Les conditions, les opérateurs et les boucles

Sommaire

Les repères de lecture


Retour au chapitre


Définition


Objectif(s)


Espace Élèves


Vidéo / Audio


Point important / À retenir


Remarque(s)


Pour aller plus loin


Normes et lois


Quiz

I. Les conditions

A. If, else

Le couple *if / else* (qui se traduit par *si / sinon*) est un élément incontournable de tout langage de programmation. Il s'agit d'une structure conditionnelle qui permet de vérifier des conditions et de procéder ensuite à des opérations en fonction des résultats de cette vérification.

Par exemple dans le langage courant nous dirions : si ce film m'intéresse j'irai le voir au cinéma sinon je lirai un livre.

Le mot « si » introduit la notion de condition qui est posée par « si ce film m'intéresse ». L'action qui en découle est « j'irais le voir au cinéma ». L'alternative est posée par « sinon » et l'action est « sinon je lirais un livre ».

La structure de cette condition peut donc être posée ci-dessous :

Pour voir la syntaxe en PHP, voici un autre exemple simple de type mathématique que nous allons ensuite tester en PHP : si le nombre $a est plus grand que $b, alors nous affichons *$a* sinon nous affichons *$b*.

```
1 if ($a > $b) // si $a est plus grand que $b
2 {
3     echo $a; // affichons $a
4 }
5 else // sinon
6 {
7     echo $b; // affichons $b
8 }
```

Fig. 1

L'explication de la syntaxe est la suivante :

- **La première ligne** contient l suivi de ce que nous voulons vérifier entre parenthèses : *$a* plus grand que *$b*. (*$a* > *$b*) ;
- **La deuxième ligne** contient une ouverture d'accolade *{*
  Les accolades contiennent l'ensemble des d'instructions (ou bloc d'instructions) qu'il est prévu d'exécuter si la condition posée est vérifiée. Ici il y a seulement une instruction (*echo $a*).
- **Troisième ligne** : nous procédons ici à l'action prévue qui est d'afficher *$a*.
- **Quatrième ligne** : l'action que nous voulons effectuer est terminée, nous fermons donc l'accolade avec le signe}.
- **Cinquième ligne** : le mot-clef *else* (sinon) introduit l'alternative (le cas où *$a* n'est pas supérieur à *$b*).
- **Sixième ligne** : accolade de début d'action.

– **Septième ligne** : l'action souhaitée dans le cas où la condition n'est pas réalisée (afficher *$b*).

– **Huitième ligne** : accolade de fin d'action.

À noter que le *else* est facultatif. Il est possible comme dans l'exemple suivant de ne pas l'utiliser.

Exemple sans* else *:

```
if ($variable > 10)
{
      echo 'La variable est supérieure à 10';
}
```

Fig. 2

Dans cet exemple, si la variable est inférieure à 10, nous ne faisons rien.

Le principe des conditions fonctionne toujours de la même façon mais être enrichi en fonction des besoins. Par exemple nous aimerions poser plusieurs conditions en simultané. Si nous voulions vérifier que cette variable est supérieure à 10 et en même temps inférieure à 15, nous utiliserions l'opérateur AND qui permet d'effectuer deux conditions en même temps.

L'opérateur AND est abordé dans la section sacrée aux opérateurs logiques.

Voici le code utilisant AND qui permet de vérifier cette double condition :

```
if ($variable > 10 AND $variable < 15 )
{
      echo 'La variable est supérieure à 10 et inférieure à 15';
}
```

Fig. 3

Afin que le code soit plus clair, il est recommandé d'ajouter des parenthèses autour de chaque condition. En effet les conditions ne sont pas toujours aussi simples et peuvent parfois être nombreuses. Bien qu'à première vue, il semble que cet ajout alourdisse le code, en réalité et avec la pratique nous nous rendons compte que les parenthèses permettent de bien séparer les conditions, et éventuellement de les compter.

```
if ( ($variable > 10) AND ($variable < 15) )
{
      echo 'La variable est supérieure à 10 et inférieure à 15';
}
```

Fig. 4

1. Elseif

Le mot-clef *elseif* (sinon si) est très pratique car il permet d'ajouter une ou plusieurs conditions supplémentaires.

Exemple

Nous voulons afficher « c'est le matin » s'il n'est pas midi, « c'est l'après-midi » s'il est moins de 18h, « c'est la soirée » s'il est moins de 22h, « c'est la nuit » s'il est plus de 22h et moins de 6h.

