Info 303

Introduction au PHP

Alin F. Rabat C.

Université de Reims Champagne Ardenne

6 septembre 2019

Sommaire

PHP est un langage de script généraliste et Open Source, conçu pour le développement d'applications web. PHP est l'acronyme de *PHP : Hypertext Processor*. Les scripts PHP sont exécutés côté serveur.
Un fichier PHP dispose de l'extention *.php*. Il peut contenir du texte, des balises HTML du CSS, du JavaScript, et naturellement du PHP.

Listing 1 – Hello world !

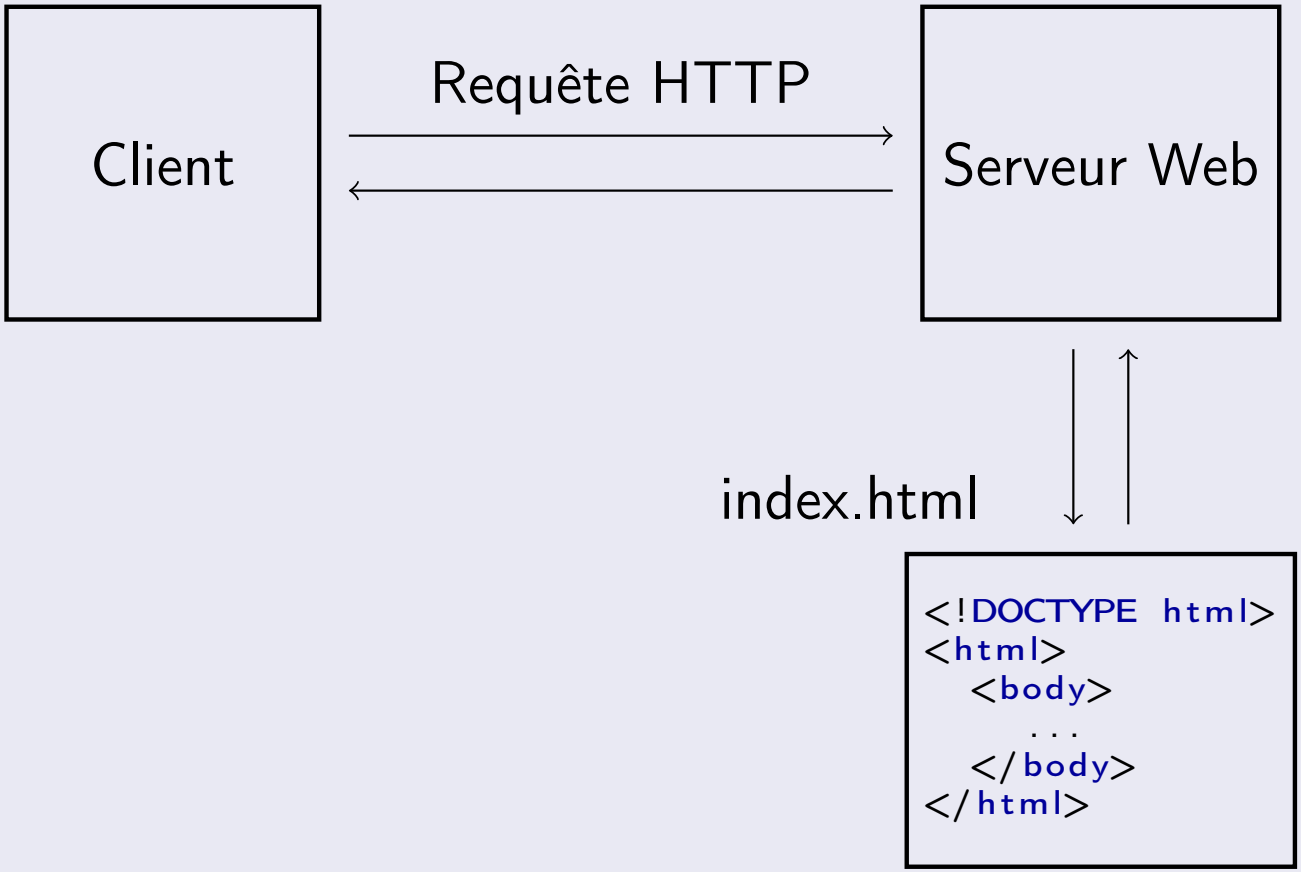
```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Exemple</title>
  </head>
  <body>
    <?php
      echo "Hello World !";
    ?>
  </body>
</html>
```

Avec PHP vous allez pouvoir :

