# BACK #1319 - PHP

# 2 - Traitements de base

## 2.1 - Les opérateurs

> Les opérateurs sont des symboles qui servent à manipuler des expressions.

Ils sont utilisés pour effectuer des opérations, affecter ou comparer des valeurs, etc ...

### Opérateur d'affectation

L'opérateur d'affectation est le signe égal (=). La variable située à gauche du signe égal prend la valeur spécifiée à sa droite

```
$variable = valeur;
```

Cet opérateur égal renvoie la valeur affectée. On peut donc réaliser des affectations en chaîne :

```
$variable1 = $variable2 = valeur;
```

#### Affectation par copie et par référence

> Par défaut, l'affectation des variables se fait par copie

```
$variable1 = $variable2;
```

Les valeurs des deux variables sont indépendantes et affecter `$valeur1` n'affectera pas `$valeur2`.

Au lieu de ce comportement, on peut souhaiter créer un lien fort entre les deux valeurs.

On utilise alors le signe `&` après le signe égal :

```
$variable1 = &$variable2;
```

### Opérateurs arithmétiques

> Les opérateurs arithmétiques du langage permettent d'effectuer les opérations arithmétiques basiques.

PHP gère les opérations arithmétiques suivantes :

| Opération | Exemple |
|---|---|
| Inverse | $a = -$b; |
| Addition | $a = 2 + 3; |
| Soustraction | $a = 4 – 1.3; |
| Multiplication | $a = 2 * 6; |

| Opération | Exemple |
|---|---|
| Division | $a = 500 / 2e2; |
| Modulo | $a = 54 % 8; |
| Puissance | $a = 3 ** 2; |

## Opérateurs d'incrémentation / décrémentation

> PHP fournit la possibilité d'incrémenter et décrémenter de 1 un entier à travers deux opérateurs : `++` et `--`

Ces opérateurs peuvent être utilisés en pré ou post évaluation :

```php
// Pré décrémentation / pré incrémentation:
$a=5;
echo --$a; // echo ++$a;

// Post décrémentation / post incrémentation :
$a=5;
echo $a--; // echo $a++;
```

## Opérateurs sur les bits

> Ces opérateurs permettent de transformer la représentation binaire d'un nombre entier

| Nom | Exemple | Description |
|---|---|---|
| ET (AND) | 5 & 9 | Les bits positionnés à 1 côté gauche **ET** côté droit sont positionnés à 1 |
| OU (OR) | 5 \| 9 | |
| Ou exclusif (XOR) | 5 ^ 9 | Les bits positionnés à 1 côté gauche **ou** côté droit, **mais pas des 2 côtés**, sont positionnés à 1 |
| Décalage à gauche | 5 << 9 | Les bits positionnés à gauche sont décalés de x fois vers la gauche (nombre côté droit). Equivaut à multiplier par 2 X fois |
| Décalage à gauche | 5 >> 9 | Les bits positionnés à gauche sont décalés de x fois vers la droite (nombre côté droit). Equivaut à diviser par 2 X fois |
| NON (Not) | ~ 5 | Les bits qui sont positionnés à 0 sont positionnés à 1 et inversement |

## Opérateurs combinés

> Les opérateurs combinés permettent d'associer l'opérateur d'affectation aux opérateurs arithmétiques

| Opérateur | Opération |
|---|---|
| += | $a += 2; |

| Opérateur | Opération |
|---|---|
| -= | $a -= 2; |
| *= | $a *= 4; |
| **= | $a **= 2; |
| %= | $a %= 5; |
| /= | $a /= 5; |
| .= | $b .= 'chaine 2'; |

### Opérateurs de comparaison

> Ils retournent `true` si la comparaison est vérifiée, `false` sinon.

| Opérateur | Siginification | Exemple |
|---|---|---|
| == | Egal à | $a == $b |
| < | Inférieur à | $a < $b |
| > | Supérieur à | $a > $b |
| <= | Inférieur ou égal à | $a <= $b |
| >= | Supérieur ou égal à | $a >= $b |
| <=> | Supérieur (renvoie 1), inférieur (renvoie -1), égal (renvoie 0) | $a <=> $b |
| != | Différent (NOT =) | $a != $b |
| === | Indentique (type et valeur) | $a === $b |
| !== | Non identique en type et en valeur | $a !== $b |

### Opérateurs logiques

| Opérateur | Exemple | Description |
|---|---|---|
| ET | $a && $b, $a AND $b | Renvoie `true` si $a **ET** $b valent `true` |
| OU | $a \|\| $b, $a OR $b | Renvoie `true` si $a **OU** $b vaut `true` |
| OU exclusif | $a XOR $b | Renvoie `true` si $a **OU** $b vaut `true` (mais pas les deux) |
| NON | !$a | Renvoie `true` si $a vaut `false` et réciproquement |

## 2.2 Les structures de contrôle

> Les structures de contrôle permettent de **répéter des actions** ou de soumettre certaines exécutions à des **conditions**.

Elles fonctionnent pour la plupart à partir de l'évaluation d'une expression. Cette évaluation doit rendre une valeur comprise comme un booléen.

On utilisera la plupart du temps les opérateurs logiques et de comparaison. Il est également possible d'avoir une expression complexe comprenant des appels de fonctions et des affectations de variables.

## Les conditions

> Les conditions permettent de déterminer si une action (ou un ensemble d'actions) doit être exécutée ou non.