La structure est donc la suivante :

Tableau n°1

CONDITION	ACTION
Si l'heure <12	Alors afficher c'est le matin
Sinon si l'heure < 18	Alors afficher c'est l'après-midi
Sinon si l'heure <22	Alors afficher c'est le soir
Sinon si l'heure >22 et l'heure<6	Alors afficher c'est la nuit

Écrivons le code en utilisant les elseif :

```
$heure = 9 ;
if ($heure < 12) // si l'heure est inférieure à 12
{
      echo 'matin';
}
elseif ($heure < 18) // sinon : si l'heure est inférieure à 18
{
      echo 'après-midi';
}
elseif ($heure < 22) // sinon : si l'heure est inférieure à 22
{
      echo 'soir';
}
elseif ($heure > 22 AND $heure < 6) // sinon : si l'heure est > 22 et < 6
{
      echo 'nuit';
}
```

Fig.5

B. Switch

L'instruction *switch* est équivalente à plusieurs instructions *if/else*. Il arrive souvent que nous devions travailler sur des conditions avec beaucoup de valeurs différentes afin de procéder aux actions souhaitées.

Si les instructions *if/else/elseif* sont très utiles, il peut parfois être judicieux d'utiliser l'instruction *switch* afin d'obtenir un code plus simple.

Comparaison entre *if/else/elseif* et *switch* :

```
// avec if/else/elseif
if ($i == 0)
{
    echo "i égal 0";
}
elseif ($i == 1)
{
    echo "i égal 1";
}
elseif ($i == 2)
{
    echo "i égal 2";
}
```

Fig.6

```
// avec switch
switch ($i)
{
    case 0:
        echo "i égal 0";
        break;
    case 1:
        echo "i égal 1";
        break;
    case 2:
        echo "i égal 2";
        break;
}
```

Fig.7

Le résultat sera exactement identique mais le code avec *switch* est plus simple et plus adapté puisqu'en l'occurrence, nous testons ici la même variable *$i*.

La syntaxe de *switch* repose sur l'utilisation du mot-clef *case* et du mot-clef *break*.

Le mot-clef *case* signifie littéralement « **dans le cas où** » et permet donc de comprendre aisément le code comme suit :

Tableau n°2

Case 0	dans le cas où $i a pour valeur 0 afficher « i égal 0 »
Case 1	dans le cas où $i a pour valeur 1 afficher « i égal 1 »
Case 2	dans le cas où $i a pour valeur 2 afficher « i égal 2 »

La commande *break* a pour rôle d'interrompre le processus si la condition est vérifiée (*to break* en anglais). Ainsi, si *$i* est égal à une des valeurs proposées (par exemple 0), l'affichage va être effectué, puis la commande *break* va interrompre le *switch*.

Le mot-clef *default* peut être utilisé pour gérer les autres cas comme le montre l'exemple suivant.

```
switch ($i)
{
    case 0:
        echo 'i égal 0';
        break;
    case 1:
        echo 'i égal 1';
        break;
    case 2:
        echo 'i égal 2';
        break;
      default :
      echo 'i n\'est pas égal aux valeurs 0, 1 ou 2';
}
```

Fig.8

Le *switch* peut également être utilisé avec des expressions. Ainsi l'exemple portant sur les heures que nous avons traité avec *if/else* peut être codé avec *switch* :

```
switch($heure){
 case ($heure < 12) :
 echo 'matin';
 break;
 case ($heure < 18) :
 echo 'après-midi';
 break;
 case ($heure < 22) :
 echo 'soir';
 break;
 case ($heure > 22 AND $heure < 6)
 echo 'nuit';
 break;
 }
```

Fig.9

Concernant la syntaxe, il est important de bien noter :
– les doubles points après la valeur ou l'expression qui suivent le *case* ou le *défaut* ;
– les points-virgules classiques après l'instruction *echo* et le *break*.

II. Les opérateurs

Un opérateur est une instruction spéciale qui s'écrit avec des caractères particuliers.

Nous avons déjà vu par exemple l'opérateur = et l'opérateur > qui respectivement permettent l'assignation de valeur à une variable ou la comparaison de grandeur. Cette leçon a pour objectif de passer en revue l'ensemble des opérateurs disponibles en PHP.

Les opérateurs sont de 5 types :
- opérateurs d'assignation ;
- opérateurs de comparaison ;
- opérateurs de calcul ;
- opérateurs combinés ;
- opérateurs logiques.

A. Opérateurs d'assignation

Nous avons également déjà vu l'opérateur d'assignation lors de la déclaration de variable.

