CHAPTER 11

Using Functions

Redundant code is rarely a good thing. Rewriting code over and over again is not time efficient and looks rather shoddy from a layout point of view. Like any good programming language, PHP alleviates the problem of redundant code in a number of ways; the most commonly used and simple-to-implement way is by using functions.

A *function* is basically a block of code that performs a given action from the script that has access to it, via includes, code insertions, or other methods. Rather than repeatedly rewrite the same block of code to, say, check if the current user is logged into your site, you can put the code into what is essentially a code wrapper and then call it at your convenience with a simple function call.

To be truly versatile, functions can receive values passed into them, perform some sort of functionality, and then return a value (or set of values using an array or object). Taking an entire block of code that was redundantly placed all over your scripts and replacing it with a one-line function call does wonders for the cleanliness of your code and is the first step to becoming an efficient programmer.

11-1. Accessing Function Parameters

The first thing any good programmer should realize about a function is that in order to do something meaningful with an exterior set of data, you must pass the function the values that are to be worked with. Parameters in PHP 5 are passed when the function itself is called and then worked on within the block of code. Because of PHP's ease of use with data types, passing a value to a function as a parameter is quite simple. The following example passes a username and password to the function to confirm that a valid match exists.

The Code

```
<?php

  //sample11_1.php

  //A function to validate a username and password.
  function validatelogin ($username, $password){
    //Typically the username and password would be validated against information
    //in the database. For the sake of simplicity in this example, the username
    //and password are hard-coded into variables.
    $actualuser = "myusername";
```

```
    $actualpass = "mypassword";

    //Now, you do a quick comparison to see if the user ➥
has entered the correct login.
    if (strcmp ($username, $actualuser) == 0 &&➥
 strcmp ($password, $actualpass) == 0){
      return true;
    } else {
      return false;
    }
  }

  //You then call the function and pass in the values you want checked.
  if (validatelogin ("myusername","mypassword")){
    echo "You are logged in correctly";
  } else {
    echo "You have an incorrect username and/or password";
  }
?>
```

```
You are logged in correctly
```

How It Works

This is a basic example of how easy it is to pass to, and then access, a set of parameters. In this case, the function receives two values from the function (denoting a username and password) and then checks to see that they match with the existing username and password (preferably in a database). If you receive a valid match, then the function returns a `true` boolean type; if not, then the function returns a `false` boolean type. Note how much easier it is to call the function `validatelogin()` rather than type out that entire block of code. Not only is it much cleaner and more efficient, but it also alleviates the problem of redundancy when you undoubtedly call the function again.

As for accessing the actual values within the script, you simply access them according to whatever you named them in the function's argument list. In this case, you named them `$username` and `$password`, allowing you to reference them using their variable names within the function.

11-2. Setting Default Values for Function Parameters

When you are passing arguments to a function, you may want the parameters to default to a certain value. Doing so within a PHP function is simple. In most programming languages, any values you are concerned might not be passed to the function properly (or at all) can be defaulted to a certain value. You might prefer to default the parameters being passed to a function for two reasons. First, you do not have to worry so much about exception handling and can rest assured that any argument that does get passed in properly will override the default. Second, when using functions that generally receive the same values but sometimes

require different values to be passed in, having the defaults in place prevents you from constantly having to pass in the same set of values. The following example returns the sum of three values.

The Code

```
<?php

  //sample11_2.php

  //A function to return the sum of three values.
  function addvalues ($value1 = 0, $value2 = 0, $value3 = 0){
    //Now the function takes the three values and adds them.
    $total = $value1 + $value2 + $value3;
    return $total;
  }

  //Now, if you forget a value or two, it will still work.
  echo addvalues (1) . "<br />"; //Echoes a 1.
  //If you pass all the arguments, you will still get a valid result.
  echo addvalues (1,2,3); //Echoes a 6.
?>
```

```
1
6
```

How It Works

Now, if you had not defaulted the values in the argument list to zeros, the function call you just made would have returned a warning telling you that you were missing arguments to your function call. Rather than face the possibility of an incorrectly called function, you can cover all your bases by defaulting the values to zeros. Therefore, if someone were to call the function (as you did in the previous example) with an incorrect number of arguments, the function would still perform its given action using the default values assigned to its arguments.