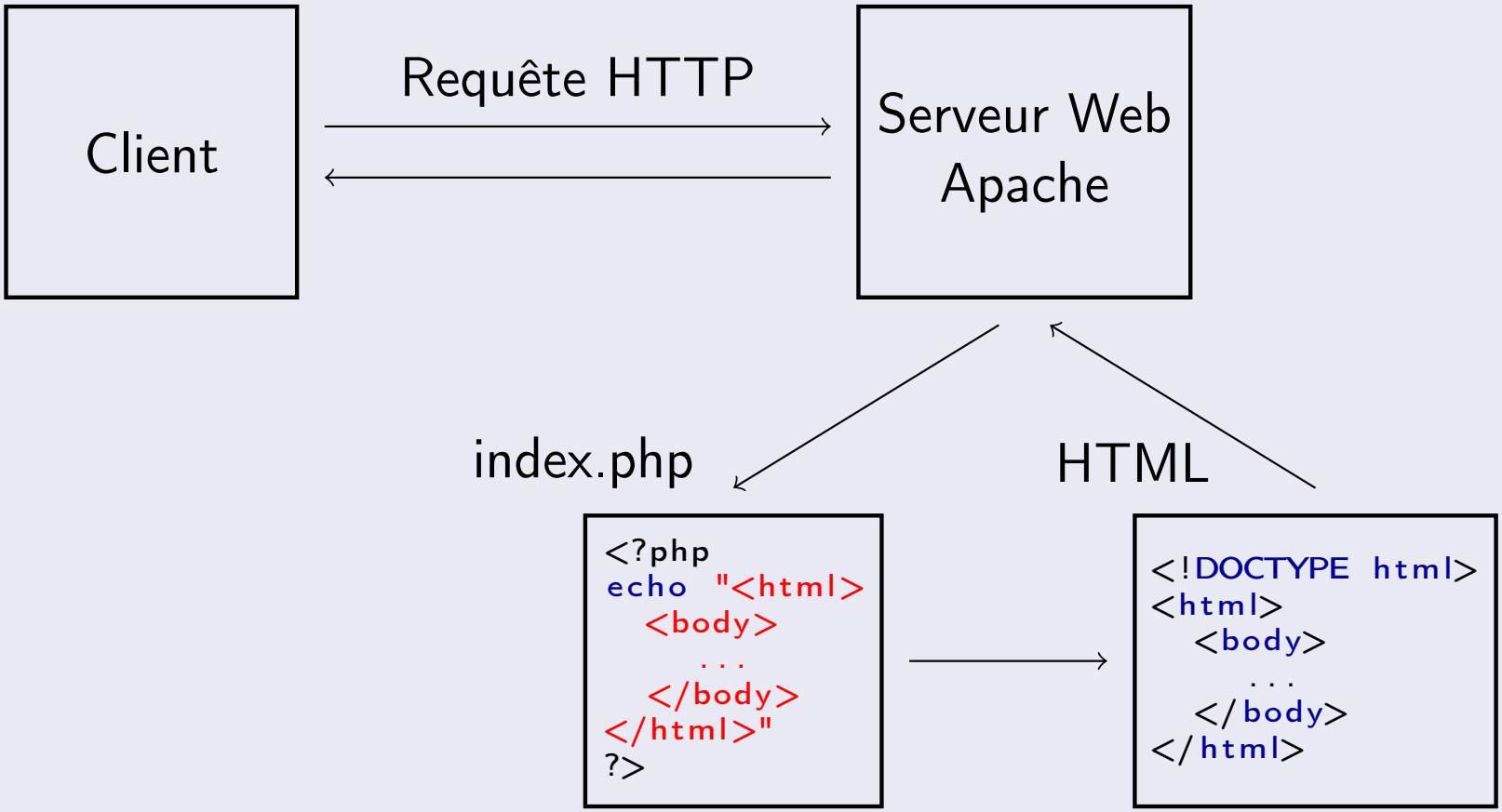
- Générer des contenus de page dynamiquement ;
- Créer, ouvrir, lire, écrire, effacer et fermer des fichiers sur le serveur ;
- Récupérer les données issues de formulaires ;
- Envoyer ou recevoir des cookies ;
- Interagir avec des bases de données ;
- Gérer le contrôle d'accès à vos pages ;
- Faire du cryptage de données ;
- Et bien d'autres choses encore.

PHP est disponible sur l'essentiel des plateformes actuelles (Windows, Linux, Unix, Mac OS X, ...). Il est disponible sur les serveurs Apache, IIS, ... et permet d'accéder à de nombreuses bases de données. Vous trouverez des ressources utiles sur le site officiel `www.php.net`

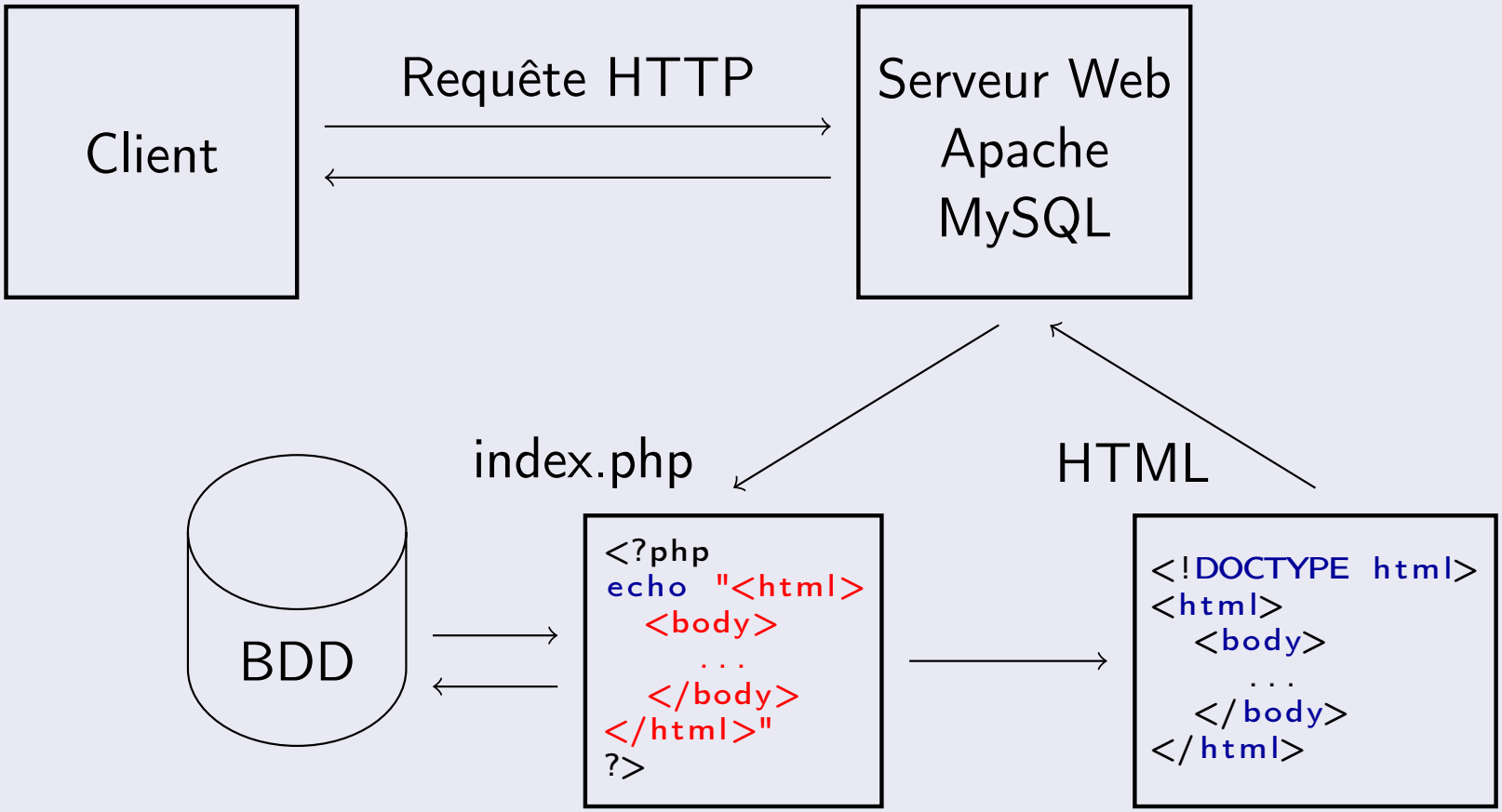
Page HTML Statique



Page dynamique PHP



Page dynamique PHP



Script PHP

- Par défaut un document contenant un script PHP utilise l'extension *.php* .
- Si l'on demande à un serveur un fichier présentant cette extension, il transfère automatiquement le fichier au processeur PHP.
- Le comportement du serveur web étant entièrement paramétrable il est possible de le configurer afin d'imposer par exemple que les fichiers d'extension *.htm* ou *.html* soient également transmis au processeur PHP.
- Le processeur PHP se charge alors de générer un fichier ne contenant que du code HTML susceptible d'être analysé par un navigateur.

