

La programmation Orientée Objet : concepts de base

Table des matières

I. Contexte	3
II. Introduction à la Programmation Orientée Objet	3
III. Exercice : Appliquez la notion	4
IV. Les classes	5
V. Exercice : Appliquez la notion	8
VI. Les instances	8
VII. Exercice : Appliquez la notion	11
VIII. Les méthodes	11
IX. Exercice : Appliquez la notion	13
X. Les constructeurs	14
XI. Exercice : Appliquez la notion	16
XII. Les destructeurs	16
XIII. Exercice : Appliquez la notion	18
XIV. Une classe : un nouveau type	19
XV. Exercice : Appliquez la notion	22
XVI. Auto-évaluation	23
A. Exercice final	23
B. Exercice : Défi.....	25
Solutions des exercices	27

I. Contexte

Durée : 1 h

Environnement de travail : Local

Pré-requis : Avoir installé et configuré un serveur de test local

Contexte

La Programmation Orientée Objet est un **paradigme de programmation**, c'est-à-dire une **manière précise d'architecturer son code**.

Ce terme peut paraître complexe, mais c'est en réalité très simple : depuis le début de votre apprentissage du PHP, sans même vous en rendre compte, vous utilisiez déjà un autre paradigme, appelé **programmation procédurale**.

Cela signifie simplement que le code est séparé en fonctions. La Programmation Procédurale est performante et très efficace pour les petits projets, mais arrive rapidement à ses limites.

Dans ce chapitre, nous allons voir quelles sont les limites du procédural et comment la Programmation Orientée Objet nous aide à les surmonter.

II. Introduction à la Programmation Orientée Objet

Objectifs

- Comprendre les limites de la programmation procédurale
- Découvrir la Programmation Orientée Objet
- Apprendre à créer des classes et des objets

Mise en situation

Jusqu'à présent, les données utilisées étaient stockées dans des tableaux associatifs. Des fonctions permettaient de manipuler ces tableaux et d'accéder aux valeurs qu'ils contenaient.

Méthode Les limites de tableaux associatifs

Prenons le code suivant comme exemple pour illustrer cela.

Imaginons qu'une fonction `loadUser` ait été définie dans le fichier `userLoading.php` et que celle-ci retourne un tableau contenant le nom et prénom d'un utilisateur donné en paramètre.

```
1 <?php
2
3 function sayHello(array $user): void
4 {
5     echo 'Bonjour ' . $user['name'] . ' ' . $user['surname'];
6 }
7
8 include("functions/userLoading.php");
9
10 $user1 = loadUser(1);
11 $user2 = loadUser(2);
12
13 sayHello($user1); // Affiche "Bonjour Robert Dupont"
```

```
14 sayHello($user2); // Affiche "Bonjour Laure Dupond"
```

Admettons maintenant que nous devions ajouter une fonction `sayGoodbye` qui s'afficherait au moment de la déconnexion de notre utilisateur.

Elle prendrait en paramètre un utilisateur et afficherait "Au revoir", suivi d'une appellation ("Monsieur", "Madame" par exemple)

```
1 <?php
2
3 function sayGoodbye(array $user): void
4 {
5     echo 'Au revoir ! '.$user['...']; // Comment déterminer où retrouver cette appellation ?
6 }
```

Il nous manque une information : on ne sait pas à quel champ correspond cette appellation.

Pour la trouver, nous pourrions aller voir comment l'utilisateur est chargé dans la fonction `loadUser`, pour récupérer tous les champs qui le composent.

Il serait également possible d'utiliser la fonction `var_dump` pour afficher la structure de `$user`, mais ce n'est vraiment pas pratique.

C'est une limite des tableaux associatifs : il n'est pas possible de connaître leur structure à l'avance.

Autre problème : que se passerait-il si on appelait la fonction `sayHello()` avec un tableau qui contiendrait d'autres données que celles d'un utilisateur ? Au mieux, le comportement ne sera pas celui attendu, mais, dans le pire des cas, cela provoquerait une erreur à l'exécution, que l'on n'aurait pas pu détecter avant.

La Programmation Orientée Objet (POO) va nous permettre de créer des structures fixes que l'on va pouvoir utiliser dans nos fonctions.

Syntaxe À retenir

- Les tableaux associatifs sont très utiles, mais leur structure dynamique fait qu'ils ne sont pas très pratiques à utiliser pour les applications plus conséquentes.
- Il faudrait alors un outil permettant de définir des structures plus précises et plus faciles à manipuler : c'est là que la POO entre en jeu.

Complément

La syntaxe de base (documentation officielle)¹

III. Exercice : Appliquez la notion

Dans cet exercice, nous allons prendre un cas pratique pour démontrer les limites des tableaux associatifs.

Pour réaliser cet exercice, vous pouvez travailler sur l'environnement de travail :

<https://sandbox.onlinephpfunctions.com/>

Replit a sa version PHP en 7.2 certains scripts exemples auront besoin d'être exécutés sous PHP 7.4 minimum.

¹ <https://www.php.net/manual/fr/language.oop5.basic.php>

Question 1

[solution n°1 p.29]

Nous sommes en charge d'une application gérant le stock de produits informatiques. Pour simuler le fait que nous recevons des informations d'une base de données distante (et à laquelle nous n'avons pas accès), nous allons utiliser la ligne de code suivante :

```
1 <?php
2
3 $product =
    json_decode(base64_decode('eyJyZW1lIjoiQ2FydGUgbVx1MDBlOHJlIiwic3RvY2siOiIxMiIsImJyYW5kIjoiQXN1cyJ9'),
    true);
```

`$product` est un tableau associatif contenant les données d'un produit que nous avons chargé dans notre base de données.

Affichez le nom du produit, suivi de sa marque et de la quantité qu'il reste en stock pour ce produit.

Indice :

Vous pouvez utiliser `var_dump` pour visualiser la structure du produit.

Question 2

[solution n°2 p.29]

Une mise à jour de la base de données a eu lieu pour rajouter l'information du prix. Le code a également évolué pour utiliser la Programmation Orientée Objet plutôt que les tableaux associatifs. Ainsi, vous tombez sur le code suivant :

```
1 <?php
2
3 class Product {
4     public string $name;
5     public string $brand;
6     public string $stock;
7     public float $price;
8 }
```

Sans même vous y connaître encore en POO, sauriez-vous déterminer quel champ va contenir le prix de notre produit ?

Indice :

Ne regardez que ce qui ressemble à des variables pour le moment. Ignorez tous les autres mots-clés, nous allons voir ça plus tard.

IV. Les classes

Objectifs

- Comprendre les limites de la programmation procédurale
- Découvrir la programmation orientée objet

Mise en situation

Maintenant que nous avons vu quelles problématiques la POO va nous aider à régler, il est temps de découvrir ce qu'est la "Programmation Orientée Objet".

Définition La POO : un moyen de structurer ses données

Prenons un exemple éloigné de l'informatique pour illustrer ce principe : imaginons que nous sommes des fabricants de vases, nous avons quelques modèles qui peuvent chacun se décliner en plusieurs couleurs, mais tous les modèles ont la même forme.



Nous pourrions faire chacun de nos vases à la main en modelant et colorant nous-même la matière première.

Le problème avec cette façon de faire, c'est que cela nous prendrait un temps très long, sans compter les petites imperfections qui pourraient apparaître d'un vase à l'autre.

Il serait donc plus efficace de créer un moule pour chaque forme de vase que l'on propose : il nous suffirait ensuite de couler la matière de la couleur souhaitée à l'intérieur et de laisser sécher.

La Programmation Orientée Objet, c'est exactement ça : nous allons définir une structure (donc un "moule") qui va définir les données (la "forme") dont nous allons avoir besoin.

Cette structure s'appelle une **classe** et nous allons l'utiliser pour créer des **objets** (les "vases").

Méthode Créer une classe vide

Reprenons nos utilisateurs : ceux-ci disposent tous d'un nom, d'un prénom et d'un identifiant permettant de les retrouver.

Le but sera donc de définir une structure (une **classe**) qui va avoir les champs "nom", "prénom" et "identifiant".

Pour cela, nous avons besoin du mot-clé `class` suivi du nom de la classe écrit en *PascalCase*. Comme pour une fonction, le contenu de la classe va être entouré d'accolades.