L'opérateur d'assignation est le signe «=» égal.

Par exemple, pour déclarer une variable $a qui a la valeur 'il fait beau', l'instruction est *$a = 'il fait beau'*.

Ce qui est à gauche du signe égal se voit affecter la valeur de l'expression qui est à droite du signe égal.

B. Opérateurs de comparaison

Les opérateurs de comparaison permettent de comparer des valeurs entre elles.

Tableau n°3

==	Égal	L'égalité repose sur **l'opérateur double égal**. **Exemple : $a == $b**
!=	Différent	La différence est marquée par l'opérateur **point d'exclamation suivi du signe égal.** **Exemple : $a != $b** Notons que le point d'exclamation exprime la négation.
===	Identique	Pour savoir si une valeur est identique en valeur et en type, nous utilisons cette forme d'égalité stricte sur laquelle nous reviendrons. Retenons qu'elle s'effectue avec l'opérateur **tripe égal**. **Exemple : $a === $b**
>	Supérieur strictement	**Exemple : $a > $b**
<	Inférieur strictement	**Exemple : $a < $b**
>=	Supérieur ou égal	**Exemple : $a >= $b**
<=	Inférieur ou égal	**Exemple : $a <= $b**

C. Opérateurs de calcul

Les opérateurs de calcul permettent d'effectuer des opérations mathématiques.

Tableau n°4

+	Addition	**Exemple** : $a + $b
-	Soustraction	**Exemple** : $a - $b
*	Multiplication	**Exemple** : $a * $b
/	Division (quotient)	**Exemple** : $a / $b
%	Modulo (le reste d'une division)	**Exemple** : $a % $b

Exemples de syntaxe :

```
echo 2 + 3; //affiche résultat d'une addition
echo $a - $b; //affiche résultat d'une soustraction
```

Fig. 10

Exemple de script contenant du calcul, une condition et une comparaison :

```
$b = 10;
$c = 2;
if (($b*$c) >= 20)
{
      echo $b*$c;
}
else
{
      echo 'Autre résultat';
}
```

Fig. 11

D. Opérateurs combinés

Les opérateurs combinés :

– fonctionnent pour les opérateurs arithmétiques et les opérateurs sur les chaînes de caractères ;
– utilisent la valeur d'une variable dans une expression et affectent le résultat de cette expression à cette variable.

***Exemple 1** : plus égal*

```
$a = 3;
$a += 5; /* affecte la valeur 8 à la variable $a
correspond à l'instruction '$a = $a + 5' */
```

Fig. 12

***Exemple 2** : incrémentation*

```
$i = 4;
$i++; // syntaxe abrégée signifiant $i + 1
echo $i; // affiche 5
```

Fig. 13

Exemple 3 *: décrémentation*

```
$i = 4;
$i--; // syntaxe abrégée signifiant $i - 1
echo $i; // affiche 3
```

Fig. 14

Exemple 4 *: point égal - concaténation abrégée*

```
$text = 'Bonjour' ;
$text .= ' tout le monde !'; // identique à $text = $text . ' tout le monde !';
echo $text ; // affiche «  bonjour tout le monde !»
```

Fig. 15

E. Opérateurs logiques : AND, OR

1. AND

AND : c'est le ET logique qui signifie que les propositions doivent être vraies en même temps.

Par exemple, si nous cherchons un nombre qui soit en même temps plus grand que 5 et plus petit que 7, le résultat est 6.

Exemple :

```
$b = 10;
$c = 2;
$d = 8;
$e = 12;
/*
Si $b multiplié par $c est supérieur ou égal à 20
ET SI DANS LE MÊME TEMPS (AND)
$d additionné à $e est égal à 20
ALORS
on affiche le résultat de $b multiplié par $c
SINON
on affiche la chaîne de caractères « Autre résultat »
*/

if ((($b * $c) >= 20) AND ($d + $e == 20 ))
{
      echo $b * $c; // dans cet exemple, ceci sera affiché
}
else
{
      echo 'Autre résultat';
}
```

Fig. 16

a. OR

OR : c'est le **OU logique et XOR est le OU logique exclusif,** qui signifie ou bien : soit l'une ou soit l'autre des propositions doit être vraie. Par exemple, si nous cherchons un nombre qui soit plus grand que 5 ou plus petit que 7, le résultat peut être n'importe quel nombre.

