**Creating PHP Functions**

I'm sure you've been using built-in PHP functions in your applications. *Functions* are nothing more than PHP code someone else wrote that you can use. Instead of having to copy all of the code into your application, you just use the function name, such as when we use the mysql\_connect() function to connect to a MySQL server.

PHP also allows us to create our own functions to use in programs and share with others. Once you define a function, you can use it throughout your program. This saves typing if you use a common routine or block of code in lots of places in your application. All you need to do is write the code once as a function and then call the function everywhere else it appears in the application!

The basic format for a function looks like this:

function *name*(*parameters*)  
{  
*function code*  
return *value*  
}

The function *name* must uniquely identify the function. It can't be one of the existing function names in PHP, and it can't start with a number (although numbers can appear anywhere else in the function name).

The function *parameters* identify one or more variables that the calling program passes to the function (actually, you can also have a function that uses no parameters). If there is more than one variable, you must separate them using commas. You can use the variables anywhere within the function code, but they only apply to the function. You can't access the passed variables anywhere else in your PHP code.

It's important to note that any variables you define within the function apply only to the function. You can't use any function variables in your outside PHP code.

At this point you might be saying, "But I perform this really cool calculation in my function! How do I pass the answer back to my program if I can't share variables?" That's the purpose of the *return* statement.

The return statement is the last statement in the function. It defines a single function variable that the function returns to the calling program. Thus, you can only pass one variable back to the rest of the PHP program.

The best way to learn functions is to start building them. Let's look at an example:

<?php

function factorial($value1)  
{  
$factorial = 1;  
$count = 1;  
while($count <= $value1)  
{  
$factorial = $factorial \* $count;  
$count = $count + 1;  
}  
return $factorial;  
}

echo "The factorial of 10 is " . factorial(10) . "<br>\n";  
echo "The factorial of 5 is " . factorial(5) . "<br>\n";  
?>

Notice that all of the *factorial* code is self-contained within the function braces. When the PHP program uses the new factorial() function, it passes a single value that the function assigns to the $value1 variable. The function returns the final factorial result.

Remember, you can't use the actual $factorial variable in your PHP code because it's defined inside the function. Instead, the factorial function now represents the value the function returns in the code. This can be somewhat difficult to grasp if you've never used functions before.

When you use the function in your PHP code, it acts like a variable in that the value it returns appears wherever you declare the function. Let's see how this works by testing it out.

1. Create a file in the WampServer www folder called *functest.php*.
2. Open the file using Microsoft Notepad or your favorite text editor.
3. Enter the code snippet from the factorial function example shown earlier.
4. Save the functest.php file.

|  |
| --- |
| **Note:** If you're using Microsoft Notepad to edit and save your PHP files, be careful when saving. You must put double quotes around the filename in the Save As dialog box (like "functest.php"), or Notepad will add a .txt extension to the file. |

1. Open a browser window, and go to http://localhost/functest.php

.

You should see the output of the functest.php program as shown below.

The functest.php program output

Notice how the result from the function appears exactly where you place the function in the echo statement.

**Creating Function Libraries**

The functest.php example defines the factorial function at the top of the program code and then uses it later in the same program. This isn't a requirement. You can define the function anywhere in the PHP code and then use it anywhere else in the same code. What comes in real handy, though, is creating a function library file.

You can define the function in a completely separate file, then use the PHP include() function to add the function code to your PHP program. This lets you create the function once and use it in as many programs as you want without having to retype it!

Let's test this feature out. First, create the factorial function file.

1. Create a file in the WampServer www folder called *factorial.php*.
2. Open the file using Microsoft Notepad or your favorite text editor.
3. Enter the following PHP code:

<?php  
function factorial($value1)  
{  
$fact = 1;  
$count = 1;  
while($count <= $value1)  
{  
$fact = $fact \* $count;  
$count = $count + 1;  
}  
return $fact;  
}  
?>

1. Save the factorial.php file.
2. Create another file in the WampServer www folder called *functest2.php*.
3. Open the file using Microsoft Notepad or your favorite text editor.
4. Enter the following PHP code:

<html>  
<body>  
<h2>This is a test of defining a function in a separate file</h2>  
<?php  
include("factorial.php");

echo "The first test. The factorial of 3 is " . factorial(3) . "<br>\n";  
?>

<h2>Now, let's try to use it again</h2>

<?php  
echo "The last factorial test. The factorial of 5 is " . factorial(5) . "<br>\n";  
?>

<h2>This is the end of the test</h2>  
</body>  
</html>

1. Save the functest2.php file.
2. Open your browser, and go to http://localhost/functest2.php.

You should see something that looks like this:

The output from the functest2.php program

Notice that after using the include() function to add the factorial.php code to your program, your new factorial() function worked no matter where in the Web page you used it! This is a great time-saving feature of PHP.

Okay, enough about functions. Let's move on to the next topic—handling images.