Tutoriel de PHP pour débutant

Table de matière

Avant propos

Ce document est destiné au débutant qui souhaite se lancer dans l'apprentissage du PHP.

La version de PHP à l'écriture de ce document est la version 8.3.

Versions

Version	Date	Commentaire
1.0	01/11/2024	Première version

Pré-requis

Pour pouvoir apprécier ce tutoriel comme il se doit, il faudra répondre aux critères suivants :

- Connaître un minimum les algorithmes ou au moins savoir ce que c'est.
- Connaître le HTML dans les grands principes.

Sources

Ce document s'appuie fortement sur la documentation officiel de PHP

Chapitre 1 : Qu'est ce que le PHP ?

Définition

PHP, acronyme de PHP Hypertext Preprocessor, est un langage de script open source.

Fonctionnement

Dans le cas d'un site web, voici comment fonctionne PHP :
Un visiteur consulte une page web, son navigateur envoie une requête HTTP au serveur correspondant. Le serveur détermine via l'extension du script que c'est un fichier php (.php) et appelle l'interprète PHP qui va traiter et générer le code finale de la page. Une fois terminé, le contenu est renvoyé au serveur qui le renvoie au visiteur.

Chapitre 2 : Environnement de travail

Toute personne qui veut se lancer dans le code à besoin d'un environnement de travail complet pour pouvoir développer sereinement.

Cette partie du tutoriel vous présente quelques manières parmi tant d'autres d'installer son environnement de travail.

Installation d'un environnement complet PHP

Windows

Wamp est un logiciel complet qui permet en une seule installation d'avoir un environnement complet c'est à dire PHP, serveur Apache et une base de données MYSQL.

URL	Commentaire
https://www.wampserver.com/	Site officiel
https://wampserver.aviatechno.net/?lang=fr	Aide installation + Addon

Mac

Mamp est un logiciel complet qui permet en une seule installation d'avoir un environnement complet c'est à dire PHP, serveur Apache et une base de données MYSQL.

URL	Commentaire
https://www.mamp.info/en/mac/	Site officiel

Linux

Lamp est un logiciel complet qui permet en une seule installation d'avoir un environnement complet c'est à dire PHP, serveur Apache et une base de données MYSQL.

URL	Commentaire
https://doc.ubuntu-fr.org/lamp	Site officiel

Docker

Docker est une plateforme de containerisation qui permet de créer des environnements légers indépendants de façon plus ou moins simple.

URL	Commentaire
https://www.docker.com/	Site officiel

Autres façons

Il existe de multiples façons d'installer PHP et l'environnement qui gravite autour. Retrouvez ces façons dans la doc officiel de PHP

Chapitre 3 : IDE

IDE ou encore Environnement de Développement Intégré est un logiciel qui permet d'aider au développement informatique.

Un IDE apporte en autres les fonctions suivantes :

- Coloration syntaxique, indentation
- Autocomplétion du code
- Debugger
- Assistant de partage de code (git) etc...

Il n'existe pas un IDE meilleur qu'un autre, il est important de trouver celui qui vous convient le mieux et où vous serez le plus à l'aise.

Voici une liste non exhaustive d'IDE pour faire du PHP :

IDE	OS	Prix
Visual Studio Code	Windows / Mac / Linux	Gratuit
Notepad++	Windows	Gratuit
Eclipse PDT	Windows / Mac / Linux	Gratuit
PHPStorm	Windows / Mac / Linux	Payant
Smultron	Mac	Payant

Chapitre 4 : PHP - Premier pas

Mon premier script

Nous avons maintenant un environnement qui fonctionne, un IDE pour travailler, tout ce qu'il faut pour se lancer dans l'apprentissage de PHP.

Un fichier PHP est tout simplement un fichier avec comme extension ".php" à la fin. C'est cette extension qui va dire à l'interpréteur que ce fichier contient du PHP. Pour que ce fichier soit correctement interprété, il faut le placer dans le dossier défini comme "virtual host" dans votre environnement. Par exemple sur wamp, c'est le dossier "www" qui est à

l'adresse suivante (installation par défaut) : C:\wamp64\www. Pour plus de lisibilité, créer un dossier que l'on appellera tuto dans notre exemple dans le dossier www.

Ouvrez votre IDE, et créez un fichier que vous allez appeler index.php. Pourquoi ce nom précisément ? Car ce nom précis indique à l'interpréteur de l'interpréter par défaut si on pointe vers la racine de votre dossier tuto.

Un script PHP commence toujours par la balise `<?php` et se termine par `?>` dans le cas d'un script dans une page HTML. Si votre fichier contient que du PHP alors il n'y à pas besoin de balise de fermeture.

Voici un exemple de fichier index.php

```php
<?php
echo 'hello world';
// Pas de balise de fermeture
```

Remarquer que l'instruction `echo 'hello world';` finit par un ; (point virgule). C'est une norme de PHP, toutes les instructions de votre script doivent se terminer par un ; sans cela votre script ne pourra pas s'exécuter.

Si vous allez sur l'url : http://localhost/tuto vous aurez alors un "Hello word".
Félicitation vous avez fait votre premier script PHP

Les variables

Une variable permet de stocker une donnée, elle comment toujours un $ et ne doit pas posséder d'espace ou d'accent

```php
$maVariable = 'Une variable'; // Ceci est une variable
```

PHP étant un langage typé dynamiquement, le type va être défini au moment de l'exécution du script. On verra plus tard qu'il est possible de typer statiquement certains aspects du langage en utilisant les déclarations de type.

Il existe de nombreux types qui sont :

- **null** : valeur par défaut de toutes variables non définies
- **bool** : ne possède que 2 valeurs true ou false, on parle de valeur de vérité
- **int** : nombre entier
- **float** : nombre à virgule flottante
- **string** : chaine de caractère défini entre les caractères ' ' (quote)
- **array** : tableau de données
- **object** : objet PHP
- **callable** : fonction de rappel

- **ressource** : variable spéciale, contenant une référence vers une ressource externe

Exemple de type

```
$var = null;    // Type null
$var = true;    // Type Boolean
$var = 10;      // Type int
$var = 10.5;    // Type float
$var = 'chaine';        // Type string
$var = [];              // Type array
$var = new Objet();     // Type object
```

Les tableaux

Le type array ou tableau est en gros une carte ordonnée sous la forme de clef => valeur.

```
$array = [
    'pomme', 'banane', 'raisin'
];
var_dump($array);
/*  le résultat affiché est
array(3) {
  [0]=> string(5) "pomme"
  [1]=> string(7) "banane"
  [2]=> string(6) "raisin"
} */
```

Un tableau peut être considéré comme un tableau, une liste, une table de hachage, un dictionnaire, une collection, une pile, une file d'attente et probablement plus. On peut avoir, comme valeur d'un tableau, d'autres tableaux, multidimensionnels ou non.

Voici un exemple de tableau à 2 dimensions avec 2 syntaxes différentes

```
$array = [
    'user_1' => [
        'nom' => 'Doe',
        'Prenom' => 'John'
        ],
    'user_2' => [
        'nom' => 'Macfly',
        'Prenom' => 'Marty'
        ],
    ];

$array = array(
```

```
    'user_1' => array(
        'nom' => 'Doe',
        'Prenom' => 'John'
        ),
    'user_2' => array(
        'nom' => 'Macfly',
        'Prenom' => 'Marty'
        ),
    );
```

Il existe 2 méthodes pour créer un tableau, la syntaxe dite classique en utilisant la structure de langage `array()` ou la syntaxe courte qui est []

Les objets

Les objets sont un type bien particulier auquel nous reviendrons plus tard dans ce tutoriel. (Voir Initiation à la POO)

Les constantes

Une constante est un identifiant (un nom) qui représente une valeur simple. Comme son nom le suggère, cette valeur ne peut jamais être modifiée durant l'exécution du script. Par convention, les constantes sont toujours en majuscule.

