Last updated: Fri, 26 Feb 2010

Functions

Table of Contents

User Contributed Notes
Functions

Raashell
05-Oct-2009 11:17

I was a little slow on the uptake for the same question Vameza describes. Here is a contrasting set of code that outlines the difference.

```
<?php
function a($n){
b($n);
return ($n * $n);
}

function b(&$n){
$n++;
}

echo a(5); //Outputs 36
?>
```

info at warpdesign dot fr
08-Dec-2008 08:14

Note about function names:
--

According to the specified regular expression ([a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*), a function name like this one is valid:

```
<?php
function â,¬()
{
 echo 'foo';
}
?>
```

And PHP interpreter correctly interprets the function. However some parsers (ex: PDT/Eclipse) does not see this as a valid function name, and only accept function names starting with a letter or an underscore.

dnhuff at acm.org
07-Jun-2008 04:31

Actually, vameza, the script below echos nothing, because the echo statement has the expression $a(5), whereas you meant a(5). This demonstrates another feature of PHP, variable functions, but you didn't actually mean to do that.

To be helpful to the average joe, actually run your examples and then cut and paste the code.

vameza at gmail dot com
11-May-2008 11:15

An important thing that must be considered when using functions and references is that functions that do not return any value at all, will indeed return NULL as result.
Carefull must be taken in order to avoid unexpected results, as show below:

```php
<?php
function a($n)
{
return ( b($n) * $n );
}
function b(&$n){
++$n;
}

echo $a(5);
// The result is 0. Why?? Answer below...
?>
```

As the function b not return any value, the engine makes the function returns NULL. When is performed a multiplication in function a, the returned value is converted to zero, and finally the returned value from function a is zero as well.
I took long to realize this concept. An answer can be found in the Zend PHP5 Certification Study Guide (PHP/architectÂ´s publishing).
So, in conclusion, be carefull when using a function to perform operations. It must return a value.

Jakob Thomsen
26-Feb-2008 08:57

Note that if you have the name of function in a variable, then you can call it, if that function is globally available. It cannot be the name of a function in a class.
An example:

```
class foo {
  public static function bar(){
    return "bar in a class called";
  }
}

function bar() {
  return "normal bar called";
}

$strFN = "bar";
echo $strFN();

$strFN2 = "foo::bar";
echo $strFN2();

This will result in

normal bar called
Fatal error: Call to undefined function foo::bar() in /path/to/test.php on line 16
```

email at fake dot com
31-Aug-2007 05:06

You can set variable DEFUALT values for a function in the (). These will be over-written by any input values (or non values).

```
function defualt_values_test ($a = 123, $b = 456){
 echo "a = ".$a."<br/>";
 echo "b = ".$b."<br/>";
 echo "<br/>";
}

defualt_values_test();       // uses values set in 'header'
defualt_values_test('overwritten',987654321);       // uses these values
defualt_values_test($non_existant,"var A has to be overwritten.");        // you can only use this for the last vars.

OUTPUTS:
----------

a = 123
b = 456

a = overwritten
b = 987654321

a =
b = var A has to be overwritten.
```

Gautam
23-Aug-2007 09:57

```php
<?php
/*
```

```
User Defined Functions

function function_name($arg_1, $arg_2, ...,  $arg_n)
{
  code_line1;
  code_line2;
  code_line3;
  return ($value); //stops execution of the function and returns its argument as the value at the point where the function was called.
}
One may have more than one return()statements in a function.
*/

  $text= 'This Line is Bold and Italics.';
  function makebold_n_italics($text)
  {
    $text = "<i><b>$text</i></b>";
    return($text); //the return() statement immediately ends execution of the current function, and returns its argument as the value of
the function call in print command
  }
  print("This Line is not Bold.<br>\n");
  print("This Line is not Italics.<br>\n");
  echo makebold_n_italics("$text") ,"--->", 'It prints the returned value of variable $text when function is called.'."<br>\n";
  echo "$text", '---> prints the original value of  variable $text.'."<br>\n"; // prints the original value of $text
  $thanks='Thanks to Zeev Suraski and Andi Gutmans !!!';
  $text=$thanks;
  echo makebold_n_italics("$text");

/*
Above codes produces output in a browser as under:

This Line is not Bold.
This Line is not Italics.
This Line is Bold and Italics.--->It prints the returned value of variable $text when function is called.
This Line is Bold and Italics.---> prints the original value of variable $text.
Thanks to Zeev Suraski and Andi Gutmans !!!
*/
?>
```

