Info 303

Introduction au PHP

Alin F. Rabat C.

Université de Reims Champagne Ardenne

21 septembre 2018

Sommaire

1. Page dynamique - Notion de client serveur
2. Script PHP
3. Variables
4. Opérateurs
5. Les structures de contrôle

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 !

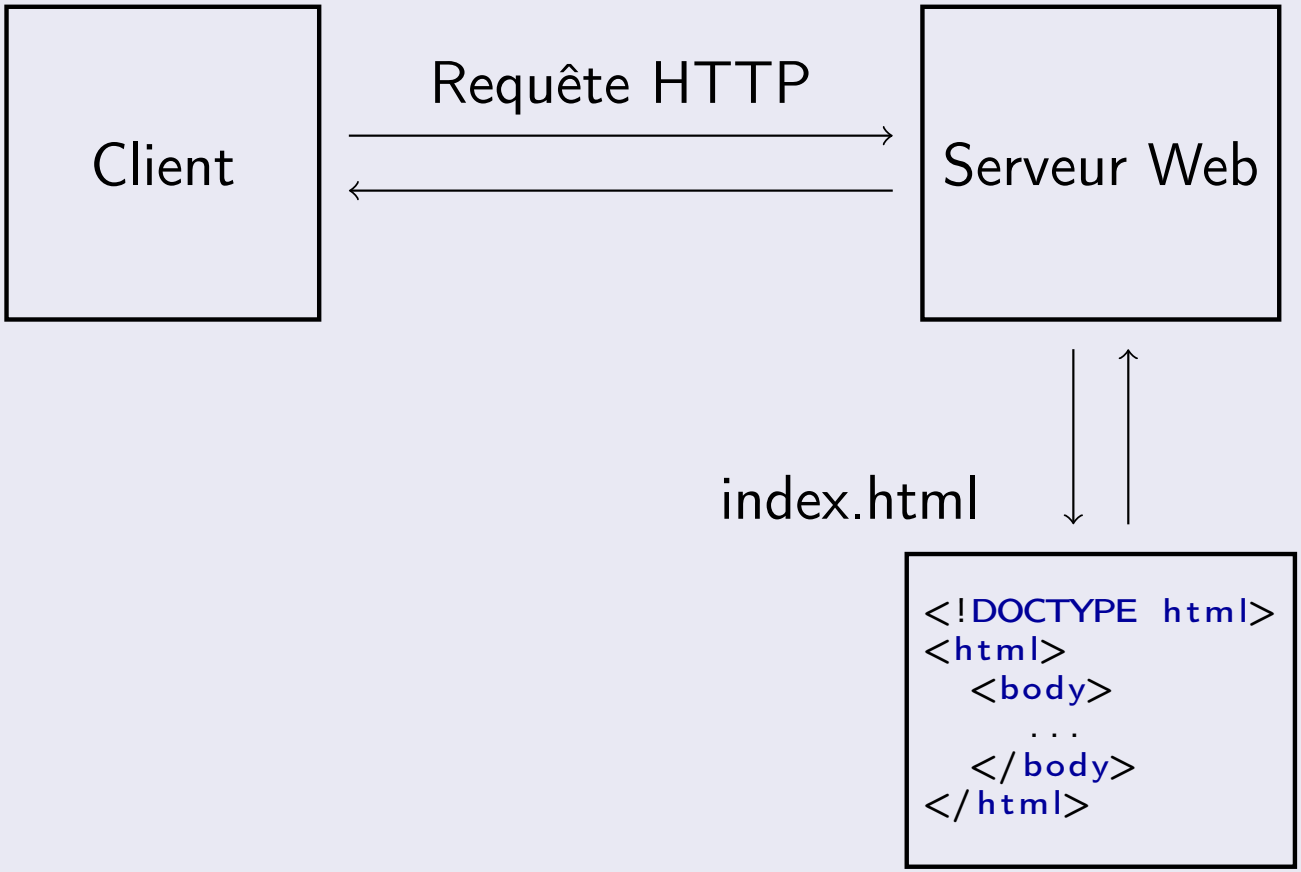
```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Exemple</title>
  </head>
  <body>
    <?php
      echo "Hello World !";
    ?>
  </body>
</html>
```

Avec PHP vous allez pouvoir :

