

# Créer ses propres fonctions

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Créer ses propres fonctions</b>	<b>3</b>
<b>III. Exercice : Appliquez la notion</b>	<b>5</b>
<b>IV. Les paramètres</b>	<b>5</b>
<b>V. Exercice : Appliquez la notion</b>	<b>8</b>
<b>VI. Les valeurs de retour</b>	<b>8</b>
<b>VII. Exercice : Appliquez la notion</b>	<b>10</b>
<b>VIII. La portée des variables</b>	<b>11</b>
<b>IX. Exercice : Appliquez la notion</b>	<b>15</b>
<b>X. Les fonctions anonymes</b>	<b>15</b>
<b>XI. Exercice : Appliquez la notion</b>	<b>17</b>
<b>XII. Les fonctions récursives</b>	<b>17</b>
<b>XIII. Exercice : Appliquez la notion</b>	<b>18</b>
<b>XIV. Auto-évaluation</b>	<b>18</b>
A. Exercice final .....	18
B. Exercice : Défi .....	20
<b>Solutions des exercices</b>	<b>21</b>

## I. Contexte

**Durée** : 1 h

**Environnement de travail** : Repl.it

**Pré-requis** : Bases de PHP

### Contexte

Comme nous le savons, PHP dispose nativement d'un certain nombre de fonctions déjà disponibles.

Mais nous avons également la possibilité de créer nos propres fonctions (que l'on appelle parfois les « fonctions utilisateur »). Celles-ci nous permettent d'éviter la répétition d'instructions à différents niveaux du code.

À travers ce cours, nous allons découvrir comment créer ces fonctions, comment les utiliser et quelle est la portée de leur intégration dans le code.

## II. Créer ses propres fonctions

### Objectifs

- Comprendre ce qu'est la déclaration d'une fonction
- Identifier les règles de déclaration d'une fonction

### Mise en situation

Avant de pouvoir déclarer nos propres fonctions, il est nécessaire de connaître quelques règles que nous allons identifier ci-après.

### Définition La déclaration d'une fonction

On appelle la définition d'une fonction la "déclaration". La déclaration peut se faire n'importe où dans le code et se fait grâce au mot-clé `function`.

Il est nécessaire qu'une fonction ait été déclarée pour pouvoir être utilisée dans un programme.

PHP doit connaître :

- son nom,
- ses paramètres éventuels,
- les instructions qu'elle va exécuter.

```
1 <?php
2
3 function functionName($param1, $param2)
4 {
5     //liste d'instructions
6 }
7
```

## Fondamental Les règles de définition d'une fonction

Les règles de définition d'une fonction sont les suivantes :

- Son nom commence par une lettre ou un `_` et peut comporter des lettres, des chiffres et le caractère `_`.
- Contrairement aux idées reçues, son nom n'est pas sensible à la casse.
- Les paramètres d'une fonction sont facultatifs, mais les parenthèses sont obligatoires.

Après que notre fonction ait été déclarée, nous pourrons l'utiliser. Celle-ci ne sera pas exécutée tant que nous n'y faisons pas appel.

## Méthode Appeler ses propres fonctions

Une fois déclarée, nous pouvons appeler notre fonction autant de fois que nous le souhaitons.

Il suffit pour cela d'indiquer :

- son nom,
- suivi d'une parenthèse ouvrante,
- des paramètres que nous souhaitons y passer si nécessaire,
- d'une parenthèse fermante,
- et d'un point-virgule.

```
1 <?php
2
3 // Déclaration de notre fonction
4 function sayHi()
5 {
6     echo 'Salut ' . PHP_EOL;
7 }
8
9 // Appel de notre fonction
10 sayHi();
```

## Complément Fonction dans une autre fonction

Il est tout à fait possible d'appeler une fonction au sein d'une autre fonction. Que nous faisons appel à l'extérieur ou à l'intérieur d'une fonction, les règles sont les mêmes.

```
1 <?php
2
3 function sayHi()
4 {
5     echo 'Salut ' . PHP_EOL;
6 }
7
8 function sayThankYou()
9 {
10     echo 'Merci! ' . PHP_EOL;
11 }
12
13 function bePolite()
14 {
15     sayHi();
16     sayThankYou();
17 }
18
```

```
19 bePolite();
```

**Syntaxe** **À retenir**

- Pour déclarer une fonction, il faut utiliser le mot-clé `function`, suivi du nom de la fonction.
- Pour utiliser la fonction, il suffit d'écrire son nom, suivi de deux parenthèses.

### III. Exercice : Appliquez la notion

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

**Question**

[solution n°1 p.23]

Déclarez une fonction qui permettra, lorsque nous y ferons appel, d'afficher la date et l'heure actuelle, dont l'affichage sera au format `jj/mm/yyyy hh:mm`.

**Indice :**

Vous pouvez afficher la date courante grâce à la fonction `date`<sup>2</sup>.