Exemple de constantes

```
// Noms valides
define("FOO",     "une valeur");
define("FOO2",    "Une autre");
define("FOO_BAR", "Autre valeur");

// Noms invalides
define("2FOO",    "Marche pas");
```

Exemple de constante dans une classe

```
class MaClass
{

   /**
    * Ma constante
    * @var int
    */
   public const MA_CONSTANTE = 12;
```

```
}
```

Echo

Echo est une instruction qui permet d'afficher la valeur d'une variable, cela permet par exemple d'afficher un texte, un résultat de valeur dans vos script PHP.

```
echo '<p>Je suis un texte</p>';     // Affichera "je suis un texte" dans
votre navigateur
```

Dans du HTML il existe une syntaxe raccourcie qui est la suivante :

```
<p>J'ai <?=$nb?> pommes.</p>
```

Quote ou double quote ?

Comme vous avez dû le remarquer, lorsque l'on déclare une variable de type string on encapsule la chaîne de caractère entre 2 quotes (ou apostrophes ‘), il est aussi possible d'utiliser le caractère double quote (ou guillemet “”).

```
$var = 'ma chaine'; // on utilise les '
$var = "ma chaine"; // fonctionne aussi avec les "

// Ce que l'on peut faire du coup
$nb = 2;
//Ceci affichera directement : J'ai 2 pommes.
echo "J'ai $nb pommes.";

// Avec des simples quote, il faut échapper le ' de j'ai et utiliser
l'opérateur de concaténation "." pour afficher la variable $nb
echo 'J\'ai '.$nb.' pommes.';
```

Mais alors quelle est la meilleure syntaxe à utiliser ? Et bien PHP recommande d'utiliser la simple quote car cela permet de pouvoir afficher correctement les caractères d'échappement.

Chapitre 5 : Les opérateurs

Un opérateur est quelque chose qui prend une ou plusieurs valeurs et qui produit une autre valeur.

Les opérateurs arithmétiques

Les opérateurs arithmétiques sont tout simplement les opérateurs pour réaliser des calculs simples comme à l'école.

Voici un exemple d'opérateurs arithmétiques :

```
$var = 1;
$var = $var +1; // Addition
echo $var; // Affichera 2

$var = $var -1; // Soustraction
echo $var; // Affichera 1 (2-1)

$var = 5*2; // Multiplication
echo $var; // Affichera 10

$var = 100/10; // Division
echo $var; // Affichera 10

$var = 10+1*2;
echo $var; // Affichera 12 car la multiplication est prioritaire

$var = (10+1)*2;
echo $var; // Affichera 22 car 10+1 est entre parenthèses

$var = 2**2; // Exposant, équivalant à 2 puissance 2
echo $var; // Affichera 4
```

Attention, les règles de priorités mathématiques sont prises en compte.

L'opérateur modulo est un peu plus complexe, et s'écrit avec la syntaxe suivante `e1 % e2` . Il génère le reste donné par l'expression suivante, où e1 est le premier opérande et e2 est le second : e1 - (e1 / e2) * e2, où les deux opérandes sont de types intégraux.

Le résultat de l'opération modulo % a le même signe que le premier opérande, ainsi le résultat de $a % $b aura le signe de $a.

Le résultat d'un modulo sera toujours converti en int, même si le résultat final est un float.

Ce qui donne niveau code

```
echo (5 % 3);           // affiche 2
echo (5 % -3);          // affiche 2
echo (-5 % 3);          // affiche -2
echo (-5 % -3);         // affiche -2
```

Opérateurs d'incrémentation et décrémentation

Ces opérateurs permettent d'ajouter ou soustraire 1 à une valeur, attention au placement de l'opérateur "++" et "--" qui va influencer la valeur de la variable dans le code.

```
$a = 5;     // Post-incrément
echo $a++;  // Affiche 5
echo $a;    // Affiche 6
$a = 5;     // Pre-incrément
echo ++$a;  // Affiche 6;
echo $a;    // Affiche 6;
$a = 5;     // Post-décrément
echo $a--;  // Affiche 5
echo $a;    // Affiche 4
$a = 5;     // Pre-décrément:
echo --$a;  // Affiche 4
echo $a;    // Affiche 4
```

Les opérateurs d'affectation

Les opérateurs d'affectations permette d'affecter une valeur à une variable
Le plus courant est le "=" mais il en existe d'autres.

```
$var = 5; // On affecte 5 à $var
$var += 5; // $var vaut maintenant 10 (5+5)

$var = "Hello"; // On affecte la chaîne de caractère hello à $var
$var .= " World"; // $var affiche maintenant "Hello world"
```

L'affectation par référence

Par défaut lorsque l'on écrit le bout de code suivant

```
$a = 5;
$b = $a; // $b est une copie de $a
$b++;
echo $a; // Affiche 5
echo $b; // Affiche 6
```

En gros $b est une copie de $a et est indépendant de $a.
Il est parfois nécessaire que $a et $b pointe vers le même conteneur de données, dans ce cas on parle alors de référence.

Reprenons notre code et ajoutons juste un & à la variable $a.

```php
$a = 5;
$b = &$a; // On définit $b comme référence de $a
$b++;
echo $a; // Affiche 6 et non 5 comme avant
echo $b; // Affiche 6
```

Que s'est-il passé ? Avec l'opérateur & $b est devenu une référence de $a, ce n'est donc pas une copie puisque $a et $b pointe vers le même conteneur de données. Lorsque l'on modifie $b alors $a est aussi modifié.

Les combinaisons d'affectations avec les opérateurs

Il est possible de combiners les affectations avec tous les opérateurs disponibles

```php
$a += $b;   // équivalent à $a = $a + $b addition
$a -= $b;   // équivalent à $a = $a - $b soustraction
$a *= $b;   // équivalent à $a = $a * $b multiplication
$a /= $b;   // équivalent à $a = $a / $b division
$a %= $b;   // équivalent à $a = $a % $b modulo
$a **= $b;  // équivalent à $a = $a ** $b exponentiation
$a &= $b    // équivalent à $a = $a & $b opérateur And
$a |= $b    // équivalent à $a = $a | $b opérateur Or
$a ^= $b    // équivalent à $a = $a ^ $b opérateur Xor
$a <<= $b   // équivalent à $a = $a << $b décalage à gauche
$a >>= $b   // équivalent à $a = $a >> $b décalage à droite
$a .= $b    // équivalent à $a = $a . $b concaténation d'une chaîne de
caractères
$a ??= $b   // équivalent à $a = $a ?? $b opérateur de coalescence nul
```

Opérateurs de comparaison

Les opérateurs de comparaisons permettent de comparer 2 valeurs.
Voici la liste des opérateurs de comparaisons

Exemple	Nom	Résultat
$a == $b	Égal	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 inférieur, égal, ou supérieur à $b respectivement.

Les opérateurs logiques

Les opérateurs logiques permettent de vérifier si plusieurs conditions sont vrai

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.

Chapitre 6 : Les structures de contrôles

Une structure de contrôle est un ensemble d'instructions qui permet de contrôler l'exécution du code. Il en existe essentiellement deux types :

- Les structures de contrôle conditionnelles qui permettent d'exécuter certaines parties du code si une condition spécifique est remplie.
- Les structures de contrôle de boucle qui permettent d'exécuter en boucle certaines parties du code (généralement jusqu'à ce qu'une condition soit remplie).

