PHP : Les fonctions

L3 Informatique - UE Développement Web

David Lesaint

david.lesaint@univ-angers.fr

Janvier 2020

Les fonctions natives de PHP

Vérifier la disponibilité de modules/fonctions (eg. chez votre hébergeur)

Liste des modules installés

- `array get_loaded_extensions()`

Liste des fonctions disponibles dans un module d'extension

- `array get_extensions_funcs("nom_module")`

Tester l'existence d'une fonction

- `bool function_exists("nom_fonction")`

exemple7-1.php

```php
1  //Tableau contenant le nom des extensions
2  $tabext = get_loaded_extensions(); natcasesort($tabext);
3  foreach ($tabext as $cle=>$valeur) {
4   echo "<h3>Extension  $valeur </h3> ";
5   //Tableau contenant le nom des fonctions
6   $fonct = get_extension_funcs($valeur); sort($fonct);
7   for ($i=0;$i<count($fonct);$i++) {
8    echo $fonct[$i],"    \n";
9    echo "<hr />";
10  }
11 }
```

Définir ses fonctions

Différentes sortes de fonctions

- Avec ou sans paramètres.
- Avec un nombre fixe ou variable de paramètres.
- Avec paramètres par défaut ou non.
- Avec ou sans valeur de retour.
- Avec passage par référence ou non.
- Avec retour par référence ou non.
- Avec variables statiques ou non.
- Avec itération sur valeurs de retour (générateur) ou non.
- Nommée ou anonyme, dynamique ou non.
- Avec capture de l'environnement lexical (fermeture) ou non.
- Typée fortement ou faiblement.

Définition de fonction

Déclaration=Définition

- Pas de déclaration séparément de la définition.
- Peut être définie n'importe où dans un fichier.

En-tête

- Mot-clé `function`.
- Nommage : mêmes règles que pour les variables.
- Paramètres : liste de variables.

Appel

- Peut être appelée avant sa définition dans le script.
- Peut être appelée sans arguments même si elle en attend (avertissement émis) !

Fonctions sans valeur retour

exemple7-2.php

```
<!DOCTYPE html>
<html lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
  <title>Fonctions ne retournant pas de valeur</title>
 </head>
 <body>
  <div>
// Fonction sans argument
function ladate() {
 echo "<table ><tr><td

style=\"background-color:blue;color:yellow;border-width:10px;border-style:groove;
bordercolor:FFCC66;font-style:fantasy;font-size:30px\"> ";
 echo "$a ",date("\l\\e d/m/Y \i\l \\e\s\\t H:i:s");
 echo "</td></tr></table><hr />";
}
// Fonction avec un argument
function ladate2($a) {
 echo "<table><tr><td
style=\"background-color:blue;color:yellow;border-width:10px;
border-style:groove;border-color:FFCC66;font-style:fantasy;font-size:30px\">";
 echo "$a ",date("\l\\e d/m/Y \i\l \\e\s\\t H:i:s");
 echo "</td></tr></table><hr />";
}
// Appels des fonctions
echo ladate();
echo ladate2("Bonjour");
echo ladate2("Salut");
echo ladate2();//Provoque un avertissement (Warning)
echo @ladate2();//Empêche l'apparition du message d'avertissement
</div>
```

Fonctions sans valeur retour

exemple7-3.php

```
 1 <!DOCTYPE html>
 2 <html lang="fr">
 3  <head>
 4   <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
 5   <title>Fonction de lecture de tableaux</title>
 6  </head>
 7  <body>
 8   <div>
 9 // Définition de la fonction
10 function tabuni($tab,$bord,$lib1,$lib2)
11 {
12  echo "<table border=\"$bord\" width=\"100%\"><tbody><tr><th>$lib1</th>
<th>$lib2 </th></ tr>";
13  foreach ($tab as $cle=>$valeur)
14  {
15   echo "<tr><td>$cle</td> <td>$valeur </td></tr>";
16  }
17  echo "</tbody> </table><br />";
18 }
19 // Définition des tableaux
20 $tab1 = array("France"=>"Paris","Allemagne"=>"Berlin","Espagne"=>"Madrid");
21 $tab2 = array("Poisson"=>"Requin","Cétacé"=>"Dauphin","Oiseau"=>"Aigle");
22 // Appels de la fonction
23 tabuni($tab1,1,"Pays","Capitale");
24 tabuni($tab2,6,"Genre","Espèce");
25 </div>
26 </body>
27 </html>
```