### IV. Les paramètres

**Objectifs**

- Comprendre l'utilité des paramètres et comment les définir
- Apprendre à définir des paramètres par défaut

**Mise en situation**

Lorsque l'on définit ses propres fonctions, le but est de pouvoir faire communiquer notre programme principal avec ces fonctions. Cette communication pourra se faire au moyen de paramètres.

**Syntaxe** **Les paramètres d'une fonction**

Les paramètres permettent de faire passer des informations entre le code qui appelle la fonction et la fonction elle-même. Ils pourront ensuite être utilisés pour exécuter les instructions composant la fonction. Il est conseillé de préciser leur type même si ce n'est pas obligatoire.

```
1 <?php
2
3 function sayHiTo(string $name)
4 {
5     echo 'Salut ' . $name . PHP_EOL;
6 }
7
8 sayHiTo('Julie');
9 sayHiTo('Anthony');
```

1 <https://repl.it/>

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

```

1 <?php
2
3 function sayHiTo($name)
4 {
5     echo 'Salut ' . $name . PHP_EOL;
6 }
7
8 sayHiTo('Julie');
9 sayHiTo('Anthony');
```

Il peut s'agir de simples variables, mais aussi de tableaux ou d'objets.

```

1 <?php
2
3 function sayHiToEverybody(array $names)
4 {
5     foreach ($names as $name) {
6         echo 'Salut ' . $name . PHP_EOL;
7     }
8 }
9
10 sayHiToEverybody(['Julie', 'Anthony']);
```

#### Rappel

Il existe 8 grands types de données différents :

- « chaîne de caractères » ou String en anglais ;
- « nombre entier » ou Integer en anglais
- « nombre décimal » ou Float en anglais
- « booléen » ou Boolean en anglais ;
- « tableau » ou Array en anglais
- « objet » ou Object en anglais
- « NULL » qui se dit également NULLE en anglais
- « ressource » ou Resource en anglais

#### Remarque

#### Différencier les notions de paramètres et d'arguments

Les paramètres sont aussi appelés *arguments* dans certains cas. Ces deux termes sont très proches, mais il existe une nuance entre ces notions : on parle de paramètres lors de la **déclaration d'une fonction**, tandis que l'on parle d'arguments dans **son utilisation au sein du code**.

#### Syntaxe

#### Valeurs par défaut

Lorsque nous déclarons une fonction, il est possible d'attribuer une valeur par défaut à certains de ses paramètres.

Pour cela, il suffit de déclarer le nom du paramètre, puis de lui affecter une valeur, comme pour une variable classique.

Lorsque nous appellerons la fonction, nous aurons alors la possibilité de préciser ou non les arguments optionnels.

```
1 <?php
2
3 function whoAmI(string $name, int $age = null)
4 {
5     echo 'Votre prénom est '.$name.PHP_EOL;
6
7     if ($age) {
8         echo 'Je connais votre âge, vous avez '.$age.' ans'.PHP_EOL;
9     }
10 }
11
12 whoAmI('Anthony', 25); // Affiche "Your name is Anthony I know your age, you are 25"
13 whoAmI('Julie'); // Affiche "Your name is Julie"
```

**Attention**

1. Les paramètres disposant d'une valeur par défaut étant optionnels, ceux-ci doivent être situés en dernier.
2. L'ordre de déclaration devant être respecté, on doit déclarer un premier paramètre optionnel, puis déclarer un second paramètre optionnel situé après le premier.

Dans l'exemple ci-après, on ne peut pas déclarer l'âge, puis déclarer la taille :

```
1 <?php
2
3 function whoAmI(string $name, int $age = null, int $height = null)
4 {
5     echo 'Votre prénom est '.$name.PHP_EOL;
6
7     if ($age) {
8         echo 'Je connais votre âge, vous avez '.$age.' ans'.PHP_EOL;
9     }
10
11     if ($height) {
12         echo 'Je connais votre taille : '.$height.'cm'.PHP_EOL;;
13     }
14 }
15
16 whoAmI('Anthony', 25, 180); // Affiche "Your name is Anthony I know your age, you are 25 I
17                             know your height: 180cm"
18 whoAmI('Julie', 170); // Affiche "Your name is Julie I know your age, you are 170"
```

**Remarque****Typage des paramètres**

Le fait de typer les paramètres d'une fonction permet de requérir que ses arguments soient d'un certain type lors de l'appel de celle-ci. Si la valeur donnée est d'un type incorrect, alors une erreur est générée.

La documentation de PHP spécifie les types possibles, ainsi que la version minimum de PHP compatible : <https://www.php.net><sup>1</sup>.

**Syntaxe****À retenir**

- Une fonction peut posséder des paramètres, qui sont des données transmises à la fonction pour effectuer ses calculs.
- Les paramètres sont déclarés entre parenthèses dans la définition de la fonction et peuvent être typés.

