PHP Functions

Back to ACP page

Table of Contents

PHP Built-in Functions

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

Please check out our PHP reference for a complete overview of the PHP built-in functions.

PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.

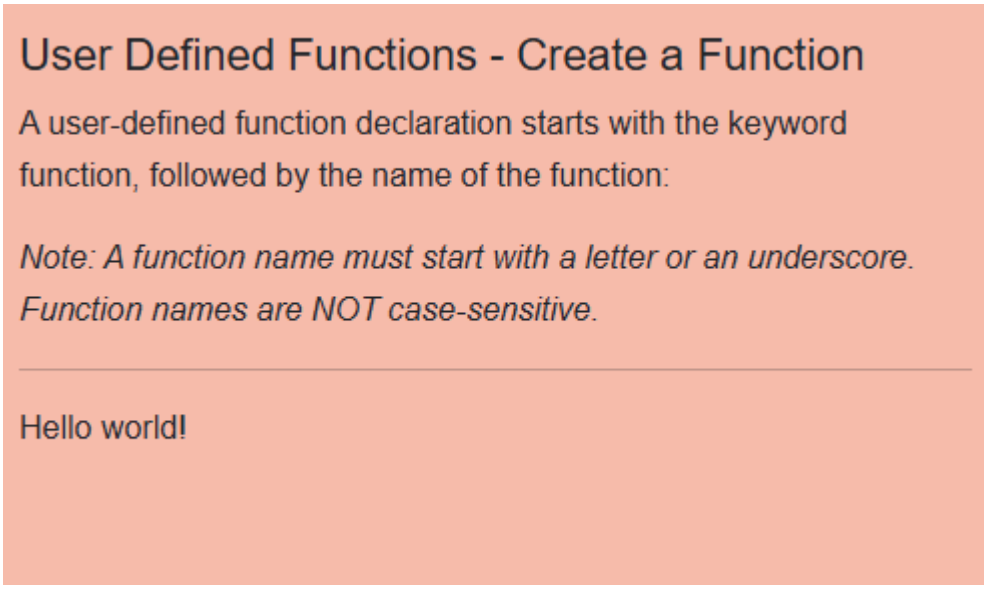
PHP 1 Create a Function - Call a Function

A user-defined function declaration starts with the keyword function, followed by the name of the function:

To call the function, just write its name followed by parentheses ():

```
<h4>User Defined Functions - Create a Function</h4>
<p>A user-defined function declaration starts with the keyword function,
followed by the name of the function:</p>
<i>Note: A function name must start with a letter or an underscore.
Function names are NOT case-sensitive.</i>
<hr>
<?php
    // Call the function
    _myMessage();


    // Create a function
    function _myMessage() {
        echo "Hello world!";
    }
?>
```