11-3. Passing Values by Reference

The default when passing a parameter to a function in PHP 5 is to pass the argument by value. In other words, when the function receives the value, it will then work on it as if that variable was an entirely separate entity to the one that was passed to it originally. If you pass by reference, however, the variable that was passed in will be manipulated within the function as if the value were still within the script it was passed in from. Think of passing arguments by value as creating a temporary copy to work with; alternatively, passing by reference uses, and can make changes to, the original copy. The following example allows you to concatenate text to an existing block of text.

The Code

```
<?php

  //sample11_3.php

  //A function to concatenate text.
  function attachtext (&$newtext = ""){
    //Now the function attaches the received text.
    $newtext = $newtext . " World!";
  }

  //Here is the current block of text.
  $mystring = "Hello";
  //Then you call the function to attach new text.
  attachtext ($mystring);
  //And when you echo the variable now...
  echo $mystring; //Outputs Hello World!
?>
```

```
Hello World!
```

How It Works

As you can see, the major difference in the argument list is that you place an ampersand (&) character in front of the passed-in variable. This tells PHP to treat the variable as a referenced object. This means any change to the passed-in value will affect the original passed-in variable. Therefore, when you output `$mystring` after the function call has been made, the new value has been concatenated onto the old value. Had you passed in the argument by value, the script would have merely output "Hello" because it would have treated the value as a copy of the original, not as an alias to the original.

11-4. Creating Functions That Take a Variable Number of Arguments

Sometimes you will need to create a function that could receive a multitude of values but the number of values to be received will not be set in stone. Take, for instance, a function that will add any number of values passed to it provided that they are integer values. In this case, you want the function to be versatile enough to add any number of values that are passed to it—kind of like a math crunching machine.

The Code

```
<?php

  //sample11_4.php

  //A function to add up any number of values.
  function addanything (){
    //Default the return value.
    $total = 0;
    //Get the full list of arguments passed in.
    $args = func_get_args ();
    //Loop through the arguments.
    for ($i = 0; $i < count ($args); $i++){
      //Make sure the value is an integer.
      if (is_int ($args[$i])){
        //And add to it if necessary.
        $total += $args[$i];
      }
    }
    //Then return the total.
    return $total;
  }

  //Now, you can pass the function any numbers.
  echo addanything (1,5,7,8,11) . "<br />"; //Outputs 32.
  echo addanything (1,1) . "<br />"; //Outputs 2.
  //It will ignore noninteger values.
  echo addanything (1,1,"Hello World"); //Still outputs 2.
?>
```

```
32
2
2
```

How It Works

The benefactor in this case happens to be the lovely `func_get_args()` function, which grabs an array of all the passed-in values. The great thing about this is that you can then cycle, or loop through, the list of arguments and do what you want with them. This sort of functionality serves you well in this case, because you loop through, adding to the total as you go. For the sake of validation, the script adds only integer values in order to keep a valid result in mind. The end result is a highly flexible function that will take care of all your integer adding needs. The prototype for `func_get_args()` and the prototype for `func_get_arg()`, which will grab an argument at a certain reference, are as follows:

```
array func_get_args ( void )
mixed func_get_arg ( int arg_num )
```

11-5. Returning More Than One Value

Naturally, it is handy to have a single value returned from a function, and it is even more helpful in some instances to have a function return multiple values. Since the return statement is really set up to return only a single value, you can get a little tricky and pass an array of items for use.

If you want to get even more involved, you can return entire objects from a function, thus allowing you to pass back whatever values were associated with the object. Through some careful manipulation, you can use functions to return whatever it is you need returned from them.

The following example is a function that allows you to return an array of values, thus getting around the problem of being able to return only a single value.

