CHAPTER 5

Introducing PHP

What You'll Learn in This Chapter:

- How PHP works with the web server as a "server-side" language
- How to include PHP code in a web page
- Beginning and ending PHP scripts
- How to use HTML, JavaScript, and PHP in the same file

In the first four chapters of this book, you worked with client-side or front-end languages: HTML and JavaScript. In this chapter, you'll learn a bit about PHP, which is a server-side scripting language that operates on a web server. You'll learn how to include PHP code into your HTML documents and how these scripts are executed when you refer to a page in your web browser that accesses a remote web server. This is a short chapter that is meant to be a bridge between the front end functionality you just learned about, and the backend functionality that you'll pick up again in Part III of this book.

How PHP Works with a Web Server

Often, when a user submits a request to a web server for a web page, the server reads a simple HTML file (that may or may not include JavaScript) and sends its contents back to the browser in response. If the request is for a

PHP file, or for an HTML document that includes PHP code, and the server supports PHP, then the server looks for PHP code in the document, executes it, and includes the output of that code in the page in place of the PHP code. Here's a simple example:

Click here to view code image

```
<!DOCTYPE html>
<html>
   <head>
     <title>There's PHP in Here</title>
   </head>
   <body>
      <?php echo "Howdy!"; ?>
   </body>
</html>
```

If this page is requested from a web server that supports PHP, the HTML sent to the browser will look like this:

Click here to view code image

```
<!DOCTYPE html>
<html>
   <head>
     <title>There's PHP in Here</title>
   </head>
   <body>
      Howdy!
   </body>
</html>
```

When the user requests the page, the web server determines that it is a PHP page rather than a regular HTML page. If a web server supports PHP, it usually treats any files with the extension `.php` as PHP pages. Assuming this page is called something like `howdy.php`, when the web server receives the request, it scans the page looking for PHP code and then runs any code it finds. PHP code is distinguished from the rest of a page by PHP tags, which you'll learn more about in the next section.

Whenever the server finds those tags, it treats whatever is within them as PHP code. That's not so different from the way things work with JavaScript, where anything inside `<script>` tags is treated as JavaScript code.

The Basics of PHP Scripts

Let’s jump straight into the fray with the simplest PHP script possible that also produces some meaningful output. To begin, open your favorite text editor. Like HTML documents, PHP files are made up of plain text. You can create them with any text editor, and most popular HTML editors and programming IDEs (integrated development environments) provide support for PHP.

TIP

If you have an IDE or simple text editor that you enjoy using for HTML and JavaScript, it’s likely to work just fine with PHP as well.