Une classe vide ressemble donc à ça :

```
1 <?php
2
3 class User
4 {
5     // Le contenu de la classe ira ici
6 }
```

Remarque

Il est possible de définir une classe dans n'importe quel fichier, même au milieu du code (comme les fonctions). Cependant, avoir un fichier par classe et utiliser le nom de la classe comme nom du fichier est une bonne pratique.

Pour l'exemple, on considérera donc que notre classe `User` est présente dans un fichier `User.php` qui lui est propre.

Définition Les propriétés

Ajoutons donc un nom, un prénom et un identifiant à notre classe. Pour cela, nous allons déclarer des variables internes à l'intérieur de notre classe. Ces variables internes s'appellent des **propriétés**.

Syntaxe

Une propriété est déclarée par le mot-clé `public` suivi de son type et de son nom, et doit respecter les conventions de nommage des variables traditionnelles.

Il est également possible de donner une valeur par défaut à un attribut en la lui affectant.

La classe `User` ressemblera donc à cela :

```
1 <?php
2
3 class User
4 {
5     public int $id; // Cet attribut n'a pas de valeur par défaut
6     public string $name = 'John'; // Le prénom par défaut de nos utilisateurs est "John"
7     public string $surname = 'Doe'; // Le nom par défaut de nos utilisateurs est "Doe"
8 }
```

Remarque

En réalité, le mot-clé `public` fait un peu plus que simplement déclarer un attribut, mais il s'agit là d'un domaine plus avancé de la POO.

Pour simplifier, partons du principe qu'il ne sert qu'à cela pour le moment. En revanche, ne soyez pas étonné de voir d'autres mots-clés plus tard.

Attention

La syntaxe des classes en PHP a beaucoup évolué avec le temps.

Pour que cette syntaxe fonctionne, assurez-vous que la version de PHP que vous utilisez sur votre environnement est supérieure ou égale à PHP 7.4, sinon le code ci-dessus ne fonctionnera pas.

Il est recommandé d'avoir votre propre serveur en local pour ce cours plutôt que d'utiliser des solutions en ligne (comme repl.it), qui pourraient ne pas proposer la version adaptée.

Complément

Notre classe n'est pas utilisable en tant que telle : elle ne représente qu'une structure, pas des données.

C'est l'équivalent du moule de nos vases : nous avons un moule d'utilisateur, mais, avant de pouvoir l'utiliser, il va falloir créer des utilisateurs à partir de ce moule.

Syntaxe **À retenir**

- Pour créer une classe, on crée un nouveau fichier qui a le nom de la classe, et on utilise le mot-clé `class`.
- Le contenu de la classe est entre accolades.
- Pour déclarer des **propriétés**, on utilise le mot-clé `public`, suivi du type de propriété, ainsi que de son nom.
- Il est possible d'affecter une valeur par défaut à une propriété.

```
1 <?php
2
3 class MaClasse
4 {
5     public int $propriete1;
6     public string $propriete2 = 'Valeur par défaut';
7 }
```

Complément

La syntaxe de base (documentation officielle)¹

V. Exercice : Appliquez la notion

Nous souhaitons gérer les caractéristiques de différentes motos : leur marque, leur couleur (pour les distinguer sur la piste) et leur vitesse maximale.

Pour réaliser cet exercice, vous pouvez travailler sur l'environnement de travail :

<https://sandbox.onlinephpfunctions.com/>

Replit a sa version PHP en 7.2 certains scripts exemples auront besoin d'être exécutés sous PHP 7.4 minimum.

Question

[solution n°3 p.29]

Créer une classe `Moto` permettant de définir cette structure.

Indice :

La marque et la couleur pourront être des chaînes de caractères, tandis que la vitesse un nombre flottant.

VI. Les instances

Objectifs

- Apprendre à instancier des objets
- Manipuler les propriétés des objets

Mise en situation

Nous avons appris à définir une structure contenant des propriétés. Mais, de la même façon qu'une fonction ne se déclenche que lorsqu'on l'appelle, les classes doivent être "appelées" pour être utilisées.

Fondamental **Différencier le contenant du contenu**

Prenons notre classe `User`, par exemple :

```
1 <?php
2
3 class User
4 {
5     public int $id;
6     public string $name = 'John';
7     public string $surname;
8 }
```

Cette classe ne représente pas un utilisateur en particulier, mais la structure générale de nos utilisateurs.

Nous savons qu'ils disposeront tous d'un nom, d'un prénom et d'un id. Cependant, nous ne connaissons pas encore leurs valeurs.

Bien que certaines propriétés aient des valeurs par défaut, elles ne représentent pas les valeurs réelles de tous nos utilisateurs.

Nous allons voir comment transformer notre classe en objet manipulable et leur donner de vraies valeurs.

¹ <https://www.php.net/manual/fr/language.oop5.basic.php>

Méthode **Instancier un objet**

En créant notre classe, nous avons aussi créé une fonction éponyme (ici, `User()`) qui permet de créer un **objet**.

La seule différence avec une fonction classique est que nous devons utiliser le mot-clé `new` au moment de l'appel : il permet de faire comprendre à PHP que l'on va créer un nouvel objet. On appelle ça **instancier** un objet.

```
1 <?php
2
3 require_once('User.php'); // Si notre classe est dans un fichier à part, il faut l'inclure
4
5 $user = new User(); // On remarque le mot-clé "new" suivi de l'appel à la fonction
   correspondant à notre classe
```

Méthode **Accéder aux valeurs d'un objet**

Ici, la variable `$user` contient un objet de type "User" possédant un champ "id", un champ "name" et un champ "surname".

Pour accéder aux valeurs de ces champs, il faut utiliser l'opérateur `->` :

```
1 <?php
2
3 include('User.php');
4
5 $user = new User();
6 echo $user->name; // Affiche "John", qui était la valeur par défaut de notre champ
```

Méthode **Assigner une valeur à une propriété**

Il est possible d'assigner une valeur à une propriété comme on le ferait pour une variable :

```
1 <?php
2
3 include('User.php');
4
5 $robert = new User(); // On crée une instance de User qu'on assigne à la variable $robert
6 $robert->name = 'Robert'; // Le champ "name" de la variable $robert contient maintenant
   "Robert", écrasant la valeur par défaut "John".
7
8 $laure = new User(); // On crée une deuxième instance de User qu'on assigne à la variable
   $laure
9 $laure->name = 'Laure'; // Le champ "name" de la variable $laure contient maintenant "Laure",
   écrasant la valeur par défaut "John".
10 $laure->surname = 'Dupond'; // Le champ "surname" de la variable $laure contient maintenant
   "Dupond"
11
12 echo $laure->name; // Affiche "Laure"
13 echo $robert->name; // Affiche "Robert"
14
15 // Avec le même "moule", on a créé deux objets : Laure et Robert !
```

Remarque **Qu'est-ce qu'une instance ?**

Si vous jouez à des jeux massivement multi-joueurs, vous êtes sûrement déjà familier avec la notion d'instance.

Dans ce type de jeu, un "donjon" est un type de terrain qui n'est accessible, à un moment donné, que pour un groupe de joueurs donné.

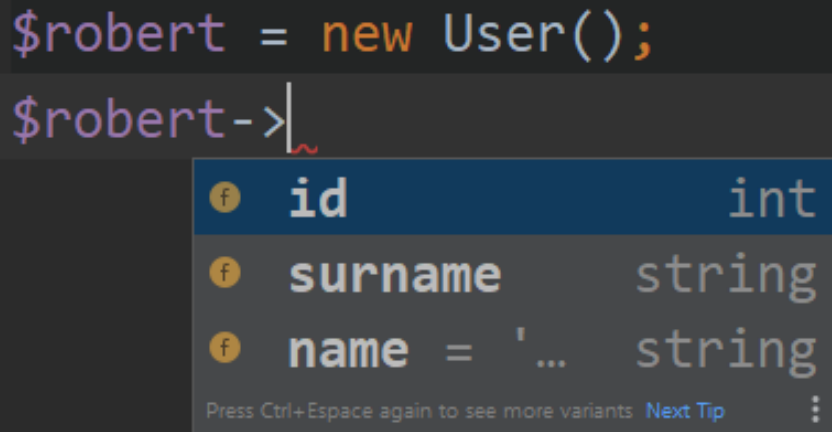
On dit alors que les donjons sont "instanciés" pour chaque groupe de joueurs, c'est-à-dire qu'il existe un "moule" de donjon qui est systématiquement dupliqué, pour permettre à différents groupes de jouer sur un même type de carte, sans pour autant se croiser et intervenir sur la partie d'autres joueurs.