Pour déclencher l'analyse du code il faut insérer à l'intérieur de la balise :

Script PHP

```
<?php
        // Votre code php
?>
```

Un script PHP peut être placé n'importe où dans un document.

Tout ce qui se trouve en dehors d'une paire de balises ouvrantes/fermantes est ignoré par l'analyseur PHP , ce qui permet d'avoir des fichiers PHP mixant les contenus. Ceci permet à PHP d'être contenu dans des documents HTML.

```
<p>Ceci sera ignore par PHP et affiche au navigateur.</p>
<?php echo 'Alors que ceci sera analyse par PHP.'; ?>
<p>Ceci sera aussi ignore par PHP et affiche au navigateur.</p>
```

PHP est très proche dans sa syntaxe des langages Java ou C++ par exemple. Ainsi la forme des commentaires est la même. Vous avez la possibilité d'écrire vos commentaires en ligne :

```
// Un commentaire sur la ligne entiere
echo "Hello World!"; // Un commentaire en fin de ligne
```

Vous pouvez aussi les construire sur plusieurs lignes :

```
/* Ceci est un commentaire
reparti sur plusieurs ligne */
```

Attention, il n'est pas possible d'imbriquer les commentaires.
En PHP toute instruction se termine par un point virgule. L'oubli de ce point virgule entraîne un message d'erreur de type *parse error*.

Le nom des variables PHP débute obligatoirement par le signe dollar $ suivi du nom de la variable. Le nom est sensible à la casse. Un nom de variable valide doit commencer par une lettre ou un caractère (_), suivi de lettres, chiffres ou soulignés. Exprimé sous la forme d'une expression régulière, cela donne :

```
[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*
```

PHP supporte 10 types basiques, 4 types scalaires, 4 types composés et 2 types spéciaux :

- boolean, integer, float, string ;
- array, object, callable, iterable,
- resource, null.

Listing 2 – Déclaration de variables

```php
<?php
$a_bool = TRUE;   // un booleen
$a_str  = "foo";  // une chaine de caracteres
$a_str2 = 'foo';  // une chaine de caracteres
$an_int = 12;     // un entier

echo gettype($a_bool); // affiche :  boolean
echo gettype($a_str);  // affiche :  string

// Si c'est un entier, increment de 4
if (is_int($an_int)) {
    $an_int += 4;
}

// Si $a_bool est une chaine de caracteres, on l'affiche
if (is_string($a_bool)) {
    echo "String: $a_bool";
}
?>
```

Remarque :

Pour vérifier le type et la valeur d'une expression, utilisez la fonction `var_dump()`. Pour afficher une représentation humainement lisible d'un type aux fins de déboguage, utilisez la fonction `gettype()`. Pour vérifier un certain type, n'utilisez pas la fonction `gettype()`, mais plutôt les fonctions `is_type()`.

C'est le type le plus simple. Un booléen représente une valeur de vérité. Il peut valoir `TRUE` ou `FALSE`. Pour spécifier un booléen littéral, utilisez la constante `TRUE` ou `FALSE`. Les deux sont insensibles à la casse.

```
<?php
$foo = true; // assigne la valeur TRUE a $foo
?>
```

Les entiers peuvent être spécifiés en notation décimale (base 10), hexadécimale (base 16), octale (base 8), ou binaire (base 2) optionnellement précédée d'un signe (- ou +).
Pour utiliser la notation octale, précédez le nombre d'un 0 (zéro).
Pour utiliser la notation hexadécimale, précédez le nombre d'un 0x.
Pour utiliser la notation binaire, précédez le nombre d'un 0b.

Listing 3 – Déclaration d'un entier

```php
<?php
$a = 1234; // un nombre decimal
$a = -123; // un nombre negatif
$a = 0123; // un nombre octal (equivalent a 83)
$a = 0x1A; // un nombre hexadecimal (equivalent a 26)
$a = 0b11111111; // un nombre binaire (equivalent a 255)
?>
```

La taille d'un entier est dépendante de la plate-forme, cependant, une valeur maximale d'environ 2 milliards est habituelle (cela correspond à 32 bits signés). Les plateformes 64-bit ont habituellement une valeur maximale d'environ 9E18, sauf pour Windows avant PHP 7, qui est toujours en 32 bits.
PHP ne supporte pas les entiers non signés. La taille d'un entier peut être déterminée en utilisant la constante `PHP_INT_SIZE`, la valeur maximale, en utilisant la constante `PHP_INT_MAX` depuis PHP 5.0.5, et la valeur minimale en utilisant la constante `PHP_INT_MIN` depuis PHP 7.0.0.

Si PHP rencontre un nombre supérieur au maximal d'un entier, il sera interprété comme un nombre décimal. De la même façon, une opération qui résulte en un nombre supérieur au nombre maximal d'un entier, retournera un nombre décimal.

Les nombres réels (ou décimaux) sont définis en utilisant la syntaxe suivante :

Listing 4 – Déclaration d'un réel

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

- Une constante est un identifiant qui représente une valeur simple.
- Cette valeur ne peut jamais être modifiée durant l'exécution du script (sauf les constantes magiques).
- Par défaut, le nom d'une constante est sensible à la casse.
- Par convention, les constantes sont toujours en majuscules.

Les noms de constantes suivent les mêmes règles que n'importe quel nom en PHP. Un nom de constante valide commence par une lettre ou un souligné, suivi d'un nombre quelconque de lettres, chiffres ou soulignés. Sous forme d'expression régulière, cela peut s'exprimer comme ceci :

```
[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*
```

On peut définir une constante en utilisant la fonction `define()` ou en utilisant le mot-clé `const` en dehors d'une définition de classe. Une fois qu'une constante est définie, elle ne peut jamais être modifiée, ou détruite.