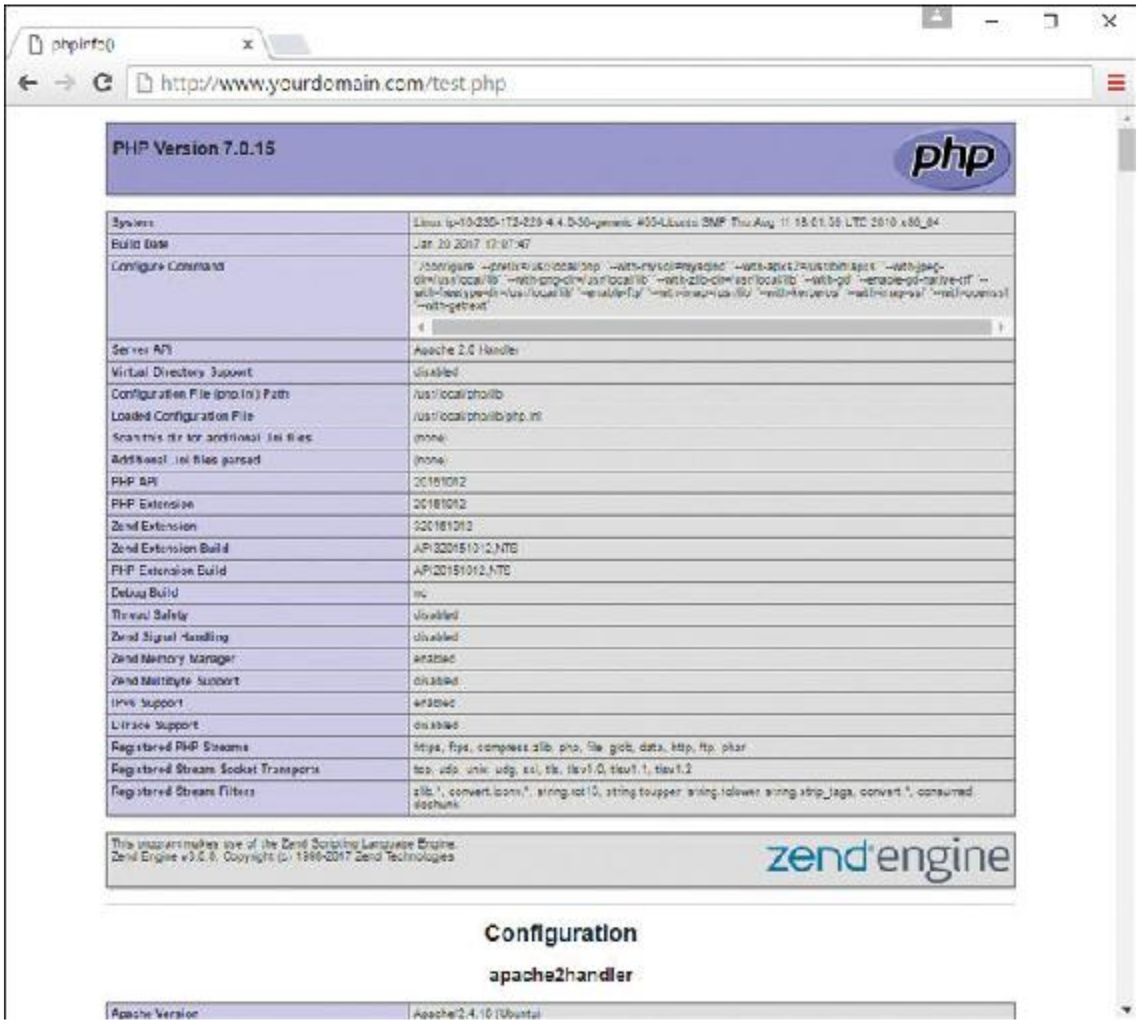
Type in the example shown in Listing 5.1 and save the file to the document root of your web server, using a name something like `test.php`.

LISTING 5.1 A Simple PHP Script

```
<?php
    phpinfo();
?>
```

This script simply tells PHP to use the built-in function called `phpinfo()`. This function automatically generates a significant amount of detail about the configuration of PHP on your system. You can also begin to see the power of a scripting language, in which a little bit of text can go a long way toward producing something useful.

If you are not working directly on the machine that will be serving your PHP script, you need to use a File Transfer Protocol (FTP) or Secure Copy Protocol (SCP) client to upload your saved document to the server. When the document is in place on the server, you should be able to access it using your browser. If all has gone well, you should see the script’s output. Figure 5.1 shows the output from the `test.php` script.



PHP Version 7.0.15

System	Linux ip-10-230-173-229 4.4.0-30-generic #50-Ubuntu SMP Thu Aug 11 18:01:35 UTC 2016 x86_64
Build Date	Jan 20 2017 17:07:47
Configure Command	[illegible]
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/php/lib
Loaded Configuration File	/usr/local/php/lib/php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,NTS
PHP Extension Build	API20151012,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.0.0, Copyright (c) 1998-2017 Zend Technologies

Configuration

apache2handler

Apache Version	Apache/2.4.10 (Ubuntu)

FIGURE 5.1
Success: the output from `test.php`.