Nous allons passer en revue l'ensemble des structures de contrôles de PHP.

If, else, elseif, else if

if

Le if est l'une des instructions les plus importantes de PHP, et on peut la résumer ainsi.

```
if (expression)
  commandes
```

Comment cela fonctionnne ? Et bien PHP va convertir l'expression en une valeur boolean, si cette valeur est true, alors on exécute les commandes, c'est aussi simple que cela

Dans le code cela ce traduit de la manière suivante :

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

On peut ajouter autant d'instruction que l'on veut entre les {}, elles ne seront prisent en compte que si l'expression vaut true

Il est possible grâce au opérateurs logiques de réaliser plusieurs expression

```
if($a > $b && $a > $c)
{
    echo "a est plus grand que b et c";
}
```

Dans le cas de conditions complexes, on peut aussi imbriquer les if les un dans les autres

```
if($a > $b) {
   echo "a est plus grand que b";
   if($a < $c) {
       echo "mais a est plus petit que c";
   }
}
```

Quand le if ne possède qu'une seule commande dans sa condition alors il est possible de l'écrire de la façon suivante :

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

else

Le else est la suite du if dans le cas ou vous voulez exécuter une instruction quand une condition n'est pas remplie

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

Il est tout à fait possible d'imbriquer les if et else ensemble mais attention dans ce cas, le else sera toujours associé avec le if le plus proche même si l'indentation vous indique le contraire

```
$a = false;
$b = true;
if ($a)
    if ($b)
        echo "b";
else
    echo "c";
// Dans ce cas le code n'affiche rien
```

Il existe une écriture simplifié du if else qui est l'opérateur ternaire, que l'on peut écrire sous la forme

```
(condition ? 'commande true' : 'commande false')
```

reprenons le if else vu précédemment et simplifions le

```
$a = 5;
$b = 2;
// Version classique
if ($a > $b) {
 echo "a est plus grand que b";
} else {
 echo "a est plus petit que b";
}
```

```php
// La ternaire
$var = ($a > $b) ? "a est plus grand que b" : "b est plus grand que a";
echo $var; // $var affiche a est plus grand que b
```

Dans le cas d'une ternaire, le résultat doit être soit être stocké dans une variable soit affiché via un echo avant la condition.
Attention tout de même à ne pas abuser de cet opérateur en particulier dans le cas de condition complexe, cela peut nuire à la lisibilité du code et le rendre plus complexe que nécessaire.
Un bon cas d'utilisation d'une ternaire est la vérification de l'existence d'une valeur dans une variable pour pouvoir utiliser cette valeur, ce qui donne :

```php
$a = null;
$b = 2;

$c = $a ? $a : $b;
echo $c; // C affiche 2 car $a est null

$a = 1;
$b = 2;

$c = $a ? $a : $b;
echo $c; // c affiche 1 car $a = 1
```

elseif/else if

Le elseif ou encore else if est tout simplement une combinaison de if et de else. Cela permet de réaliser une nouvelle expression si la première expression dans le if n'est pas bonne.

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

Il est tout à fait possible d'avoir plusieurs elseif qui se suive.

La boucle while

La boucle while est le moyen le plus simple de faire une boucle en PHP, tant que l'expression vaut true alors la boucle continue.

```
while (expression)
    commandes
```

Exemple de code

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

Dans cet exemple, tant que $i ne vaut pas 10 alors l'expression vaut true et la boucle continue.
Il n'y à pas de limite au nombre de commandes que l'on peut mettre dans un while

Tout comme le if on peut écrire un while sans accolade de la façon suivante :

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

Cet exemple est exactement le même que précédemment mais avec une syntaxe légèrement différente. Noté dans ce cas de la présence du endwhile pour signifier à PHP la fin du block while.

La boucle do-while

La boucle do-while est la même chose que la boucle while à la seule différence que l'expression est testé à la fin de l'itération et non au début, ce qui veut dire que la première boucle de l'itération sera toujours exécutée.

```
$i = 0;
do {
   echo $i;
} while ($i > 0);
// Affiche 0
```

La boucle for

La boucle for est la boucle la plus complexe en PHP car elle fonctionne en fonction de plusieurs instructions

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

La première expression (expr1) est 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 exécutée.

Voici ce que cela donne niveau code

```
for ($i = 1; $i <= 10; $i++) {
   echo $i;
}
// Ce qui affichera 12345678910
```

On utilise essentiellement les boucles for quand on souhaite itéré un tableau php, voici un exemple de code

```
$array = [
    'pomme', 'banane', 'raisin'
];

// On initialise $i et $size à la première itération pour éviter de
devoir recalculer à chaque boucle la taille du tableau
for ($i = 0, $size = count($array); $i < $size; ++$i ) {
    echo $array[$i] . ' <br />';
}
/* output
pomme
banane
raisin
*/
```

La boucle foreach

La structure de langage foreach est la méthode la plus simple pour parcourir les tableaux PHP. Vous ne pouvez utiliser foreach que sur un tableau ou un objet initialisé.

Il existe 2 syntaxes qui sont :

```
foreach (iterable_expression as $value){
    //commandes
}
foreach (iterable_expression as $key => $value){
    //commandes
}
```

La première forme passe en revue le tableau iterable_expression. À chaque itération, la valeur de l'élément courant est assignée à $value.
La seconde forme assignera en plus la clé de l'élément courant à la variable $key à chaque itération.
Vous pouvez modifier facilement les éléments d'un tableau en précédent $value d'un &. Ceci assignera une référence au lieu de copier la valeur.

Exemple de code

```
$array = [
    'pomme', 'banane', 'raisin'
];

// Foreach classique
foreach($array as $valeur) {
    echo $valeur . ' <br />';
}
/* output
pomme
banane
raisin
*/

// Foreach avec clé => valeur
foreach($array as $key => $valeur) {
    echo $key . ' => ' . $valeur . ' <br />';
}
/* output
0 => pomme
1 => banane
2 => raisin
*/
```

Exemple de code en modifiant la valeur d'un tableau par la référence

```
$array = [
```

```
    'pomme', 'banane', 'poire'
];

// On modifie la référence d'une ligne du tableau
foreach($array as &$valeur) {
    $valeur .= " mûre";
}

var_dump($array);
/*
array(3) {
  [0]=> string(11) "pomme mûre"
  [1]=> string(12) "banane mûre"
  [2]=> string(11) "poire mûre"
}
*/
```

Exemple de code en imbriquant les foreach pour parcourir les tableaux à multidimension

```
$array = [
    'pomme' => [
    'Gala', 'Golden', 'Dalinette'
    ], 'banane', 'poire'
];

foreach($array as $key => $valeur) {
    // SI $valeur est un tableau alors on boucle dessus
    if(is_array($valeur)) {
        echo $key . ": [Espèces : ";
        foreach($valeur as $espece) {
            echo $espece . ', ';
        }
        echo "]";
    }
    // Sinon on affiche sa valeur
    else {
        echo $valeur . ' ';
    }
}

/* Affiche
pomme: [Espèces : Gala, Golden, Dalinette]
banane
poire*/
```

```
var_dump($array);
/*
array(3) {
  ["pomme"]=>
  array(3) {
    [0]=> string(4) "Gala"
    [1]=> string(6) "Golden"
    [2]=> string(9) "Dalinette"
  }
  [0]=> string(6) "banane"
  [1]=> string(5) "poire"
}

*/
```