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

- Lors de l'appel de la fonction, il faut renseigner tous les paramètres obligatoires.
- Il est possible de rendre des paramètres facultatifs en leur affectant une valeur dans la définition de la fonction.

## V. Exercice : Appliquez la notion

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



### Question

[solution n°2 p.23]

Modifiez la fonction suivante afin de rendre le nom de l'utilisateur et son âge paramétrables.

Le nom est une chaîne de caractères obligatoire, tandis que l'âge est un entier qui ne sera pas obligatoire.

```
1 <?php
2
3 function greetings()
4 {
5     echo "Bonjour, je m'appelle Jules" . PHP_EOL;
6     echo "J'ai 27 ans" . PHP_EOL;
7 }
8
9 greetings();
```

## VI. Les valeurs de retour

### Objectif

- Identifier comment une fonction peut retourner un résultat et quelles sont ses limites

### Mise en situation

Nous allons voir grâce aux valeurs de retour que nos fonctions peuvent aussi communiquer avec notre programme principal.

### Récupérer le résultat d'une fonction

Il est possible de récupérer une valeur de retour suite à l'exécution d'une fonction. Pour cela, il suffit d'utiliser l'instruction `return`.

Lorsque celle-ci est rencontrée, **l'exécution des instructions s'arrête** et nous avons la possibilité de récupérer la valeur qui suit cette instruction.

1 <https://repl.it/>



**Méthode** Syntaxe de l'instruction return

La syntaxe de l'instruction `return` est la suivante :

```
1 <?php
2
3 function returnSomething()
4 {
5     return 'Something';
6 }
7
8 echo returnSomething();
9
```

Ce code retournera “Something” sur votre page web.

Il n'est pas possible pour une fonction de retourner plusieurs valeurs en même temps. En revanche, il est tout à fait possible de renvoyer un tableau de valeurs.

```
1 <?php
2
3 function returnAnArray()
4 {
5     return ['Something', 'Something Else', 'Another value'];
6 }
7
8 $values = returnAnArray();
9
10 foreach ($values as $value) {
11     echo $value.PHP_EOL;
12 }
```

Ce code retournera “Something Something Else Another value” sur votre page web.

**Complément**

Nous pouvons définir plusieurs instructions `return` au sein d'une fonction (à travers des conditions, par exemple).

Dans ce cas, la première instruction `return` rencontrée mettra fin à l'exécution de la fonction.

Le code suivant retourne “non, 8 est plus petit que 10 oui, 12 est plus grand que 10”.

```
1 <?php
2
3 function isGreaterThanTen($value)
4 {
5     if (10 < $value) {
6         return 'oui, '.$value.' est plus grand que 10'.PHP_EOL;
7     } else {
8         return 'non, '.$value.' est plus petit que 10'.PHP_EOL;
9     }
10 }
11
12 echo isGreaterThanTen(8);
13 echo isGreaterThanTen(12);
```

### Remarque Typage des paramètres de retour

Depuis PHP 7.0, il est possible de déclarer les types de retours des fonctions. Il s'agit des mêmes types de retour que pour les paramètres d'une fonction. Il est possible de les retrouver ici : <https://www.php.net/><sup>1</sup>.

Le typage peut parfois être préfixé par un point d'interrogation : cette notation, valable depuis PHP 7.1, permet d'indiquer que la fonction peut retourner le type de valeur indiqué ou une valeur nulle.

```
1 <?php
2
3 function returnAnArray(): array
4 {
5     return ['Something', 'Something Else', 'Another value'];
6 }
```

### Complément

Il existe un type de retour particulier lorsque notre fonction ne retourne rien. Ce type de retour est `void` et on peut l'utiliser comme ceci :

```
1 <?php
2
3 function returnAnArray(): void
4 {
5     echo 'Salut'.PHP_EOL;
6 }
```

### Syntaxe À retenir

- Une variable peut retourner une valeur grâce au mot-clé `return`.
- Il est possible de réceptionner la valeur retournée par une fonction en la stockant dans une variable.
- On peut définir un type de retour dans le cas où la fonction retourne une valeur, sinon on peut spécifier que le type sera `void`.

## VII. Exercice : Appliquez la notion

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



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

<sup>2</sup> <https://repl.it/>

## Question

[solution n°3 p.23]

La fonction suivante affiche la date du jour et l'heure courante au format européen. Modifiez-la pour qu'elle retourne cette date plutôt qu'elle ne l'affiche. Renommez-la également en conséquence : `getDateTime`.

```
1
2 function printDateTime()
3 {
4     echo date('d/m/Y h:i');
5 }
6
7 printDateTime();
```

## VIII. La portée des variables

### Objectifs

- Comprendre ce qu'est la portée des variables
- Identifier la différences entre des variables locales, globales et statiques

### Mise en situation

On appelle « portée d'une variable » l'espace dans lequel elle est définie et peut être utilisée.