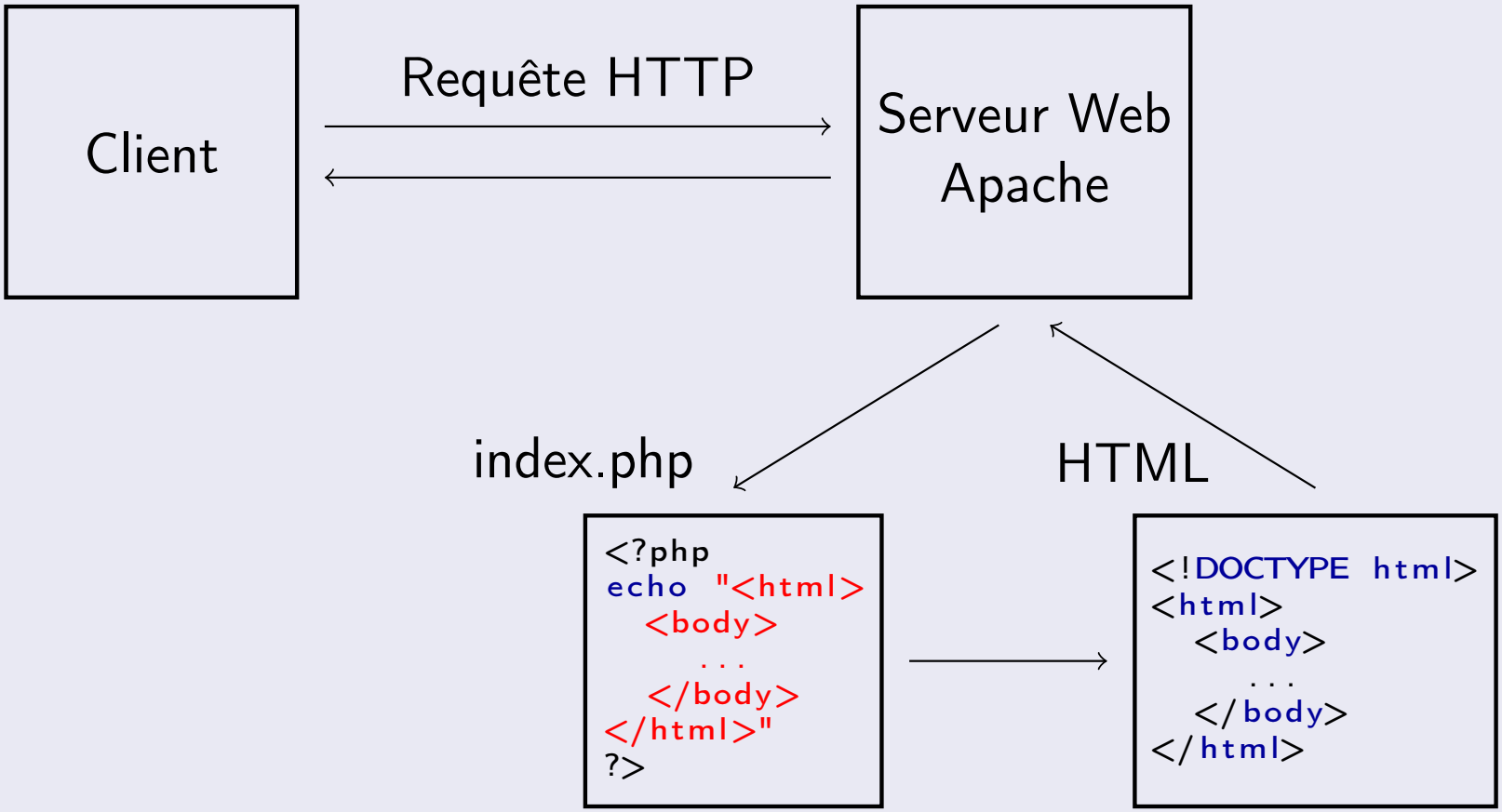
- Générer des contenus de page dynamiquement ;
- Créer, ouvrir, lire, écrire, effacer et fermer des fichiers sur le serveur ;
- Récupérer les données issues de formulaires ;
- Envoyer ou recevoir des cookies ;
- Interagir avec des bases de données ;
- Gérer le contrôle d'accès à vos pages ;
- Faire du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