Switch

Le switch est l'équivalent de plusieurs instructions if qui se suivent. Il arrive par moment que vous devez comparer la même variable avec de nombreuses valeurs différentes et exécuter du code différent pour chaque cas. Il est tout à fait possible de le faire avec des if mais cela risque de rendre rapidement le code illisible, le switch est là pour pallier au problème.

```
// Ce switch:

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

// Équivaut à:

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

```
}
```

Match

Match est apparu avec PHP 8.0, cette structure fonctionne globalement comme un switch mais avec quelques petites différences :

- Une expression match compare les valeurs de manière stricte (===) et non de manière faible comme le fait l'instruction switch.
- Une expression match renvoie une valeur.
- Les expressions match ne passent pas aux cas suivants comme le font les instructions switch.
- Une expression match doit être exhaustive.

Exemple de code avec match

```php
$age = 18;

$output = match (true) {
   $age < 2 => "Bébé",
   $age < 13 => "Enfant",
   $age <= 19 => "Adolescent",
   $age > 19 => "Jeune adulte",
   $age >= 40 => "Adulte âgé"
};

var_dump($output);
/* output
string(9) "Adolescent"
*/
```

Break

L'instruction break est très simple, elle permet de sortir d'une structure for, foreach, while, do-while ou switch

break accepte un argument numérique optionnel qui vous indiquera combien de structures emboîtées doivent être interrompues. La valeur par défaut est 1, seulement la structure emboîtée immédiate est interrompue.

Voici 2 exemples de break

```php
$array = [
    'pomme', 'banane', 'raisin'
];
```

```php
// Foreach classique
foreach($array as $valeur) {
    echo $valeur . ' <br />';
    if($valeur === 'banane') {
        break;
    }
}
/* output
pomme
banane
*/

// Cas avec structures emboîtées
$i = 0;
while (++$i) {
   switch ($i) {
       case 5:
           echo "$i = 5";
           break 1;  /* Termine uniquement le switch. */
       case 10:
           echo "$i = 10 stop while";
           break 2;  /* Termine le switch et la boucle while. */
       default:
           break;
   }
}
```

Continue

L'instruction continue permet de passer à l'itération suivante sans forcément finir les instruction en cours

```php
$arr = ['zéro', 'un', 'deux', 'trois', 'quatre', 'cinq', 'six'];
foreach ($arr as $key => $value) {
   if (0 === ($key % 2)) { // évite les membres pairs
       continue;
   }
   echo $value . "\n";
}
/* output
un
trois
cinq
*/
```

Chapitre 7 : Les fonctions

Une fonction est une série d'instructions qui effectue une action et qui renvoie un résultat.

Voici un exemple de fonction

```php
/**
* @param int $age
* @return bool
*/
function isAdult(int $age): bool
{
   if ($age < 18) {
       return false;
   }
   return true;
}

$age = 15;
var_dump(isAdult($age)); // Retourne false

$age = 50;
var_dump(isAdult($age)); // Retourne true
```

Que fait cette fonction ? Elle indique tout simplement si on est majeur ou non. SI $age est inférieur à 18 alors la fonction renvoi false, sinon true.

Attention tout de même, le nom de la fonction doit respecter une certaine syntaxe qui est la suivante : Un nom de fonction valide commence par une lettre ou un "_" (underscore), suivi par un nombre quelconque de lettres, de nombres ou d' "_" (underscore).

Définition de base

Les fonctions n'ont pas besoin d'être définies avant d'être utilisées, SAUF lorsqu'une fonction est définie conditionnellement, comme montré dans les deux exemples suivants.

Lorsqu'une fonction est définie de manière conditionnelle, comme dans les exemples ci-dessous, leur définition doit précéder leur utilisation.

```php
$makefoo = true;

/* Impossible d'appeler foo() ici,
  car cette fonction n'existe pas.
```

```
  Mais nous pouvons utiliser bar() */

bar();

if ($makefoo) {
 function foo()
 {
   echo "Je n'existe pas tant que le programme n'est pas passé ici.\n";
 }
}

/* Maintenant, nous pouvons appeler foo()
  car $makefoo est évalué à vrai */

if ($makefoo) foo();

function bar()
{
 echo "J'existe dès le début du programme.\n";
}
```

En PHP, toutes les fonctions ont une portée globale, être appelées à l'extérieur d'une fonction si elles ont été définies à l'intérieur et vice-versa.

```
function foo()
{
 function bar()
 {
   echo "Je n'existe pas tant que foo() n'est pas appelé.\n";
 }
}

/* Impossible d'appeler bar() ici
  car il n'existe pas. */

foo();

/* Maintenant, nous pouvons appeler bar(),
  car l'utilisation de foo() l'a rendue
  accessible. */

bar();
```

Il est aussi possible qu'une fonction s'appelle elle-même, dans ce cas on appelle cette fonction une fonction récursive. Ce genre de fonction est très pratique quand on doit lire

le contenu d'un tableau qui contient "n" tableau(x) sans en connaître le nombre exact ni la profondeur.

Exemple d'une fonction récursive qui affiche le contenu d'un tableau PHP

```php
/**
* Affiche le contenu du tableau
* @param array $tb
* @return void
*/
function parseArray(array $tb): void
{
   foreach ($tb as $key => $value) {

       // Si $value est un array
       if (is_array($value)) {
           echo $key . ' :<ul>';
           parseArray($value);
           echo '</ul><br />';
       } else { //Sinon, c'est un élément à afficher
           echo '<li>' . $value . '</li>';
       }
   }
}

$tb = [
   'Légumes' => [
       'Choux',
       'Haricots',
       'Épinards'
   ],
   'Fruits' => [
       'Agrumes' => [
           'Oranges',
           'Clémentines',
           'Citrons'
       ],
       'Fruits rouges' => [
           'Cassis',
           'Framboises',
           'Mûres',
           'Groseilles'],
       'Autres fruits' => [
           'Pommes',
           'Poires'
       ]
   ]
```

```
];
parseArray($tb);

/*
* Output
* Légumes
*      Choux
*      Haricots
*      Epinards
* Fruits
*      Agrumes
*          Oranges
*          Clémentines
*      Citrons
*      Fruits rouges
*          Cassis
*          Framboises
*          Mûres
*          Groseilles
*      Autres fruits
*          Pommes
*          Poires
*/
```

Les arguments

Pour faire passer des informations à une fonction on utilise les arguments ou paramètres.
Chaque argument est séparé par une virgule.
Les arguments sont toujours lus de gauche à droite avant l'appel de la fonction.

```
/**
* @param string $arg
* @param string|null $arg2
* @return void
*/
function maFonction(string $arg, string $arg2 = null): void
{
   // Instruction
}

/**
* Autre syntaxe possible
* @param string $arg
```

```
* @param string|null $arg2
* @return void
*/
function maFonction(
   string $arg,
   string $arg2 = null
): void
{
   // Instruction
}
```

Argument de référence

Lorsque l'on passe un argument à une fonction, par défaut celui-ci sera passé par valeur (ainsi, changer la valeur d'un argument dans la fonction ne change pas sa valeur à l'extérieur de la fonction). Si vous souhaitez pouvoir changer la valeur de vos arguments il faut alors utiliser la référence de l'argument.