### La portée des variables

En PHP, il est possible de déclarer une variable n'importe où : en début de script, dans une boucle ou à l'intérieur d'une fonction.

La portée d'une variable dépend du contexte dans lequel la variable est définie. Pour la majorité des variables, la portée concerne la totalité d'un script PHP. Mais lorsque vous définissez une fonction, la portée d'une variable définie dans cette fonction est locale à la celle-ci.

```
1 <?php
2
3 function sayHiTo($name)
4 {
5     $name = 'Raphael';
6     echo 'Salut ' . $name . PHP_EOL;
7 }
8
9
10 $name = 'Anthony';
11
12 function sayHi()
13 {
14     echo 'Salut ' . $name . PHP_EOL;
15 }
16
17
18 // Affiche "Anthony"
19 echo $name . PHP_EOL;
20
21 // Affichera un warning car la variable $name n'est pas définie dans la fonction
22 echo sayHi();
23
24 // Affiche "Salut Raphael" car la valeur de la variable $name est modifiée à l'intérieur de la
    fonction
```

```

25 sayHiTo($name);
26
27 // Affiche "Anthony" car, même si la variable $name est modifiée à l'intérieur de la fonction
  sayHiTo, la portée de celle-ci se limite à l'intérieur de la fonction
28 echo $name . PHP_EOL;

```

Dans l'exemple ci-dessus, il existe en réalité deux variables `$name` : celle qui est définie à l'extérieur de la fonction, et celle qui est définie à l'intérieur. Modifier l'une d'entre elles ne modifiera pas l'autre.

## Les variables globales

Il peut arriver que nous ayons besoin de nous servir, à l'intérieur d'une fonction, de variables possédant une **portée globale**, c'est-à-dire des variables ayant été définies en dehors de la fonction.

Pour cela, il est possible d'utiliser le mot-clé `global` avant la déclaration des variables. Cela permet d'indiquer que la variable utilisée ici doit être accessible de façon globale.

```

1 <?php
2
3 function sayMyName($name)
4 {
5     echo 'Votre nom est ' . $name . PHP_EOL;
6 }
7
8 function renamePeople()
9 {
10     // On définit la variable $name comme étant globale en modifiant sa valeur
11     global $name;
12
13     $name = 'Laure';
14     echo 'Votre nouveau nom est ' . $name . PHP_EOL;
15 }
16
17
18 $name = 'Julie';
19
20 // Affiche "Votre nom est Julie"
21 sayMyName($name);
22
23 // La variable $name a une portée globale : c'est elle qui va être récupérée par la fonction
  renamePeople()
24 renamePeople();
25
26 // La variable globale ayant été modifiée, cette fonction affiche maintenant "Votre nom est
  Laure"
27 sayMyName($name);
28
29 print_r($GLOBALS['name']);
30

```

Une deuxième méthode pour accéder aux variables globales est d'utiliser le tableau associatif prédéfini `$GLOBALS`.

Le tableau `$GLOBALS` est un tableau associatif contenant l'ensemble des variables globales comme clé et les valeurs des éléments du tableau comme valeur des variables.

### Remarque

Le `print_r()` affiche les informations d'une variable de façon à ce qu'elles soient lisibles par les humains. Par exemple, les valeurs contenues dans un tableau sont décrites dans un format qui montre les clés et les éléments.

## Les variables statiques

Une autre caractéristique importante de la portée des variables est la notion de **variable statique**.

En définissant une variable de façon locale, celle-ci sera détruite après que notre fonction ait été exécutée. Or, on pourrait avoir besoin de conserver la valeur finale d'une variable (dans le cas d'un compteur, par exemple).

Il est possible d'empêcher la destruction d'une variable en la précédant du mot-clé `static` lors de sa déclaration.

Celle-ci aura toujours une portée locale, mais sa valeur ne sera pas détruite lors de la fin de l'exécution de la fonction : elle sera conservée pour pouvoir être réutilisée lors d'une prochaine exécution. La valeur de cette fonction n'est initialisée qu'au premier appel de la fonction.

```
1 <?php
2
3 function countElements()
4 {
5     static $count = 0;
6
7     $count++;
8
9     echo $count . PHP_EOL;
10 }
11
12 countElements();
13 countElements();
14 countElements();
```

### Remarque La différence entre ++\$i et \$i++

Ici, nous parlons de post-incrémentation (`$i++`) et de pré-incrémentation (`++$i`).

