Info 303

Introduction au PHP

Alin F. Rabat C.

Université de Reims Champagne Ardenne

21 septembre 2018

Sommaire

Les noms de fonctions suivent les mêmes règles que les autres labels en PHP. Un nom de fonction valide commence par une lettre ou un souligné, suivi par un nombre quelconque de lettres, de nombres ou de soulignés. Ces règles peuvent être représentées par l'expression rationnelle suivante :

```
[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*..
```

Listing 1 – Exemple de déclaration d'une fonction

```php
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Exemple de fonction.\n";
    return $retval;
}
?>
```

Les fonctions n'ont pas besoin d'être définies avant d'être utilisées. Par contre lorsqu'une fonction est définie de manière conditionnelle,sa définition doit nécessairement précéder son utilisation.

Toutes les fonctions et classes en PHP ont une portée globale - elles peuvent être appelées à l'extérieur d'une fonction si elles ont été définies à l'intérieur et vice-versa.

PHP ne supporte pas la surchage, la destruction et la redéfinition de fonctions déjà déclarées.

Les noms de fonctions sont insensibles à la casse, et il est généralement admis que les fonctions doivent être appelées avec le nom utilisé dans leur déclaration, y compris la casse.

Il est possible d'appeler des fonctions récursives en PHP.

Exemple :

```php
<?php
function recursion($a)
{
    if ($a < 20) {
        echo "$a\n";
        recursion($a + 1);
    }
}
?>
```

Les paramètres sont passés à une fonction en utilisant une liste d'arguments, dont chaque expression est séparée par une virgule.
Les arguments sont évalués de la gauche vers la droite.
PHP supporte le passage d'arguments par valeur (comportement par défaut). Il est aussi possible de passer des arguments par référence. On peut définir des valeurs d'arguments par défaut. Enfin une fonction peut accepter liste variable d'arguments.

Par défaut, un argument est passé à la fonction par valeur, c'est à dire que tout changement de la valeur du paramètre à l'intérieur de la fonction ne modifie pas le contenu de la variable passée comme paramètre . Si l'on souhaite qu'une fonction puisse changer la valeur d'argument, il faut passer l'argument par référence.
Pour passer un argument par référence, on ajoute un '&' devant l'argument dans la déclaration de la fonction :

Listing 2 – Passage de paramètre par référence

```php
<?php
function add_some_extra(&$string)
{
    $string .= ', et un peu plus.';
}
$str = 'Ceci est une chaine';
add_some_extra($str);
echo $str; // Affiche : 'Ceci est une chaine, et un peu plus.'
?>
```

Il est possible de définir des valeurs par défaut pour les arguments de type scalaire :

Listing 3 – Valeur par défaut des arguments de fonctions

```php
<?php
function servir_cafe ($type = "cappuccino")
{
    return "Servir un $type.\n";
}
echo servir_cafe();
echo servir_cafe(null);
echo servir_cafe("espresso");
?>
```

Dans ce cas si le paramètre est omis, c'est la valeur par défaut qui est utilisée.

Il est possible d'utiliser des tableaux ainsi que le type spécial NULL comme valeur de paramètre par défaut.

Listing 4 – Utilisation de type non scalaire comme valeur par défaut

```php
<?php
function servir_cafe($types = array("cappuccino"),
         $coffeeMaker = NULL)
{
    $device = is_null($coffeeMaker) ? "les mains" : $coffeeMaker;
    return "Preparation d'une tasse de ".join(", ", $types)." avec $device.\n";
}
echo servir_cafe();
echo servir_cafe(array("cappuccino", "lavazza"), "une cafetiere");
?>
```

La valeur par défaut d'un argument doit obligatoirement être une constante, et ne peut être ni une variable, ni un membre de classe, ni un appel de fonction.

Si l'on utilise simultanément des arguments présentant valeur par défaut et d'autres sans valeur par défaut, les premiers doivent être placés à la suite de tous les paramètres sans valeur par défaut. Sinon, cela ne fonctionnera pas.

Considérons le code suivant :

```php
<?php
function faireunyaourt ($type = "acidophilus", $flavour)
{
    return "Preparer un bol de $type $flavour.\n";
}
// ne fonctionne pas comme voulu
echo faireunyaourt("framboise");
?>
```

Ce code ne fonctionne pas car les paramètres sont affectés aux arguments de gauche à droite. Il n'est donc pas possible de ne pas fournir de valeur à l'argument `$type`. Pour que cela fonctionne il faut la définir la fonction ainsi :

```php
<?php
function faireunyaourt ($flavour, $type = "acidophilus")
{
    return "Preparer un bol de $type $flavour.\n";
}
 // fonctionne comme voulu
echo faireunyaourt ("framboise");
?>
```

Les déclarations de type permettent aux fonctions de requérir que certains paramètres soient d'un certain type lors de l'appel de celles-ci. Si la valeur donnée est d'un type incorrect alors PHP 7 lève une exception **TypeError**.
Pour spécifier une déclaration de type, on précise le type du paramètre avant son nom. La déclaration accepte la valeur NULL si la valeur par défaut du paramètre est définie à NULL.

Type	Description
Nom de la Classe/interface	Le paramètre doit être une instanceof de la classe ou interface donnée.
self	Le paramètre doit être une instanceof de la même classe qui a défini la méthode. Ceci ne peut être utilisé que sur les méthodes des classes et des instances.
array	Le paramètre doit être un array.
callable	Le paramètre doit être un callable valide.
bool	Le paramètre doit être un boolean.
float	Le paramètre doit être un nombre flottant (float).
int	Le paramètre doit être un integer.
string	Le paramètre doit être une string.
iterable	Le paramètre doit être soit un array ou une instanceof Traversable.
object	Le paramètre doit être un object.