C'est exactement le même principe ici, d'où le terme "d'instance" que l'on retrouve à la fois dans les jeux et dans la programmation.

Conseil L'un des avantages d'utiliser des objets avec un IDE

Le fait d'utiliser une classe permet de savoir à tout moment ce que contient notre variable, sans avoir besoin d'aller fouiller dans des `var_dump` ou des bases de données.

Un IDE sera capable d'apporter son aide lors de l'utilisation d'un objet en proposant les propriétés qu'il est possible d'utiliser.



```
$robert = new User();
$robert->
```

f	id	int
f	surname	string
f	name = '...	string

Press Ctrl+Space again to see more variants Next Tip

Syntaxe À retenir

- Pour créer un nouvel objet, on utilise le mot-clé `new` en appelant une fonction qui a le même nom que la classe.
- Une fois l'objet créé, il est possible d'accéder à ses propriétés grâce à l'opérateur `->`.
- Il est possible de donner une valeur à une propriété en utilisant l'opérateur d'affectation.

```
1 <?php
2
3 class MaClasse
4 {
5     public int $propriete1;
6     public string $propriete2;
7 }
8
9 $objet = new MaClasse(); // On crée un objet à partir d'une classe
10 $objet->propriete1 = 'valeur'; // Affectation
11 echo $objet->propriete1; // Affiche 'valeur'
```

Complément

La syntaxe de base (documentation officielle)¹

¹ <https://www.php.net/manual/fr/language.oop5.basic.php>

VII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous pouvez travailler sur l'environnement de travail :

<https://sandbox.onlinephpfunctions.com/>

Replit a sa version PHP en 7.2 certains scripts exemples auront besoin d'être exécutés sous PHP 7.4 minimum.

Question 1

[solution n°4 p.29]

Soit la classe `Moto` suivante :

```
1 <?php
2 // fichier Moto.php
3 class Moto
4 {
5     public string $brand;
6     public string $color;
7     public float $maxSpeed;
8 }
```

Importez cette classe dans votre script et instanciez une moto "Yamaha" de couleur "rouge" et allant à 210 km/h maximum.

Question 2

[solution n°5 p.29]

Créez maintenant une "Suzuki" "bleue" allant à 220 km/h maximum.

VIII. Les méthodes

Objectifs

- Ajouter des méthodes dans nos classes et les utiliser
- Organiser son code proprement avec le paradigme POO

Mise en situation

Pour le moment, nous avons utilisé nos classes comme de purs éléments de structuration de données : nous avons défini une structure composée de plusieurs variables, que nous avons remplies.

C'est déjà très utile, mais il est possible d'aller plus loin : en plus de déclarer des variables dans des classes, il est possible d'y déclarer des fonctions, qui vont permettre d'interagir avec les différentes propriétés.

Définition	Méthode
------------	---------

Une **méthode** est une fonction interne à une classe. Comme pour les propriétés, elle se déclare grâce au mot-clé `public`.

Une fois une méthode déclarée, elle peut être appelée depuis un **objet** via l'opérateur `->`.

Une méthode a accès aux informations définies dans la classe dans laquelle elle se trouve, c'est-à-dire les propriétés et les autres méthodes.

Pour cela, une variable appelée `$this` est automatiquement injectée dans les méthodes. Cette variable contient l'**objet courant**, c'est-à-dire l'objet qui va appeler notre méthode.

Méthode Ajouter une méthode à une classe

Créons une méthode `sayHello()` dans la classe `User`, qui va retourner la phrase de bienvenue à afficher à notre utilisateur.

```
1 <?php
2
3 // Fichier User.php
4
5 class User
6 {
7     public int $id;
8     public string $name = 'John';
9     public string $surname;
10
11     public function sayHello(): string
12     {
13         return 'Bonjour '.$this->name.' '.$this->surname.PHP_EOL; // Nous utilisons $this pour
14         récupérer le nom et le prénom de l'objet appelant.
15     }
16 }
```

La méthode peut ensuite être appelée depuis un objet :

```
1 <?php
2
3 // Fichier index.php
4
5 require_once 'User.php';
6
7 $robert = new User();
8 $robert->name = 'Robert';
9 $robert->surname = 'Rondon';
10
11 echo $robert->sayHello(); // Affiche "Bonjour Robert Rondon". Lors de l'appel, la variable
12 $this de sayHello contient les valeurs de $robert, puisque c'est cette variable qui appelle la
13 méthode
14
15 $laure = new User();
16 $laure->name = 'Laure';
17 $laure->surname = 'Dupont';
18
19 echo $laure->sayHello(); // Affiche "Bonjour Laure Dupont". Lors de l'appel, la variable $this
20 de sayHello contient les valeurs de $laure, puisque c'est cette variable qui appelle la
21 méthode
```

Contrairement à une fonction normale, il n'est pas nécessaire de passer les données de l'utilisateur en paramètres de notre méthode.

En effet, étant donné qu'elle a accès aux propriétés de notre objet, elle dispose déjà de toutes les informations nécessaires grâce à la variable `$this`.

Fondamental

La POO est un moyen très efficace d'architecturer notre code : elle permet de regrouper dans un même endroit les données dont nous avons besoin et les fonctions les utilisant. C'est ce qu'on appelle l'**encapsulation**.

C'est une manière complètement différente de penser, mais très puissante une fois maîtrisée.

Syntaxe **À retenir**

- Une classe peut avoir des fonctions internes appelées **méthodes**, déclarées avec le mot-clé `public`.
- Ces méthodes peuvent accéder à l'objet appelant via la variable `$this`, qui est automatiquement injectée dans chaque méthode.

```
1 <?php
2
3 class MaClasse
4 {
5     public string $propriete;
6
7     public function maMethode(): void
8     {
9         echo $this->propriete;
10    }
11 }
12
13 $objet = new MaClasse();
14 $objet->propriete = 'valeur';
15 $objet->maMethode(); // Affiche "Valeur"
16
```

Complément

La syntaxe de base (documentation officielle)¹

IX. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous pouvez travailler sur l'environnement de travail :

<https://sandbox.onlinephpfunctions.com/>

Replit a sa version PHP en 7.2 certains scripts exemples auront besoin d'être exécutés sous PHP 7.4 minimum.

Question

[solution n°6 p.29]

Reprenons la classe `Moto` de la partie précédente :

```
1 <?php
2 // fichier Moto.php
3 class Moto
4 {
5     public string $brand;
6     public string $color;
7     public float $maxSpeed;
8 }
```

Lorsque notre course sera finie, nous allons devoir présenter la moto qui aura terminé première.

Pour cela, créez une méthode `getDescription()` dans la classe `Moto` permettant de retourner une description de notre moto, par exemple "Honda verte ayant une vitesse maximale de 214 km/h".

Puis instanciez une Kawasaki noire allant à 238 km/h et affichez sa présentation.

¹ <https://www.php.net/manual/fr/language.oop5.basic.php>

X. Les constructeurs

Objectifs

- Doter des classes de constructeurs
- Créer un objet en utilisant un constructeur

Mise en situation

En l'état, l'initialisation de nos objets peut s'avérer fastidieuse. Fort heureusement, il existe une méthode pour faciliter cette étape.

Remarque Simplifier la construction d'un objet

Nos objets commencent à devenir de plus en plus conséquents et permettent de faire de plus en plus de choses. Cependant, pour le moment, initialiser nos objets est quelque chose d'assez fastidieux : il faut une ligne de code pour instancier notre objet, puis une ligne de code par propriété.

```
1 <?php
2
3 $laure = new User(); // On crée l'objet
4 $laure->name = 'Laure'; // On initialise la propriété "name"
5 $laure->surname = 'Dupond'; // On initialise la propriété "surname"
6 // ... Il faudrait ajouter autant de lignes que de propriétés
7
8 // Et refaire la même chose pour tous nos objets
9 $robert = new User();
10 $robert->name = 'Robert';
11 $robert->surname = 'Dupont';
12 // ...
```

Notre utilisateur n'a, pour le moment, que quelques propriétés, donc ce n'est pas forcément gênant.