- `$i++` incrémente `$i` et retourne l'ancienne valeur de `$i`
- `++$i` incrémente `$i` et retourne la nouvelle valeur de `$i`

```
1 <?php
2 //Post-incrémentation
3 $a = 2;
4 $b = $a++; // $a vaut 3 & $b vaut 2
5
6 //Pré-incrémentation
7 $a = 2;
8 $b = ++$a; // $a vaut 3 & $b vaut 3
```

## La fonction sprintf

La fonction `sprintf` retourne une chaîne formatée :

`string sprintf ( string $format [, mixed1 $args [, mixed2 $... ] ] )`

La chaîne de format peut contenir aucune, une ou plusieurs directives, soit, des caractères basiques (sauf `%`) qui sont copiés directement dans le résultats et des spécifications de conversion introduites par le caractère `%`.

1 <http://www.lephpfacile.com/manuel-php/language.pseudo-types.php#language.types.mixed>

2 <http://www.lephpfacile.com/manuel-php/language.pseudo-types.php#language.types.mixed>

### Exemple

```
1 <?php
2     $number = 5;
3     $fruit = 'pommier';
4     $format = 'Il y a %d pommes dans le %s';
5     echo sprintf($format, $number, $fruit);
```

Le code précédent affichera “Il y a 5 pommes dans le pommiers”.

%d : L'argument est traité comme un entier.

%s : L'argument est traité et présenté comme une chaîne de caractères.

### Passage de valeurs par référence

Il existe une autre méthode pour modifier la valeur d'un paramètre à l'intérieur d'une fonction, en la faisant précéder du caractère &.

Avec cette notation, on précise ainsi qu'il s'agit d'un alias : la valeur de la variable est modifiée à la sortie de la fonction. On parle alors de **passage par référence**.

Dans ce cas, on passe la référence (adresse mémoire) de la variable à la fonction, ce qui permet de modifier sa valeur de façon globale.

```
1 <?php
2
3 function sayHiToThenRename(&$name)
4 {
5     echo 'Salut ' . $name . PHP_EOL;
6
7     $name = 'Laure';
8     echo 'Votre nouveau nom est ' . $name . PHP_EOL;
9 }
10
11
12 $name = 'Julie';
13
14 echo $name . PHP_EOL;
15
16 sayHiToThenRename($name);
17
18 echo $name . PHP_EOL;
```

### Remarque

Le signe & est nécessaire uniquement dans la déclaration de la fonction, pas lors de l'appel.

### Syntaxe À retenir

- Les paramètres d'une fonction sont des variables différentes de celles qui sont passées en paramètres.
- Chaque variable a une **portée**, c'est-à-dire une zone du code dans laquelle elle peut être appelée.
- Les variables appelées dans le script principal ont une **portée globale**.
- La portée d'un paramètre est la fonction dans laquelle il a été déclaré.
- Il est possible d'appeler une variable globale depuis une fonction grâce au mot-clé **global**.
- Si on veut pouvoir modifier une variable directement dans une fonction, il faut la passer par référence (via le symbole &).

## IX. Exercice : Appliquez la notion

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



### Question 1

[solution n°4 p.23]

La fonction suivante permet d'incrémenter un compteur de deux récursivement en partant d'un chiffre donné jusqu'à 20.

```
1 <?php
2
3 function increaseToTwenty($start) {
4     echo $start . PHP_EOL;
5
6     if ($start < 20) {
7         $start = $start + 2;
8         increaseToTwenty($start);
9     }
10 }
11
12 increaseToTwenty(10);
```

Les variables statiques sont adaptées à la mise en place d'un compteur. Modifiez cette fonction afin que `$start` ne soit plus un paramètre, mais une variable statique qui continuera à être incrémentée automatiquement jusqu'à 20 lors de l'exécution de la fonction.

### Question 2

[solution n°5 p.24]

Déclarez une variable globale intitulée `city` et affectez-lui une valeur. Affichez ensuite le tableau contenant l'ensemble des variables globales.

## X. Les fonctions anonymes

### Objectif

- Découvrir ce qu'on appelle les fonctions anonymes et dans quels cas les utiliser

### Mise en situation

Certaines fonctions ne disposent pas de nom et sont déclarées à la volée : ce sont les fonctions anonymes.

#### Définition Les fonctions anonymes

On appelle **fonctions anonymes** ou **closures**, des fonctions que l'on crée à la volée et qui ne sont pas nommées.

Ces fonctions sont généralement utilisées comme des fonctions de rappel, sous le terme de **callback**, c'est-à-dire des fonctions que l'on va passer en argument d'une autre fonction pour effectuer une action spécifique.

