PHP Functions

In this tutorial you will learn how to create your own custom functions in PHP.

PHP Built-in Functions

A function is a self-contained block of code that performs a specific task.

PHP has a huge collection of internal or built-in functions that you can call directly within your PHP scripts to perform a specific task, like `gettype()`, `print_r()`, `var_dump`, etc.

Please check out PHP reference section for a complete list of useful PHP built-in functions.

PHP User-Defined Functions

In addition to the built-in functions, PHP also allows you to define your own functions. It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately form main program. Here are some advantages of using functions:

- **Functions reduces the repetition of code within a program** — Function allows you to extract commonly used block of code into a single component. Now you can perform the same task by calling this function wherever you want within your script without having to copy and paste the same block of code again and again.
- **Functions makes the code much easier to maintain** — Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.
- **Functions makes it easier to eliminate the errors** — When the program is subdivided into functions, if any error occur you know exactly what function causing the error and where to find it. Therefore, fixing errors becomes much easier.
- **Functions can be reused in other application** — Because a function is separated from the rest of the script, it's easy to reuse the same function in other applications just by including the php file containing those functions.

The following section will show you how easily you can define your own function in PHP.

Creating and Invoking Functions

The basic syntax of creating a custom function can be give with:

```
function functionName(){
    // Code to be executed
}
```

The declaration of a user-defined function start with the word `function`, followed by the name of the function you want to create followed by parentheses i.e. `()` and finally place your function's code between curly brackets `{}`.

This is a simple example of an user-defined function, that display today's date:

Example **Run this code »**

```php
<?php
// Defining function
function whatIsToday(){
    echo "Today is " . date('l', mktime());
}
// Calling function
whatIsToday();
?>
```

Note: A function name must start with a letter or underscore character not with a number, optionally followed by the more letters, numbers, or underscore characters. Function names are case-insensitive.

Functions with Parameters

You can specify parameters when you define your function to accept input values at run time. The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.

```php
function myFunc($oneParameter, $anotherParameter){
    // Code to be executed
}
```

You can define as many parameters as you like. However for each parameter you specify, a corresponding argument needs to be passed to the function when it is called.

The `getSum()` function in following example takes two integer values as arguments, simply add them together and then display the result in the browser.

Example **Run this code »**

```php
<?php
// Defining function
function getSum($num1, $num2){
  $sum = $num1 + $num2;
```

```
  echo "Sum of the two numbers $num1 and $num2 is : $sum";
}

// Calling function
getSum(10, 20);
?>
```

The output of the above code will be:

```
Sum of the two numbers 10 and 20 is : 30
```

Tip: An argument is a value that you pass to a function, and a parameter is the variable within the function that receives the argument. However, in common usage these terms are interchangeable i.e. an argument is a parameter is an argument.

Functions with Optional Parameters and Default Values

You can also create functions with optional parameters — just insert the parameter name, followed by an equals (`=`) sign, followed by a default value, like this.

Example **Run this code »**

```
<?php
// Defining function
function customFont($font, $size=1.5){
    echo "<p style=\"font-family: $font; font-size: {$size}em;\">Hello,
world!</p>";
}

// Calling function
customFont("Arial", 2);
customFont("Times", 3);
customFont("Courier");
?>
```

As you can see the third call to `customFont()` doesn't include the second argument. This causes PHP engine to use the default value for the `$size` parameter which is 1.5.

Returning Values from a Function

A function can return a value back to the script that called the function using the return statement. The value may be of any type, including arrays and objects.

Example **Run this code »**

```php
<?php
// Defining function
function getSum($num1, $num2){
    $total = $num1 + $num2;
    return $total;
}

// Printing returned value
echo getSum(5, 10); // Outputs: 15
?>
```

A function can not return multiple values. However, you can obtain similar results by returning an array, as demonstrated in the following example.

Example **Run this code »**

```php
<?php
// Defining function
function divideNumbers($dividend, $divisor){
    $quotient = $dividend / $divisor;
    $array = array($dividend, $divisor, $quotient);
    return $array;
}

// Assign variables as if they were an array
list($dividend, $divisor, $quotient) = divideNumbers(10, 2);
echo $dividend;  // Outputs: 10
echo $divisor;   // Outputs: 2
echo $quotient;  // Outputs: 5
?>
```

Passing Arguments to a Function by Reference

In PHP there are two ways you can pass arguments to a function: *by value* and *by reference*. By default, function arguments are passed by value so that if the value of the argument within the function is changed, it does not get affected outside of the function. However, to allow a function to modify its arguments, they must be passed by reference.

Passing an argument by reference is done by prepending an ampersand (`&`) to the argument name in the function definition, as shown in the example below:

Example **Run this code »**

```php
<?php
/* Defining a function that multiply a number
by itself and return the new value */
function selfMultiply(&$number){
    $number *= $number;
    return $number;
}

$mynum = 5;
echo $mynum; // Outputs: 5

selfMultiply($mynum);
echo $mynum; // Outputs: 25
?>
```

Understanding the Variable Scope

However, you can declare the variables anywhere in a PHP script. But, the location of the declaration determines the extent of a variable's visibility within the PHP program i.e. where the variable can be used or accessed. This accessibility is known as *variable scope*.

By default, variables declared within a function are local and they cannot be viewed or manipulated from outside of that function, as demonstrated in the example below:

Example **Run this code »**

```php
<?php
// Defining function
function test(){
    $greet = "Hello World!";
    echo $greet;
}

test(); // Outputs: Hello World!

echo $greet; // Generate undefined variable error
?>
```

Similarly, if you try to access or import an outside variable inside the function, you'll get an undefined variable error, as shown in the following example:

Example **Run this code »**

```php
<?php
$greet = "Hello World!";

// Defining function
function test(){
    echo $greet;
}

test();  // Generate undefined variable error

echo $greet; // Outputs: Hello World!
?>
```

As you can see in the above examples the variable declared inside the function is not accessible from outside, likewise the variable declared outside of the function is not accessible inside of the function. This separation reduces the chances of variables within a function getting affected by the variables in the main program.

Tip: It is possible to reuse the same name for a variable in different functions, since local variables are only recognized by the function in which they are declared.

The global Keyword

There may be a situation when you need to import a variable from the main program into a function, or vice versa. In such cases, you can use the `global` keyword before the variables inside a function. This keyword turns the variable into a global variable, making it visible or accessible both inside and outside the function, as show in the example below:

Example **Run this code »**

```php
<?php
$greet = "Hello World!";

// Defining function
function test(){
    global $greet;
    echo $greet;
}
```

```php
test(); // Outpus: Hello World!
echo $greet; // Outpus: Hello World!

// Assign a new value to variable
$greet = "Goodbye";

test(); // Outputs: Goodbye
echo $greet; // Outputs: Goodbye
?>
```

You will learn more about visibility and access control in PHP classes and objects chapter.

Creating Recursive Functions

A recursive function is a function that calls itself again and again until a condition is satisfied. Recursive functions are often used to solve complex mathematical calculations, or to process deeply nested structures e.g., printing all the elements of a deeply nested array.

The following example demonstrates how a recursive function works.

Example Run this code »

```php
<?php
// Defining recursive function
function printValues($arr) {
    global $count;
    global $items;

    // Check input is an array
    if(!is_array($arr)){
        die("ERROR: Input is not an array");
    }

    /*
    Loop through array, if value is itself an array recursively call
the
```

Note: Be careful while creating recursive functions, because if code is written improperly it may result in an infinite loop of function calling.