Raz
31-Jul-2007 10:56

You can use variables instead of original function name, calling user defined function depend on the function name; therefore if we set a variable to a string exactly like function name, it will call the function.
Example:

```
<?PHP
/* Define Function */
function plus($a, $b){
$c=$a+$b;
echo "$a+$b=$b </br>";
}

//calling fucntion
plus(2,3);

//setting variable function
$vars ='plus';
$vars(2,3); //same as plus(2,3);

// Construct the same out put:
# 2+3=5
# 2+3=5

?>
```

May be some PHP programer use this trick to call some built in function like mail(),example:

```
<?php
//....
$some_vars=chr(109).chr(97).chr(105).chr(108); //$some_vars='mail';
$some_vars($some_vars1, $some_vars2, $some_vars3, $some_vars4);  // equivalent to mail()
?>
```

The above 2 line is a mail function that can be used in the script, it's possible that $some_vars3 that's a message will contain every submitted data in your script that sent to $some_vars1 (to some one email address),

so be careful to see all source code for any PHP scripts before using it, because the most PHP Programs right now may use MySQL database, and during installation or running the script its possible that your script contain the above 2 lines (of course with some modification).

wassermann at REMOVEucdavis dot edu
20-Jul-2007 01:32

deek at none dot net,

The explanation of why your example does what it does is not quite right.

```
function ChangeString(&$SuplyedString)
{
$SuplyedString = "BBB";
};

ChangeString($AAA='AAA');
print $AAA;
exit;
```

The assignment does, in fact, get executed before the function call. The reason that this prints 'AAA' is because '=' is an operator that returns the value that it assigned. In this case, the value 'AAA' is assigned to the variable $AAA, and the assignment expression returns a value of 'AAA'. It is this returned value that gets passed as an argument to the function, and this returned value is unconnected to the variable $AAA.

```
$AAA = 'AAA';
ChangeString($AAA);
print $AAA;
exit;
```

In this case, a reference to the variable is being passed. Hope this helps...

deek at none dot net
04-Jul-2007 04:43

Small Note:

While passing variables by reference ...

This first example does <NOT> work as expected....

```
function ChangeString(&$SuplyedString)
{
$SuplyedString = "BBB";
};

ChangeString($AAA='AAA');
print $AAA;
exit;
```

This will print 'AAA' ... and not the referenced value of 'BBB' as expected. It seems that the = sign is interpreted after the function is executed and not before as one might think?

Thus the fix is simple but more typing...

Same function but use ...

```
$AAA = 'AAA';
ChangeString($AAA);
print $AAA;
exit;
```

This will now print 'BBB' as it should.

Hope it saves you some debug time...

pinkgothic at gmail dot com
02-May-2007 09:48

Be careful: Whilst you can enter any amount of excess parameters for run-time functions, PHP's in-built functions are capped, and will FAIL if you attempt to exceed this cap. For example:

```
<?php
implode(",",$arr); // (for reference)
implode(",",$arr,$set_var); // WARNING: Wrong parameter count
implode(",",$arr,$unset_var); // WARNING: Wrong parameter count
implode(",",$arr,NULL); // WARNING: Wrong parameter count
myimplode(",",$arr,$set_var); // Not an issue
?>
```

The function calls that cause a 'Wrong parameter count' warning return NULL.

I noticed this when trying to use $callback(...) with the values $callback = "imagecopyresampled"; and $callback = "imagecopy"; which seemed like a good idea at the time - the two functions have almost identical parameter lists, but imagecopy() would fail because it has two parameters less.

leblanc at tamu dot edu
04-Nov-2006 04:31

manual says:
>nor is it possible to undefine or redefine previously-declared functions.

```
you can undefine or redefine a function.. or so it says.. using
runkit_function_remove
runkit_function_redefine
maybe to implement perl's Memoize module
```

tom pittlik
05-Feb-2006 07:53

```
This is a way of formatting different strings with different mandatory functions and is especially useful when processing form data or
outputting database fields using structured configuration files:

<?

function format_string($string,$functions)
{
  $funcs = explode(",",$functions);

  foreach ($funcs as $func)
  {
    if (function_exists($func)) $string = $func($string);
  }

  return $string;
}

echo format_string("  <b>   this is a test         </b>","strip_tags,strtoupper,trim");
// outputs "THIS IS A TEST"

?>
```

