Tableau et programmation fonctionnelle

Qu'est-ce que la programmation fonctionnelle ?

La programmation fonctionnelle est un paradigme de programmation qui consiste à concevoir ses programmes comme un ensemble de fonctions mathématiques que l'on composent entre elles.

Les deux éléments essentiels dans la programmation fonctionnelle sont **l'utilisation de fonctions de première classe** et **l'utilisation de fonctions pures**.

Les fonctions sont des objets de première classe en PHP , **elles peuvent donc être passées en argument et donc être composées**. C'est notamment le principe des fonctions de rappel.

Les fonctions pures sont des fonctions dont le résultat ne dépend que des arguments passés et qui n'ont pas d'effets extérieurs (appelés effets de bord).

L'une des principales conséquence de privilégier la programmation fonctionnelle est que **les fonctions sont indépendantes de leur contexte d'exécution**. C'est-à-dire que pour les mêmes arguments, elles retourneront toujours le même résultat, et ce où qu'elles soient appelées. Cela permet une grande maintenabilité.

Appliquer une fonction sur tous les éléments d'un tableau

La fonction `array_map()` permet d'appliquer une fonction de rappel sur tous les éléments d'un tableau.

Prenons un premier exemple avec un tableau :

```
<?php
$tableau1 = [1, 2, 3];
$tableauSortie = array_map(fn ($el) => $el * 2, $tableau1);
echo "<pre>";
print_r($tableauSortie);
echo "</pre>";
/* Array
(
  [0] => 2
```

```
  [1] => 4
  [2] => 6
) */
```

Si vous passez plusieurs tableaux, chaque élément des tableaux seront disponibles en arguments dans la fonction de rappel :

```
<?php
$tableau1 = [1, 2, 3];
$tableau2 = [3, 4, 5];
$tableauSortie = array_map(fn ($el, $el2) => $el * $el2, $tableau1,
$tableau2);
echo "<pre>";
print_r($tableauSortie);
echo "</pre>";
/* Array
(
  [0] => 3
  [1] => 8
  [2] => 15
) */
```

Ici, `$el` contient pour chaque itération un élément de `$tableau1` et `$el2` un l'élément du `$tableau2` à la même position.

Vous pouvez bien sûr utiliser des fonctions normales et pas seulement fléchées.

Par exemple, pour calculer la valeur moyenne de trois tableaux pour chaque élément :

```
<?php
function average(...$args)
{
  $sum = 0;
  foreach ($args as $value) {
    $sum += $value;
  }
  return round($sum / count($args), 1);
}

$tableau1 = [1, 2, 3];
```

```
$tableau2 = [3, 4, 5];
$tableau3 = [42, 128, 16];
$tableauSortie = array_map("average", $tableau1, $tableau2, $tableau3);
echo "<pre>";
print_r($tableauSortie);
echo "</pre>";
/* Array
(
  [0] => 15.3
  [1] => 44.7
  [2] => 8
) */
```

C'est une bonne fonction pour revoir ce que nous avons appris jusqu'ici.

`...$args` permet d'accepter un nombre indéfini d'arguments qui seront mis dans un tableau.

Ici, notre fonction de rappel recevra un élément par tableau passé à `array_map()`. Elle recevra deux éléments pour deux tableaux, trois éléments pour trois tableaux etc.

Nous initialisons un compteur à `$sum = 0`.

Nous parcourons ensuite le tableau `$args` contenant les éléments de chaque itération des trois tableaux pour obtenir la somme des valeurs dans `$args`.

Notez que nous pourrions utiliser directement `array_sum($args)`, mais c'est pour pratiquer `foreach()`.

Nous faisons ensuite la moyenne `$sum / count($args)` et nous arrondissons avec `round()` à un chiffre après la virgule.

Cette fonction est donc utilisable quel que soit le nombre de tableau.

Nous passons notre fonction à `array_map()` en utilisant son nom : `array_map("average", $tableau1, $tableau2, $tableau3)`.

Nous obtenons un tableau avec la moyenne pour chaque clé des valeurs des trois tableaux.

Filtrer les éléments d'un tableau

La fonction `array_filter()` permet de filtrer les éléments d'un tableau en utilisant une fonction de rappel.