The Code

```
<?php

  //sample11_5.php

  //Function that will take in a set of values, calculate them,➥
 then return the values.
  function addandsubtract ($firstvalue, $secondvalue){
    //The first thing we need to do is add the values.
    $firstreturnvalue = ($firstvalue + $secondvalue);
    $secondreturnvalue = ($firstvalue - $secondvalue);

    //Now, you declare an array.
    $myarray = array ();

    //Then put the two return values into the first two indexes of the array.
    $myarray[0] = $firstreturnvalue;
    $myarray[1] = $secondreturnvalue;

    //Then you can return the entire array.
    return $myarray;
  }

  //Now, when you call the function, it will return the two values in array format.
  $myarray = array ();
  $myarray = addandsubtract (10, 3);

  echo $myarray[0] . "<br />"; //Will echo 13.
  echo $myarray[1]; //Will echo 7.
?>
```

```
13
7
```

How It Works

As you can see, the method for returning an array from a function is rather simple. All that is required is to have an array declared (and probably filled with a value or two) and then return it to the function call using the `return` method. Then, when you receive the value from the function, you can assign the result of the function to an array and use it as you would any other array.

11-6. Returning Values by Reference

Sometimes passing back an argument by value may not be all that efficient. Fortunately, PHP 5 allows returning values by reference, but you should keep in mind a few new syntaxes both when declaring the function and when calling the function.

Returning values by reference can be rather obscure, but when used properly, this technique can be quite handy in specific circumstances. The following example allows you to search through an array of objects and then return the exact object for which you are looking.

The Code

```
<?php

  //sample11_6.php

  //Create a class that stores values.
  class myclass {

    //A defining value.
    private $thevalue;
    //A word to prove you have found the right object.
    private $theword;

    public function __construct (){
      $num_args = func_num_args();

      if($num_args > 0){
        $args = func_get_args();
        $this->theword = $args[0];
      }
    }

    public function setvalue ($newvalue){
      $this->thevalue = $newvalue;
    }
    public function getvalue () {
      return $this->thevalue;
    }
    public function getword () {
      return $this->theword;
```

```
    }
  }

  //Now, create four different instances of this class.
  $myclass1 = new myclass ("Abra");
  $myclass1->setvalue (1);

  $myclass2 = new myclass ("Kadabra");
  $myclass2->setvalue (2);

  $myclass3 = new myclass ("Hocus");
  $myclass3->setvalue (3);

  $myclass4 = new myclass ("Pocus");
  $myclass4->setvalue (4);

  //Create a global array of
  $classarr = array ($myclass1,$myclass2,$myclass3,$myclass4);

  //Now, you can create a function that searches for a correct instance of a class.
  function &findclass ($whichclass,$classarr){
    for ($i = 0; $i < count ($classarr); $i++){
      if ($classarr[$i]->getvalue() == $whichclass){
        return $classarr[$i];
      }
    }
  }

  //Search for the id number 3, and return the word if it is found.
  $myobject = new myclass ("");
  $myobject =& findclass (3,$classarr);
  echo $myobject->getword();

?>
```

```
Hocus
```

How It Works

In this example, you create four objects of a certain class and fill them with four sets of values. Next, you create an array of the objects and a function that will sift through the array until it finds the object in question. If the object is found, the function can return the actual object through the magic of returning values by reference.

Although this may seem like overkill with four objects, consider if you had a hundred—or a thousand. The ability to sift through a mountain of objects and return the exact one you are looking for is incredibly valuable and can give you instant use of the object in question.

11-7. Returning Failure

A simplistic yet rather important aspect of functions is returning a failure value should something go wrong with the function. Functions can make wonderful systems for performing validation on different parts of your code, and they can be used as `true`/`false` values by simply returning a boolean result on success or failure. This sort of functionality can clean up your code and, with the right naming conventions, create code that is much easier to read. The following example returns a `true` or `false` value based on whether the e-mail value passed to it is a valid format.