```
/**
 * Utilisation d'un paramètre de référence
 * @param string $chaine
 * @return void
 */
function updateChaine(string &$chaine)
{
    $chaine .= " mise à jour";
}

$chaine = "cette chaine est";
updateChaine($chaine);
echo $chaine; // Affiche "cette chaine est mise à jour"
```

Valeur par défaut des arguments

Une fonction peut définir des valeurs par défaut pour les arguments en utilisant une syntaxe similaire à l'affectation d'une variable. La valeur par défaut n'est utilisée que lorsque le paramètre n'est pas spécifié

```
/**
 * @param string $couleur
 * @return string
 */
function couleurCheveux(string $couleur = "blond")
{
```

```
    return 'Mes cheveux sont ' . $couleur;
}

echo couleurCheveux('châtain'); // Retourne mes cheveux sont châtin
echo couleurCheveux(); // Retourne mes cheveux sont blond
echo couleurCheveux(null); // Retourne une erreur car null est un type
particulier
```

Attention à l'ordre des arguments lorsque vous utilisez des valeurs par défaut, Les arguments sans valeur par défaut doivent être passés en premier, sinon cela provoque une fatale erreur

```
/**
 * @param string $couleur
 * @param string $prenom
 * @return string
 */
function couleurCheveux(string $couleur = "blond", string $prenom)
:string
{
    return 'Je m\'appel ' . $prenom . ' et mes cheveux sont' . $couleur;
}

echo couleurCheveux('Pierre');
/* Provoque l'erreur
PHP Fatal error:  Uncaught ArgumentCountError: Too few arguments to
function couleurCheveux(), 1 passed
*/
```

Le bon ordre est le suivant

```
/**
 * @param string $prenom
 * @param string $couleur
 * @return string
 */
function couleurCheveux(string $prenom, string $couleur = "blond")
:string
{
    return 'Je m\'appel ' . $prenom . ' et mes cheveux sont' . $couleur;
}

echo couleurCheveux('Pierre'); // Retourne Je m'appel Pierre mes cheveux
sont blond
```

```
echo couleurCheveux('Marc', 'rouge'); // Retourne Je m'appel Marc mes
cheveux sont rouge
```

Les arguments nommés

Les arguments nommés permettent de passer les arguments à une fonction en s'appuyant sur le nom du paramètre, au lieu de la position du paramètre. Ceci documente automatiquement la signification de l'argument, rend l'ordre des arguments indépendant et permet d'ignorer les valeurs par défaut arbitrairement.

```
/**
 * @param string $prenom
 * @param string $couleur
 * @return string
 */
function couleurCheveux(string $prenom, string $couleur = "blond")
:string
{
    return 'Je m\'appel ' . $prenom . ' et mes cheveux sont' . $couleur;
}
echo couleurCheveux(couleur : 'rouge', prenom: 'Marc'); // Retourne Je
m'appel Marc mes cheveux sont rouge
```

Dans l'exemple ci-dessus, grâce aux arguments nommés je peux inverser l'ordre des paramètres comme bon me semble.

Typer les arguments

Depuis PHP 7.4 il est possible de typer les arguments d'une fonction afin de pouvoir effectuer un meilleur contrôle de ce que l'on envoie à notre fonction et de limiter ainsi les problèmes de changement de type qui peuvent créer des instabilités dans le code.

Pour typer un argument, il suffit de préciser le type devant la variable que vous voulez typer

Voici un exemple de fonction avec des paramètres typés :

```
/**
* @param int $id
* @param string $chaine
* @param bool $isOk $
* @param array $params
* @return void
*/
```

```php
function functionTest(int $id, string $chaine, bool $isOk = false, array
$params = []):void
{
   // ...
}
```

Si par exemple je décide d'envoyer un string au lieu d'un integer pour $id, cela va provoquer l'erreur suivante :

```
HP Fatal error:  Uncaught TypeError: functionTest(): Argument #1 ($id) must
be of type int, string given
```

Les valeurs de retour

Lorsqu'une fonction à terminé elle à la possibilité de pouvoir renvoyer son résultat grâce à la structure `return`.
Le `return` stop l'exécution en cours et rend le résultat au script appelant.

```php
/**
* @param string $role
* @return array
*/
function generateArray(string $role = 'USER'): array
{
   $array[] = ['ROLE' => 'USER'];
   if ($role !== 'ADMIN') {
       // La condition n'est pas respectée, on stop la fonction ici
       return $array;
   }
   $array[] = ['ROLE' => 'ADMIN'];
   return $array;
}

var_dump(generateArray('ROLE'));
/* output
array(1) {
 [0]=> array(1) {
   ["ROLE"]=> string(4) "USER"
 }|
}
*/
var_dump(generateArray('ADMIN'));
/* output
array(2) {
```

```
 [0]=> array(1) {
   ["ROLE"]=> string(4) "USER"
 }
 [1]=> array(1) {
   ["ROLE"]=> string(5) "ADMIN"
 }
}
*/
```

Si aucun `return` n'est déclaré alors par défaut la fonction retournera null.

Typer les valeurs de retour

Tout comme les paramètres, vous pouvez typer le retour de votre fonction afin d'avoir un meilleur contrôle de votre code.
Dans le cas ou aucun `return` n'est déclaré à la fin de votre fonction, le type de celle ci sera alors `void`

Pour typer un retour de fonction, il faut tout simplement rajouter à la fin de la fonction " : le type"

```php
/**
* @param int $id
* @return int
*/
function functionTest(int $id): int
{
    // Le retour attendu est un integer
   return $id;
}
```

Si le type de retour ne correspond pas au retour attendu, alors l'erreur suivante se déclenche :

```
PHP Fatal error:  Uncaught TypeError: functionTest(): Return value must be of type int, string returned
```

Fonctions variables

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.

```php
/**
```

```
* @param int $id
* @return int
*/
function testVariable(int $id):int {
   return $id;
}

$func = 'testVariable';
echo $func(5); // Affiche 5
```

Vous pouvez aussi appeler les méthodes d'un objet en utilisant le système des fonctions variables.

```
class testClass {
   /**
    * @param int $id
    * @return int
    */
   function testVariable(int $id):int {
       return $id;
   }
}

$class = new testClass();
$func = 'testVariable';
echo $class->$func(5); // Affiche 5
```

Fonctions anonymes

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 callable

Exemple de callback

```
echo preg_replace_callback(
   '~-([a-z])~', // On ne récupère que les lettres de l'alphabet
   function ($match) {
       return strtoupper($match[1]); // On récupère la première lettre
et on la force en majuscule
   }, 'bonjour-le-monde'); // Affichera bonjourLeMonde
```

Exemple de fonction anonyme utilisé comme valeur de variable

```php
/**
* @param $name
* @return void
*/
$greet = function($name) {
   printf("Bonjour %s\r\n", $name);
};

$greet('World'); // Affiche Bonjour World
$greet('PHP'); // Affiche Bonjour PHP
```

Chapitre 8 : Initiation à la POO, les Classes et les objets

Maintenant que nous avons passer en revu quelques bases du PHP, nous allons pouvoir nous attaquer à très gros morceau qui est la POO

Définition POO

La POO ou programmation Orienté Objet est une méthode de développement qui consiste à construire son code avec des classes et des objets. Cette méthodologie qui semble complexe à première vue est en fait bien plus simple et plus intuitive que la programmation procédurale.

La syntaxe de base

Une classe commence toujours par le mot clé Class et est suivi du nom de la classe. Tout comme les fonctions, le nom de la classe doit respecter des règles. Un nom de classe valide commence par une lettre ou un underscore, suivi de n'importe quel nombre de chiffres, ou de lettres, ou d'underscores.

Une classe peut contenir ses propres variables, constantes que l'on appel des propriétés ou encore attributs et des fonctions que l'on appel méthodes.