Example 1

Result View Example

PHP 2 Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

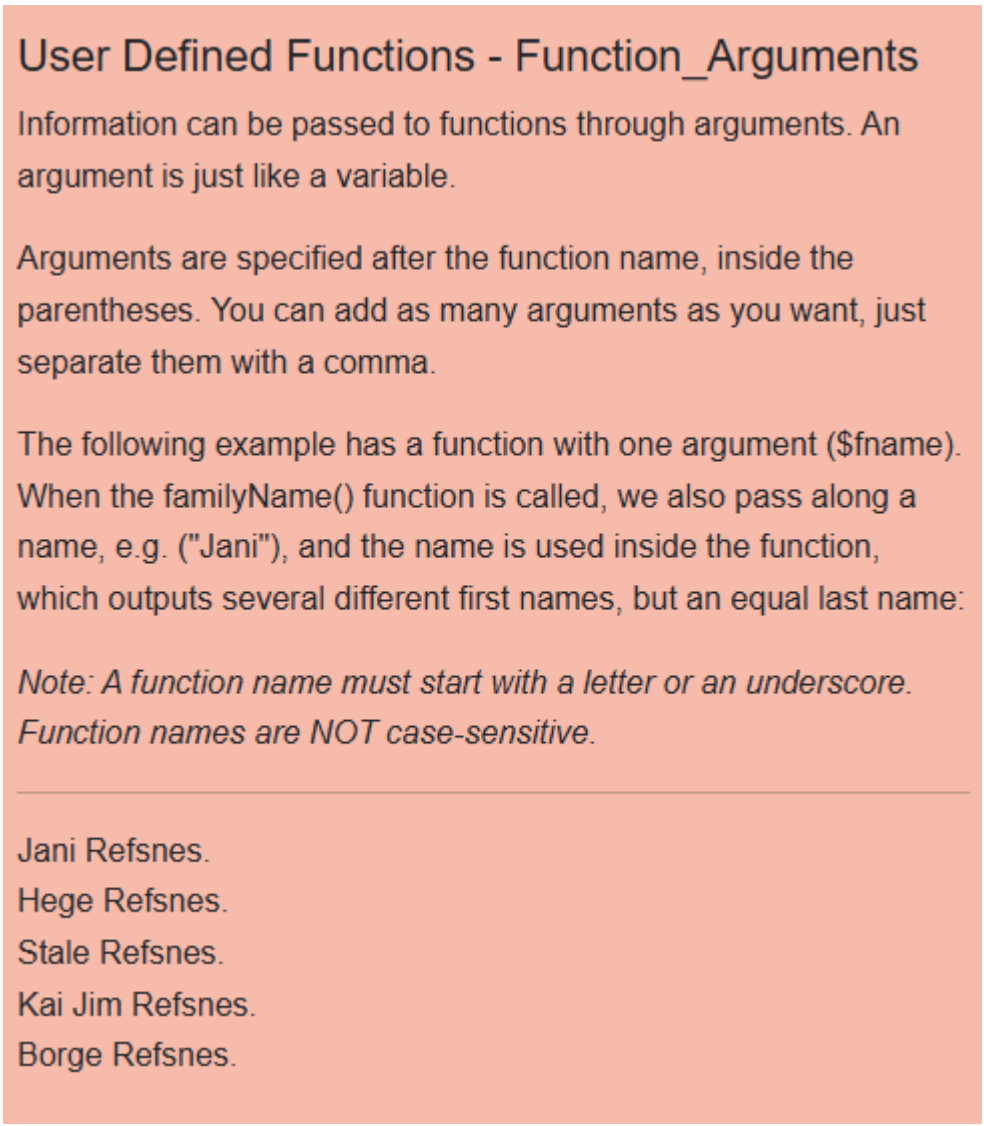
Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name, e.g. ("Jani"), and the name is used inside the function, which outputs several different first names, but an equal last name:

```
<h4>User Defined Functions - Function_Arguments</h4>
<i>Note: A function name must start with a letter or an underscore.
Function names are NOT case-sensitive.</i>
<hr>

<?php
    function familyName($fname) {
        echo "$fname Refsnes.<br>";
    }

    familyName("Jani");
    familyName("Hege");
    familyName("Stale");
    familyName("Kai Jim");
    familyName("Borge");
?>
```



User Defined Functions - Function_Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name, e.g. ("Jani"), and the name is used inside the function, which outputs several different first names, but an equal last name:

Note: A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

Jani Refsnes.
Hege Refsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.