The Code

```
<?php

  //sample11_7.php

  //A function to return a true/false value based on e-mail format.
  function validemail ($email = ""){
    return preg_match("/^([a-zA-Z0-9])+([.a-zA-Z0-9_-])*@([a-zA-Z0-9_-])➥
+(.[a-zA-Z0-9_-]+)+[a-zA-Z0-9_-]$/",$email);
  }
  $anemail = "lee@babinplanet.ca";
  //Use the function to confirm a valid e-mail.
  if (validemail ($anemail)){
    echo $anemail . " is in valid e-mail format.<br />";
  } else {
    echo $anemail . " is not valid.<br />";
  }

  //And of course, an invalid e-mail.
  $bademail = "ababemail";
  if (validemail ($bademail)){
    echo $bademail . " is in valid e-mail format.<br />";
  } else {
    echo $bademail . " is not valid.<br />";
  }
?>
```

```
lee@babinplanet.ca is in valid e-mail format.
ababemail is not valid.
```

How It Works

As you can see, the code to check the validity of an e-mail string's format is quite clear and easy to read. The function returns a `true` value if the format is valid and a `false` value if the format is incorrect. By using this in the code, you can easily see what the script is attempting to accomplish, so now you have a handy function to validate against user-submitted e-mail addresses that can be called at any time.

11-8. Calling Variable Functions

The concept of calling variable functions is an interesting one. Basically, by adding parentheses to the end of a variable you can force PHP to attempt to call a function of whatever name the value of the variable equates to. This can make for some nice conditional handling, as you can essentially determine which function is to be called on the fly by using a specific variable. Say, for instance, that you have three functions: one function adds two values, one subtracts two values, and the last multiplies two values. Based on what the user enters into a form, the script determines which function to use and then assigns a value to the variable that will be used to call the function.

The Code

```
<?php

  //sample11_8.php

  //A function to add two values.
  function addvalues ($firstvalue = 0, $secondvalue = 0){
    return $firstvalue + $secondvalue;
  }

  //A function to subtract two values.
  function subtractvalues ($firstvalue = 0, $secondvalue = 0){
    return $firstvalue - $secondvalue;
  }

  //A function to multiply two values.
  function multiplyvalues ($firstvalue = 0, $secondvalue = 0){
    return $firstvalue * $secondvalue;
  }
  //And let's assume these are the values you want to work with.
  $firstvalue = 10;
  $secondvalue = 3;

  //Let's say this value represents a user-submitted value.
  $whattodo = "addvalues";

  //You can then call the function as a variable.
  echo $whattodo($firstvalue, $secondvalue) . "<br />";
```

```
  //Let's say this value represents a user-submitted value.
  $whattodo = "subtractvalues";

  //You can then call the function as a variable.
  echo $whattodo($firstvalue, $secondvalue) . "<br />";

  //Let's say this value represents a user-submitted value.
  $whattodo = "multiplyvalues";

  //You can then call the function as a variable.
  echo $whattodo($firstvalue, $secondvalue) . "<br />";

?>
```

```
13
7
30
```

How It Works

The key aspect to note about this code is where you actually perform the function call. Does it look a little strange to you? Thanks to the power of variable function calls, you can assign a value dynamically to a variable and then have the script look for a function that is named the same as the variable's value. Naturally, if PHP cannot find a function by that name, you will get the regular errors you would get for attempting to call a function that does not exist. The powerful aspect of this code is that you can use conditional statements to determine which function gets called.

11-9. Accessing a Global Variable from Within a Function

While generally considered a quick-fix approach and not really a valid way to code because of programmers preferring more rigidly structured code (globals can easily get lost/changed), sometimes having global variables around is useful. Quite possibly the most useful aspect to global variables is using them within functions without having to pass them in as arguments. Because the variables are global, any script within the scope of the variable (basically, any script that has access to the originally declared global variable) will be able to use it without having to pass it around.

The current standard in PHP 5 is to use superglobals to access the global variables, and the following script shows you how to do it properly. PHP has the predefined superglobal variable `$GLOBALS` that can be used to create, access, and maintain global variables. The following function uses a global value that is set to tell you what the current username and password for the site are.