Typage des paramètres (PHP 7+)

Noms de types valides

Pour déclarer paramètres et valeur de retour :

- `int`, `float`, `string`, `bool`, `array`.
- `callable` : fonctions de rappel.
- Nom de classe ou interface.
  - `self` : Réfère à la classe définissant la méthode.

Valeur de retour inutile ou aucun paramètre en entrée : `void`.

Typage faible vs. typage fort (alias typage strict)

Choix du typage fort en incluant en début de fichier la directive `declare(strict_types=1);`

- Exception `TypeError` levée en cas d'erreur de type à l'appel de fonctions.

Transtypage automatique en cas de typage faible.

Typage faible et typage fort

function-typing-weak.php

```php
function sum(int $a, int $b) : int
   { return $a + $b; }
echo sum(1, 2);
// conversion automatique de float vers int
echo '\n'.sum(1.5, 2.5);
```

function-typing-strict.php

```php
declare(strict_types=1);
function sum(int $a, int $b) : int { return $a + $b; }
try {
      echo sum(1.5, 2.5);
} catch (TypeError $e) {
      echo 'Erreur : '.$e->getMessage();
}
```

Retourner plusieurs valeurs avec un tableau

exemple7-5.php

```
 1 <!DOCTYPE html>
 2 <html lang="fr">
 3  <head>
 4   <meta charset="UTF-8" />
 5   <title>Nombres complexes</title>
 6  </head>
 7  <body>
 8   <div>
 9 function modarg($reel,$imag) {
10      // $mod= hypot($reel,$imag);
11      // ou encore si vous n'avez pas la fonction hypot()du module standard
12      $mod =sqrt($reel*$reel + $imag*$imag);
13      $arg = atan2($imag,$reel);
14      return array("module"=>$mod,"argument"=>$arg);
15 }
16 // Appels de la fonction
17 $a= 5;
18 $b= 8;
19 $complex= modarg($a,$b);
20 echo "<b>Nombre complexe $a + $b i:</b><br /> module = ", $complex["module"]
, "<br />argument = ",$complex["argument"]," radians<br />";
21 </div>
22 </body>
23 </html>
```

Paramètres par défaut

Passage de paramètres par défaut

Les paramètres “les plus à droite” peuvent avoir des valeurs par défaut.

function-default.php

```
1 function f($a, $b=2) { return $a*$b; }
2 echo f(10);
```

Fonctions variadiques

Emulation à l'aide d'un argument de type tableau (exemple7-7.php)

```
<!DOCTYPE html>
<html lang="fr">
 <head>
  <meta charset="UTF-8" />
  <title>Nombre de paramètres variable</title>
 </head>
 <body>
  <div>
function prod($tab) {
 $n=count($tab);
 echo "Il y a $n paramètres :";
 $prod = 1;
 foreach ($tab as $val) {
  echo "$val, " ; $prod *=$val;
 }
 echo " le produit vaut ";
 return $prod;
}
$tab1= range(1,10);
echo "Produit des nombres de 1 à 10 : ", prod($tab1),"<br />";
$tab2 = array(7,12,15,3,21);
echo "Produit des éléments : ", prod($tab2),"<br />";
</div>
</body>
</html>
```

Informations sur les paramètres

Nombre d'arguments passés à une fonction

- `int func_num_args()`

Accès à paramètre individuel par position (0 pour le 1er)

- `mixed func_get_arg(int)`

Tableau indicé des paramètres

- `array func_get_args()`

exemple7-8.php

L'opérateur ... (PHP 5.6+)

Pour définir des fonctions variadiques

Syntaxe : `f($x,...$tab);`

- `$tab` traité comme un tableau comportant un nombre quelconque de paramètres supplémentaires.
- Lecture classique de `$tab` avec boucle.