Beginning and Ending a Block of PHP Statements

When writing PHP, you need to inform the PHP engine that you want it to execute your commands. If you don’t do this, the code you write will be mistaken for HTML and will be output to the browser. You can designate your code as PHP with special tags that mark the beginning and end of PHP code blocks. Table 5.1 shows the two possible PHP delimiter tags.

TABLE 5.1 PHP Start and End Tags

Tag Style	Start Tag	End Tag

Standard tags	`<?php`	`?>`
Short tags	`<?`	`?>`

Standard tags are highly recommended and are the default expectation by the PHP engine. If you want to use short tags, you must explicitly enable short tags in your PHP configuration by making sure that the `short_open_tag` switch is set to `On` in `php.ini`:

```
short_open_tag = On;
```

Such a configuration change will require a restart of your web server. This is largely a matter of preference, although if you intend to include XML in your script, you should not enable the short tags (`<?` and `?>`) and should work with the standard tags (`<?php` and `?>`). Standard tags will be used throughout this book, but you should be aware you may run into this other type of tag if you are reading other people's code; if you use that code, you must adjust the PHP tags before you use it.

CAUTION

The character sequence `<?` tells an XML parser to expect a processing instruction and is therefore often included in XML documents. If you include XML in your script and have short tags enabled, the PHP engine is likely to confuse XML processing instructions and PHP start tags. Definitely do not enable short tags if you intend to incorporate XML in your document. Again, the standard tags will really serve you best in the long run.

Let's run through the two ways in which you can legally write the code in Listing 5.1, including using short tags if that configuration option is enabled:

```
<?php
    phpinfo();
?>
```

```
<?
     phpinfo();
?>
```

You can also put single lines of code in PHP on the same line as the PHP start and end tags:

```
<?php phpinfo(); ?>
```

Now that you know how to define a block of PHP code with PHP tag delimiters, let's move forward.

The `echo` and `print()` Statements

The following bit of PHP code does exactly one thing—it displays "Hello!" on the screen:

```
<?php
     echo "Hello!";
?>
```

The `echo` statement, which is what we used here, outputs data. In most cases, anything output by `echo` ends up viewable in the browser. Alternatively, you could have used the `print()` statement in place of the `echo` statement. Using `echo` or `print()` is a matter of taste; when you look at other people's scripts, you might see either used, which is why I mention both here.

Referring back to the code you have seen so far, note the only line of code in Listing 5.1 ended with a semicolon. The semicolon informs the PHP engine that you have completed a statement, and it's probably the most important bit of coding syntax you could learn at this stage.

A *statement* represents an instruction to the PHP engine. Broadly, it is to PHP what a sentence is to written or spoken English. A sentence should usually end with a period; a statement should usually end with a semicolon. Exceptions to this rule include statements that enclose other statements and statements that end a block of code. In most cases, however, failure to end a statement with a semicolon will confuse the PHP engine and result in an error.

Combining HTML and PHP

The script in Listing 5.1 is pure PHP, but that does not mean you can only use PHP in PHP scripts. You can use PHP and HTML (and JavaScript too!) in the same document by simply ensuring that all of the PHP is enclosed by PHP start and end tags, as shown in Listing 5.2.