1 <https://repl.it/>

Elles se déclarent ainsi :

```
1 <?php
2 function()
3 {
4     // instructions
5 };
```

Ces fonctions peuvent être stockées au sein de variables. Ainsi, nous pouvons utiliser la syntaxe suivante :

```
1 <?php
2
3 $cube = function($n)
4 {
5     return ($n * $n * $n);
6 };
7
8 // Affichera 27
9 echo $cube(3);
10
11
```

Nous serons amenés à utiliser ce type de fonctions lorsque nous voudrions faire appel à des fonctions telles qu'`array_map`<sup>1</sup>, qui permettent d'appliquer une fonction sur tous les éléments d'un tableau.

```
1 <?php
2
3 $cube = function($n)
4 {
5     return ($n * $n * $n);
6 };
7
8 // La fonction range permet de générer un tableau contenant un intervalle d'éléments
9 $array = range(1, 5);
10
11 // Affichera la valeur au cube de tous les éléments du tableau
12 print_r(array_map($cube, $array));
```

Nous pouvons également utiliser une variable extérieure grâce au mot-clé `use`.

```
1 <?php
2
3 $by = 5;
4 $multiplyBy = function ($n) use ($by)
5 {
6     return ($n * $by);
7 };
8
9 $array = range(1, 5);
10 print_r(array_map($multiplyBy, $array));
```

## Syntaxe À retenir

- Une **fonction anonyme** est une fonction sans nom qui peut être stockée dans une variable.
- Elle est utilisée lorsque des fonctions prennent en paramètre une autre fonction.

<sup>1</sup> <https://www.php.net/manual/fr/function.array-map.php>



## XI. Exercice : Appliquez la notion

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



### Question

[solution n°6 p.24]

Déclarez un tableau contenant des nombres entiers de 1 à 27, de trois en trois.

À l'aide de la fonction `array_map`<sup>2</sup>, ajoutez 7 à chacun des éléments du tableau. Le chiffre que vous ajoutez à chacun des éléments devra être paramétrable.

#### Indice :

La fonction `range` vous permet de définir un pas.

## XII. Les fonctions récursives

### Objectif

- Découvrir ce qu'on appelle les fonctions récursives et dans quels cas les utiliser

### Mise en situation

Dans certains cas, une même fonction peut nécessiter de faire appel à elle-même : ce sont les fonctions récursives.

#### Définition Les fonctions récursives

Nous avons vu qu'il était possible d'appeler une fonction au sein d'une autre fonction. Cependant, une fonction peut également faire appel à elle-même. On appelle cela une **fonction récursive**.

```
1 <?php
2
3 function decreaseToZero(int $start): void {
4     echo $start.PHP_EOL;
5
6     if ($start > 0) {
7         $start--;
8
9         decreaseToZero($start);
10    }
11 }
12
13 decreaseToZero(20);
```

1 <https://repl.it/>

2 <https://www.php.net/manual/fr/function.array-map.php>

### Attention

Lorsque l'on fait appel à une fonction récursive, attention à ce que celle-ci ne soit pas infinie ! Sinon notre fonction sera appelée en boucle et notre programme se terminera en erreur.

C'est par exemple le cas du code ci-dessous :

```
1 <?php
2
3 function decreaseForever(int $start): void {
4     echo $start.PHP_EOL;
5
6     $start--;
7     decreaseForever($start);
8 }
9
10 decreaseForever(20);
```

### Syntaxe À retenir

- Une **fonction récursive** est une fonction qui s'appelle elle-même. C'est une autre manière de faire des boucles.

### Remarque

Les fonctions récursives représentent une affaire complexe, le fait de ne pas les comprendre tout de suite n'est pas un problème.

## XIII. Exercice : Appliquez la notion

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



### Question

[solution n°7 p.24]

Déclarez et faites appel à une fonction récursive qui, pour un chiffre donné, l'incrémentera de deux tant qu'il ne sera pas supérieur à 20.

#### Indice :

Vous pouvez vous aider de la fonction `decreaseToZero` que nous avons déclarée précédemment.

## XIV. Auto-évaluation

### A. Exercice final

#### Exercice 1

[solution n°8 p.24]

Exercice

1 <https://repl.it/>

Le nom d'une fonction est sensible à la casse.

- ☐ Vrai
- ☐ Faux

#### Exercice

Parmi ces propositions concernant les règles de déclaration d'une fonction, lesquelles sont correctes ?

- ☐ Une fonction doit obligatoirement disposer d'un nom
- ☐ Une fonction doit obligatoirement disposer de paramètres
- ☐ Le nom d'une fonction peut comporter des chiffres
- ☐ Les parenthèses sont facultatives
- ☐ Le nom d'une fonction doit commencer par une lettre ou un \_

#### Exercice

Les paramètres par défaut d'une fonction doivent être situés :

- ☐ En premier
- ☐ En dernier

#### Exercice

Lorsqu'une fonction fait appel à elle-même dans ses instructions, on appelle cela :

- ☐ Une fonction anonyme
- ☐ Une fonction récursive

#### Exercice

Une fonction créée à la volée, généralement utilisée comme une fonction de rappel, est dite :

- ☐ récursive
- ☐ anonyme

#### Exercice

Selon la déclaration de cette fonction, ses paramètres de retour indiquent :

```
1 <?php
2
3 function myFunction(): ?array
4 {
5     // instructions
6 }
```

- ☐ Qu'elle peut retourner une valeur nulle
- ☐ Qu'elle peut retourner n'importe quelle valeur
- ☐ Qu'elle peut retourner un tableau

#### Exercice

Par défaut, la portée d'une variable au sein d'une fonction est considérée comme :

- ☐ Globale
- ☐ Locale

Exercice

Il est possible d'accéder à l'ensemble des variables globales au moyen d'un tableau associatif. Quel est le nom de ce tableau ?