```php
class MaClass {

   /**
    * Déclaration d'une constante
    */
   public const MA_CONSTANTE = 12;

   /**
    * Déclaration d'une propriété
    * @var string
```

```
     */
    public string $name;

    /**
     * Déclaration d'une méthode
     * @return string
     */
    public function getName():string
    {
        return $this->name;
    }

}
```

De manière conventionnelle, on définit qu'un fichier = une classe mais il est tout à fait possible de définir plusieurs classes dans un seul fichier.

Le mot clé new

Le mot clé new permet de créer une nouvelle instance de classe, dans ce cas un nouvel objet sera créé. Les classes devraient être définies avant l'instanciation.

```
// J'importe ma classe
use Composer\Util\Filesystem;

// Je créer mon objet
$file = new Filesystem();
```

Le mot-clé extends

Une classe peut hériter des constantes, méthodes et des propriétés d'une autre classe en utilisant le mot-clé extends dans la déclaration. Il n'est pas possible d'étendre plusieurs classes : une classe peut uniquement hériter d'une seule classe de base.

Les constantes, méthodes et propriétés héritées peuvent être redéfinies en les re-déclarant avec le même nom que dans la classe parente. Il est possible d'accéder aux méthodes ou propriétés statiques redéfinies en y faisant référence avec l'opérateur parent::.

Exemple

```
class SimpleClass {
    function displayVar() {
        echo "Classe Parent";
    }
```

```
}

class ExtendClass extends SimpleClass
{
 // Redéfinition de la méthode parente
 function displayVar()
 {
   echo "Classe étendue\n";
   parent::displayVar();
 }
}

$extended = new ExtendClass();
$extended->displayVar();
/* output
Classe étendue
Classe Parent
*/
```

Le mot clé :class

Le mot clé class est également utilisé pour la résolution des noms de classes. Il est possible d'obtenir le nom complètement qualifié d'une classe ClassName en utilisant ClassName::class.

Reprenons l'exemple précédent

```
echo ExtendClass::class; // Affiche ExtendClass
```

Les propriétés

Une propriété est tout simplement une variable au sein de votre classe. A partir de PHP 7.4 il est possible de typer les propriété, il est recommandé de le faire systématiquement.

Exemple de propriétés

```
class myClass {
   /**
    * Propriété de type int
    * @var int
    */
   public int $id;

   /**
    * Propriété de type string
```

```
    * @var string
    */
   public string $name;
}
```

La visibilité des propriétés

par défaut une propriété est publique (`public`) c'est-à-dire qu'elle est accessible partout dans le code.
Il existe deux autres déclaration de visibilité qui sont :

- `protected` : rend accessible la propriété uniquement dans la classe mère ou dans ses enfants.
- `private` : rend accessible la propriété uniquement dans la classe qui l'a défini

Exemple de visibilités :

```
class SimpleClass {

   /**
    * Propriété public
    * @var string
    */
   public string $var1 = 'Je suis public';

   /**
    * Propriété protected
    * @var string
    */
   protected string $var2 = 'Je suis protected';

   /**
    * Propriété private
    * @var string
    */
   private string $var3 = 'Je suis private';

   public function getVar2() {
       return $this->var2;
   }

   public function getVar3() {
       return $this->var3;
   }
}
```

```php
class ExtendClass extends SimpleClass
{
   public function getExtendVar1(): string
   {
       return $this->var1;
   }

   public function getExtendVar2():string
   {
       return $this->var2;
   }

   public function getExtendVar3():string
   {
       return $this->var3;
   }
}

$class = new SimpleClass();
echo $class->var1; // Affiche je suis public
echo $class->var2; // Fatal error cannot access protected property
SimpleClass::$var2
echo $class->var3 // Fatal error cannot access private property
SimpleClass::$var3
echo $class->getVar2(); // Affiche je suis protected
echo $class->getVar3(); // Affiche je suis private
echo $class->getExtendVar1(); // Affiche je suis public
echo $class->getExtendVar2(); // Affiche je suis protected
echo $class->getExtendVar3(); // Fatal error cannot access private
property SimpleClass::$var3
```

Le readOnly

La propriété readOnly empêche la modification de la propriété après l'initialisation.

```php
class Test {
   public readonly string $prop;

   public function __construct(string $prop) {
       // Initialization légale.
       $this->prop = $prop;
   }
}

$test = new Test("foobar");
```

```
// Lecture légale.
var_dump($test->prop); // string(6) "foobar"

// Réaffectation illégal. Ce n'importe pas que la valeur assigné soit
identique.
$test->prop = "foobar";
// Error: Cannot modify readonly property Test::$prop
```

Le parcours d'objets

Il est tout à fait possible, via un foreach de parcourir l'ensemble des propriétés d'un objet en fonction de la visibilité des propriétés

Exemple de code :

```
class MyClass
{
   public string $var1 = 'valeur 1';
   public string $var2 = 'valeur 2';
   public string $var3 = 'valeur 3';

   protected string $protected = 'variable protégée';
   private string $private   = 'variable privée';

   function iterateVisible(): void
   {
       echo "MyClass::iterateVisible:\n";
       foreach ($this as $key => $value) {
           print "$key => $value\n";
       }
   }
}

$class = new MyClass();

foreach($class as $key => $value) {
   print "$key => $value\n";
}
echo "\n";

$class->iterateVisible();
```

Ce qui affiche le résultat suivant :

```
var1 => valeur 1
var2 => valeur 2
```

```
var3 => valeur 3

MyClass::iterateVisible:
var1 => valeur 1
var2 => valeur 2
var3 => valeur 3
protected => variable protégée
private => variable privée
```

Les constantes de classe

Une constante de classe est une valeur qui reste identique et non modifiable. Par défaut, une constante de classe est publique.

Exemple d'utilisation de constantes :

```php
class MyClass
{
   const CONSTANT = 'valeur constante';

   function showConstant() {
       echo  self::CONSTANT;
   }
}

echo MyClass::CONSTANT; // Affiche valeur constante

$classname = "MyClass";
echo $classname::CONSTANT; // Affiche valeur constante

$class = new MyClass();
$class->showConstant(); // Affiche valeur constante

echo $class::CONSTANT; // Affiche valeur constante
```

Constructeurs et destructeurs

Le constructeur

Le constructeur est une méthode qui est appelée à chaque nouvelle instance de l'objet. Le constructeur est utilisé pour initialiser certaines données dont l'objet à besoin pour fonctionner correctement.

Syntaxe de base d'un constructeur

```
__construct(mixed ...$values = ""): void
```

Dans le cas d'une classe enfant, le constructeur parent n'est pas appelé par défaut, il est nécessaire de le déclarer de la façon suivante : `parent::construct()`

Exemple de constructeurs

```php
class BaseClass {
   function __construct() {
       print "Dans le constructeur de BaseClass\n";
   }
}

class SubClass extends BaseClass {
   function __construct() {
       parent::__construct();
       print "Dans le constructeur de SubClass\n";
   }
}

class OtherSubClass extends BaseClass {
   // Constructeur hérité de BaseClass
}

// Dans le constructeur de BaseClass
$obj = new BaseClass();

// Dans le constructeur de BaseClass
// Dans le constructeur de SubClass
$obj = new SubClass();

// Dans le constructeur de BaseClass
$obj = new OtherSubClass();
```

Argument dans un constructeur

Avant PHP 8, voici la méthode la plus simple pour initialiser des propriétés d'une classe