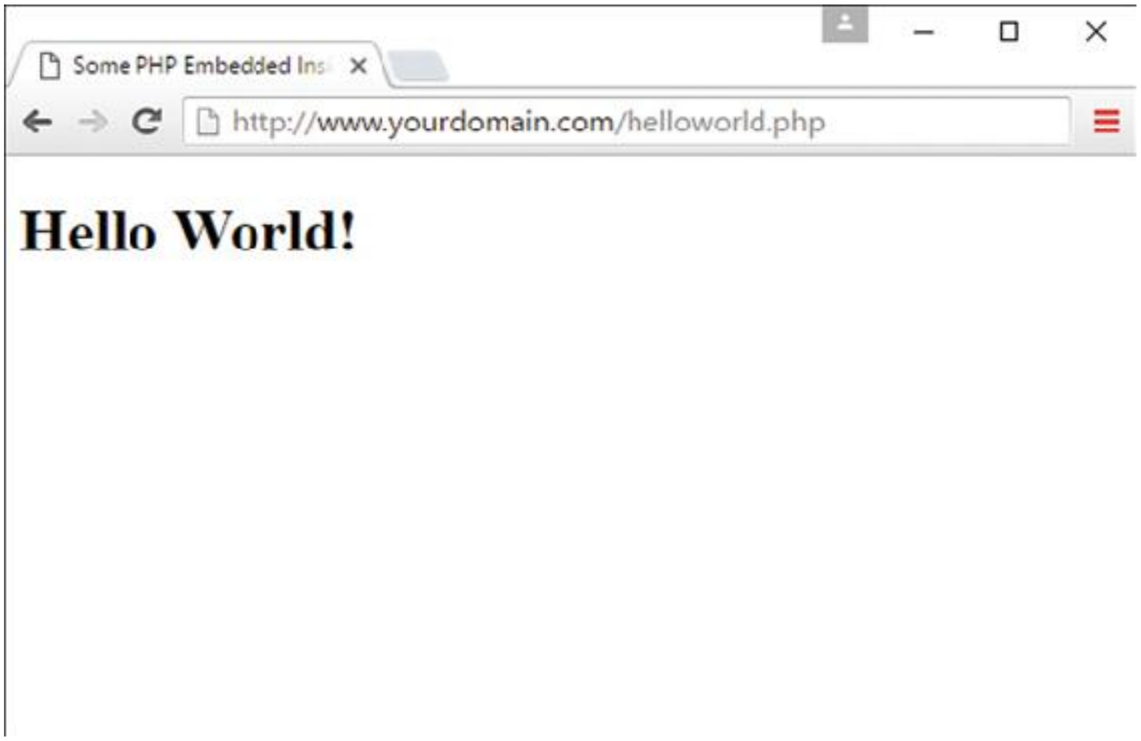
LISTING 5.2 Some PHP Embedded Inside HTML