Exemples :

Listing 5 – Définir une constante

```php
<?php
  define("CONSTANTE", "Bonjour le monde.");
  echo CONSTANTE; // affiche "Bonjour le monde."
  echo Constante; // affiche "Constante" et genere une exception.
?>
```

Listing 6 – Définir des constantes en utilisant le mot-clé const

```php
<?php
// Fonctionne depuis PHP 5.3.0
const CONSTANT = 'Bonjour le monde !';
echo CONSTANT;
// Fonctionne depuis PHP 5.6.0
const ANOTHER_CONST = CONSTANT.'; Au revoir le monde !';
echo ANOTHER_CONST;
const ANIMALS = array('chien', 'chat', 'oiseaux');
echo ANIMALS[1]; // affiche "chat"
// Fonctione depuis PHP 7
define('ANIMALS', array(
    'chien',
    'chat',
    'oiseaux'
));
echo ANIMALS[1]; // affiche "chat"
?>
```

Quand vous utilisez des mots-clés `const`, seuls les types de données scalaires (boolean, integer, float et string) peuvent être utilisés. Il est possible de définir une constante à l'aide d'une expression scalaire. Il est également possible de définir une constante de type tableau.
Vous pouvez accéder à la valeur d'une constante en spécifiant simplement son nom. Contrairement aux variables, vous ne devez pas préfixer le nom de la constante avec $. Vous pouvez aussi utiliser la fonction `constant()`, pour lire dynamiquement la valeur d'une constante, dont vous obtenez le nom dynamiquement (retour de fonction, par exemple). Utilisez la fonction `get_defined_constants()` pour connaître la liste de toutes les constantes définies.

Il y a des différences entre les constantes et les variables :

- Les constantes ne commencent pas par le signe $.
- Les constantes sont définies et accessibles à tout endroit du code, globalement.
- Les constantes ne peuvent pas être redéfinies ou indéfinies une fois qu'elles ont été définies.
- Les constantes ne peuvent contenir que des scalaires. Il est possible de définir des constantes de tableaux en utilisant le mot clé `const` et les constantes de tableaux peuvent aussi être définies en utilisant `define()`. Vous pouvez utiliser des tableaux dans les expressions scalaires des constantes (par exemple, `const FOO = array(1,2,3)[0];`, mais le résultat final doit être une valeur scalaire de type autorisé.

PHP fournit un grand nombre de constantes magiques. Certaines constantes sont définies par différentes extensions, et ne seront présentes que si ces extensions sont compilées avec PHP, ou bien si l'extension a été chargée dynamiquement.
Il y a neuf constantes magiques qui changent suivant l'emplacement où elles sont utilisées. Par exemple, la valeur de `__LINE__` dépend de la ligne où vous l'utilisez dans votre script. Toutes ces constantes "magiques" sont évaluées au moment de la compilation, contrairement aux constantes classiques, qui sont évaluées au moment de l'exécution. Ces constantes spéciales sont insensibles à la casse.

Table – Quelques constantes PHP magiques

Nom	Description
`__LINE__`	La ligne courante dans le fichier
`__FILE__`	Le chemin complet et le nom du fichier courant avec les liens symboliques résolus. Si utilisé pour une inclusion, le nom du fichier inclus est retourné.
`__DIR__`	Le dossier du fichier. Si utilisé dans une inclusion, le dossier du fichier inclus sera retourné. C'est l'équivalent de `dirname(__FILE__)`. Ce nom de dossier ne contiendra pas de slash final, sauf si c'est le dossier racine.
`__FUNCTION__`	Le nom de la fonction.
`__CLASS__`	Le nom de la classe courante. Le nom de la classe contient l'espace de nom dans lequel cette classe a été déclarée (i.e. `Foo\Bar`). `__CLASS__` fonctionne aussi dans les traits. Lorsqu'elle est utilisée dans une méthode de trait, `__CLASS__` est le nom de la classe dans laquelle le trait est utilisé.
`__TRAIT__`	Le nom du trait. Le nom du trait inclut l'espace de nom dans lequel il a été déclaré (e.g. `Foo\Bar`).
`__METHOD__`	Le nom de la méthode courante.
`__NAMESPACE__`	Le nom de l'espace de noms courant.
`ClassName::class`	Le nom entièrement qualifié de la classe.

Le tableau suivant résume les opérateurs arithmétiques disponibles PHP :

Exemple	Nom	Résultat
+$a	Identité	Conversion de $a vers int ou float, selon le plus approprié.
-$a	Négation	Opposé de $a.
$a + $b	Addition	Somme de $a et $b.
$a - $b	Soustraction	Différence de $a et $b.
$a * $b	Multiplication	Produit de $a et $b.
$a / $b	Division	Quotient de $a et $b.
$a % $b	Modulus	Reste de $a divisé par $b.
$a ** $b	Exponentielle	Résultat de l'élévation de $a à la puissance $b.

Table – Opérateurs arithmétiques

L'opérateur de division `/` retourne une valeur à virgule flottante sauf si les 2 opérandes sont des entiers (ou une chaîne de caractères qui a été convertie en entiers) et que leur division est exacte (i.e. a pour reste 0), auquel cas une valeur entière sera retournée. Pour la division entière, voir `intdiv()`.
Les opérandes du modulo sont converties en entiers (en supprimant la partie décimale) avant exécution. Pour le modulo sur des nombres décimaux, voir `fmod()`.
Le résultat de l'opération modulo `%` a le même signe que le premier opérande, ansi le résultat de `$a % $b` aura le signe de `$a`.

Exemple :

```php
<?php
echo (5 % 3)."\n";          // affiche 2
echo (5 % -3)."\n";         // affiche 2
echo (-5 % 3)."\n";         // affiche -2
echo (-5 % -3)."\n";        // affiche -2
?>
```

PHP supporte les opérateurs de pré- et post-incrémentation et décrémentation, comme en langage C.

Les opérateurs d'incrémentation/décrémentation n'affectent que les nombres et les chaînes de caractères. Les tableaux, objets et ressources ne sont pas affectés. La décrémentation des valeurs NULL n'a également aucun effet, mais leur incrémentation donnera comme résultat 1.