```php
class Point {
   protected int $x;
   protected int $y;

   public function __construct(int $x, int $y = 0) {
       $this->x = $x;
```

```
        $this->y = $y;
    }
}
```

À partir de PHP 8.0.0, les paramètres du constructeur peuvent être promus pour correspondre à une propriété de l'objet. Il est très commun pour les paramètres d'un constructeur d'être assignés à une propriété sans effectuer d'opérations dessus. La promotion du constructeur fournit un raccourci pour ce cas d'utilisation.

Exemple de code en reprenant l'exemple précédent

```
class Point {
    public function __construct(protected int $x, protected int $y = 0) {
    }
}
```

Le destructeur

La méthode destructeur est appelée dès qu'il n'y a plus de référence sur un objet donné, ou dans n'importe quel ordre pendant la séquence d'arrêt.

Exemple d'un destructeur

```
class MyDestructableClass
{
    function __construct() {
        print "In constructor\n";
    }

    function __destruct() {
        print "Destroying " . __CLASS__ . "\n";
    }
}

$obj = new MyDestructableClass();
```

Tout comme avec le constructeur, dans le cas d'une classe enfant, le constructeur parent n'est pas appelé par défaut, il est nécessaire de le déclarer de la façon suivante : `parent::destruct()`

L’héritage

Lorsqu'une classe est étendue, la classe enfant hérite de toutes les méthodes publiques et protégées, propriétés et constantes de la classe parente. Tant qu'une classe n'écrase pas ces méthodes, elles conservent leurs fonctionnalités d'origine.
L'héritage est très utile pour définir et abstraire certaines fonctionnalités communes à plusieurs classes, tout en permettant la mise en place de fonctionnalités supplémentaires dans les classes enfants, sans avoir à réimplémenter en leur sein toutes les fonctionnalités communes.

Les méthodes privées d'une classe parente ne sont pas accessibles à la classe enfant. Par conséquent, les enfants peuvent ré-implémenter une méthode privée eux-mêmes sans se soucier des règles d'héritage normales.

Exemple de code avec héritage

```php
class Foo
{
   public function printItem($string): void
   {
       echo 'Foo: ' . $string;
   }

   public function printPHP(): void
   {
       echo 'PHP est super';
   }
}

class Bar extends Foo
{
   public function printItem($string): void
   {
       echo 'Bar: ' . $string;
   }
}

$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Affiche : 'Foo: baz'
$foo->printPHP();       // Affiche : 'PHP est super'
$bar->printItem('baz'); // Affiche : 'Bar: baz'
$bar->printPHP();       // Affiche : 'PHP est super'
```

L'opérateur de résolution de portée (::)

L'opérateur de résolution de portée ou, en termes plus simples, le symbole "double deux-points" (::), fournit un moyen d'accéder aux membres une constante, une propriété statique, ou une méthode statique d'une classe ou de l'une de ses classes parentes. Lorsque vous référencez ces éléments en dehors de la définition de la classe, utilisez le nom de la classe.

Exemple en dehors d'une classe

```
class MyClass {
    const CONST_VALUE = 'Une constante';
}

$classname = 'MyClass';
echo $classname::CONST_VALUE; // Affiche une constante
echo MyClass::CONST_VALUE; // Affiche une constante
```

Exemple dans une classe

```
class MyClass
{
    const CONST_VALUE = 'Une constante';
}

class OtherClass extends MyClass
{
    public static string $myStatic = 'variable statique';

    public static function doubleColon(): void
    {
        echo parent::CONST_VALUE . ' - ';
        echo self::$myStatic;
    }
}

$classname = 'OtherClass';
$classname::doubleColon(); // Affiche Une constante - variable statique
OtherClass::doubleColon(); // Affiche Une constante - variable statique
```

Le mot clé Static

Le fait de déclarer des propriétés ou des méthodes comme statiques vous permet d'y accéder sans avoir besoin d'instancier la classe. Celles-ci peuvent alors être accédées statiquement depuis une instance d'objet.

Les méthodes statiques

```php
<?php
class Foo
{
   public static function aStaticMethod() {
       // ...
   }
}

Foo::aStaticMethod();
$classname = 'Foo';
$classname::aStaticMethod();
?>
```

Les propriétés statiques

```php
<?php
class Foo
{
   public static $my_static = 'foo';

   public function staticValue() {
       return self::$my_static;
   }
}

class Bar extends Foo
{
   public function fooStatic() {
       return parent::$my_static;
   }
}


print Foo::$my_static . "\n";

$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static . "\n";      // "Propriété" my_static non définie

print $foo::$my_static . "\n";
$classname = 'Foo';
print $classname::$my_static . "\n";
```

```php
print Bar::$my_static . "\n";
$bar = new Bar();
print $bar->fooStatic() . "\n";
?>
```

Abstraction de classes

Les classes définies comme abstraites ne peuvent pas être instanciées, et toute classe contenant au moins une méthode abstraite doit elle-aussi être abstraite. Les méthodes définies comme abstraites déclarent simplement la signature de la méthode

L'intérêt principal de définir une classe comme abstraite va être de fournir un cadre plus strict lors du développement en forçant à définir certaines méthodes.
En effet, une classe abstraite ne peut pas être instanciée directement et contient généralement des méthodes abstraites. L'idée ici va donc être de définir des classes mères abstraites et de pousser les développeurs à étendre ces classes.

```php
abstract class AbstractClass
{
    // Notre méthode abstraite ne doit que définir les arguments requis
    abstract protected function prefixName($name);

}

class ConcreteClass extends AbstractClass
{

    // Notre classe enfant peut définir des arguments optionnels non
présents dans la signature du parent
    public function prefixName($name, $separator = "."): string
    {
        $prefix = match ($name) {
            "Marc" => "Mr",
            "Claire" => "M",
            default => "",
        };
        return $prefix . $separator . ' ' . $name;
    }
}

$class = new ConcreteClass;
echo $class->prefixName("Marc"); // Affiche Mr. Marc
echo $class->prefixName("Claire"); // Affiche M.Claire
```

Interfaces

Les interfaces objet vous permettent de créer du code qui spécifie quelles méthodes une classe doit implémenter, sans avoir à définir comment ces méthodes fonctionneront. Les interfaces partagent l'espace de nom avec les classes et les traits, donc elles ne peuvent pas utiliser le même nom.

On définit une interface par le mot clé `interface` au lieu de `class`.

En pratique les interfaces servent deux rôles complémentaires :

- Permettre aux développeurs de créer des objets de classes différentes qui peuvent être utilisés de façon interchangeable, car elles implémentent la ou les mêmes interfaces. Un exemple commun sont plusieurs services d'accès à des bases de données, plusieurs gestionnaires de paiement ou différentes stratégies de cache. Différentes implémentations peuvent être échangées sans nécessiter des changements dans le code qui les utilisent.
- Pour permettre à une fonction ou méthode d'accepter et opérer sur un paramètre qui conforme à une interface, sans se soucier de quoi d'autre l'objet peut faire ou comment c'est implémenté. Ces interfaces sont souvent nommées Iterable, Cacheable, Renderable, etc. pour décrire la signification de leur comportement.

Exemple d'implémentation d'interface

```
// Declaration de l'interface 'Template'
interface Template
{
   public function setVariable(string $name, mixed $var);

   public function getHtml(string $template);
}

// Implémentation de l'interface
class WorkingTemplate implements Template
{
   private array $vars = [];

   public function setVariable(string $name, mixed $var): void
   {
       $this->vars[$name] = $var;
   }

   public function getHtml(string $template): array|string
   {
       foreach ($this->vars as $name => $value) {
           $template = str_replace('{' . $name . '}', $value,
```