Cependant, dans la pratique, nous allons être amenés à manipuler des classes ayant beaucoup plus de champs. Initialiser de telles classes pourrait ainsi devenir très verbeux, mais aussi source d'erreurs ou d'oublis.

Heureusement, il existe un moyen beaucoup plus simple pour initialiser un objet : les **constructeurs**.

Méthode Ajouter un constructeur à une classe

Un constructeur est une méthode particulière, toujours nommée `__construct`.

Cette méthode va s'occuper d'initialiser notre objet en renseignant une valeur à chaque propriété.

Elle va prendre en paramètres les valeurs que l'on veut donner à nos propriétés et va s'occuper d'affecter chaque valeur à la bonne propriété en utilisant la variable `$this`.

```
1 <?php
2
3 class User
4 {
5     public int $id;
6     public string $name = 'John';
7     public string $surname;
8
9     // On déclare notre constructeur
10    public function __construct(string $name, string $surname)
11    {
12        $this->name = $name; // La propriété "name" de l'objet courant prend la valeur du
    paramètre $name
```

```
13     $this->surname = $surname; // La propriété "surname" de l'objet courant prend la
14     valeur du paramètre $surname
15 }
16 public function sayHello(): string
17 {
18     return 'Bonjour ' . $this->name . ' ' . $this->surname;
19 }
20 }
```

Le constructeur est automatiquement appelé au moment de faire le `new`, et les paramètres sont passés dans la fonction qui a le même nom que notre classe.

```
1 <?php
2
3 $laure = new User('Laure', 'Dupond'); // On passe directement nos valeurs dans la "fonction"
4 $robert = new User('Robert', 'Rondon');
5 // Nos objets sont instanciés en une seule ligne !
6
7 echo $robert->sayHello(); // Affiche bien "Bonjour Robert Rondon"
```

Fondamental

Un constructeur ne sert pas seulement à initialiser des propriétés.

Dans des cas complexes, c'est aussi lui qui peut réaliser diverses actions nécessaires au bon fonctionnement de notre objet : vérifier les connexions qui seront utilisées par l'objet, écrire dans un fichier pour logger l'appel à notre objet, etc.

Syntaxe À retenir

- Les classes peuvent avoir un **constructeur** : `__construct`.
- Cette méthode est appelée au moment d'un `new` et permet d'initialiser les propriétés de l'objet avec des valeurs données en paramètres.

```
1 <?php
2
3 class MaClasse
4 {
5     public string $propriete1;
6
7     public function __construct(string $valeurPropriete1)
8     {
9         $this->propriete1 = $valeurPropriete1;
10    }
11 }
12
13 $objet = new MaClasse('valeur');
14
```

Complément

Les constructeurs et destructeurs (documentation officielle)¹

¹ <https://www.php.net/manual/fr/language.oop5.decon.php>

XI. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous pouvez travailler sur l'environnement de travail :

<https://sandbox.onlinephpfunctions.com/>

Replit a sa version PHP en 7.2 certains scripts exemples auront besoin d'être exécutés sous PHP 7.4 minimum.

Question

[solution n°7 p.30]

Agrémentez notre classe `Moto` d'un constructeur pour pouvoir créer des objets plus simplement :

```
1 <?php
2
3 // fichier Moto.php
4 class Moto
5 {
6     public string $brand;
7     public string $color;
8     public float $maxSpeed;
9
10    public function getDescription(): string
11    {
12        return $this->brand.' '.$this->color.' ayant une vitesse maximale de '.$this->maxSpeed.' km/h'.PHP_EOL;
13    }
14 }
```

Puis instanciez une Piaggio violette allant à 217 km/h et affichez sa description.

XII. Les destructeurs

Objectifs

- Comprendre la notion de méthode magique
- Doter des classes de destructeurs

Mise en situation

Il existe certaines méthodes qui sont appelées automatiquement selon l'état de l'objet : on appelle cela les méthodes magiques.

Définition Les méthodes magiques

En PHP, les méthodes commençant par deux *underscores* (comme `__construct`) sont appelées **méthodes magiques**.

Cela signifie qu'elles ne sont **jamais appelées directement par le développeur**, mais qu'elles se déclenchent automatiquement dans des cas précis.

Par exemple, dans le cas du constructeur, on ne fait jamais un appel explicite à `$monObjet->__construct()`.

C'est simplement le fait de faire un `new MaClasse()` qui provoque cet appel : le constructeur est appelé "magiquement".

Il existe d'autres méthodes magiques qu'il est possible d'implémenter dans nos objets, comme les **destructeurs**.

Définition La notion de destructeur

Le **destructeur** permet de définir le comportement de notre objet au moment où il est détruit.

Il s'utilise en implémentant la méthode magique `__destruct`. Cette méthode est appelée lorsqu'un objet n'existe plus du tout dans notre code, c'est-à-dire :

- À la fin de notre script PHP (que ce soit en cas de fin normale du script ou lors d'une erreur fatale, de l'utilisation d'un `die` ou d'un `exit`). Cette méthode se lance automatiquement.
- Lorsqu'il n'y a plus aucune variable qui contient notre objet (par un `unset()` ou un écrasement de valeur)

```
1 <?php
2
3 class DestroyMe {
4     function __destruct() {
5         echo 'Je suis mort !';
6     }
7 }
8
9 $destroyMe1 = new DestroyMe();
10 unset($destroyMe1); // Affiche "Je suis mort !"
11
12 $destroyMe2 = new DestroyMe();
13
14 $destroyMe3 = new DestroyMe();
15 $destroyMe3 = null; // Affiche "Je suis mort !"
16
17
18
19 //Affiche "Je suis mort !" car le script est terminé, donc l'objet $destroyMe2 est détruit
    automatiquement
```

Utilisée beaucoup plus rarement, elle a quand même son utilité lorsqu'une action particulière doit être réalisée avant de complètement supprimer l'objet.

C'est le cas par exemple lorsque l'on travaille avec des fichiers ou des bases de données, donc des ressources externes : la connexion est ouverte dans le constructeur et est fermée dans le destructeur, pour s'assurer que la ressource soit libérée à la fin du script.

Complément

Il existe beaucoup d'autres méthodes magiques qu'il est possible d'implémenter dans certaines situations.

Il est par exemple possible de définir le comportement d'un objet lorsque l'on appelle une méthode ou une propriété qui n'existe pas, ou lorsque l'on essaye de convertir l'objet en chaîne de caractères.

Attention toutefois : la seule méthode magique considérée comme étant une bonne pratique de développement est l'utilisation d'un constructeur.

L'utilisation des autres méthodes magiques, bien qu'elles peuvent parfois se révéler utiles, sont considérées comme de mauvaises pratiques, justement à cause du caractère "magique" des appels.

Exemple

Imaginons que vous récupériez ce morceau de code :

```
1 <?php
2
3 $user = new User('John');
4 $user->echo();
```

Et que ce code, à l'exécution, affiche "John". Jusque-là, rien d'anormal, mais en allant voir le contenu de la classe `User`, quelque chose paraît étrange :

```
1 <?php
2
3 class User
4 {
5     public string $name;
6
7     public function __construct(string $name) {
8         $this->name = $name;
9     }
10
11     public function __call(string $name, array $arg) {
12         eval($name.'('.$this->name.')');
13     }
14 }
```

La méthode `echo` n'existe pas !

Nous pourrions la chercher pendant des heures, mais c'est en fait la méthode magique `__call` qui est appelée automatiquement.

En effet, `__call` est appelée lorsque l'on essaye d'appeler une méthode qui n'existe pas dans notre objet. Mais comme tout cela est "magique", c'est difficile à suivre. Attention à l'utilisation des méthodes magiques !

Syntaxe À retenir

- Une méthode magique est une méthode qui n'est jamais appelée explicitement, mais appelée automatiquement dans certains cas.
- Le constructeur est un exemple de méthode magique, mais il en existe d'autres, comme le **destructeur** `__destruct`, qui est appelé lorsque l'objet est détruit.