Par défaut, PHP convertit les mauvais types vers le type scalaire attendu quand c'est possible. Il est possible d'activer un *typage strict* fichier par fichier. Dans ce mode, seule une variable du type attendu dans la déclaration sera acceptée sinon on obtient une erreur TypeError. La seule exception à cette règle est qu'un entier (integer) peut être passé à une fonction attendant un nombre flottant (float). Les appels aux fonctions depuis des fonctions internes ne seront pas affectés par la déclaration `strict_types`.

Pour activer le typage strict, il faut appeler la fonction `declare` avec le paramètre `strict_types` :

```php
<?php
declare(strict_types=1);

function sum(int $a, int $b) {
    return $a + $b;
}

var_dump(sum(1, 2));
var_dump(sum(1.5, 2.5));
?>
```

Le typage strict s'applique aux appels de fonction effectués depuis l'intérieur d'un fichier dont le typage strict est actif, et non aux fonctions déclarées dans ce fichier. Si un fichier dont le typage strict n'est pas activé effectue un appel à une fonction qui a été définie dans un fichier dont le type strict est actif, la préférence de l'appelant sera respecté, et la valeur sera forcée.

PHP supporte les fonctions à nombre d'arguments variable. Pour cela on utilise le mot clé...
La liste des arguments peut inclure le mot clé ... pour indiquer que cette fonction accepte un nombre variable d'arguments. Les arguments seront passés dans la variable fournie sous forme d'un tableau.

Listing 5 – Nombre de paramètres variables

```php
<?php
function sum(...$numbers) {
    $acc = 0;
    foreach ($numbers as $n) {
        $acc += $n;
    }
    return $acc;
}
echo sum(1, 2, 3, 4);
?>
```

Les valeurs sont renvoyées en utilisant une instruction de retour optionnelle. Tous les types de variables peuvent être renvoyés, tableaux et objets compris. Cela fait que la fonction finit son exécution immédiatement et passe le contrôle à la ligne appelante. Voir return pour plus d'informations.

Si return est omis, c'est la valeur NULL qui est retournée.

Listing 6 – Utilisation de return

```php
<?php
function carre($num)
{
    return $num * $num;
}
echo carre(4); // Affiche '16'
?>
```

Une fonction ne peut pas renvoyer plusieurs valeurs en même temps, mais vous pouvez obtenir le même résultat en renvoyant un tableau.

Listing 7 – Utiliser un tableau pour retourner plusieurs valeurs

```php
<?php
function petit_nombre()
{
    return array (0, 1, 2);
}
list ($zero, $un, $deux) = petit_nombre();
?>
```

Il est possible de retourner la référence à une variable. Pour cela il faut utiliser l'opérateur & à la fois dans la déclaration de la fonction et dans l'assignation de la valeur de retour.

```php
<?php
function &retourne_reference()
{
    return $uneref;
}

$newref =& retourne_reference();
?>
```

- PHP 7 ajoute le support des déclarations du type de retour. Les mêmes types sont disponibles pour le typage du retour que pour le typage des arguments.
- Le typage strict affecte aussi les déclarations du type de retour.
- Dans le mode faible par défaut, les valeurs retournées seront transtypées vers le type demandé si elles ne sont pas déjà de celui-ci.
- Dans le mode strict, la valeur retournée doit être du bon type sinon il y a génération une exception TypeError.

Lors de la surcharge d'une méthode parente, la méthode fille doit retourner une valeur cohérente avec le type de valeur retournée par la méthode parente. Si le parent ne définit par de type de retour, il doit en être de même pour la fille.

Listing 8 – Déclaration du type du retour

```php
<?php
function sum($a, $b): float {
    return $a + $b;
}

// Note qu'une valeur flottante sera retournee.
var_dump(sum(1, 2));
?>
```

PHP supporte le concept de fonctions variables. Cela signifie que si le nom d'une variable est suivi de parenthèses, PHP recherchera une fonction de même nom, et essaiera de l'exécuter. Cela peut servir, entre autres, pour faire des fonctions de rappel, des tables de fonctions...

Les fonctions variables ne peuvent pas fonctionner avec les éléments de langage comme les `echo`, `print`, `unset()`, `isset()`, `empty()`, `include`, `require` etc. Vous devez utiliser votre propre gestion de fonctions pour utiliser un de ces éléments de langage comme fonctions variables.

Listing 9 – Exemple de fonction variable

```php
<?php
function foo() {
    echo "dans foo()<br />\n";
}
function bar($arg = '')
{
    echo "Dans bar(); l'argument etait '$arg'.<br />\n";
}
// Ceci est une fonction detournee de echo
function echoit($string)
{
    echo $string;
}
$func = 'foo';
$func();        // Appel foo()
$func = 'bar';
$func('test');  // Appel bar()
$func = 'echoit';
$func('test');  // Appel echoit()
?>
```

Les fonctions anonymes, aussi appelées fermetures ou closures permettent la création de fonctions sans préciser leur nom. Elles sont particulièrement utiles comme fonctions de rappel, mais leur utilisation n'est pas limitée à ce seul usage.
Les fonctions anonymes sont implémentées en utilisant la classe Closure.

Listing 10 – Fonctions anonymes

```php
<?php
echo preg_replace_callback('~-([a-z])~', function ($match) {
    return strtoupper($match[1]);
}, 'bonjour-le-monde');
?>
```

Les fonctions anonymes peuvent aussi être utilisées comme valeurs de variables. PHP va automatiquement convertir ces expressions en objets Closure. Assigner une fermeture à une variable est la même chose qu'une assignation classique, y compris pour le point-virgule final.