Les clés du tableau sont préservées.

```php
<?php
$tableau = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
$tableauSortie = array_filter($tableau, fn ($el) => $el % 2 === 0);
echo "<pre>";
print_r($tableauSortie);
echo "</pre>";
/* Array
(
  [1] => 2
  [3] => 4
  [5] => 6
  [7] => 8
  [9] => 10
) */
```

La fonction de rappel passée doit retourner `true` ou `false`. Si elle retourne `true`, l'élément est ajouté au nouveau tableau. Si elle retourne `false` il n'est pas ajouté.

Ici notre fonction de rappel retourne `true` si `$el % 2 === 0`. Autrement dit, elle retourne les éléments pairs.

Notez que les clés sont maintenues. Pour des clés numériques, ce n'est souvent pas souhaitable. On utilise `array_values()` pour réindexer numériquement le tableau sans trou dans les index :

```php
<?php
$tableau = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
$tableauSortie = array_filter($tableau, fn ($el) => $el % 2 === 0);
echo "<pre>";
print_r(array_values($tableauSortie));
echo "</pre>";
/* Array
(
  [0] => 2
  [1] => 4
  [2] => 6
  [3] => 8
  [4] => 10
) */
```

Notez qu'il est possible de passer les clés ou les clés et les valeurs à la fonction de rappel, en utilisant les options `ARRAY_FILTER_USE_KEY` ou `ARRAY_FILTER_USE_BOTH`.

Par exemple :

```
<?php
function filtre($val, $cle)
{
  return str_starts_with($cle, "a") && $val % 2 === 0;
}

$tableau = [
  "afze" => 1, "aa" => 2, "bc" => 3, "bde" => 4,
  "bzd" => 5, "agre" => 6, "ac" => 7, "bedz" => 8,
  "bda" => 9, "as" => 10
];
$tableauSortie = array_filter($tableau, "filtre", ARRAY_FILTER_USE_BOTH);
echo "<pre>";
print_r($tableauSortie);
echo "</pre>";
/* Array
(
  [aa] => 2
  [agre] => 6
  [as] => 10
) */
```

Ici un élément n'est ajouté au tableau de sortie que si sa clé commence par `"a"` et si sa valeur est paire.

Réduire un tableau en une valeur

La fonction `array_reduce()` permet de réduire un tableau en une valeur calculée.

La signature de la fonction est :

```
array_reduce ( array $array , callable $callback , mixed $initial = null
) : mixed
```

Le premier argument est le tableau à réduire.

Le deuxième élément est la fonction de rappel, que nous étudierons après.

Le dernier argument est la valeur initiale de l'accumulateur.

La signature de la fonction de rappel est la suivante :

```
callback ( mixed $carry , mixed $item ) : mixed
```

`$carry` ou l'accumulateur, est la valeur qui est sauvegardée entre chaque itération et passée à chaque nouvelle itération.

`$item` est l'élément de l'itération en cours.

Nous allons voir un premier exemple simple :

```
<?php
$tableau = [1, 2, 3, 4, 5];
$valeur = array_reduce($tableau, fn ($acc, $el) => $acc + $el, 0);
echo "<pre>";
print_r($valeur); // 15
echo "</pre>";
```

Lors de l'initialisation nous avons `$acc = 0`.

Lors de la première itération, `$el` est la première valeur du tableau donc 1. `$acc` vaut 0. Nous retournons 0 + 1 donc 1 comme prochaine valeur de `$acc`.

Lors de la seconde itération, `$el` est la deuxième valeur du tableau donc 2. `$acc` vaut 1. Nous retournons 1 + 2 donc 3 comme prochaine valeur de `$acc`.

Lors de la troisième itération, `$el` est la troisième valeur du tableau donc 3. `$acc` vaut 3. Nous retournons 3 + 3 donc 6 comme prochaine valeur de `$acc`.

Lors de la quatrième itération, `$el` est la quatrième valeur du tableau donc 4. `$acc` vaut 6. Nous retournons 6 + 4 donc 10 comme prochaine valeur de `$acc`.

Lors de la dernière itération, `$el` est la cinquième valeur du tableau donc 5. `$acc` vaut 10. Nous retournons 10 + 5 donc 15 comme valeur finale de `$acc` qui est le résultat de la réduction et la valeur retournée.