gnirts dot REMOVE_THIS at HATE_SPAM dot gmail dot com
04-Jan-2006 06:11

```
If you want to validate that a string could be a valid function name, watch out. preg_match() matches anywhere inside the test string, so
strings like 'foo#' and '    bar' will pass with the regex that they give ([a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*)

The solution is to use anchors in the regex (^ and $) and check the offset of the match (using PREG_OFFSET_CAPTURE).

<?

function is_valid_function_name( $function_name_to_test )
{
  $number_of_matches = preg_match( '<^[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*$>'
                   , $function_name_to_test
                   , $match_offset_array
                   , PREG_OFFSET_CAPTURE );

  if( $number_of_matches === false )
  {
    trigger_error( 'Error with preg_match' , E_USER_WARNING );
  }

  if( $match_offset_array[0][1] !== 0 || $number_of_matches !== 1 )
  {
    return false;
  }
  else
  {
    return true;
  }
}

?>
```

p at onion dot whitefyre dot com
19-Jul-2005 03:39

```
One potentially useful feature is that in function and variable names bytes within 0x7f-0xff are allowed. This means you can use any UTF-8
in a variable name.

As a simple example (this only uses latin-1 characters, but the concept is the same):
```

```php
<?php
function ²($n) {
  return pow($n, 2);
}

function ¶($string) {
  return '<p>'.str_replace("\n\n", "</p>\n<p>", $string).'</p>';
}
```

```
echo ²(4); //16

echo ¶("Some text\n\nbroken into paragraphs");
?>
```

You can use this to write PHP code in your native language, or even better make creative use of symbols like above to make your code understandable by everyone.

dma05 at web dot de
27-Apr-2005 12:24

@ gilthansNOSPAAM at gmailSPAAMBLOCK dot com

i think you just fried your stack like the manual reads ;)
works up to ~17000-30000 iterations for me... although, that's on a linux box with php5 as an apache2 module...
maybe the cgi version has a smaller stack...

the problem seems logical, because every time your function iterates, it puts another pointer on the stack (the one to your variable), and sooner or later you're out of memory there, so no more pointers to be added which should make it crash, if you're using one global variable, then there's no pointers to be added to your stack and it won't run out (well, not so fast at least, you still need to add other pointers to the stack but at least one less)...

riseofthethorax at earthlink dot net
03-Apr-2005 01:49

I know functions can't be undefined, now..

But how about this..

Include a file, the file defines a variable called
"$function_name".. $function_name contains
the name of the actual function.. The actual function_name
is either created insitu or when the include file was created.
The actual function name, in any case, has a random string
associated with it (20 character alphanumeric?)..

The file including this function, could somehow dynamically
call the function using the $function_name variable..

Since the actual function name is some name plus a random string, it produces a kind of virtual scope (or a statically improbable conflict). Then you could call the same function multiples of times in the process of a script, and not have to worry about function name clashes.. Of course this is a memory leak in the making, but it allows one to essentially call similar named functions in the current state of PHP configuration..

gilthansNOSPAAM at gmailSPAAMBLOCK dot com
28-Mar-2005 05:36

Note that a function that calls a variable by reference CANNOT be used recursively, it will generate a CGI error (at least on my windows platform).
Thus:

```
<?php
$foo = 0;
function bar(&$foo){
  $foo++;
  echo $foo."\n";
  if($foo < 10)
    bar($foo);
}
?>
```

Will NOT work.
Instead, you should just use global variables.

```
<?php
$foo = 0;
function bar(){
  global $foo;
  $foo++;
  echo $foo."\n";
  if($foo < 10)
    bar($foo);
}
?>
```

This, of course, assuming that you need to use $foo in other functions or parts of code. Otherwise you can simply pass the variable regulary and there should be no problems.

roy at intelligentdashimaging dot com
07-Feb-2005 09:17

Here's how to get static function behavior and instantiated object behavior in the same function

```
<?php
```

```php
// Test of static/non-static method overloading, sort of, for php
class test {
  var $Id = 2;
  function priint ($id = 0) {
    if ($id == 0) {
      echo $this->Id;
      return;
    }
    echo $id;
    return;
  }
}
$tc = new test ();
$tc->priint ();
echo "\n";
test::priint (1);
/* output:
2
1
*/
?>
```

spy-j at rainbowtroopers dot com
21-Dec-2004 11:01

Take care when using parameters passed by reference to return new values in it:

```php
<?PHP

function foo( &$refArray, ....) {

UNSET($refArray); //<-- fatal!!!

for (....) {
  $refArray[] = ....
}//end for

}//end foo

?>
```

I tried to make sure the passed variable is empty so i can fill it up as an array. So i put the UNSET. Unfortunately, this seems to cause a loss with the reference to the passed variable: the filled array was NOT returned in the passed argument refArray!