Exemple	Nom	Résultat
++$a	Pre-incrémente	Incrémente $a de 1, puis retourne $a.
$a++	Post-incrémente	Retourne $a, puis incrémente $a de 1.
--$a	Pré-décrémente	Décrémente $a de 1, puis retourne $a.
$a--	Post-décrémente	Retourne $a, puis décrémente $a de 1.

Table – Opérateurs d'incrémentation et décrémentation

PHP utilise indifféremment le symbole ou le nom de l'opérateur pour les opérateurs logiques. Les symboles sont les mêmes que ceux utilisés dans le langage C.

Exemple	Nom	Résultat
`$a and $b`	And (Et)	TRUE si `$a` ET `$b` valent TRUE.
`$a or $b`	Or (Ou)	TRUE si `$a` OU `$b` valent TRUE.
`$a xor $b`	XOR	TRUE si `$a` OU `$b` est TRUE, mais pas les deux en même temps.
`!$a`	Not (Non)	TRUE si `$a` n'est pas TRUE.
`$a && $b`	And (Et)	TRUE si `$a` ET `$b` sont TRUE.
`$a || $b`	Or (Ou)	TRUE si `$a` OU `$b` est TRUE.

Table – Les opérateurs logiques

Les opérateurs sur les bits permettent de manipuler les bits dans un entier.

Exemple	Nom	Résultat
`$a & $b`	And (Et)	Les bits positionnés à 1 dans `$a` ET dans `$b` sont positionnés à 1.
`$a | $b`	Or (Ou)	Les bits positionnés à 1 dans `$a` OU `$b` sont positionnés à 1.
`$a ^ $b`	Xor (ou exclusif)	Les bits positionnés à 1 dans `$a` OU dans `$b` mais pas dans les deux sont positionnés à 1.
`~ $a`	Not (Non)	Les bits qui sont positionnés à 1 dans `$a` sont positionnés à 0, et vice-versa.
`$a << $b`	Décalage à gauche	Décale les bits de `$a`, `$b` fois sur la gauche (chaque décalage équivaut à une multiplication par 2).
`$a >> $b`	Décalage à droite	Décale les bits de `$a`, `$b` fois par la droite (chaque décalage équivaut à une division par 2).

Table – Les opérateurs sur les bits

Les opérateurs de comparaison permettent de comparer deux valeurs. Le tableau suivant résume les différents tests possibles :

Exemple	Nom	Résultat
`$a == $b`	Egal	TRUE si `$a` est égal à `$b` après le transtypage.
`$a === $b`	Identique	TRUE si `$a` est égal à `$b` et qu'ils sont de même type.
`$a != $b`	Différent	TRUE si `$a` est différent de `$b` après le transtypage.
`$a <> $b`	Différent	TRUE si `$a` est différent de `$b` après le transtypage.
`$a !== $b`	Différent	TRUE si `$a` est différent de `$b` ou bien s'ils ne sont pas du même type.
`$a < $b`	Plus petit que	TRUE si `$a` est strictement plus petit que `$b`.
`$a > $b`	Plus grand	TRUE si `$a` est strictement plus grand que `$b`.
`$a <= $b`	Inférieur ou égal	TRUE si `$a` est plus petit ou égal à `$b`.
`$a >= $b`	Supérieur ou égal	TRUE si `$a` est plus grand ou égal à `$b`.
`$a <=> $b`	Combiné	Un entier inférieur, égal ou supérieur à zéro lorsque `$a` est respectivement inférieur, égal, ou supérieur à `$b`.

Table – Opérateurs de comparaison

Opérateur ternaire :

Il existe un autre opérateur conditionnel qui est l'opérateur ternaire.

Syntaxe :

```
expr1 ? expr2 : expr3
```

PHP propose l'opérateur " ??" (ou fusion null).
L'expression `(expr1) ?? (expr2)` retourne expr2 si expr1 est NULL, et expr1 dans les autres cas.

Listing 7 – Assigner une valeur par défaut

```php
<?php
// Exemple d'utilisation pour: Operateur de fusion Null
$action = $_POST['action'] ?? 'default';

// le code ci-dessus est equivalent a cette structure if/else
if (isset($_POST['action'])) {
    $action = $_POST['action'];
} else {
    $action = 'default';
}
```

If

L'instruction `if` permet l'exécution conditionnelle d'une partie de code. Les fonctionnalités de l'instruction `if` sont les mêmes en PHP qu'en C :

```
if (expression)
  commandes
```

L'exemple suivant affiche la phrase a est plus grand que b si `$a` est plus grand que `$b` :

```
<?php
if ($a > $b)
  echo "a est plus grand que b";
?>
```

If

L'exemple suivant affiche a est plus grand que b, si `$a` est plus grand que `$b`, puis assigne la valeur de `$a` à la variable `$b`

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

Il est possible d'imbriquer indéfiniment des instructions `if` dans d'autres instructions `if`, ce qui permet une grande flexibilité dans l'exécution d'une partie de code suivant un grand nombre de conditions.

else

Dans l'exemple suivant, ce bout de code affiche a est plus grand que b si la variable `$a` est plus grande que la variable `$b`, et a est plus petit que b sinon :

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

Les instructions après le else ne sont exécutées que si l'expression du `if` est FALSE

elseif ou else if

`elseif` est une combinaison de `if` et de `else`. Comme l'expression `else`, il permet d'exécuter une instruction après un `if` dans le cas où le "premier" `if` est évalué comme FALSE. Mais, à la différence de l'expression `else`, il n'exécutera l'instruction que si l'expression conditionnelle `elseif` est évaluée comme TRUE. L'exemple suivant affichera a est plus grand que b, a est égal à b ou a est plus petit que b :

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

while

La boucle while est le moyen le plus simple d'implémenter une boucle en PHP. Cette boucle se comporte de la même manière qu'en C. L'exemple le plus simple d'une boucle while est le suivant :

```
while (expression)
    commandes
```

while

Comme avec le `if`, vous pouvez regrouper plusieurs instructions dans la même boucle `while` en les regroupant à l'intérieur d'accolades ou en utilisant la syntaxe suivante :

```
while (expression):
    commandes
    ...
endwhile;
```