Exercice

À la différence d'une variable locale, la valeur d'une variable statique est :

- ☐ Détruite lors de la fin de l'exécution de la fonction
- ☐ Conservée lors de la fin de l'exécution de la fonction

Exercice

Quel symbole permet d'indiquer que la valeur d'un paramètre sera passée par référence lors de la déclaration d'une fonction ?

- ☐ &
- ☐ ?
- ☐ #

## B. Exercice : Défi

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



### Question 1

[solution n°9 p.26]

Déclarez une fonction `greetings` qui permettra à un utilisateur d'afficher plusieurs informations le concernant :

- Son nom : il doit s'agir d'une chaîne de caractères
- Son âge : il doit s'agir d'un entier
- Ses langages de programmation préférés : il doit s'agir d'un tableau qui, par défaut, peut être vide

La fonction devra afficher les éléments suivants :

- "Bonjour, je m'appelle ... et j'ai ... ans."
- Si l'utilisateur n'a pas défini de langages de programmation préférés, affichez "Je n'ai pas de langage de programmation favori"
- Si l'utilisateur a défini des langages de programmation préférés, affichez "J'ai N langages de programmation favoris, il s'agit de : ... .."

1 <https://repl.it/>

**Indice :**

Vous pouvez vous aider des fonctions `sprintf`<sup>1</sup>, `empty`<sup>2</sup>, et `implode`<sup>3</sup>.

**Question 2**

[solution n°10 p.27]

Reprenez la solution de la question précédente.

Déclarez une seconde fonction, `countGreetings`, qui implémentera un compteur qui sera incrémenté à chaque appel.

Cette fonction devra retourner la valeur de son compteur.

La fonction `greetings` devra appeler la fonction `countGreetings` à chaque fois qu'elle est appelée, et affichera le message "La fonction a été exécutée ... fois".

Réalisez plusieurs appels de votre fonction et affichez la valeur du compteur après avoir réalisé le dernier appel.

**Indice :**

Les variables statiques sont idéales pour utiliser un compteur.

**Solutions des exercices**

---

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

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

3 <https://www.php.net/manual/fr/function.implode.php>



**p. 5 Solution n°1**

```
1 <?php
2
3 function printDateTime()
4 {
5     echo date('d/m/Y h:i');
6 }
7
8 printDateTime();
```

**p. 8 Solution n°2**

Vous devriez avoir obtenu un résultat similaire :

```
1 <?php
2
3 function greetings(string $name, int $age = null)
4 {
5     echo sprintf("Bonjour, je m'appelle %s", $name) . PHP_EOL;
6
7     if (!empty($age)) {
8         echo sprintf("J'ai %s ans ", $age) . PHP_EOL;
9     }
10 }
11
12 greetings('Jules', 27);
13 greetings('Sophia');
```

**p. 11 Solution n°3**

```
1 <?php
2
3 function getDateTime(): string
4 {
5     return date('d/m/Y h:i');
6 }
7
8 $currentDate = getDateTime();
9 echo $currentDate;
```

**p. 15 Solution n°4**

```
1 <?php
2
3 function increaseToTwenty() {
4     static $start = 10;
5
6     echo $start . PHP_EOL;
7
8     if ($start < 20) {
9         $start = $start + 2;
10        increaseToTwenty();
11    }
```

```
11     }
12 }
13
14 increaseToTwenty();
```

#### p. 15 Solution n°5

```
1 <?php
2
3 global $city;
4
5 $city = 'Montpellier';
6
7 print_r($GLOBALS);
```

#### p. 17 Solution n°6

Vous avez dû obtenir une fonction similaire à celle-ci :

```
1 <?php
2
3 $numberToAdd = 7;
4 $add = function ($n) use ($numberToAdd)
5 {
6     return ($n + $numberToAdd);
7 };
8
9 $array = range(1, 27, 3);
10 print_r(array_map($add, $array));
```

#### p. 18 Solution n°7

Vous avez dû obtenir une fonction similaire à celle-ci :

```
1 <?php
2
3 function increaseToTwenty($start) {
4     echo $start . PHP_EOL;
5
6     if ($start < 20) {
7         $start = $start + 2;
8         increaseToTwenty($start);
9     }
10 }
11
12 increaseToTwenty(10);
```

#### Exercice p. 18 Solution n°8



**Exercice**

Le nom d'une fonction est sensible à la casse.

- ☐ Vrai
- ☒ Faux

**Exercice**

Parmi ces propositions concernant les règles de déclaration d'une fonction, lesquelles sont correctes ?