Click here to view code image

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Some PHP Embedded Inside HTML</title>
  </head>
  <body>
     <h1><?php echo "Hello World!"; ?></h1>
  </body>
</html>
```

As you can see, incorporating PHP code into a predominantly HTML document is simply a matter of typing in the code because the PHP engine ignores everything outside the PHP start and end tags.

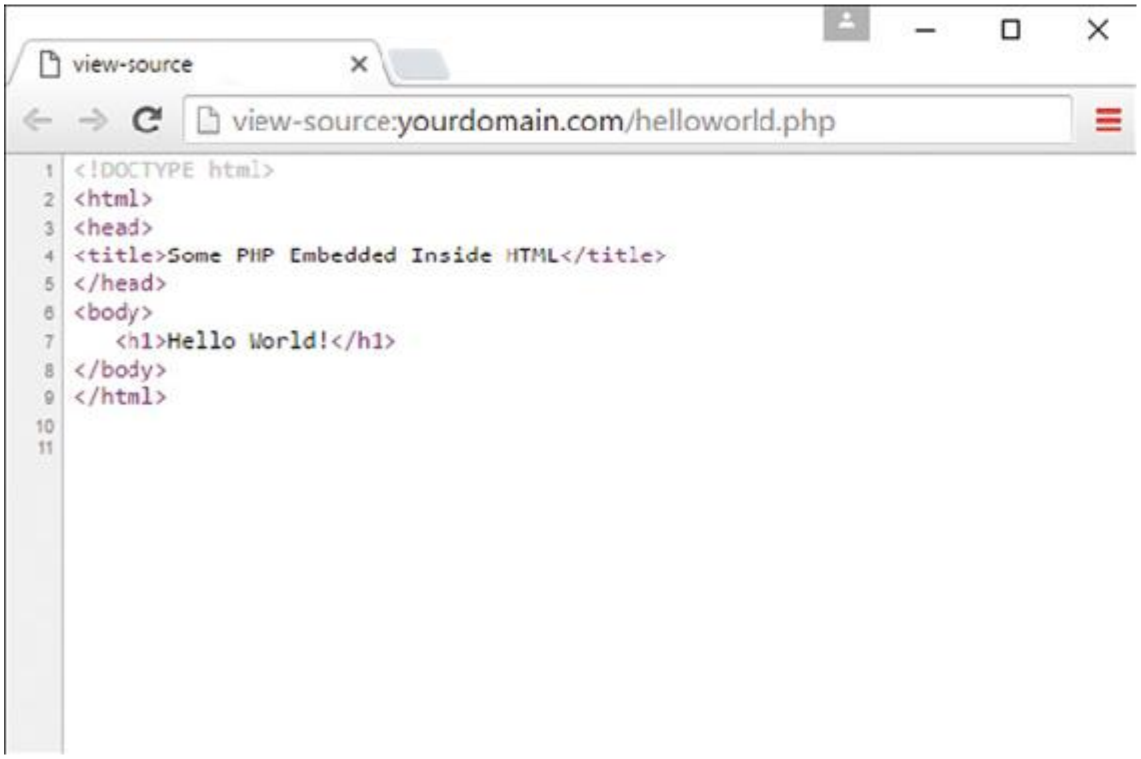
Save the contents of Listing 5.2 as `helloworld.php`, place it in your document root, and then view it with a browser. You should see the string `Hello World!` in a large, bold heading, as shown in Figure 5.2. If you were to view the document source, as shown in Figure 5.3, the listing would look exactly like a normal HTML document, because all of the processing by the PHP engine will have already taken place before the output gets to the browser for rendering.



Hello World!

FIGURE 5.2
The output of `helloworld.php` as viewed in a browser.

You can include as many blocks of PHP code as you need in a single document, interspersing them with HTML however it makes sense. Although you can have multiple blocks of code in a single document, they combine to form a single continuous script as far as the PHP engine is concerned.



```
<!DOCTYPE html>
<html>
<head>
<title>Some PHP Embedded Inside HTML</title>
</head>
<body>
   <h1>Hello World!</h1>
</body>
</html>
```

FIGURE 5.3
The output of `helloworld.php` as HTML source code.

Adding Comments to PHP Code

Code that seems clear at the time you write it can seem like a hopeless tangle when you try to amend it 6 months later. Adding comments to your code as you write can save you time later on and make it easier for other programmers to work with your code.

A *comment* is text in a script that is ignored by the PHP engine. Comments can make code more readable or can be use to annotate a script.

Single-line comments begin with two forward slashes (`//`), which is the preferred style, or a single hash or pound sign (`#`), which you may see in other people's code if those folks are more used to working in Perl or other languages where that comment type is more the norm. Regardless of which valid comment type you use, the PHP engine ignores all text between these marks and either the end of the line or the PHP close tag:

```
// this is a comment
#  this is another comment
```

Multiline comments begin with a forward slash followed by an asterisk (`/*`) and end with an asterisk followed by a forward slash (`*/`):