Complément

Les méthodes magiques¹

Les constructeurs et destructeurs (documentation officielle)²

XIII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous pouvez travailler sur l'environnement de travail :

<https://sandbox.onlinephpfunctions.com/>

Replit a sa version PHP en 7.2 certains scripts exemples auront besoin d'être exécutés sous PHP 7.4 minimum.

Question

[solution n°8 p.31]

Continuons à manipuler nos motos. Ajoutez un destructeur à la classe `Moto` qui affiche un message indiquant que la moto rentre au garage.

```
1 <?php
2
3 // fichier Moto.php
4 class Moto
5 {
```

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

2 <https://www.php.net/manual/fr/language.oop5.decon.php>

```
6 public string $brand;
7 public string $color;
8 public float $maxSpeed;
9
10 public function __construct(string $brand, string $color, float $maxSpeed)
11 {
12     $this->brand = $brand;
13     $this->color = $color;
14     $this->maxSpeed = $maxSpeed;
15 }
16
17 public function getDescription(): string
18 {
19     return $this->brand.' '.$this->color.' ayant une vitesse maximale de '.$this->
20     >maxSpeed.' km/h'.PHP_EOL;
21 }
```

Par exemple, "Kawasaki rouge rentre au garage" est un message valide. Instanciez ensuite deux motos de votre choix et lancez le script pour afficher le message au moment où les objets seront supprimés, à la fin du script.

XIV. Une classe : un nouveau type

Objectifs

- Utiliser une classe comme un type
- Comprendre la différence entre "passage par valeur" et "passage par référence"
- Manipuler des objets dans des fonctions

Mise en situation

Les classes sont des éléments très pratiques à manipuler : leur structure est fixe et centralisée dans un même endroit et notre IDE nous propose directement les champs dont on dispose.

Mais, plus qu'une simple structure de données, une classe représente également un tout nouveau type que l'on peut utiliser pour typer nos variables.

Cependant, on ne manipule pas un objet exactement de la même manière qu'un tableau : ils ne sont pas gérés pareil en mémoire et cela peut parfois poser quelques problèmes.

Nous allons donc voir précisément ce qu'il est possible de faire avec nos objets, et ce à quoi nous devons faire attention.

Méthode

Jusqu'à présent, lorsque l'on voulait typer nos variables dans nos signatures de fonction (en paramètre ou en type de retour), nous n'avions pas d'autre choix que d'utiliser des types simples : `int`, `float`, `bool`, etc.

Mais en créant la classe `User` dans la partie précédente, nous avons également créé le type `User`, qu'il est possible d'utiliser comme les autres types.

Il est donc maintenant possible de dire à une fonction qu'elle attend un objet de type `User`, ou qu'elle retourne un objet de ce type, ou qu'une propriété est de type `User`.

Par exemple, créons une classe `Couple` qui représente un duo d'utilisateurs :

```
1 <?php
2
3 class Couple
4 {
5     // Un couple est composé de deux utilisateurs :
6     public User $user1;
7     public User $user2;
8
9     public function __construct(User $user1, User $user2)
10    {
11        $this->user1 = $user1;
12        $this->user2 = $user2;
13    }
14 }
15
16 $laure = new User('Laure', 'Dupont');
17 $robert = new User('Robert', 'Durand');
18
19 $couple = new Couple($laure, $robert);
```

Remarque Valeurs et références

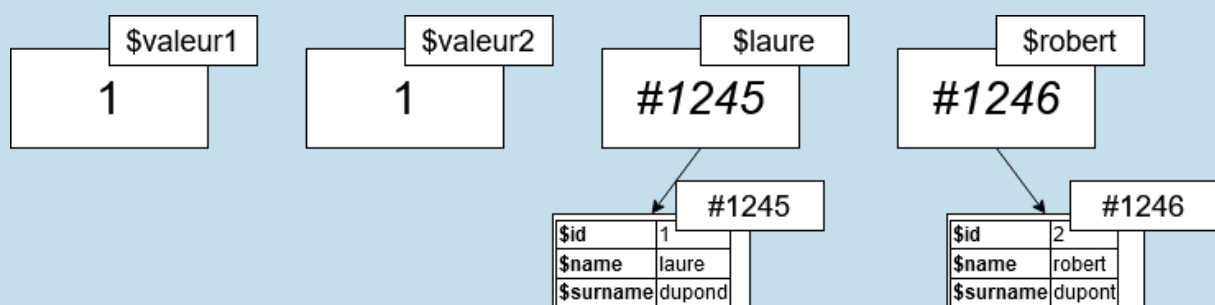
Attention, PHP ne gère pas les objets de la même manière que les autres variables.

Les variables qui ont un type simple, comme des entiers, des chaînes de caractères ou des tableaux, sont **passées par valeur**. Cela signifie que chaque variable contient une **copie de la donnée originale** :

```
1 <?php
2
3 $valeur1 = 1;
4 $valeur2 = $valeur1; // Ici, le contenu de $valeur2 est une copie de cette de $valeur 1
5 $valeur1 = 99;
6 echo $valeur1; // Affiche "99"
7 echo $valeur2; // Affiche "1", car même si le contenu de $valeur1 a été modifié, cela n'a pas
   impacté le contenu de $valeur2
```

Les objets, quant à eux, sont **passés par référence**.

Chaque variable ne contient pas vraiment notre objet, mais une référence vers un objet qui, lui, est stocké ailleurs :

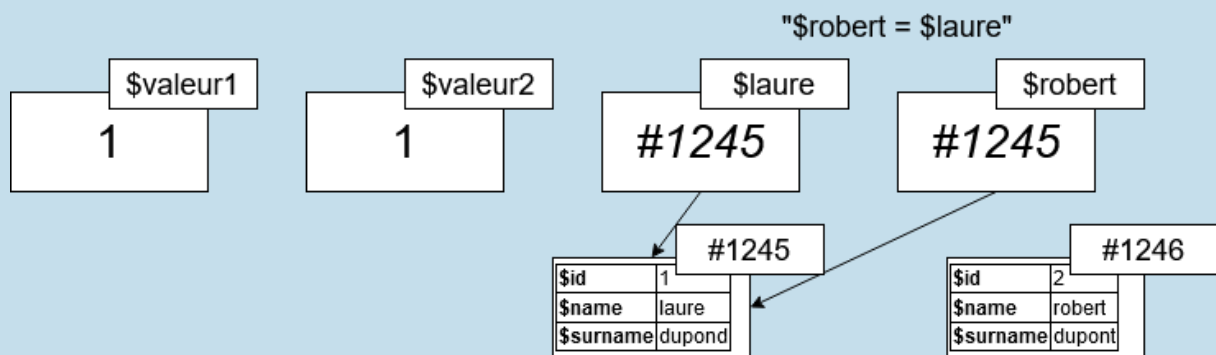


La conséquence directe de cela est que **chaque variable contenant la même référence manipule le même objet**.

```

1 <?php
2
3 $laure = new User();
4 $laure->name = 'Laure';
5 $laure->surname = 'Dupond';
6
7 $robert = $laure; // Ici, $robert contient une référence vers l'objet $laure, et non pas une
   copie de l'objet, comme on pourrait s'y attendre !
8 $robert->name = 'Robert';
9
10 echo $robert->name; // Affiche "Robert"
11 echo $laure->name; // Affiche "Robert" également !

```



Lorsqu'il n'existe plus de référence vers un objet, il n'est plus possible de le manipuler.

Il est considéré comme perdu et est automatiquement détruit : c'est à ce moment là que son destructeur est appelé.

Attention

Les objets sont **toujours** passés par référence, **même dans les fonctions et les méthodes !**

Si une fonction était amenée à modifier l'objet qui lui est passé en paramètre, l'objet avec lequel notre fonction est appelée sera modifié également.

```

1 <?php
2
3 function modifyUser(User $user): void
4 {
5     $user->surname = 'Doe';
6 }
7
8 $laure = new User('Laure', 'Dupond'); // Le nom de $laure est "Dupond"
9 modifyUser($laure); // La fonction modifie le champ "surname" de notre objet
10 echo $laure->surname; // Affiche "Doe" !

