as (comme)
- le nom de la variable que l'on souhaite utiliser à chaque itération pour récupérer la clé
- les caractères =>
- le nom de la variable que l'on souhaite utiliser à chaque itération pour récupérer la valeur.

**Exercice 5**

1. Faites un tableau qui stocke les jours de la semaine. A l'aide de foreach, affichez les jours de la semaine (un jour par ligne).
2. Faites un tableau qui stocke le nom des planètes du système solaire. A l'aide de foreach, affichez les planètes du système solaire (une planète par ligne).
3. Faites un tableau associatif où la clé est le mois de l'année et la valeur le nombre de jour de ce mois. A l'aide d'une boucle foreach affichez chaque entrée sous la forme :  
« Le mois de janvier contient 31 jours »
4. Voici la distance qui sépare les planètes du système solaire du soleil.  
Mercure : 57 909 050 km  
Vénus : 108 208 475 km  
La Terre : 149 598 023 km  
Mars : 227 939 200 km  
Jupiter : 778 340 821 km  
Saturne : 1 426 666 422 km  
Uranus : 2 870 658 186 km  
Neptune 4 498 396 441 km

Faites un tableau associatif dont la clé est le nom de la planète et la valeur la distance. A l'aide de foreach, affichez le tableau sous la forme :  
« Mercure est à 57 909 050 km du Soleil »