```
/*
this is a comment
none of this will
be parsed by the
PHP engine
*/
```

Code comments, just like HTML and JavaScript (and anything else that isn't PHP), are ignored by the PHP engine and will not cause errors that halt the execution of your scripts.

Code Blocks and Browser Output

In the previous section, you learned that you can slip in and out of HTML mode at will using the PHP start and end tags. In later chapters, you'll learn more about how you can present distinct output to the user according to a decision-making process you can control by using what are called *flow control statements*, which exist in both JavaScript and PHP. Although flow control in both languages will be discussed at length as the book moves forward, I use a basic example here to continue the lesson about mingling PHP and HTML with ease.

Imagine a script that outputs a table of values only when some condition is true, and doesn't output anything when a condition is false. Listing 5.3 shows a simplified HTML table constructed with the code block of an `if` statement.

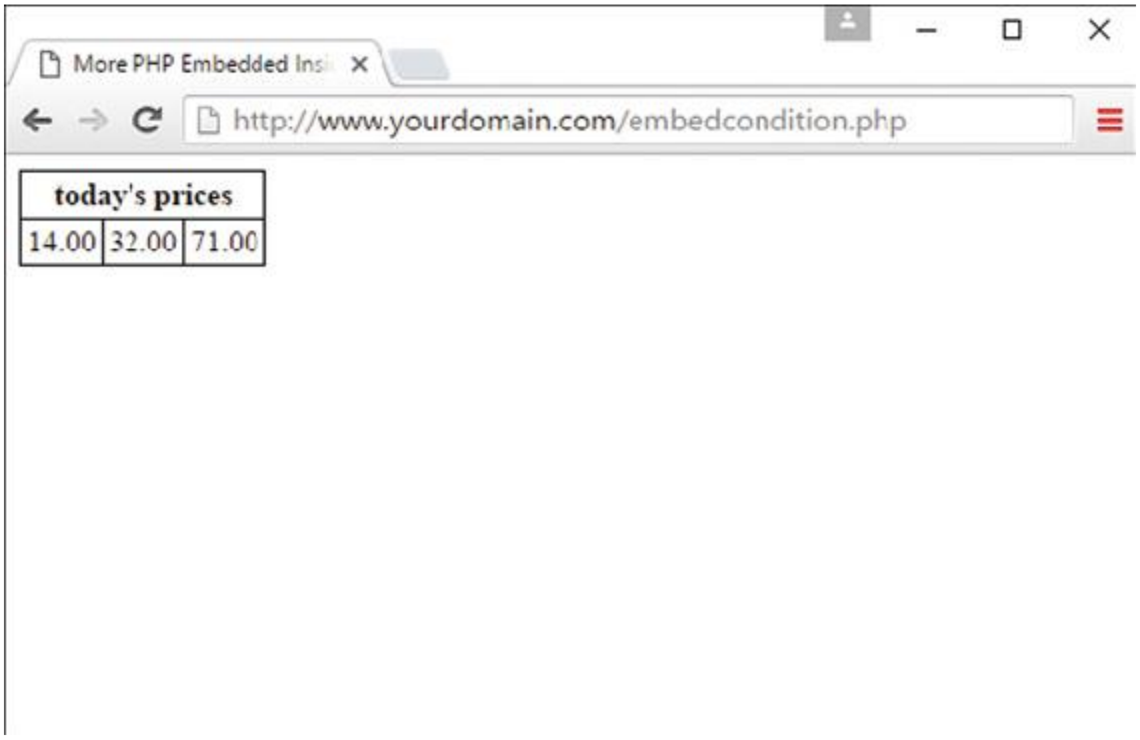
LISTING 5.3 PHP Displays Text if a Condition Is True

Click here to view code image

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>More PHP Embedded Inside HTML</title>
   <style type="text/css">
   table, tr, th, td {
        border: 1px solid #000;
        border-collapse: collapse;
        padding: 3px;
   }
   th {
        font-weight: bold;
   }
   </style>
   </head>
   <body>
   <?php
   $some_condition = true;
   if ($some_condition) {
      echo "<table>
      <tr><th colspan=\"3\">
      Today's Prices
      </th></tr>
      <tr><td>14.00</td><td>32.00</td><td>71.00</td></tr>
      </table>";
    }
    ?>
    </body>
</html>
```

If the value of `$some_condition` is `true`, the table is printed. For the sake of readability, we split the output into multiple lines surrounded by one `echo` statement, and we use the backslash to escape any quotation marks used in the HTML output.