The Code

```
<?php

  //sample11_9.php

  $GLOBALS['user'] = "myusername";
  $GLOBALS['pass'] = "mypassword";

  //A function to check the validity of a login.
  function validatelogin ($username, $password){
    //Now, you do a quick comparison to see if the user➥
 has entered the correct login.
    if (strcmp ($username, $GLOBALS['user']) == 0 &&➥
 strcmp ($password, $GLOBALS['pass']) == 0){
      return true;
    } else {
      return false;
    }
  }

  //You then call the function and pass in the values you want checked.
  if (validatelogin ("myusername","mypassword")){
    echo "You are logged in correctly";
  } else {
    echo "You have an incorrect username and/or password";
  }
?>
```

```
You are logged in correctly
```

How It Works

You will notice that this example looks like recipe 11-1. You will, however, notice one key difference. Rather than assigning the current correct username and password values within the function, you can set the values anywhere within the scope of the script using the superglobal `$GLOBALS`. This means that rather than having to search the database within the function for the current proper login, you can search it within a hidden include file and then reference the values. It looks a little cleaner and helps hide what is potentially hazardous information from the wrong viewer.

11-10. Creating Dynamic Functions

One of the advantages of using PHP functions is that you can create conditional occurrences that allow you to write functions only if strictly necessary. By placing function declarations within conditional statements, you can force PHP to create a function only if a condition has been met. By using this sort of functionality, you can actually create functions dynamically by allowing functions to be born based on a certain condition.

Let's say you want to take in a value from the user, and based on that value you create a function that performs a certain task. For instance, based on what the user enters, you need a function either to add two values, to subtract two values, or to multiply two values. Rather than clutter your code with functions you may not use, you can create the valid function on the fly and call it by just one name.

The following example is useful in a site where a user can log in and log out based upon their current status.

The Code

```
<?php

  //sample11_10.php

  if ($_GET['go'] == "yes"){

    //Now, if you are logged in, you want the function to log you out.
    if ($_GET['loggedin'] == "true"){

      //Create a logout function.
      function dosomething (){
        $_GET['loggedin'] = false;
        echo "You have been successfully logged out.<br />";
      }

    }

    //Now, if you were not logged in, you want to be able to log in.
    if ($_GET['loggedin'] == "false"){

      //Create a login function.
      function dosomething (){
        $_GET['loggedin'] = true;
        echo "You have been successfully logged in.<br />";
      }
    }

    dosomething();

  }
```

```
  if ($_GET['loggedin']){
    ?><a href="sample11_10.php?go=yes&amp;loggedin=true">➥
click here to log out</a><?php
  } elseif (!$_GET['loggedin']){
    ?><a href="sample11_10.php?go=yes&amp;loggedin=false">➥
click here to log in</a><?php
  }
?>
```

If you click to log in, you should get this message and hence be logged in:

```
You have been successfully logged in.
click here to log out
```

If, however, you click to log out, you should get the following result:

```
You have been successfully logged out.
click here to log in
```

How It Works

This particular instance is based on a login principle. If a person is logged in, you want the function to allow them to log out. If, however, the person is logged out, you want to provide them with a means to log in. Through the power of dynamic function creation, you can make the same function call but actually have it perform two (or more) different actions.

Summary

As you can see, PHP 5 not only supports a myriad of ways to clean up and modularize your code, but it also allows you to manipulate your functions in a wide variety of ways. By using functions to ensure that you are never using redundant code in your applications, you cut back on the time you will spend coding and make your code more applicable both for others to use and for you to clean up should the need arise.

PHP 5 supports passing and receiving values by reference as well as by value, and you should always use the defaults if you think the validity of the code calling the function could ever come into question. The ideal way to do things is to evaluate the task at hand and then select the most efficient method for the job. Passing and returning by reference can be an ideal solution for keeping integrity within a variable or group of variables, and passing and returning by value is ideal for working with a given data set.

PHP also supports many ways to base your code upon dynamic dealings. By using dynamic functions or variable function calls, you can reduce the processing and preloading time of your script by deciding on the fly what calls are necessary and which function declarations are important. This allows for a wide range of ingenuity and good, clean coding.

All in all, you can make a powerful set of PHP code that much more efficient by proper, smart function use, and the amount of time it will save you in the end is well worth the initial investment.

Looking Ahead

In the next chapter, we will introduce a topic that is quite far from basic, web basics. We will cover a wide variety of important web aspects to show you how to turn a bland, static website into a dynamic, living, breathing entity. No good web application is complete without the upcoming knowledge contained within Chapter 12.