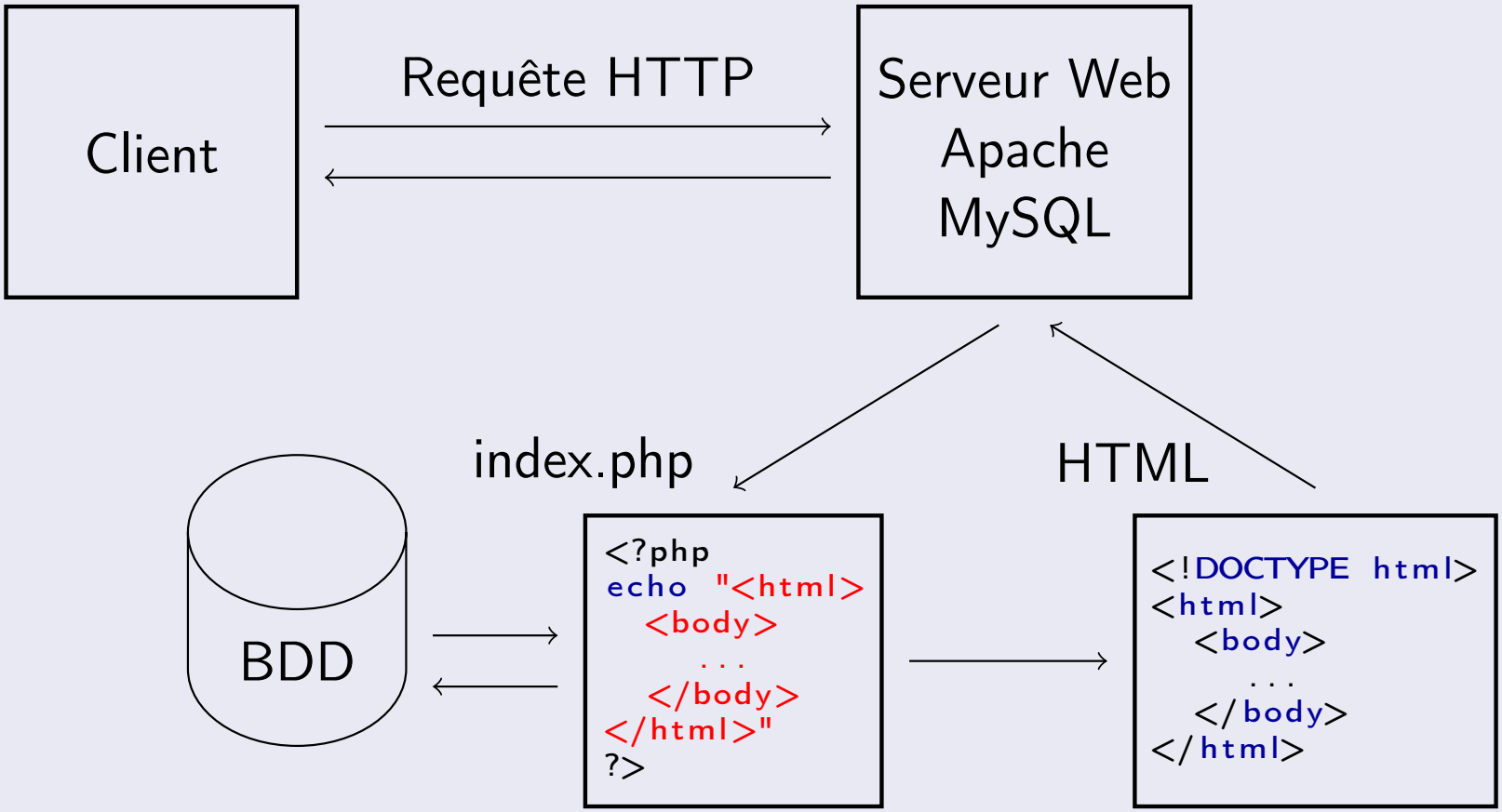
Page HTML Statique



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. Cependant, pour des questions de lisibilité, de fiabilité et de maintenabilité du code, nous donnerons plus loin quelques règles de bonne pratique.

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, pour créer par exemple des *templates*.

```
<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>
```

La balise fermante d'un bloc PHP à la fin d'un fichier est optionnelle, et parfois, il est utile de l'omettre lors de l'utilisation de la fonction include ou de la fonction require.