Exemple :

- si le visiteur a entre 18 et 34 ans, je lui assigne le profil "jeune adulte"
- si le visiteur a plus de 65 ans, je lui assigne le profil "senior"

### L'instruction `if`

> L'instruction `if` permet d'exécuter une suite d'instructions en fixant une condition.

```
if(expression) {
    // instructions
}
```

Exemple :

```
if( $a < 10) {
    echo "a est plus petit que 10";
}
```

### La clause `else`

> La clause `else` permet de spécifier une suite d'instructions lorsque la condition n'est pas vérifiée

```
if(expression) {
    // instructions si la condition est vérifiée
} else {
    // instructions si la condition n'est pas vérifiée
}
```

Exemple :

```
if( $a < 10) {
    echo "a est plus petit que 10";
} else {
    echo "a est égal à 10 ou plus grand que 10";
}
```

### L'instruction `elseif`

> L'instruction `elseif` permet d'enchaîner une série de `if` sans avoir à les imbriquer

```
if(expression1) {
    // instructions si l'évaluation de expression1 retourne true
} elseif(expression2) {
    // instructions si l'évaluation de expression2 retourne true
} elseif(expression3) {
    //instructions si l'évaluation de expression3 retourne true
} else {
    //instructions si les conditions ne sont pas vérifiées
}
```

Remarque : `elseif`peut également s'écrire `else if`

### Les accolades dans les conditions

> Les accolades délimitent les blocs d'instructions.

Il est possible de s'en passer lorsque le bloc d'instructions ne contient qu'une instruction (si les accolades ne sont pas placées, seule la première instruction est exécutée dans le cadre du test).

Attention : cette écriture peut être source d'erreurs.

### L'instruction `switch`

> L'instruction switch permet d'effectuer plusieurs tests sur la valeur d'une variable.

Elle évite d'imbriquer des `if` et simplifie la lecture du code :

```
switch($variable) {
    case valeur1 :
        // instructions
        break;
    case valeur2 :
        // instructions
        break;
    case valeur3 :
        // instructions
        break;
    default :
        //instructions
}
```

## Les boucles

> Les boucles permettent de répéter l'exécution d'une instruction ou d'un bloc d'instructions si une condition est vérifiée (ou plusieurs).

**L'instruction `while`**

> L'instruction `while` correspond à une boucle **tant que**

On pourra exécuter des instructions tant que la condition est vérifiée

```
while(condition) {
    // instructions
}
```

Attention : Le bloc d'instructions doit impérativement contenir du code susceptible d'influer sur la condition sous peine de créer une boucle infinie.

**L'instruction `do ... while`**

```
do {
    // instructions
} while(condition1;
```

Attention : La condition n'est testée qu'après l'exécution du bloc d'instructions => le bloc sera exécuté au moins une fois

**L'instruction `for`**

> L'instruction `for` est le plus souvent utilisée lorsqu'il est possible de quantifier le nombre d'itérations

```
for(expression1; condition; expression2) {
    //instructions
}
```

- **expression1** est exécutée une seule fois lors de l'entrée dans la boucle,
- **condition1** est testée à chaque passage dans la boucle,
- **expression2** est exécutée à la fin de chaque passage dans la boucle

```
for( $i=1; $i < 10; $i++) {
    // instructions
};
```

**L'instruction `foreach`**

> L'instruction `foreach` constitue un moyen simple de parcourir un tableau

```
// Permet de parcourir les valeurs du tableau
foreach($tableau as $valeur) {
   // instructions
}

// Permet de parcourir les paires clé/valeur du tableau
foreach($tableau as $cle => $valeur) {
   // instructions
}
```

Remarque : L'instruction `foreach` travaille sur une copie du tableau spécifié et non sur le tableau lui-même. Les modifications ne seront donc pas prises en compte pour l'exécution de la boucle elle-même.

```
$tableau = ['a', 'b', 'c'];

foreach ($tableau as $cle => $valeur) {
    echo $cle . '/' . $valeur . '<br/>';
    $tableau = [];
}

var_dump($tableau);
```

**L'instruction `foreach` (utilisation par référence) :**

> Il est possible de travailler avec le tableau sur les valeurs par référence et non par copie.

```
foreach($tableau as &$valeur) {
    // instructions
}

foreach($tableau as $cle=>&$valeur) {
    // instructions
}
```

Attention : Si une valeur est modifiée, elle l'est dans le tableau original.

**Les instructions d'arrêt : `break`**

> L'instruction break entraîne la sortie d'une structure conditionnelle ( for, while, foreach, switch).

Elle accepte un paramètre optionnel qui indique le nombre de niveaux de structures emboitées qui doivent être interrompus (1 par défaut).

```php
for($i = 0; $i <10; $i++) {
    // instructions
    if( $i == 5) break;
}
```

**Les instructions d'arrêt : `continue`**

> L'instruction continue entraîne la sortie de l'itération en cours et le passage à l'itération suivante.

Les instructions comprises entre `continue` et la fin de la boucle ne sont pas exécutées pour cette itération.

```php
for($i = 0; $i <10; $i++) {
    //instructions
    if( $i == 5) continue;
    //instructions non exécutées si i=5
}
```

## 2.3 - Inclusion de fichiers

> PHP propose deux fonctions très simples pour réutiliser du code (stocké dans un autre fichier) : `require` et `include`

`include` : le fichier à inclure est inséré. Une erreur de niveau Warning est générée en cas d'erreur

`require` : le fichier à inclure est inséré. Une erreur fatale est générée en cas d'erreur, ce qui arrêtera le script

```php
include 'chemin/du/fichier/nom_de_fichier.php';
require 'chemin/du/fichier/nom_de_fichier.php';
```

`include_once`, `require_once` : permet de n'inclure le fichier qu'une seule fois. Si le même fichier a déjà été inséré, l'instruction n'a aucun effet.

Par défaut, ces fonctions retournent `1` si l'inclusion s'est bien déroulée. Toutefois si le fichier inclus retourne une valeur, celle-ci sera considérée comme valeur de retour.

Attention : La référence en terme de chemin (`./`) reste le script appelant.