Différents appels possibles

- Une seule valeur liée à `$x` : `f(1)`
- Une liste de paramètres, le premier lié à `$x`, les autres à `$tab` : `f(1,3,4)`
- Un tableau lié à `$tab` : `f(...[1,3,4])`
- Une valeur et un tableau liés respectivement à `$x` et `$tab` : `f(1,...[3,4])`

L'opérateur . . . (PHP 5.6+)

exemple7-9.php

```php
function prod($a,...$tab) {
  foreach ($tab as $nb) { $a *=$nb; }
  echo " Le produit vaut ";
  return $a;
}
echo "Produit 1",prod(2*3*4*5),"<br/>";//120
echo "Produit 2",prod(2,3,4,5),"<br/>";//120
$tab1= range(2,5);
echo "Produit 3",prod(...$tab1),"<br/>";//120
$tab2 = array(3,4,5);
echo "Produit 4",prod(2,...$tab2),"<br/>";//120
```

Les générateurs (PHP 5.5+)

Fonction retournant plusieurs valeurs à la demande

- Alternative efficace à la création et au renvoi de tableau.
- Crée et renvoie un itérateur (objet de classe `Generator`).
- Le code appelant peut itérer sur les valeurs générées avec `foreach`.
- Syntaxe avec mot-clé `yield` :
  - `yield $value;` pour générer des valeurs sans clés.
  - `yield $key=>$value;` pour générer des valeurs avec clés.

exemple7-10.php

```
1 function suite($min,$max,$pas) {
2   for($i=$min;$i<=$max;$i+=$pas) { yield $i; }
3 }
4 echo "Suite : ";
5 foreach(suite(0,50,10) as $nb) { echo $nb, " /// "; }
```

Les générateurs (PHP 5.5+)

Avec instruction `return`

Valeur de retour accessible par invocation de la méthode `getReturn()` sur le générateur.

exemple7-11.php

```php
function suite($min,$max,$pas) {
  $total=0;
  for($i=$min;$i<=$max;$i+=$pas) {
    yield $i;
    $total+=$i;
  }
  return $total;
}
echo "Suite : ";
$gener = suite(0,45,5);
foreach($gener as $nb) {
      echo $nb, " ";
}
echo "<br/> Total = ",$gener->getReturn();
```

Délégation de générateurs (PHP 7+)

Appel de générateur à générateur avec `yield from`

exemple7-12.php

```php
function nom($prenom,$nom) {
 yield $prenom;
 yield $nom;
 yield from adresse();
 yield from code();
}
function adresse() {
 yield "Paris"; yield "France";
}
function code() {
 yield "75006";
}
foreach(nom("Jan","Geelsen") as $coord) {
 echo $coord,"\n";
}
```

Portée des variables

Déterminée selon le contexte

Portée globale pour toute variable déclarée en dehors d'une fonction ou d'une classe (portée limitée au script et aux scripts inclus).

variable-scope-include.php

```php
echo $a;
```

variable-scope.php

```php
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
</head>
<body>
$a = 1;
include 'variable-scope-include.php';
<div></div>
echo $a;
</body>
</html>
```

Portée des variables

Au sein des fonctions

Portée locale pour toute variable définie dans une fonction ou une classe SAUF si redéclarée avec `global` ou utilisée avec `$GLOBALS[]`.

variable-scope-function.php

```php
$a = $b = $c = 1;
function test() {
    global $b;
    echo $b;
    echo $GLOBALS['c'];
    echo $a; // undefined variable $a
}
test();
```

Variables statiques

Variable statique

- Variable locale déclarée avec `static` dans une fonction.
- Garde sa valeur en sortie d'appel à la fonction pendant la durée du script.

variable-static.php

```php
function test() {
    static $a = 0;
    echo $a;
    $a++;
}
test(); // 0
test(); // 1 ...
```

Paasage par référence

Référence en PHP

- Semblable à la notion de lien sur fichier en LINUX.
- Différent de la notion de pointeur en C (pas d'arithmétique sur références ...).