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
?>
```

Pour convertir explicitement une valeur en booléen, utilisez `bool` ou `boolean`.

Lors d'une conversion en booléen, les valeurs suivantes sont considérées comme `FALSE` :

- le booléen `FALSE`, lui-même
- l'entier 0 (zéro)
- le nombre à virgule flottante 0.0 (zéro)
- la chaîne vide, et la chaîne "0"
- un tableau avec aucun élément
- le type spécial NULL (incluant les variables non définies)
- les objets SimpleXML créés depuis des balises vides

Toutes les autres valeurs sont considérées comme `TRUE`

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.

Pour convertir explicitement une valeur en un entier, utilisez soit le mot-clé `(int)`, soit `(integer)`. Cependant, dans la plupart des cas, ce mot-clé n'est pas nécessaire vu qu'une valeur sera automatiquement convertie si un opérateur, une fonction ou une structure de contrôle demande un entier en guise d'argument. Une valeur peut également être convertie en un entier en utilisant la fonction `intval()`.

- Depuis un booléen :
  FALSE correspond à 0 (zéro), et TRUE correspond à 1 (un).
- Depuis un nombre à virgule flottante :
  Lorsque l'on convertit un nombre décimal en un entier, la partie décimale est tronquée.
  Si le nombre à virgule flottante est au delà des limites des entiers (habituellement, $+/-2.15e^9 = 2^{31}$ sur les plate-formes 32-bit et $+/-9.22e^{18} = 2^{63}$ sur les plate-formes 64-bit autre que Windows), le résultat sera indéfini, sachant que le nombre à virgule flottante n'a pas une précision suffisante pour donner un résultat entier exact. Aucune alerte n'est émise lorsque ce comportement survient !

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;
?>
```

La comparaison de nombres décimaux peut parfois s'avérer problématique en raison de la façon dont ces grandeurs sont représentées en interne. Ainsi pour tester l'égalité de valeurs de nombres décimaux, une borne supérieure de l'erreur relative à l'arrondi est utilisée. Cette valeur est connue comme étant l'epsilon de la machine, ou le unit roundoff, et est la différence la plus petite acceptable dans les calculs.

Listing 5 – Comparaison de deux réels à 10^{-5} près

```php
<?php
$a = 1.23456789;
$b = 1.23456780;
$epsilon = 0.00001;
if(abs($a-$b) < $epsilon) {
    echo "true";
}
?>
```

- 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.

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.

Les constantes et les variables globales utilisent deux espaces de noms différents. Ce qui implique que `TRUE` et `$TRUE` sont généralement différents (en tous cas, ils peuvent avoir des valeurs différentes).

Si l'on utilise une constante non définie, PHP considère que vous souhaitez uniquement le nom de la constante elle-même, comme si vous l'appeliez comme étant une chaîne de caractères (CONSTANT vs "CONSTANT"). Une alerte de type `E_NOTICE` sera émise lorsque ce cas se produit. Lisez également l'entrée du manuel qui explique pourquoi `$foo[bar]` est faux (tant que vous ne définissez pas bar comme étant une constante). Ceci ne s'applique pas pour les constantes (fully) qualified, ce qui déclencherait une erreur fatale si ce n'est pas défini. Si vous voulez simplement vérifier qu'une constante est définie, utilisez la fonction `defined()`.

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.

Exemples :

Listing 6 – Définir une constante

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

Listing 7 – 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"
?>
```

Contrairement aux constantes définies en utilisant l'instruction `define()`, les constantes définies en utilisant le mot-clé `const` doivent être déclarées au plus haut niveau du contexte, car elles seront définies au moment de la compilation. Cela signifie qu'elles ne peuvent être déclarées à l'intérieur de fonctions, boucles, instructions `if` ou blocs `try catch`.

Les constantes définies en utilisant le mot clé `const` sont toujours sensibles à la casse, alors que les constantes définies avec la fonction `define()` ne sont pas sensibles à la casse.

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

Le décalage de bits en PHP est arithmétique :

- Les bits qui sont décalés hors de l'entier sont perdus.
- Les décalages à gauche font apparaître des zéros à droite, tandis que le bit de signe est décalé à gauche, ce qui signifie que le signe de l'entier n'est pas préservé.
- Les décalages à droite décalent aussi le bit de signe sur la droite, ce qui signifie que le signe est préservé.

Utilisez des parenthèses pour vous assurer que la précédence voulue est bien appliquée. Par exemple, `$a & $b == true` applique d'abord l'égalité, et ensuite le ET logique, alors que `($a & $b) == true` applique d'abord le ET logique, puis l'égalité.

- Si les deux opérandes pour les opérateurs `&`, `|` et `^` sont des chaînes de caractères, alors l'opération sera réalisée sur les valeurs ASCII des caractères et le résultat sera une chaîne de caractères. Dans tous les autres cas, les deux opérandes seront converties en entier et le résultat sera un entier.
- Si l'opérande pour l'opérateur `~` `operator` est une chaîne de caractères, l'opération sera effectuée sur les caractères ASCII composant la chaîne et le résultat sera une chaîne de caractères. Sinon l'opérande et le résultat seront traités comme des entiers.
- Les opérandes et le résultat des opérateurs `<<` et `>>` sont traités comme des entiers.

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

Si vous comparez un nombre avec une chaîne ou bien que la comparaison implique des chaînes numériques, alors chaque chaîne sera convertie en un nombre et la comparaison sera effectuée numériquement. Ces règles s'appliquent également à l'instruction `switch`. La conversion de type n'intervient pas lorsque la comparaison est `===` ou `!==` vu que ceci engendre aussi bien une comparaison de type que de valeur.

Opérateur ternaire :

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

Syntaxe :