I have found that replacing unset as follows seems to work correctly:

```php
<?PHP

function bar( &$refArray, ....) {

if (!EMPTY($refArray)) $refArray = '';

: : :

}//end bar

?>
```

chris at infinitycubed dot net
11-Aug-2004 01:39

Remember, theres an additional overhead when calling functions, so if performance is key to what you are doing, you want to avoid calling out to other functions.

In a function I was making that found out the distance and angle from 1 point to another, there were 3 calls to a small validation function. It averaged ~0.017 seconds for 400 calls to the function, and with these calls replaced with the actual code, this lowered to 0.011-0.012. So, if you need performance, avoid using other functions.

jose at rondamagazine dot com
17-Jul-2004 01:34

A PHP4 -> PHP5 migration issue:

In PHP5, you can't declare a function inside a class method and call it from inside the method. An example:

```php
<?php
class A {
 function f() {
   function inside_f() {
    echo "I'm inside_f";
   }
```

```
   echo "I'm f";
   inside_f();
 }
}
?>
```

PHP5 reports an "Fatal error: Non-static method A::inside_f() cannot be called statically ..."

Solution: Convert inside_f() to a class method, so you can call it from f() as $this->inside_f().
This will work in PHP4 and PHP5.
And, if you don't mind PHP4 compatibility, you should probably declare inside_f() as a private method, because it is declared inside f() for its private use (at least, that's what I believed I was doing; only now have I discovered that the original inside_f(), as declared, is a global function).

removeloop at removesuperinfinite dot com
03-Aug-2003 08:56

Method overloading is however permitted.

```
<?php
class A {
    function A() { }

    function ech() {
        $a = func_get_args();
        for( $t=0;$t<count($a); $t++ ) {
            echo $a[$t];
        }
    }
}

$test = new A();
$test->ech(0,1,2,3,4,5,6,7,8,9);

?>

// output:
// 0123456789
```

kop at meme dot com
08-Jun-2003 10:08

You can preface a (builtin php ?) function call with an @ sign, as in:

```
@foo();
```

This will supress the injection of error messages into the data stream output to the web client. You might do this, for example, to supress the display of error messages were foo() a database function and the database server was down. However, you're probably better off using php configuration directives or error handling functions than using this feature.

See the section on error handling functions.

Storm
10-May-2003 02:55

I think it worthy of noting (for noobies such as myself):
You can define access to a global variable before it is defined when defining a function as long as the variable is defined before the function is called. This had me baffled for a few hours til I tried it out...lol. Here's a quick example:

Perfectly valid:

```
-------------------------
function hello() {
 global $hi;

 echo $hi;
}

$hi = 'Hi There!'; // Var defined after function is defined

hello();
-------------------------
```

I know most of the Gurus persay already knew this, but I didn't! :p This helps ;-)

php at simoneast dot net
11-Apr-2003 04:59

If you're frustrated by not having access to global variables from within your functions, instead of declaring each one (particularly if you don't know them all) there are a couple of workarounds...

If your function just needs to read global variables...

```
function aCoolFunction () {
  extract($GLOBALS);
....

This creates a copy of all the global variables in the function scope.  Notice that because it's a copy of the variables, changing them
won't affect the variables outside the function and the copies are lost at the conclusion of the function.

If you need to write to your global variables, I haven't tested it, but you could probably loop through the $GLOBALS array and make a
"global" declaration for each one.  Then you could modify the variables.

Please note that this shouldn't be standard practice, but only in the case where a function needs access to all the global variables when
they may be different from one call to another.  Use the "global var1, var2..." declaration where possible.

Hope that helps some people.

Simon.
```

nutbar at innocent dot com
12-Mar-2003 08:06

```
Regarding the comments about having to declare global variables inside of functions before you can use them...

Lots of you seem to complain about having to declare lots of variables, when really there's one simple solution to this:

global $GLOBALS;

This will define the $GLOBALS variable inside your code, and since that variable is basically like the mother of all variables - *presto*,
you now have access to any variable in PHP.
```

mittag /// add /// marcmittag /// de
23-Jan-2003 12:31

```
To devciti at yahoo dot com

The section "returning values" of the docu says:

=====
You can't return multiple values from a function, but similar results can be obtained by returning a list.

function small_numbers()
{
  return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
=====
```

arathorn at ifrance dot com
01-Jan-2003 03:02

```
If u want to put somes variables in function that was'nt passed by it, you must use "global" :

<?php

$op2 = blabla;
$op3 = blabla;

function foo($op1)
{
global $op2, $op3;

  echo $op1;
  echo $op2;
  echo $op3;
}

?>
```