```

Syntaxe À retenir

- Il est possible d'utiliser une classe comme un type pour signer les fonctions.
- Attention toutefois, car les objets sont passés **par référence**, donc toutes les propriétés modifiées directement avec l'opérateur `->` seront réellement modifiées dans l'objet, où qu'il soit utilisé.
- Il faut donc faire très attention quand il s'agit de manipuler les objets dans des fonctions.

Complément

Déclaration de types (documentation officielle)¹

Passage par référence (documentation officielle)²

XV. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous pouvez travailler sur l'environnement de travail :

<https://sandbox.onlinephpfunctions.com/>

Replit a sa version PHP en 7.2 certains scripts exemples auront besoin d'être exécutés sous PHP 7.4 minimum.

Question

[solution n°9 p.31]

Il est maintenant temps d'organiser la course entre nos motos !

Soit la classe `Moto` suivante :

```
1 <?php
2
3 // fichier Moto.php
4 class Moto
5 {
6     public string $brand;
7     public string $color;
8     public float $maxSpeed;
9
10    public function __construct(string $brand, string $color, float $maxSpeed)
11    {
12        $this->brand = $brand;
13        $this->color = $color;
14        $this->maxSpeed = $maxSpeed;
15    }
16
17    public function getDescription(): string
18    {
19        return $this->brand.' '.$this->color.' ayant une vitesse maximale de '.$this->
20    >maxSpeed.' km/h'.PHP_EOL;
21    }
22
23    public function __destruct() {
24        echo $this->brand.' '.$this->color.' rentre au garage'.PHP_EOL;
25    }
26 }
```

Écrivez une classe `Race` qui va organiser une course entre deux motos : chaque moto sera une propriété, et la classe `Race` aura une méthode `startRace` qui va lancer la course et retourner la moto qui aura gagné, c'est-à-dire la moto la plus rapide.

Dans votre script principal, instanciez deux motos et une course et affichez la description de la moto gagnante. Assurez-vous ensuite que toutes les motos retournent au garage.

Indice :

La méthode `startRace` compare les deux `maxSpeed` des motos de la course. On ne traite pas les cas d'égalité.

¹ <https://www.php.net/manual/fr/functions.arguments.php#functions.arguments.type-declaration>

² <https://www.php.net/manual/fr/language.references.pass.php>

XVI. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°10 p.32]

Exercice

Dans le code d'une application pour gérer des documents, vous trouvez cette page permettant d'afficher les données d'un document :

```
1 <?php
2
3 $document = loadDocumentFromDatabase($_GET['id']);
4 echo $document['name'].' : '.$document['filePath'];
```

Vous devez rajouter le poids d'un document sur cette page. Comment le récupérez-vous ?

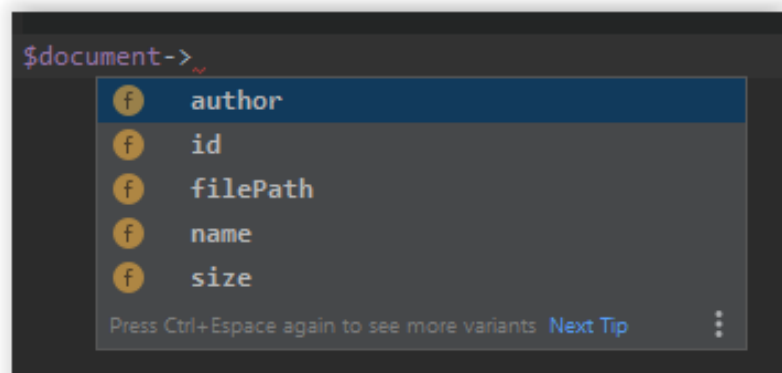
- ☐ \$document['size']
- ☐ \$document['fileSize']
- ☐ \$document['weight']
- ☐ \$document['fileWeight']
- ☐ Je n'ai pas assez d'informations pour savoir quel champ utiliser, je dois aller voir en base de données ou utiliser la fonction `var_dump` sur mon tableau.

Exercice

En réalité, le code précédent était une ancienne version. Voici l'actuelle :

```
1 <?php
2
3 $document = loadDocumentFromDatabase($_GET['id']);
4 echo $document->name.' : '.$document->filePath;
```

En commençant à taper, votre IDE vous affiche les informations suivantes :



Comment rajoutez-vous la taille du fichier ?

- ☐ \$document->size
- ☐ \$size->document
- ☐ \$document=>size
- ☐ \$document['size']

- ☐ Je n'ai pas assez d'informations pour savoir quel champ utiliser, je dois aller voir en base de données ou utiliser la fonction `var_dump` sur mon objet.

Exercice

Dans le code ci-dessus, qu'est-ce que `$document` ?

- ☐ Une classe
- ☐ Une propriété
- ☐ Un objet
- ☐ Un tableau

Exercice

Que sont "id" et "name" de notre variable `$document` ?

- ☐ Des paramètres
- ☐ Des propriétés
- ☐ Des classes
- ☐ Des objets

Exercice

Qu'est-ce qu'un constructeur ?

- ☐ Une fonction permettant de créer des objets
- ☐ Une méthode permettant de donner une valeur aux propriétés de nos objets et de faire toutes les opérations nécessaires au bon fonctionnement de notre objet
- ☐ Une espèce de boa qui tue ses proies en les serrant très fort
- ☐ Une méthode qui est appelée automatiquement au tout début de notre script PHP

Exercice

Que se passe-t-il lorsque l'on passe une variable par référence à une autre variable ?

- ☐ Le contenu de la première variable est vidé et est transféré dans la deuxième variable
- ☐ Les deux variables manipulent la même valeur : modifier la valeur d'une des variables modifie également l'autre
- ☐ La valeur de la première variable est copiée dans la deuxième variable

Exercice

Comment sont passés les objets ?

- ☐ Toujours par référence
- ☐ Par valeur, sauf si on met le symbole "&" devant le nom de la variable
- ☐ Par référence quand elles sont passées en paramètres, et par valeur quand elles sont retournées par une fonction
- ☐ Par référence quand elles sont retournées par une fonction, et par valeur quand elles sont passées en paramètres

Exercice

Pour quoi peut-on utiliser une classe ?

- ☐ Pour créer des objets
- ☐ Pour typer des paramètres de fonction
- ☐ Pour typer des propriétés
- ☐ Pour typer le retour d'une fonction

Exercice

Comment peut-on s'assurer de recevoir un objet avec une propriété "name" dans une fonction ?

- ☐ En mettant en signature de fonction : `function myFunction(array $param['name'])`
- ☐ En mettant dans les commentaires de la fonction :
`// SVP merci de ne passer que des objets avec une propriété name, merci !`
- ☐ En créant une classe `NamedClass` contenant un champ "name" et en mettant en signature de fonction :
`function myFunction(NamedClass $param)`

B. Exercice : Défi

Pendant le développement d'un projet informatique, il est important d'avoir des outils permettant de suivre l'exécution du code et de conserver ces informations. Cela permet, en cas de dysfonctionnement, de récupérer les informations et d'avoir une vision plus claire de ce qui a provoqué l'erreur.

Une manière de faire ça est d'utiliser un `logger` ("Journaliseur", en français) qui va créer des `logs` ("Journaux") sous la forme de fichiers textes retraçant un événement et notifiant leur date. Dans ce défi, vous allez devoir créer votre propre `logger`.

Pour réaliser cet exercice, vous pouvez travailler sur l'environnement de travail :

<https://sandbox.onlinephpfunctions.com/>

Replit a sa version PHP en 7.2 certains scripts exemples auront besoin d'être exécutés sous PHP 7.4 minimum.

Question

[solution n°11 p.35]

Créez une classe `Logger` qui va permettre d'insérer des messages dans des fichiers textes. Chaque fichier de log aura pour nom la date du jour (au format AAAA-MM-JJ) avec, pour extension, **.log**.