Trois usages

- Affectation par référence : opérateur `=&`
- Passage par référence : `function f(&$a)`
- Retour par référence : `function &f(); $a= &f();`

Affectation par référence - Destruction de référence

reference1.php

```php
// affectation par référence
$b = 10;
$a =& $b;
echo "\$a=".$a; //10
$a += 1;
echo "\n\$b=".$b; //11
```

reference2.php

```php
// destruction de référence
$b = "b";
$c = "c";
$a =& $b;
echo "\n\$a=".$a; // b
$a =& $c;
echo "\n\$a=".$a; // c
echo "\n\$b=".$b; // b
unset($a);
echo "\n\$a=".$a; // Undefined
echo "\n\$c=".$c; // c
```

Fonctions : passage et retour par référence

reference3.php

```php
// passage par référence
function foo(&$var) { $var++; }
function bar($var) { $var++; }
foo($a);
echo "\n\$a=".$a;
bar($a);
echo "\n\$a=".$a;
```

reference4.php

```php
// retour par référence
function &collector() {
    static $collection = array();
    print_r($collection);
    return $collection;
}
// !! préfixer l'appel avec &
$collect = &collector(); //Array()
$collect[] = 'foo';
collector();//Array([0]=>foo)
```

Fonctions dynamiques

Nom de fonction déterminé par une variable `string`

Cf. variables dynamiques.

function-dynamic.php

```php
$f1 = "function_exists";
$f2 = "date";
echo $f1($f1);// équivaut à function_exists("function_exists");
echo "\n";
echo $f2("d/M/Y");  // équivaut à date("d/M/Y");
// Bloc équivalent à ce qui précède
$tf = ["function_exists","date"];
echo $tf[0]($tf[0]);
echo "\n";
echo $tf[1]("d/M/Y");
```

Fonctions de rappel

Fonction de rappel (alias callback)

- Fonction nommée ou anonyme, méthode d'objet ou méthode statique de classe.
- Pseudo-type `callable`.
- Peut être passée comme argument à d'autres fonctions (eg. fonctions de tri sur tableaux) :
  - Par la chaîne la dénommant.
  - Par la variable objet la représentant si elle est anonyme : on parle alors de fermeture (alias closure).

Fonctions anonymes (alias fermetures)

function-anonymous.php

```php
$tab=array(1,2,3);
// fonction anonyme
$tab1=array_map(function($n) { return $n*$n; },$tab);
print_r($tab1);
```

function-anonymous2.php

```php
$tab=array(1,2,3);
// fermeture (objet Closure)
$f = function($n) {return $n*$n;};
echo $f(10);
$tab1=array_map($f,$tab);
print_r($tab1);
```

Accès d'une fermeture à l'environnement lexical

Accès aux propriétés d'un objet

- Par appel à la méthode `call()` de la fermeture (objet `Closure`).

Accès par référence (ou non) à variable non locale

- Avec syntaxe `use(&$x)`.

exemple7-22.php

```php
class Nombre {
  public $valeur1=22;
  }
$varadd = function ($a) use(&$text) { return $text.($this->valeur1 + $a); };
$text = "Somme = ";
//Appel en créant un objet
echo $varadd->call(new Nombre,7); //Somme = 29
//Appel avec un objet déjà créé
$nb = new Nombre();
echo $varadd->call($nb,7); //Somme = 29
$text = "Addition = ";
$nb->valeur1=55;
echo $varadd->call($nb,7);//Addition = 62
```