Les exemples suivants sont identiques et affichent tous les nombres de 1 jusqu'à 10 :

```php
<?php
/* exemple 1 */

$i = 1;
while ($i <= 10) {
    echo $i++;  /* La valeur affichee est $i avant l'incrementation
                   (post-incrementation)  */
}

/* exemple 2 */

$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
?>
```

do while

Les boucles `do while` ressemblent aux boucles `while`, mais l'expression est testée à la fin de chaque itération plutôt qu'au début.
Il n'y a qu'une syntaxe possible pour les boucles `do while` :

```php
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```

La boucle ci-dessus ne va être exécutée qu'une seule fois, car lorsque l'expression est évaluée, elle vaut FALSE (car la variable `$i` n'est pas plus grande que 0) et l'exécution de la boucle s'arrête.

Les utilisateurs familiers du C sont habitués à une utilisation différente des boucles `do while` , qui permet de stopper l'exécution de la boucle au milieu des instructions, en encapsulant dans un `do while(0)` la fonction `break`. Le code suivant montre une utilisation possible :

```php
<?php
do {
    if ($i < 5) {
        echo "i n'est pas suffisamment grand";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit) {
        break;
    }
   echo "i est bon";

    /* ...traitement de i... */

} while (0);
?>
```

for

Les boucles for sont les boucles les plus complexes en PHP. Elles fonctionnent comme les boucles for du langage C(C++). La syntaxe des boucles for est la suivante :

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

La première expression `expr1` est évaluée (exécutée), quoi qu'il arrive au début de la boucle.
Au début de chaque itération, l'expression `expr2` est évaluée. Si l'évaluation vaut TRUE, la boucle continue et les commandes sont exécutées. Si l'évaluation vaut FALSE, l'exécution de la boucle s'arrête.
À la fin de chaque itération, l'expression `expr3` est évaluée (exécutée).

Listing 8 – affichage des chiffres de 1 à 10

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

/* exemple 2 */
for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}

/* exemple 3 */
$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}

/* exemple 4 */
for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
```

PHP supporte la syntaxe alternative suivante pour les boucles for :

```
for (expr1; expr2; expr3):
    commandes
    ...
endfor;
```

switch

L'instruction `switch` équivaut à une série d'instructions `if`.
Les deux exemples suivants sont deux manières différentes d'écrire la même chose

```php
<?php
if ($i == 0) {
    echo "i egal 0";
} elseif ($i == 1) {
    echo "i egal 1";
} elseif ($i == 2) {
    echo "i egal 2";
}
```

```php
switch ($i) {
    case 0:
        echo "i egal 0";
        break;
    case 1:
        echo "i egal 1";
        break;
    case 2:
        echo "i egal 2";
        break;
}
?>
```

Listing 9 – Instruction switch utilisant une chaine de caractere

```php
<?php
switch ($i) {
    case "apple":
        echo "i est une pomme";
        break;
    case "bar":
        echo "i est une barre";
        break;
    case "cake":
        echo "i est un gateau";
        break;
}
?>
```

La liste de commandes d'un case peut être vide, auquel cas PHP utilisera la liste de commandes du cas suivant.

```php
<?php
switch ($i) {
    case 0:
    case 1:
    case 2:
        echo "i est plus petit que 3 mais n'est pas negatif";
        break;
    case 3:
        echo "i egal 3";
}
?>
```

Un cas spécial est `default`. Ce cas est utilisé lorsque tous les autres cas ont échoué. Par exemple :

```php
<?php
switch ($i) {
    case 0:
        echo "i egal 0";
        break;
    case 1:
        echo "i egal 1";
        break;
    case 2:
        echo "i egal 2";
        break;
    default:
       echo "i n'est ni egal a 2, ni a 1, ni a 0.";
}
?>
```

La syntaxe alternative pour cette structure de contrôle est la suivante :

```php
<?php
switch ($i):
    case 0:
        echo "i egal 0";
        break;
    case 1:
        echo "i egal 1";
        break;
    case 2:
        echo "i egal 2";
        break;
    default:
        echo "i n'est ni egal a 2, ni a 1, ni a 0";
endswitch;
?>
```

Sommaire

Un tableau en PHP est en fait une map ordonnée. Une map est un type qui associe des valeurs à des clés. On peut avoir, comme valeur d'un tableau, d'autres tableaux, multidimensionnels ou non.

Pour créer un tableau on utilise la commande `array()`. Elle prend un nombre illimité de paramètres, chacun séparé par une virgule, sous la forme d'une paire key => value.

```
array(
    key1  => value1,
    key2 => value2,
    key3 => value3,
    ...
)
```

La virgule après le dernier élément d'un tableau est optionnelle et peut ne pas être ajoutée.

Il est également possible d'utiliser la syntaxe courte `[]`.

Listing 10 – Tableau simple

```php
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

- La clé key est un entier, ou une chaîne de caractères. La valeur value peut être de n'importe quel type.
- Si plusieurs éléments dans la déclaration d'un tableau utilisent la même clé, seule la dernière sera utilisée, écrasant ainsi toutes les précédentes.

PHP effectue les modifications de type suivant pour la clé key :

- Les chaînes de caractères contenant un entier valide, sauf si le nombre est précédé d'un signe + seront modifiées en un type entier. I.e. la clé "8" sera actuellement stockée comme l'entier 8. D'un autre côté, "08" ne sera pas modifié, sachant que ce n'est pas un entier décimal valide.
- Les nombres à virgule flottante seront aussi modifiés en entier, ce qui signifie que la partie après la virgule sera tronquée. I.e. la clé 8.7 sera stockée sous l'entier 8.
- Les booléens seront modifiés en entier également, i.e. la clé true sera stockée sous l'entier 1 et la clé false sous l'entier 0.
- La valeur Null sera modifiée en une chaîne vide, i.e. la clé null sera stockée sous la chaîne de caractère "".
- Les tableaux et les objets ne peuvent pas être utilisés comme

La clé key est optionnelle. Si elle n'est pas spécifiée, PHP utilisera un incrément de la dernière clé entière utilisée.

Listing 11 – Indexation automatique des tableaux

```php
<?php
$array = array("foo", "bar", "hello", "world");
?>
```

Pour accéder aux éléments d'un tableau on utilise la syntaxe à base de crochets. Les éléments d'un tableau sont accessibles en utilisant la syntaxe `array[key]`.

Listing 12 – Accès aux éléments d'un tableau

```php
<?php
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
         "dimensional" => array(
             "array" => "foo"
         )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
?>
```

Un tableau existant peut être modifié en y assignant explicitement des valeurs. L'assignation d'une valeur dans un tableau est effectuée en spécifiant la clé, entre crochets. Si aucune clé n'est fournie alors PHP attribue automatique comme clé le premier entier disponible.