En effet, 24 sera plus grand que 5, tandis que 3 sera plus petit que 7.

```
$b = 10;
$c = 2;
$d = 8;
$e = 12;
/*
Si $b est plus grand que $c OU BIEN (OR) $d est plus petit que $e
ALORS
on affiche $d
SINON
on affiche $b
*/
if (($b > $c) OR ($d < $e ))
{
      echo $d ; // dans cet exemple, ceci sera affiché
}
else
{
      echo $b;
}
```

Fig. 17

III. Les boucles

Les boucles sont des structures permettant d'effectuer une ou plusieurs instructions généralement répétitives tant que les conditions posées sont vérifiées.

Prenons l'exemple d'une recette de cuisine : il faut faire bouillir l'eau des pâtes tant que celles-ci ne sont pas bien cuites. Cet exemple illustre bien la définition : la condition est le degré de cuisson des pâtes, l'instruction est de faire bouillir l'eau.

En développement, nous rencontrerons souvent les boucles suivantes :
- *while* ;
- *for* ;
- *foreach*.

Bien qu'elles soient utilisées dans des cas de figure différents, ces structures fonctionnent toutes selon le principe de base : tant que la condition est vraie, alors des instructions sont effectuées.

A. La boucle *while*

La boucle *while* n'est pas la boucle la plus utilisée mais sa simplicité d'utilisation permet de bien comprendre le fonctionnement des boucles. *While* en anglais signifie tant que. Avec une boucle *while*, l'instruction souhaitée s'exécute tant que l'expression de la boucle *while* est vraie.

Exemple de code contenant une boucle while :

```
$i = 1; //affectons à la variable $i la valeur 1
while ($i < 5) // tant que $i est inférieur à 5 ...
{
 echo $i .'<br>'; //... alors nous affichons la valeur de la variable
 $i++; // puis nous exécutons l'addition $i = $i + 1 au format abrégé
}
```

Fig. 18

Cette boucle *while* va afficher :

```
1
2
3
4
```

Fig. 19

Analysons la syntaxe de la boucle *while* :

Première ligne :
- utilisation du mot-clef *while* ;
- puis entre parenthèses nous posons une condition qui indique que la variable *$i* est inférieure strictement à 5.

Ainsi, tant que la variable *$i* sera inférieure à 5, la boucle pourra fonctionner. Notons l'utilisation du double point après la parenthèse.

Deuxième ligne : première instruction souhaitée. Ici un affichage de la variable $i avec *echo*. Nous avons ajouté un saut de ligne *HTML
* pour bien lister les affichages.

Troisième ligne : seconde instruction *$i++* qui est la syntaxe abrégée de l'addition *$i + 1*.

Notons que nous pouvons rencontrer la syntaxe avec le mot-clef *endwhile*, sans parenthèse mais avec les double-points comme ceci :

```
while ($i < 5):
 echo $i .'<br>';
 $i++;
endwhile;
```

Fig. 20

La boucle *while* ne doit pas être confondue avec la boucle do *while*, fonctionnant selon le même principe mais pour laquelle l'expression est testée à la fin de tour de boucle et non pas au début.

B. La boucle *for*

La boucle for est une structure de contrôle très utilisée.

S'agissant d'une boucle, le principe vu précédemment est donc le même : les instructions contenues dans la boucle seront effectuées **tant que** la condition posée est vérifiée.

Voici un exemple que nous allons analyser pour bien la comprendre.

```
for ( $i = 1; $i < 5; $i++ )
{
      echo $i;
}
```

Fig. 21

La boucle **for** comporte 3 instructions :

- **l'initialisation** : nous affectons une valeur à la variable. Ici : *$i = 1* ;

- **la condition de continuation** : c'est-à-dire que la boucle sera effectuée tant que cette condition sera vraie. Ici : $i < 5 ($i inférieur à 5). Cela signifie que la boucle sera effectuée tant que *$i* sera inférieur à 5 ;

– **la manière d'incrémenter le compteur qui est ici une addition**. Cette dernière instruction est exécutée à la fin de chaque tour de boucle. La boucle effectue donc une addition.

En résumé, la boucle *for* dans notre exemple peut se traduire par la phrase suivante : nous allons afficher la variable $i puis additionner celle-ci à 1 tant que le résultat de cette addition sera inférieur à 5.

Détaillons la boucle dans son intégralité :

1. Premier tour de boucle (on dit aussi : itération)

La boucle vérifie la **condition de continuation**. Ici la boucle vérifie donc que la valeur de *$i* est inférieure à 5 (*$i < 5*).

Puisque l'**instruction d'initialisation** a affecté la valeur 1 à la variable *$i* et que 1 est bien inférieur à 5, la boucle peut être effectuée.