```
$template);
        }
        return $template;
    }
}

// Ceci ne fonctionnera pas
// Fatal error: Class BadTemplate contains 1 abstract methods
// and must therefore be declared abstract (Template::getHtml)
class BadTemplate implements Template
{
    private array $vars = [];

    public function setVariable($name, $var): void
    {
        $this->vars[$name] = $var;
    }
}
```

Les traits

Un trait est semblable à une classe, mais il ne sert qu'à grouper des fonctionnalités d'une manière intéressante. Il n'est pas possible d'instancier un Trait en lui-même. C'est un ajout à l'héritage traditionnel, qui autorise la composition horizontale de comportements, c'est-à-dire l'utilisation de méthodes de classe sans besoin d'héritage.

Exemple de trait

```
trait TraitA {

    public function sayHello(): void
    {
        echo 'Hello';
    }
}

trait TraitB {
    public function sayWorld(): void
    {
        echo 'World';
    }
}

class MyHelloWorld
```

```
{
    use TraitA, TraitB; // Une class peut avoir de multiples traits

    public function sayHelloWorld(): void
    {
        $this->sayHello();
        echo ' ';
        $this->sayWorld();
        echo "!\n";
    }
}

$myHelloWorld = new MyHelloWorld();
$myHelloWorld->sayHelloWorld();
```

Résolution de conflits

Si deux Traits insèrent une méthode avec le même nom, une erreur fatale est levée si le conflit n'est pas explicitement résolu.
Pour résoudre un conflit de nommage entre des Traits utilisés dans la même classe, il faut utiliser l'opérateur insteadof pour choisir une des méthodes en conflit.

Exemple de résolution de conflits

```
trait A {
    public function smallTalk() {
        echo 'a';
    }
    public function bigTalk() {
        echo 'A';
    }
}

trait B {
    public function smallTalk() {
        echo 'b';
    }
    public function bigTalk() {
        echo 'B';
    }
}

class Talker {
    use A, B {
        B::smallTalk insteadof A;
```

```
        A::bigTalk insteadof B;
    }
}

class Aliased_Talker {
    use A, B {
        B::smallTalk insteadof A;
        A::bigTalk insteadof B;
        B::bigTalk as talk;
    }
}
```

Les class anonymes

Les classes anonymes sont utiles lorsque de simples objets uniques ont besoin d'être créés. On peut leur passer des arguments via le constructeur, elles peuvent étendre d'autres classes, implémenter des interfaces ou utiliser des traits comme avec une classe normale.

Exemple de class anonyme

```
// Utilisant une classe explicite
class Logger
{
    public function log($msg): void
    {
        echo $msg;
    }
}

$util->setLogger(new Logger());

// Utilisant une classe anonyme
$util->setLogger(new class {
    public function log($msg): void
    {
        echo $msg;
    }
});
```

Surcharge magique

La surcharge magique en PHP permet de "créer" dynamiquement des propriétés et des méthodes. Ces entités dynamiques sont traitées via des méthodes magiques établies que l'on peut positionner dans une classe pour divers types d'actions.

Les surcharges de propriétés

```
public __set(string $name, mixed $value): void
```

__set() est sollicitée lors de l'écriture de données vers des propriétés inaccessibles (protégées ou privées) ou inexistantes.

```
public __get(string $name): mixed
```

__get() est appelée pour lire des données depuis des propriétés inaccessibles (protégées ou privées) ou inexistantes.

```
public __isset(string $name): bool
```

__isset() est sollicitée lorsque isset() ou empty() sont appelées sur des propriétés inaccessibles (protégées ou privées) ou inexistantes.

```
public __unset(string $name): void
```

__unset() est invoquée lorsque unset() est appelée sur des propriétés inaccessibles (protégées ou privées) ou inexistante.

L'argument $name est le nom de la propriété avec laquelle on interagit. L'argument $value de la méthode __set() spécifie la valeur à laquelle la propriété $name devrait être définie.

Exemple de code :

```
class PropertyTest
{
   /**  Variable pour les données surchargées.  */
   private array $data = array();

   /**  La surcharge n'est pas utilisée sur les propriétés déclarées.
*/
   public int $declared = 1;

   /**  La surcharge n'est lancée que lorsque l'on accède à cette
propriété depuis l'extérieur de la classe.  */
   private int $hidden = 2;

   public function __set($name, $value)
   {
       echo "Définition de '$name' à la valeur '$value'\n";
```

```
        $this->data[$name] = $value;
    }

    public function __get($name)
    {
        echo "Récupération de '$name'\n";
        if (array_key_exists($name, $this->data)) {
            return $this->data[$name];
        }

        $trace = debug_backtrace();
        trigger_error(
            'Propriété non-définie via __get() : ' . $name .
            ' dans ' . $trace[0]['file'] .
            ' à la ligne ' . $trace[0]['line'],
            E_USER_NOTICE);
        return null;
    }

    public function __isset($name)
    {
        echo "Est-ce que '$name' est défini ?\n";
        return isset($this->data[$name]);
    }

    public function __unset($name)
    {
        echo "Effacement de '$name'\n";
        unset($this->data[$name]);
    }

    /**  Ce n'est pas une méthode magique, nécessaire ici que pour
l'exemple.  */
    public function getHidden(): int
    {
        return $this->hidden;
    }
}


echo "<pre>\n";

$obj = new PropertyTest;

$obj->a = 1;
echo $obj->a . "\n\n";
```

```php
var_dump(isset($obj->a));
unset($obj->a);
var_dump(isset($obj->a));
echo "\n";

echo $obj->declared . "\n\n";

echo "Manipulons maintenant la propriété privée nommée 'hidden' :\n";
echo "'hidden' est visible depuis la classe, donc __get() n'est pas
utilisée...\n";
echo $obj->getHidden() . "\n";
echo "'hidden' n'est pas visible en dehors de la classe, donc __get()
est utilisée...\n";
echo $obj->hidden . "\n";
```

Ce qui va afficher

```
Définition de 'a' à '1'
Récupération de 'a'
1

Est-ce que 'a' est défini ?
bool(true)
Effacement de 'a'
Est-ce que 'a' est défini ?
bool(false)

1

Manipulons maintenant la propriété privée nommée 'hidden' :
'hidden' est visible depuis la classe, donc __get() n'est pas
utilisée...
2
'hidden' n'est pas visible en dehors de la classe, donc __get()
est utilisée...
Récupération de 'hidden'


Notice: Propriété non-définie via __get() : hidden dans <file> à
la ligne 64 dans <file> à la ligne 28
```

Surcharge de méthodes

```
public __call(string $name, array $arguments): mixed
```

__call() est appelée lorsque l'on invoque des méthodes inaccessibles dans un contexte objet.

```
public static __callStatic(string $name, array $arguments): mixed
```

__callStatic() est lancée lorsque l'on invoque des méthodes inaccessibles dans un contexte statique.

Exemple de code :

```php
class MethodTest
{
    public function __call($name, $arguments)
    {
        // Note : la valeur de $name est sensible à la casse.
        echo "Appel de la méthode '$name' "
             . implode(', ', $arguments). "\n";
    }

    public static function __callStatic($name, $arguments)
    {
        // Note : la valeur de $name est sensible à la casse.
        echo "Appel de la méthode statique '$name' "
             . implode(', ', $arguments). "\n";
    }
}

$obj = new MethodTest;
$obj->runTest('dans un contexte objet');

MethodTest::runTest('dans un contexte static');
```

Ce qui va afficher

```
Appel de la méthode 'runTest' dans un contexte objet
Appel de la méthode statique 'runTest' dans un contexte static
```

Les méthodes magiques

https://www.php.net/manual/fr/language.oop5.magic.php