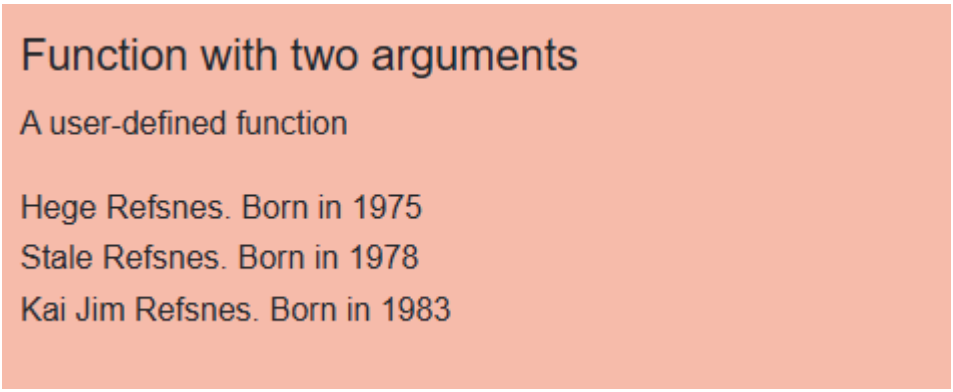
Example 2

Result View Example

PHP 3 function with two arguments

The following example has a function with two arguments ($fname, $year):

```
<h4>Function with two arguments</h4>
<p>A user-defined function</p>
<?php
function familyName($fname, $year)
{
    echo "$fname Refsnes. Born in $year <br>";
}
familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```



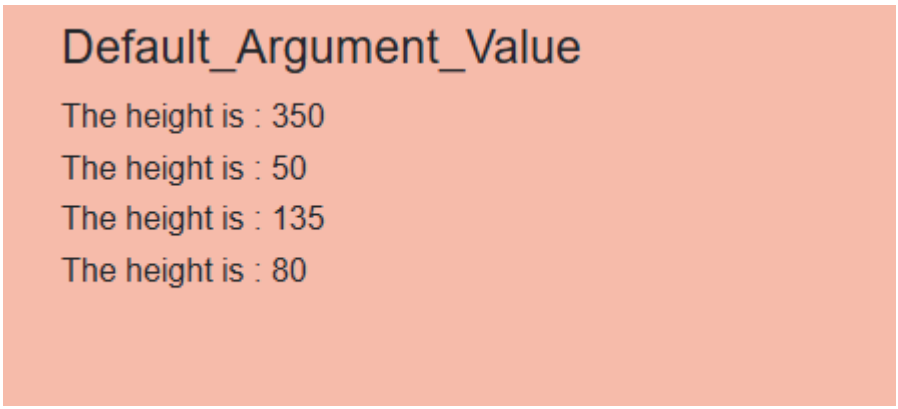
Example 3

Result View Example

PHP 4 Default Argument Value

```
<h4>Default_Argument_Value</h4>
<?php
function setHeight($minheight = 50)
{
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight();
setHeight(135);
setHeight(80);
?>
```



Example 4

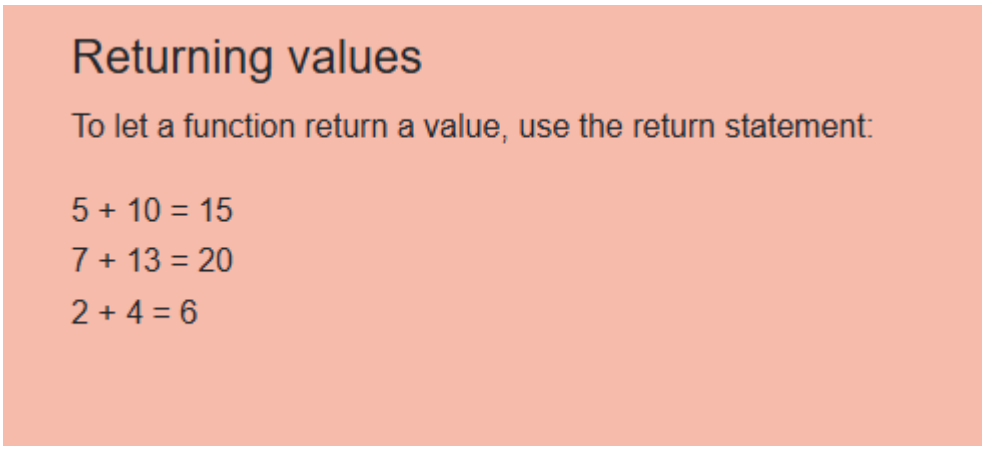
Result View Example

PHP 5 Functions - Returning values

To let a function return a value, use the return statement:

```
<h4>Returning values</h4>
<p>To let a function return a value, use the return statement:</p>
<?php
function sum($x, $y)
{
    $z = $x + $y;
    return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```



Example 5

Result View Example

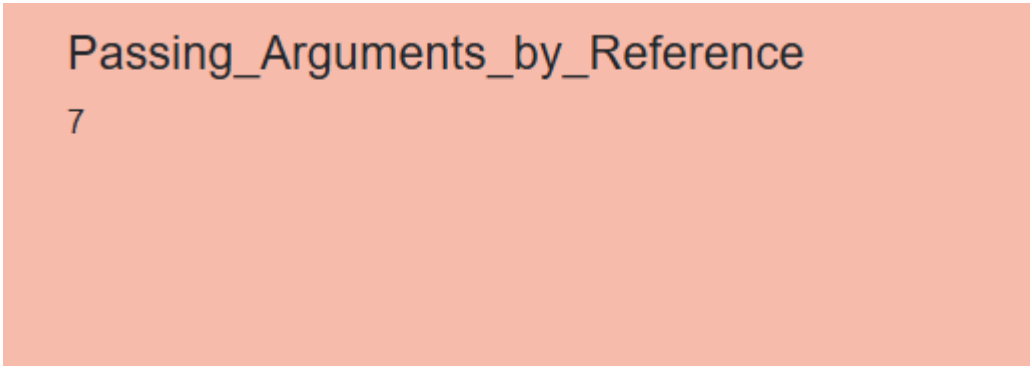
PHP 6 Passing Arguments by Reference

In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed.