$i a la valeur 1, l'instruction *echo* affiche donc 1.

Ensuite l'addition *$i*++ est effectuée.

$i*++ est la syntaxe abrégée de l'addition *$i + 1.

Désormais *$i* a pour valeur 2.

2. Deuxième tour de boucle (deuxième itération) :

La boucle vérifie la **condition de continuation**. Ici la boucle vérifie donc que la valeur de $i est inférieure à 5 (*$i < 5*).

Puisque $i a pour valeur 2 et qu'il est vrai que 2 est inférieur à 5, alors la boucle peut être effectuée et l'instruction *echo* affiche donc 2.

Ensuite l'addition $i++ est effectuée, désormais $i a pour valeur 3.

3. Troisième tour de boucle (troisième itération) :

La boucle vérifie la **condition de continuation**. Ici la boucle vérifie donc que la valeur de $i est inférieure à 5 (*$i < 5*).

Puisque $i a pour valeur 3 et qu'il est vrai que 3 est inférieur à 5, alors la boucle peut être effectuée.

Suite au deuxième tour de boucle, la valeur de $i est donc 3 et l'instruction *echo* affiche donc 3.

Ensuite l'addition *$i*++ est effectuée et désormais $i a pour valeur 4.

4. Quatrième tour de boucle (quatrième itération) :

La boucle vérifie la **condition de continuation**.

Puisque *$i* a pour valeur 4 et qu'il est vrai que 4 est inférieur à 5, alors la boucle peut être effectuée et l''instruction *echo* affiche donc 4.

Ensuite l'addition *$i*++ est effectuée et désormais $i a pour valeur 5.

5. Cinquième tour de boucle (cinquième itération) :

La boucle vérifie la **condition de continuation**. Ici la boucle vérifie donc que la valeur de $i est inférieure à 5 (*$i < 5*). $i a désormais pour valeur 5 donc **puisqu'il n'est pas vrai que 5 est inférieur à 5, alors la boucle s'interrompt.**

Analysons maintenant la syntaxe précise de la boucle *for* :

Première ligne :
- utilisation du mot-clef *for* ;
- puis ouverture d'une parenthèse ;
- rédaction de l'instruction de l'initialisation *$i = 1* suivi d'un point-virgule ;
- rédaction de la condition de continuation *$i < 5* suivi d'un point-virgule ;
- rédaction de l'instruction d'incrémentation *$i* ++ ;
- fermeture de la parenthèse.

Deuxième ligne : ouverture de l'accolade contenant le bloc des instructions

Troisième ligne : instruction souhaitée. Ici un affichage de la variable *$i* avec *echo*.

Notons que plusieurs instructions sont possibles.

Quatrième ligne : fermeture de l'accolade.

Ainsi le code :

```
for ( $i = 1; $i < 5; $i++ )
{
      echo $i;
}
```

Fig. 22

Affiche :

```
1
2
3
4
```

Fig. 23

Notons que nous pouvons rencontrer la syntaxe avec le mot-clef *endfor*, sans parenthèse mais avec les double points comme ceci :

```
for ( $i = 1; $i < 5; $i++ ) :
echo $i .'<br>';
endfor;
```

Fig.24

Précaution concernant les boucles *for* : si la condition posée est toujours vraie, nous avons le risque de produire des boucles infinies, comme dans cet exemple qui stipule que tant que *$i* sera supérieur à 0, alors on l'additionne à 1 :

```
for ($i = 1, $i > 0, $i++)
```

Fig.25

Attention si nous testons ce code, le navigateur court le risque de planter. Heureusement, PHP a une procédure dite de time-out qui interrompt les scripts dont le temps de calcul dépasse un certain laps de temps (30 secondes par défaut).

C. La boucle *foreach*

La boucle foreach est utilisée uniquement dans le cas des tableaux de données (array) et est détaillée dans le chapitre des tableaux de données.

1. Interruption d'une boucle avec l'instruction break

La commande *break* permet d'interrompre une boucle.

La syntaxe est simple, il suffit de placer *break* à l'endroit souhaité et éventuellement le soumettre à une condition.

Cette instruction peut être très utile en fonction des besoins, pour arrêter une boucle avant la fin prévue de celle-ci comme dans cet exemple :

```
$i = 1;
while ($i < 5)
{
 echo $i .'<br>';
 $i++;

 if($i == 3) // si $i a la valeur 3, alors on interrompt la boucle
 {
      break;
```

Fig.26

L'instruction break marque donc la fin de la boucle qui va afficher :

```
1
2
3
```

Fig.27