```
$arr[cle] = valeur;
$arr[] = valeur;
// cle peut etre un entier ou une chaine de caracteres
// valeur peut etre n'importe quel type
```

Pour modifier une valeur en particulier, il convient d'assigner une valeur en spécifiant sa clé. Pour effacer une paire clé/valeur, il convient d'appeler la fonction `unset()` sur la clé désirée.

```php
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // Identique a $arr[13] = 56;
                // a cet endroit du script

$arr["x"] = 42; // Ceci ajoute un nouvel element au
                // tableau avec la cle "x"

unset($arr[5]); // Ceci efface l'element du tableau

unset($arr);    // Ceci efface completement le tableau
?>
```

- Si aucune clé n'est spécifiée, l'indice maximal existant est repris, et la nouvelle clé sera ce nombre, plus 1 (mais au moins 0).
- Si aucun indice entier n'existe, la clé sera 0 (zéro).
- Notez que la clé entière maximale pour cette opération n'a pas besoin d'exister dans le tableau au moment de la manipulation. Elle doit seulement avoir existé dans le tableau à un moment ou un autre depuis la dernière fois où le tableau a été ré-indexé.

```php
<?php
// Creation d'un tableau simple.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Maintennant, on efface tous les elements,
// mais on conserve le tableau :
foreach ($array as $i => $value) {
    unset($array[$i]);
}
print_r($array);

// Ajout d'un element
// (notez que la nouvelle cle est 5, et non 0).
$array[] = 6;
print_r($array);

// Re-indexation :
$array = array_values($array);
$array[] = 7;
print_r($array);
?>
```

Sommaire

Les noms de fonctions suivent les mêmes règles que les autres labels en PHP. Un nom de fonction valide commence par une lettre ou un souligné, suivi par un nombre quelconque de lettres, de nombres ou de soulignés. Ces règles peuvent être représentées par l'expression rationnelle suivante :

```
[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*..
```

Listing 13 – Exemple de déclaration d'une fonction

```php
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Exemple de fonction.\n";
    return $retval;
}
?>
```

Les fonctions n'ont pas besoin d'être définies avant d'être utilisées. Par contre lorsqu'une fonction est définie de manière conditionnelle,sa définition doit nécessairement précéder son utilisation.

Toutes les fonctions et classes en PHP ont une portée globale - elles peuvent être appelées à l'extérieur d'une fonction si elles ont été définies à l'intérieur et vice-versa.

PHP ne supporte pas la surchage, la destruction et la redéfinition de fonctions déjà déclarées.

Les noms de fonctions sont insensibles à la casse, et il est généralement admis que les fonctions doivent être appelées avec le nom utilisé dans leur déclaration, y compris la casse.

Il est possible d'appeler des fonctions récursives en PHP.

Exemple :

```php
<?php
function recursion($a)
{
    if ($a < 20) {
        echo "$a\n";
        recursion($a + 1);
    }
}
?>
```

Les paramètres sont passés à une fonction en utilisant une liste d'arguments, dont chaque expression est séparée par une virgule.
Les arguments sont évalués de la gauche vers la droite.
PHP supporte le passage d'arguments par valeur (comportement par défaut). Il est aussi possible de passer des arguments par référence. On peut définir des valeurs d'arguments par défaut. Enfin une fonction peut accepter liste variable d'arguments.

Par défaut, un argument est passé à la fonction par valeur, c'est à dire que tout changement de la valeur du paramètre à l'intérieur de la fonction ne modifie pas le contenu de la variable passée comme paramètre . Si l'on souhaite qu'une fonction puisse changer la valeur d'argument, il faut passer l'argument par référence.
Pour passer un argument par référence, on ajoute un '&' devant l'argument dans la déclaration de la fonction :

Listing 14 – Passage de paramètre par référence

```php
<?php
function add_some_extra(&$string)
{
    $string .= ', et un peu plus.';
}
$str = 'Ceci est une chaine';
add_some_extra($str);
echo $str; // Affiche : 'Ceci est une chaine, et un peu plus.'
?>
```

Il est possible de définir des valeurs par défaut pour les arguments de type scalaire :

Listing 15 – Valeur par défaut des arguments de fonctions

```php
<?php
function servir_cafe ($type = "cappuccino")
{
    return "Servir un $type.\n";
}
echo servir_cafe();
echo servir_cafe(null);
echo servir_cafe("espresso");
?>
```

Dans ce cas si le paramètre est omis, c'est la valeur par défaut qui est utilisée.

Il est possible d'utiliser des tableaux ainsi que le type spécial NULL comme valeur de paramètre par défaut.

Listing 16 – Utilisation de type non scalaire comme valeur par défaut

```php
<?php
function servir_cafe($types = array("cappuccino"),
         $coffeeMaker = NULL)
{
    $device = is_null($coffeeMaker) ? "les mains" : $coffeeMaker;
    return "Preparation d'une tasse de ".join(", ", $types)." avec $device.\n";
}
echo servir_cafe();
echo servir_cafe(array("cappuccino", "lavazza"), "une cafetiere");
?>
```

La valeur par défaut d'un argument doit obligatoirement être une constante, et ne peut être ni une variable, ni un membre de classe, ni un appel de fonction.

Si l'on utilise simultanément des arguments présentant valeur par défaut et d'autres sans valeur par défaut, les premiers doivent être placés à la suite de tous les paramètres sans valeur par défaut. Sinon, cela ne fonctionnera pas.

Considérons le code suivant :

```php
<?php
function faireunyaourt ($type = "acidophilus", $flavour)
{
    return "Preparer un bol de $type $flavour.\n";
}
// ne fonctionne pas comme voulu
echo faireunyaourt("framboise");
?>
```

Ce code ne fonctionne pas car les paramètres sont affectés aux arguments de gauche à droite. Il n'est donc pas possible de ne pas fournir de valeur à l'argument `$type`. Pour que cela fonctionne il faut la définir la fonction ainsi :