- ☐ Une fonction doit obligatoirement disposer d'un nom
- ☐ Une fonction doit obligatoirement disposer de paramètres
- ☒ Le nom d'une fonction peut comporter des chiffres
- ☐ Les parenthèses sont facultatives
- ☒ Le nom d'une fonction doit commencer par une lettre ou un \_

**Exercice**

Les paramètres par défaut d'une fonction doivent être situés :

- ☐ En premier
- ☒ En dernier

**Exercice**

Lorsqu'une fonction fait appel à elle-même dans ses instructions, on appelle cela :

- ☐ Une fonction anonyme  
*Les fonctions anonymes sont des fonctions créées à la volée, utilisées notamment comme des fonctions de rappel*
- ☒ Une fonction récursive

**Exercice**


Une fonction créée à la volée, généralement utilisée comme une fonction de rappel, est dite :

- ☐ récursive  
*Les fonctions récursives sont des fonctions qui font appel à elles-même dans leurs instructions*
- ☒ anonyme

**Exercice**


Selon la déclaration de cette fonction, ses paramètres de retour indiquent :

```
1 <?php
2
3 function myFunction(): ?array
4 {
5     // instructions
6 }
```

- ☒ Qu'elle peut retourner une valeur nulle
- ☐ Qu'elle peut retourner n'importe quelle valeur
- ☒ Qu'elle peut retourner un tableau
-  Depuis PHP 7.1, la notation ci-dessus permet d'indiquer que la fonction peut retourner le type de valeur indiqué, ou une valeur nulle, symbolisée par le ?.

### Exercice


Par défaut, la portée d'une variable au sein d'une fonction est considérée comme :

- ☐ Globale
- ☒ Locale
-  Par défaut, une variable possède le niveau local, c'est-à-dire qu'elle ne sera modifiée qu'à l'intérieur de la fonction où elle est utilisée et retrouvera la valeur qu'elle avait juste avant l'exécution de la fonction.

### Exercice


Il est possible d'accéder à l'ensemble des variables globales au moyen d'un tableau associatif. Quel est le nom de ce tableau ?

\$GLOBALS

-  Le tableau \$GLOBALS est un tableau associatif contenant l'ensemble des variables globales comme clé et les valeurs des éléments du tableau comme valeur des variables.


### Exercice

À la différence d'une variable locale, la valeur d'une variable statique est :

- ☐ Détruite lors de la fin de l'exécution de la fonction
- ☒ Conservée lors de la fin de l'exécution de la fonction
-  La valeur d'une variable statique ne sera pas détruite lors de la fin de l'exécution de la fonction, elle sera conservée pour pouvoir être réutilisée lors d'une prochaine exécution.

### Exercice

Quel symbole permet d'indiquer que la valeur d'un paramètre sera passée par référence lors de la déclaration d'une fonction ?

- ☒ &
- ☐ ?
- ☐ #
-  Pour modifier une variable, il est possible de la faire précéder du caractère &.
- Avec cette notation, on précise ainsi qu'il s'agit d'un alias : la valeur de la variable est modifiée à la sortie de la fonction. On parle alors de passage par référence.

Vous devriez avoir obtenu un résultat similaire :

```

1 <?php
2
3 function greetings(string $name, int $age, array $favoriteLanguages = [])
4 {
5     echo(sprintf("Bonjour, je m'appelle %s et j'ai %s ans. ", $name, $age));
6
7     if (empty($favoriteLanguages)) {
8         echo "Je n'ai pas de langage de programmation favori" . PHP_EOL;
9     } else {
10        echo(sprintf("J'ai %s langages favoris, il s'agit de : %s " . PHP_EOL,
11        count($favoriteLanguages), implode(', ', $favoriteLanguages)));
12    }
13 }
14 greetings('Laure', 27, ['PHP', 'Python']);
15 greetings('Julie', 26);
16

```

#### p. 21 Solution n°10

Vous devriez avoir obtenu un résultat similaire :

```

1 <?php
2
3 function greetings(string $name, int $age, array $favoriteLanguages = [])
4 {
5     echo(sprintf("Bonjour, je m'appelle %s et j'ai %s ans. ", $name, $age));
6
7     if (empty($favoriteLanguages)) {
8         echo "Je n'ai pas de langage de programmation favori" . PHP_EOL;
9     } else {
10        echo(sprintf("J'ai %s langages favoris, il s'agit de : %s " . PHP_EOL,
11        count($favoriteLanguages), implode(', ', $favoriteLanguages)));
12    }
13
14    echo sprintf('La fonction a été exécutée %s fois' . PHP_EOL, countGreetings());
15 }
16
17 function countGreetings()
18 {
19     static $count = 0;
20
21     return ++$count;
22 }
23 greetings('Laure', 27, ['PHP', 'Python']);
24 greetings('Julie', 26);
25

```