```
expr1 ? expr2 : expr3
```

Il est possible d'omettre la partie centrale de l'opérateur ternaire. L'expression `expr1 ?: expr3` retourne expr1 si expr1 vaut TRUE, et expr3 sinon.

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

Listing 8 – 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` est une des plus importantes instructions de tous les langages, PHP inclus. Elle 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`, comme son nom l'indique, 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";
}
?>
```

Vous pouvez avoir plusieurs `elseif` qui se suivent les uns après les autres, après un `if` initial. Le premier `elseif` qui sera évalué à TRUE sera exécuté. En PHP, vous pouvez aussi écrire `else if` en deux mots et son comportement sera identique à la version en un seul mot. La sémantique des deux expressions est légèrement différente, mais au bout du compte, le résultat sera exactement le même.
L'expression `elseif` est exécutée seulement si le `if` précédent et tout autre `elseif` précédent sont évalués comme FALSE, et que votre elseif est évalué à TRUE.

A noter que `elseif` et `else if` sont traités de la même façon seulement quand des accolades sont utilisées, comme dans l'exemple ci-dessus. Quand vous utilisez " :" pour définir votre condition if/elseif, vous ne devez pas séparer `else if` en deux mots, sans quoi PHP soulèvera une erreur d'interprétation.

```php
<?php
/* Mauvaise methode : */
if ($a > $b):
    echo $a." est plus grand que ".$b;
else if ($a == $b): // ne compilera pas
    echo "La ligne ci-dessus provoque une erreur d'interpretation";
endif;

/* Bonne methode : */
if ($a > $b):
    echo $a." est plus grand que ".$b;
elseif ($a == $b): // Les deux mots sont colles
    echo $a." egal ".$b;
else:
    echo $a." est plus grand ou egal a ".$b;
endif;
?>
```

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).

Les expressions peuvent éventuellement être laissées vides ou peuvent contenir plusieurs expressions séparées par des virgules. Dans `expr2`, toutes les expressions séparées par une virgule sont évaluées mais le résultat est obtenu depuis la dernière partie. Si l'expression `expr2` est laissée vide, cela signifie que c'est une boucle infinie (PHP considère implicitement qu'elle vaut TRUE, comme en C). Cela n'est pas vraiment très utile, à moins que vous souhaitiez terminer votre boucle par l'instruction conditionnelle break.

Listing 9 – 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;
```

Il est possible d'itérer grâce à des tableaux :

```php
<?php
/*
 * Ceci est un tableau avec des donnees que nous voulons modifier
 * au long de la boucle
 */
$people = array(
    array('name' => 'Kalle', 'salt' => 856412),
    array('name' => 'Pierre', 'salt' => 215863)
);

for($i = 0; $i < count($people); ++$i) {
    $people[$i]['salt'] = mt_rand(000000, 999999);
}
?>
```

Ce code peut être lent parce qu'il doit calculer la taille du tableau à chaque itération. Étant donné que la taille ne change jamais, il peut facilement être optimisé en utilisant une variable intermédiaire pour stocker la taille au lieu d'appeler de façon répétitive la fonction `count()` :

```php
<?php
$people = array(
    array('name' => 'Kalle', 'salt' => 856412),
    array('name' => 'Pierre', 'salt' => 215863)
);

for($i = 0, $size = count($people); $i < $size; ++$i) {
    $people[$i]['salt'] = mt_rand(000000, 999999);
}
?>
```

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 10 – 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;
?>
```

Il est possible d'utiliser un point-virgule plutôt que deux points après un `case`, comme ceci :

```php
<?php
switch($beer)
{
    case 'leffe';
    case 'grimbergen';
    case 'guinness';
        echo 'Bon choix';
    break;
    default;
        echo 'Merci de faire un choix...';
    break;
}
?>
```