```php
<?php
function faireunyaourt ($flavour, $type = "acidophilus")
{
    return "Preparer un bol de $type $flavour.\n";
}
 // fonctionne comme voulu
echo faireunyaourt ("framboise");
?>
```

Les déclarations de type permettent aux fonctions de requérir que certains paramètres soient d'un certain type lors de l'appel de celles-ci. Si la valeur donnée est d'un type incorrect alors PHP 7 lève une exception **TypeError**.
Pour spécifier une déclaration de type, on précise le type du paramètre avant son nom. La déclaration accepte la valeur NULL si la valeur par défaut du paramètre est définie à NULL.

Type	Description
Nom de la Classe/interface	Le paramètre doit être une instanceof de la classe ou interface donnée.
self	Le paramètre doit être une instanceof de la même classe qui a défini la méthode. Ceci ne peut être utilisé que sur les méthodes des classes et des instances.
array	Le paramètre doit être un array.
callable	Le paramètre doit être un callable valide.
bool	Le paramètre doit être un boolean.
float	Le paramètre doit être un nombre flottant (float).
int	Le paramètre doit être un integer.
string	Le paramètre doit être une string.
iterable	Le paramètre doit être soit un array ou une instanceof Traversable.
object	Le paramètre doit être un object.

Par défaut, PHP convertit les mauvais types vers le type scalaire attendu quand c'est possible. Il est possible d'activer un *typage strict* fichier par fichier. Dans ce mode, seule une variable du type attendu dans la déclaration sera acceptée sinon on obtient une erreur TypeError. La seule exception à cette règle est qu'un entier (integer) peut être passé à une fonction attendant un nombre flottant (float). Les appels aux fonctions depuis des fonctions internes ne seront pas affectés par la déclaration `strict_types`.

Pour activer le typage strict, il faut appeler la fonction `declare` avec le paramètre `strict_types` :

```php
<?php
declare(strict_types=1);

function sum(int $a, int $b) {
    return $a + $b;
}

var_dump(sum(1, 2));
var_dump(sum(1.5, 2.5));
?>
```

Le typage strict s'applique aux appels de fonction effectués depuis l'intérieur d'un fichier dont le typage strict est actif, et non aux fonctions déclarées dans ce fichier. Si un fichier dont le typage strict n'est pas activé effectue un appel à une fonction qui a été définie dans un fichier dont le type strict est actif, la préférence de l'appelant sera respecté, et la valeur sera forcée.

PHP supporte les fonctions à nombre d'arguments variable. Pour cela on utilise le mot clé...
La liste des arguments peut inclure le mot clé ... pour indiquer que cette fonction accepte un nombre variable d'arguments. Les arguments seront passés dans la variable fournie sous forme d'un tableau.

Listing 17 – Nombre de paramètres variables

```php
<?php
function sum(...$numbers) {
    $acc = 0;
    foreach ($numbers as $n) {
        $acc += $n;
    }
    return $acc;
}
echo sum(1, 2, 3, 4);
?>
```

Les valeurs sont renvoyées en utilisant une instruction de retour optionnelle. Tous les types de variables peuvent être renvoyés, tableaux et objets compris. Cela fait que la fonction finit son exécution immédiatement et passe le contrôle à la ligne appelante. Voir return pour plus d'informations.

Si return est omis, c'est la valeur NULL qui est retournée.

Listing 18 – Utilisation de return

```php
<?php
function carre($num)
{
    return $num * $num;
}
echo carre(4); // Affiche '16'
?>
```

Une fonction ne peut pas renvoyer plusieurs valeurs en même temps, mais vous pouvez obtenir le même résultat en renvoyant un tableau.

Listing 19 – Utiliser un tableau pour retourner plusieurs valeurs

```php
<?php
function petit_nombre()
{
    return array (0, 1, 2);
}
list ($zero, $un, $deux) = petit_nombre();
?>
```

Il est possible de retourner la référence à une variable. Pour cela il faut utiliser l'opérateur & à la fois dans la déclaration de la fonction et dans l'assignation de la valeur de retour.

```php
<?php
function &retourne_reference()
{
    return $uneref;
}

$newref =& retourne_reference();
?>
```

- PHP 7 ajoute le support des déclarations du type de retour. Les mêmes types sont disponibles pour le typage du retour que pour le typage des arguments.
- Le typage strict affecte aussi les déclarations du type de retour.
- Dans le mode faible par défaut, les valeurs retournées seront transtypées vers le type demandé si elles ne sont pas déjà de celui-ci.
- Dans le mode strict, la valeur retournée doit être du bon type sinon il y a génération une exception TypeError.

Lors de la surcharge d'une méthode parente, la méthode fille doit retourner une valeur cohérente avec le type de valeur retournée par la méthode parente. Si le parent ne définit par de type de retour, il doit en être de même pour la fille.

Listing 20 – Déclaration du type du retour

```php
<?php
function sum($a, $b): float {
    return $a + $b;
}

// Note qu'une valeur flottante sera retournee.
var_dump(sum(1, 2));
?>
```

PHP supporte le concept de fonctions variables. Cela signifie que si le nom d'une variable est suivi de parenthèses, PHP recherchera une fonction de même nom, et essaiera de l'exécuter. Cela peut servir, entre autres, pour faire des fonctions de rappel, des tables de fonctions...

Les fonctions variables ne peuvent pas fonctionner avec les éléments de langage comme les `echo`, `print`, `unset()`, `isset()`, `empty()`, `include`, `require` etc. Vous devez utiliser votre propre gestion de fonctions pour utiliser un de ces éléments de langage comme fonctions variables.

Listing 21 – Exemple de fonction variable

```php
<?php
function foo() {
    echo "dans foo()<br />\n";
}
function bar($arg = '')
{
    echo "Dans bar(); l'argument etait '$arg'.<br />\n";
}
// Ceci est une fonction detournee de echo
function echoit($string)
{
    echo $string;
}
$func = 'foo';
$func();        // Appel foo()
$func = 'bar';
$func('test');  // Appel bar()
$func = 'echoit';
$func('test');  // Appel echoit()
?>
```

Les fonctions anonymes, aussi appelées fermetures ou closures permettent la création de fonctions sans préciser leur nom. Elles sont particulièrement utiles comme fonctions de rappel, mais leur utilisation n'est pas limitée à ce seul usage.
Les fonctions anonymes sont implémentées en utilisant la classe Closure.

Listing 22 – Fonctions anonymes

```php
<?php
echo preg_replace_callback('~-([a-z])~', function ($match) {
    return strtoupper($match[1]);
}, 'bonjour-le-monde');
?>
```

Les fonctions anonymes peuvent aussi être utilisées comme valeurs de variables. PHP va automatiquement convertir ces expressions en objets Closure. Assigner une fermeture à une variable est la même chose qu'une assignation classique, y compris pour le point-virgule final.