Voici un exemple de fichier log attendu (2020-03-06.log) :

```
1 [2020-03-06 13:12:54] Ouverture des logs
2 [2020-03-06 13:12:54] Course lancée. Vitesse moto 1 : 120. Vitesse moto 2 : 112
3 [2020-03-06 13:12:54] Moto 1 gagne
4 [2020-03-06 13:12:54] Fermeture des logs
5 [2020-03-06 13:21:25] Ouverture des logs
6 [2020-03-06 13:12:54] Course lancée. Vitesse moto 1 : 98. Vitesse moto 2 : 164
7 [2020-03-06 13:21:25] Moto 2 gagne
8 [2020-03-06 13:21:25] Fermeture des logs
```

La classe `Logger` devra donc créer le fichier s'il n'existe pas et proposer une méthode `log()` permettant d'écrire une ligne dans ce fichier : le message sera passé en paramètre, mais la date devra être ajoutée à la volée.

Attention : pour des raisons de performance, il est préférable d'ouvrir le fichier au début et de le refermer à la fin du script, plutôt que de l'ouvrir et le refermer à chaque insertion.

Lorsque le `Logger` est créé, il doit créer automatiquement un message "Ouverture des logs", et un message "Fermeture des logs" doit être créé à la fin du script : cela permet de séparer les appels d'une même journée.

Une fois votre `Logger` créé, instanciez-le et créez un message de log de test. Assurez-vous qu'il soit bien placé entre les deux messages d'ouverture et de fermeture.

Note : le type "resource", qui est utilisé pour manipuler des fichiers ou des connexions, n'est pas un "vrai" type et ne peut pas être utilisé pour typer des variables. Exceptionnellement, vous aurez le droit de ne pas typer votre propriété.

Voici le contenu du fichier `index.php` :

```
1 <?php
2
3 // fichier index.php
4 include 'Moto.php';
5 include 'Race.php';
6 include 'Logger.php';
7
8 $logger = new Logger();
9
10 $moto1 = new Moto("Yamaha", "rouge", 120);
11 $moto2 = new Moto("Suzuki", "bleue", 112);
12
13 $race = new Race($moto1, $moto2);
14 $logger->log($race->startRace());
15 $logger->log($race->getWinner());
16
17 unset($logger);
18
19 $logger = new Logger();
20
21 $moto1 = new Moto("Yamaha", "rouge", 98);
22 $moto2 = new Moto("Suzuki", "bleue", 164);
23
24 $race = new Race($moto1, $moto2);
25 $logger->log($race->startRace());
26 $logger->log($race->getWinner());
27
28 unset($logger);
```

Ainsi que le contenu du fichier `Moto.php` :

```
1 <?php
2
3 class Moto
4 {
5     public string $brand;
6     public string $color;
7     public float $maxSpeed;
8
9     public function __construct(string $brand, string $color, float $maxSpeed)
10    {
11        $this->brand = $brand;
12        $this->color = $color;
13        $this->maxSpeed = $maxSpeed;
14    }
15
16    public function getDescription(): string
17    {
18        return $this->brand.' '.$this->color.' ayant une vitesse maximale de '.$this->
19    >maxSpeed.' km/h'.PHP_EOL;
20    }
21 }
```

Et le contenu du fichier `Race.php` modifié pour l'exercice :

```
1 <?php
2
3 class Race
4 {
5     public Moto $moto1;
6     public Moto $moto2;
7
8     public function __construct(Moto $moto1, Moto $moto2)
9     {
10         $this->moto1 = $moto1;
11         $this->moto2 = $moto2;
12     }
13
14     public function startRace(): string
15     {
16         return sprintf(
17             'Course lancée. Vitesse moto 1 : %s. Vitesse moto 2 : %s',
18             $this->moto1->maxSpeed,
19             $this->moto2->maxSpeed
20         );
21     }
22
23     public function getWinner(): string
24     {
25         if ($this->moto1->maxSpeed > $this->moto2->maxSpeed) {
26             return 'Moto 1 gagne';
27         }
28
29         return 'Moto 2 gagne';
30     }
31 }
32
```

Indice :

Le constructeur de la classe `Logger` doit s'occuper de vérifier si le fichier d'aujourd'hui existe, de le créer s'il n'existe pas et d'écrire le message d'ouverture. Aidez-vous des différents modes d'ouverture de fichiers¹.

Indice :

Pour le nom du fichier et la date, vous pouvez vous aider des formats de date².

Indice :

Le destructeur doit s'occuper de fermer le fichier log et d'écrire le message de fermeture.

Indice :

Le constructeur et le destructeur doivent utiliser la méthode `log()`.

Solutions des exercices

1 <https://www.php.net/manual/fr/function.fopen.php>

2 <https://www.php.net/manual/fr/function.date.php>

p. 5 Solution n°1

```
1 <?php
2
3 $product =
    json_decode(base64_decode('eyJyZW1lIjoiQ2FydGUgbVx1MDBlOHJlIiwic3RvY2siOiIxMmIsImJyYW5kIjoiQXN1cyJ9'),
    true);
4
5 echo $product['name'].' ('.$product['brand'].') : '.$product['stock'];
```

Il a fallu utiliser la fonction `var_dump` pour connaître la structure de notre tableau, ce qui n'était pas pratique !

p. 5 Solution n°2

Même sans connaître la POO, on peut se douter que la variable `$price` va servir à récupérer le prix de notre produit.

p. 8 Solution n°3

```
1 <?php
2 // fichier Moto.php
3 class Moto
4 {
5     public string $brand;
6     public string $color;
7     public float $maxSpeed;
8 }
```

p. 11 Solution n°4

```
1 <?php
2 // fichier index.php
3 require_once 'Moto.php';
4
5 $moto = new Moto();
6 $moto->brand = "Yamaha";
7 $moto->color = "rouge";
8 $moto->maxSpeed = 210;
```

p. 11 Solution n°5

```
1 <?php
2 // fichier index.php
3 require_once 'Moto.php';
4
5 $moto = new Moto();
6 $moto->brand = "Suzuki";
7 $moto->color = "bleue";
8 $moto->maxSpeed = 220;
```

p. 13 Solution n°6

```

1 <?php
2
3 // fichier Moto.php
4 class Moto
5 {
6     public string $brand;
7     public string $color;
8     public float $maxSpeed;
9
10    public function getDescription(): string
11    {
12        return $this->brand.' '.$this->color.' ayant une vitesse maximale de '.$this->
13        maxSpeed.'km/h'.PHP_EOL;
14    }
15 }

```

```

1 <?php
2
3 // fichier index.php
4 require_once 'Moto.php';
5
6 $moto = new Moto();
7 $moto->brand = "Kawasaki";
8 $moto->color = "noire";
9 $moto->maxSpeed = 238;
10
11 echo $moto->getDescription();

```

p. 16 Solution n°7

```

1 <?php
2
3 // fichier Moto.php
4 class Moto
5 {
6     public string $brand;
7     public string $color;
8     public float $maxSpeed;
9
10    public function __construct(string $brand, string $color, float $maxSpeed)
11    {
12        $this->brand = $brand;
13        $this->color = $color;
14        $this->maxSpeed = $maxSpeed;
15    }
16
17    public function getDescription(): string
18    {
19        return $this->brand.' '.$this->color.' ayant une vitesse maximale de '.$this->
20        maxSpeed.'km/h'.PHP_EOL;
21    }
22 }

```

```

1 <?php
2
3 // fichier index.php
4 require_once 'Moto.php';
5
6 $moto = new Moto("Piaggio", "violet", 217);
7 echo $moto->getDescription();

```

p. 18 Solution n°8

```

1 <?php
2
3 // fichier Moto.php
4 class Moto
5 {
6     public string $brand;
7     public string $color;
8     public float $maxSpeed;
9
10    public function __construct(string $brand, string $color, float $maxSpeed)
11    {
12        $this->brand = $brand;
13        $this->color = $color;
14        $this->maxSpeed = $maxSpeed;
15    }
16
17    public function getDescription(): string
18    {
19        return $this->brand.' '.$this->color.' ayant une vitesse maximale de '.$this->
20    >maxSpeed.'km/h'.PHP_EOL;
21    }
22
23    public function __destruct() {
24        echo $this->brand.' '.$this->color.' rentre au garage'.PHP_EOL;
25    }
26 }

```

```

1 <?php
2
3 // fichier index.php
4 require_once 'Moto.php';
5
6 $moto = new Moto("Piaggio", "violet", 217);
7 $moto = new Moto("Kawasaki", "rouge", 211);