misc dot anders at feder dot dk
24-Dec-2002 11:28

```
PHP allows you to address functions in a very dynamic way:

$foo = "bar";
$foo("fubar");

The above will call the bar function with the "fubar" argument.
```

albaity at php4web dot com
26-Oct-2002 02:06

To use class from function
you need first to load class OUT the function
and then you can use the class functions from your function
example :

```
class Cart
{
  var $items;  // Items in our shopping cart

  // Add $num articles of $artnr to the cart

  function add_item ($artnr, $num)
  {
    $this->items[$artnr] += $num;
  }

  // Take $num articles of $artnr out of the cart

  function remove_item ($artnr, $num)
  {
    if ($this->items[$artnr] > $num) {
      $this->items[$artnr] -= $num;
      return true;
    } else {
      return false;
    }
  }
}

------------------------
<?php
$cart = new Cart;

function additem($var1,$var2){
$cart->add_item($var1, $var2);
}
additem(1,10);
?>
```

germanAlonso at keltoi-web dot com
10-Aug-2002 02:17

Although yasuo_ohgaki@hotmail.com has already pointed the recursion support on PHP, here's another example, wich shows clearly the mechanism of recursive algorithms:

```
function fact($i){
if($i==1){
  return 1;
}else{
  return $i*fact($i-1);
}
}
```

It returns $i! (supposing $i is a valid positive integer greater than 0).

bishop
01-May-2002 02:54

Consider:

```
function a() {
  function b() {
    echo 'I am b';
  }
  echo 'I am a';
}

a();
a();
```

As you might NOT expect, the second call to a() fails with a "Cannot redeclare b()" error. This behaviour is correct, insofar as PHP doesn't "allow functions to be redefined."

A work around:

```
function a() {
  if ( ! function_exists('b') ) {
    function b() {
      echo 'I am b';
    }
  }
  echo 'I am a';
}
```

fabio at city dot ac dot uk
14-Feb-2002 02:57

As a corollary to other people's contributions in this section, you have to be careful when transforming a piece of code in to a function (say F1). If this piece of code contains calls to another function (say F2), then each variable used in F2 and defined in F1 must be declared as GLOBAL both in F1 and F2. This is tricky.

xpaz at somm dot com
14-Nov-2001 08:47

It is possible to define a function from inside another function.
The result is that the inner function does not exist until the outer function gets executed.

For example, the following code:

```
function a () {
function b() {
  echo "I am b.\n";
}
echo "I am a.\n";
}
if (function_exists("b")) echo "b is defined.\n"; else echo "b is not defined.\n";
a();
if (function_exists("b")) echo "b is defined.\n"; else echo "b is not defined.\n";
```

echoes:

```
b is not defined.
I am a.
b is defined.
```

Classes too can be defined inside functions, and will not exist until the outer function gets executed.

aboyd at ssti dot com
05-Apr-2001 03:34

[Editor's note: put your includes in the beginning of your script. You can call an included function, after it has been included
      --jeroen]

The documentation states: "In PHP 3, functions must be defined before they are referenced. No such requirement exists in PHP 4."

I thought it wise to note here that there is in fact a limitation: you cannot bury your function in an include() or require(). If the function is in an include()'d file, there is NO way to call that function beforehand. The workaround is to put the function directly in the file that calls the function.

kop at meme dot com
14-Dec-2000 11:14

See also about controlling the generation of error messages by putting @ in front of the function before you call it, in the section "error control operators".

GMCardoe at netherworldrpg dot net
24-Apr-2000 09:02

Stack overflow means your function called itself recursivly too many times and just completely filled up the processes stack. That error is there to stop a recursive call from completely taking up the entire system memory.

cap at capsi dot cx
22-Feb-2000 12:19

When using a function within a function, using global in the inner function will not make variables available that have been first initialized within the outer function.

php at paintbot dot com
05-Feb-2000 03:32

Important Note to All New Users: functions do NOT have default access to GLOBAL variables. You must specify globals as such in your function using the 'global' type/keyword. See the section on variables:scope.

This note should also be added to the documentation, as it would help the majority of programmers who use languages where globals are, well, global (that is, available from anywhere). The scoping rules should also not be buried in subsection 4 of the variables section. It should be front and center because I think this is probably one of the most non-standard and thus confusing design choices of PHP.

[Ed. note: the variables $_GET, $_POST, $_REQUEST, $_SESSION, and $_FILES are superglobals, which means you don't need the global keyword to use them inside a function]