Put these lines into a text file called `embedcondition.php` and place this file in your web server document root. When you access this script through your web browser, it should look like Figure 5.4.



today's prices		
14.00	32.00	71.00

FIGURE 5.4
Output of `embedcondition.php`.

There’s nothing wrong with the way this is coded, but you may find your code more readable to slip back into HTML mode within the code block itself. Listing 5.4 does just that.

LISTING 5.4 Returning to HTML Mode Within a Code Block

Click here to view code image

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>More PHP Embedded Inside HTML</title>
   <style type="text/css">
   table, tr, th, td {
        border: 1px solid #000;
        border-collapse: collapse;
        padding: 3px;
   }
   th {
        font-weight: bold;
   }
   </style>
   </head>
```

```
    <body>
        <?php
        $some_condition = true;
        if ($some_condition) {
        ?>
        <table>
        <tr><th colspan="3">Today's Prices</th></tr>
        <tr><td>14.00</td><td>32.00</td><td>71.00</td></tr>
        </table>
        <?php
        }
        ?>
     </body>
</html>
```

The important thing to note here is that the shift to HTML mode occurs only if the condition of the `if` statement is fulfilled. Slipping in and out of PHP or HTML mode like this can save you from the time and effort of escaping quotation marks and wrapping output in `echo` statements. This approach might, however, affect the readability of the code in the long run, especially if the script grows larger.

Summary

In this chapter, you learned the concept of using PHP as a server-side scripting language, and using the `phpinfo()` function, you produced a list of the PHP configuration values for the server you are using. You created a simple PHP script using a text editor, and you learned about the tags you can use to begin and end blocks of PHP code.

You learned how to use the `echo` and `print` statements to send data to the browser, and you brought HTML and PHP together into the same script. You also looked at a technique for using PHP start and end tags in conjunction with conditional code blocks, as another way to embed PHP within HTML.

The first five chapters in this book lay the groundwork for creating web applications by introducing you to the idea of interactive websites and basic concepts used when developing web content with HTML, JavaScript, and PHP. The PHP language and its use on the back end of applications (the server side), as opposed to the front end (your web browser), makes it the more complex of the three. However, there are many similarities in the fundamentals and syntax of both PHP and JavaScript, such that directly after this chapter we will refocus on JavaScript for a bit before returning to PHP to build interactive and dynamic applications.

Q&A

Q. Which are the best start and end tags to use?

A. It is largely a matter of preference. For the sake of portability, the standard tags (`<?php` and `?>`) are preferred.

Q. What editors should I avoid when creating PHP code?

A. Do not use word processors that format text for printing (Microsoft Word, for example). Even if you save files created using this type of editor in plain-text format, hidden characters are likely to creep into your code.

Q. When should I comment my code?

A. Again, this is a matter of preference. Some short scripts will be self-explanatory, even when you return to them after a long interval. For scripts of any length or complexity, you should comment your code. Comments in your code often save you time and frustration in the long run.

Workshop

The Workshop is designed to help you review what you’ve learned and begin putting your knowledge into practice.

Quiz

1. Can a person browsing your website read the source code of a PHP script you have successfully installed?
2. Is the following a valid PHP script that will run without errors? If so, what will display in the browser?

   ```
   <?php echo "Hello World!" ?>
   ```

3. Is the following a valid PHP script that will run without errors? If so, what will display in the browser?

   ```
   <?php
   // I learned some PHP!
   ?>
   ```

Answers

1. No, the user will see only the output of your script.
2. No, this code will produce an error because the `echo` statement is not terminated by a semicolon.
3. Yes, this code will run without errors. However, it will produce nothing in the browser because the only code within the PHP start and end tags is a PHP comment, which will be ignored by the PHP engine. Therefore, no output will be rendered.

Exercises

1. Familiarize yourself with the process of creating, uploading, and running PHP scripts. In particular, create your own "Hello World" script. Add HTML code to it as well as additional blocks of PHP.

Part II: Getting Started with Dynamic Websites