```

p. 22 Solution n°9

```

1 <?php
2
3 // fichier Race.php
4 class Race
5 {
6     public Moto $moto1;
7     public Moto $moto2;
8
9     public function __construct(Moto $moto1, Moto $moto2)

```

```

10     {
11         $this->moto1 = $moto1;
12         $this->moto2 = $moto2;
13     }
14
15     public function startRace(): Moto
16     {
17         if ($this->moto1->maxSpeed > $this->moto2->maxSpeed) {
18             return $this->moto1;
19         }
20
21         return $this->moto2;
22     }
23 }

```

```

1 <?php
2 // fichier index.php
3 include 'Moto.php';
4 include 'Race.php';
5
6 $moto1 = new Moto("Yamaha", "rouge", 210);
7 $moto2 = new Moto("Suzuki", "bleue", 180);
8
9 $race = new Race($moto1, $moto2);
10 echo $race->startRace()->getDescription();
11 // cette ligne va donc lancer la fonction startRace dans la classe Race puis retourner la
    description de la moto qui aura gagné la course. Pour rappel, la fonction startRace va
    retourner une moto gagnante (class Moto) et la fonction getDescription est présente dans la
    classe Moto.

```

Exercice p. 23 Solution n°10

Exercice


Dans le code d'une application pour gérer des documents, vous trouvez cette page permettant d'afficher les données d'un document :

```

1 <?php
2
3 $document = loadDocumentFromDatabase($_GET['id']);
4 echo $document['name'].' : '.$document['filePath'];

```

Vous devez rajouter le poids d'un document sur cette page. Comment le récupérez-vous ?

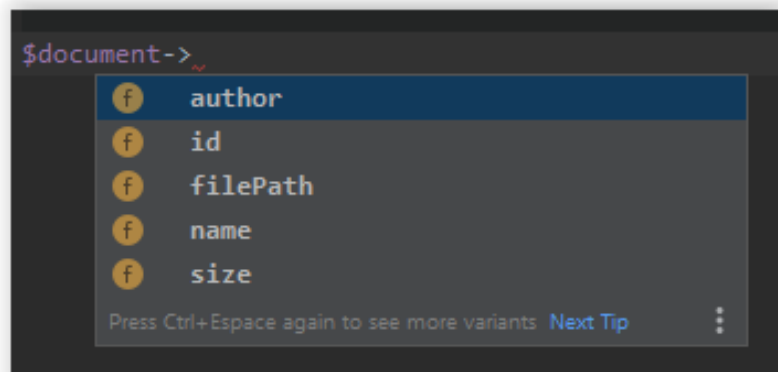
- ☐ \$document['size']
- ☐ \$document['fileSize']
- ☐ \$document['weight']
- ☐ \$document['fileWeight']
- ☒ Je n'ai pas assez d'informations pour savoir quel champ utiliser, je dois aller voir en base de données ou utiliser la fonction `var_dump` sur mon tableau.
-  Les tableaux ne permettent pas de connaître les champs qu'ils contiennent, donc il faut récupérer l'information autrement.

Exercice

En réalité, le code précédent était une ancienne version. Voici l'actuelle :

```
1 <?php
2
3 $document = loadDocumentFromDatabase($_GET['id']);
4 echo $document->name. ' : ' . $document->filePath;
```

En commençant à taper, votre IDE vous affiche les informations suivantes :



Comment rajoutez-vous la taille du fichier ?

- ☒ \$document->size
- ☐ \$size->document
- ☐ \$document=>size
- ☐ \$document['size']
- ☐ Je n'ai pas assez d'informations pour savoir quel champ utiliser, je dois aller voir en base de données ou utiliser la fonction `var_dump` sur mon objet.
- ☒ Nous savons quels champs possède notre objet grâce à notre IDE, et le champ "size" semble être exactement ce qu'on recherche. L'opérateur permettant d'accéder aux valeurs d'un objet est `->`.


Exercice

Dans le code ci-dessus, qu'est-ce que `$document` ?

- ☐ Une classe
La classe est le "moule" utilisé pour créer l'objet, mais `$document` est bel et bien un objet !
- ☐ Une propriété
Une propriété est une variable placée à l'intérieur d'une classe.
- ☒ Un objet
- ☐ Un tableau
- ☒ La variable `$document` est un objet, c'est pourquoi il faut utiliser l'opérateur `->` pour accéder à ses valeurs.

Exercice

Que sont "id" et "name" de notre variable `$document` ?

- ☐ Des paramètres
- ☒ Des propriétés
- ☐ Des classes
- ☐ Des objets
-  Ces "variables internes" à notre classe s'appellent des propriétés.


Exercice

Qu'est-ce qu'un constructeur ?

- ☐ Une fonction permettant de créer des objets
Une fonction est quelque chose d'externe à une classe, alors que les constructeurs font partie intégrante de la classe : ce sont des méthodes. De plus, les constructeurs ne créent pas d'objet, ils ne font que les initialiser : c'est le mot-clé `new` qui s'occupe de la création !
- ☒ Une méthode permettant de donner une valeur aux propriétés de nos objets et de faire toutes les opérations nécessaires au bon fonctionnement de notre objet
- ☐ Une espèce de boa qui tue ses proies en les serrant très fort
Ça s'écrit "constricteur". D'ailleurs, contrairement à la croyance populaire, il n'étouffe pas ses proies, mais bloque l'arrivée du sang dans les organes vitaux.
- ☐ Une méthode qui est appelée automatiquement au tout début de notre script PHP


Exercice

Que se passe-t-il lorsque l'on passe une variable par référence à une autre variable ?

- ☐ Le contenu de la première variable est vidé et est transféré dans la deuxième variable
- ☒ Les deux variables manipulent la même valeur : modifier la valeur d'une des variables modifie également l'autre
- ☐ La valeur de la première variable est copiée dans la deuxième variable
-  Deux variables partageant la même référence regardent dans le même emplacement mémoire, donc chaque changement de valeur est répercuté sur les deux variables.

Exercice


Comment sont passés les objets ?

- ☒ Toujours par référence
- ☐ Par valeur, sauf si on met le symbole "&" devant le nom de la variable
C'est correct pour les types scalaires, mais pas pour les objets !
- ☐ Par référence quand elles sont passées en paramètres, et par valeur quand elles sont retournées par une fonction
- ☐ Par référence quand elles sont retournées par une fonction, et par valeur quand elles sont passées en paramètres
-  Les objets sont TOUJOURS passés par référence : il faut être vigilant en les modifiant dans des méthodes ou des fonctions !

Exercice

Pour quoi peut-on utiliser une classe ?


- ☒ Pour créer des objets
- ☒ Pour typer des paramètres de fonction
- ☒ Pour typer des propriétés
- ☒ Pour typer le retour d'une fonction

 Il est possible d'utiliser les classes comme n'importe quel autre type scalaire, en plus de leur rôle premier de définir une structure qui sera utilisée pour créer des objets.

Exercice

Comment peut-on s'assurer de recevoir un objet avec une propriété "name" dans une fonction ?

- ☐ En mettant en signature de fonction : `function myFunction(array $param['name'])`
- ☐ En mettant dans les commentaires de la fonction :
`// SVP merci de ne passer que des objets avec une propriété name, merci !`
- ☒ En créant une classe `NamedClass` contenant un champ "name" et en mettant en signature de fonction :
`function myFunction(NamedClass $param)`

 Si on veut être sûr de la structure d'un paramètre d'une fonction, il faut créer une classe et typer notre paramètre en utilisant cette classe.

p. 25 Solution n°11

```

1 <?php
2
3 class Logger
4 {
5     public $file;
6
7     public function __construct()
8     {
9         $this->file = fopen(date('Y-m-d').'log', 'a');
10        $this->log('Ouverture des logs');
11    }
12
13    public function log(string $message)
14    {
15        fwrite($this->file, '['.date('Y-m-d H:i:s').'] '.$message.PHP_EOL);
16    }
17
18    public function __destruct()
19    {
20        $this->log('Fermeture des logs');
21        fclose($this->file);
22    }
23 }
24
25 $logger = new Logger();
26 $logger->log('test');
```