When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the & operator is used:

```
<h4>Passing_Arguments_by_Reference</h4>
<?php
function add_five(&$value)
{
    $value += 5;
}

$num = 2;
add_five($num);
echo $num;
?>
```



Example 6

Result View Example

PHP 7 Variable Number of Arguments

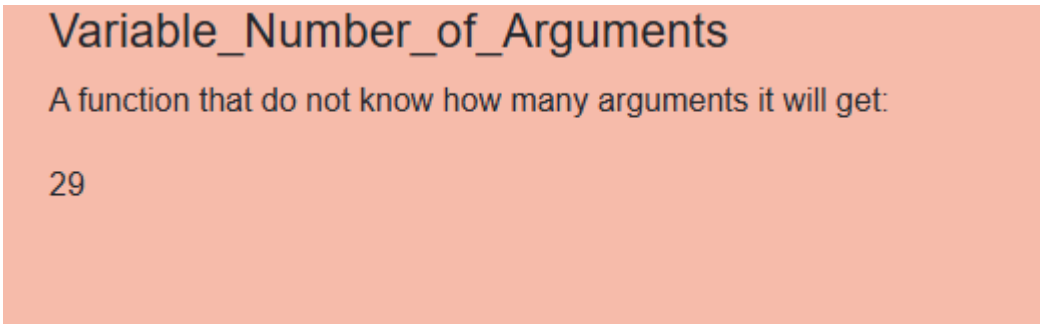
By using the ... operator in front of the function parameter, the function accepts an unknown number of arguments. This is also called a variadic function.

The variadic function argument becomes an array.

```
<h4>Variable_Number_of_Arguments</h4>
<p>A function that do not know how many arguments it will get:</p>
<?php
function sumMyNumbers(...$x)
{
    $n = 0;
    $len = count($x);
    for ($i = 0; $i < $len; $i++) {
```

```
        $n += $x[$i];
    }
    return $n;
}

$a = sumMyNumbers(5, 2, 6, 2, 7, 7);
echo $a;
?>
```



Example 7

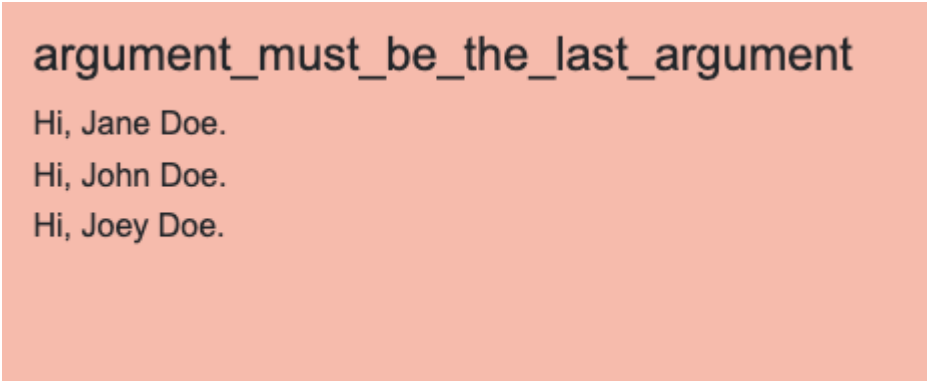
Result View Example

PHP 8 argument must be the last argument

```
<h4>argument_must_be_the_last_argument</h4>

<?php
    function myFamily($lastname, ...$firstname) {
    // function myFamily(...$firstname, $lastname) { // Error
        $txt = "";
        $len = count($firstname);
        for($i = 0; $i < $len; $i++) {
          $txt = $txt."Hi, $firstname[$i] $lastname.<br>";
        }
        return $txt;
      }

      $a = myFamily("Doe", "Jane", "John", "Joey");
      echo $a;
?>
```



Example 8

Result View Example

PHP 9 Loosely Typed Language

In the examples above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives us an option to specify the expected data type when declaring a function, and by adding the strict declaration, it will throw a "Fatal Error" if the data type mismatches.

In the following example we try to send both a number and a string to the function without using strict:

```
<?php
declare(strict_types=1); // strict requirement


/**
 * Move the declare(strict_types=1);
 * statement to the very top of the PHP file, before any other code.
 */
?>

<?php

function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is enabled and "5 days" is not an integer, an error will
be thrown
?>
```

Loosely_Typed_Language

Note: To specify strict we need to set declare(strict_types=1);. This must be on the very first line of the PHP file. In the following example we try to send both a number and a string to the function, but here we have added the strict declaration:

10

Example 9

Result View Example

PHP 10 Return Type Declarations

```
<?php
declare(strict_types=1); // strict requirement


/**
 * Move the declare(strict_types=1);
 * statement to the very top of the PHP file, before any other code.
 */
?>

<h4>Return_Type_Declarations</h4>

<?php
    declare(strict_types=1); // strict requirement
    function addNumbers(float $a, float $b) : int {
      return (int)($a + $b);
    }
    echo addNumbers(1.2, 5.2);
?>
```

Return_Type_Declarations

6

Example 10

Result View Example

Document

Document in project

You can Download PDF file.

Reference