

Basic Syntax

As stated in the book's introduction, PHP is an *HTML-embedded* scripting language. This means that you can intermingle PHP and HTML code within the same file. So to begin programming with PHP, start with a simple Web page. **Script 1.1** gives an example of a no-frills, no-content XHTML Transitional document, which will be used as the foundation for every Web page in the book (this book does not formally discuss [X]HTML; see a resource dedicated to the topic for more information).

To add PHP code to a page, place it within PHP tags:

```
<?php
```

```
?>
```

Anything placed within these tags will be treated by the Web server as PHP (meaning the PHP interpreter will process the code). Any text outside of the PHP tags is immediately sent to the Web browser as regular HTML.

Along with placing PHP code within PHP tags, your PHP files must have a proper extension. The extension tells the server to treat the script in a special way, namely, as a PHP page. Most Web servers will use `.html` or `.htm` for standard HTML pages, and normally, `.php` is preferred for your PHP files.

To make a basic PHP script:

1. Create a new document in your text editor or Integrated Development Environment (**Script 1.2**).

It generally does not matter what application you use, be it Dreamweaver (a fancy IDE), BBEdit (a great and popular Macintosh plain-text editor), or vi (a plain-text Unix editor, lacking a graphical interface). Still, some text editors and

Script 1.1 A basic XHTML 1.0 Transitional Web page.



```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.0 Transitional//EN" "http://www.w3.org/
    TR/xhtml1/DTD/xhtml1-transitional.dtd">

2  <html xmlns="http://www.w3.org/1999/
    xhtml" xml:lang="en" lang="en">

3  <head>

4      <meta http-equiv="content-type" content=
        "text/html; charset=iso-8859-1" />

5      <title>Page Title</title>

6  </head>


7  <body>

8  </body>

9  </html>

```

Script 1.2 This first PHP script doesn't do anything, per se, but does demonstrate how a PHP script is written. It'll also be used as a test, prior to getting into elaborate PHP code.



```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
   1.0 Transitional//EN" "http://www.w3.org/
   TR/xhtml1/DTD/xhtml1-transitional.dtd">
2  <html xmlns="http://www.w3.org/1999/xhtml"
   xml:lang="en" lang="en">
3  <head>
4    <meta http-equiv="content-type" content=
   "text/html; charset=iso-8859-1" />
5    <title>Basic PHP Page</title>
6  </head>
7  <body>
8    <p>This is standard HTML.</p>
9  <?php
10 ?>
11 </body>
12 </html>

```

IDEs make typing and debugging HTML and PHP easier (conversely, Notepad on Windows does some things that makes coding harder). If you don't already have an application you're attached to, search the Web or use the book's corresponding forum (www.DMCInsights.com/phorum/) to find one.

2. Start a basic HTML document.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//EN"
→ http://www.w3.org/TR/xhtml1/DTD/
→ xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/
→ xhtml" xml:lang="en" lang="en">

<head>

    <meta http-equiv="content-type"
    → content="text/html; charset=
    → iso-8859-1" />

    <title>Basic PHP Page</title>

</head>

<body>

<p>This is standard HTML.</p>

</body>

</html>

```

Although this is the syntax being used throughout the book, you can change the HTML to match whichever standard you intend to use (e.g., HTML 4.0 Strict). Again, see a dedicated (X)HTML resource if you're unfamiliar with this HTML code (see the first tip).

continues on next page

- Before the closing `body` tag, insert your PHP tags.

```
<?php
?>
```

These are the formal PHP tags, also known as XML-style tags. Although PHP supports other tag types (see the second tip), I recommend that you use the formal type, and I will do so throughout this book.

- Save the file as `first.php`.

Remember that if you don't save the file using an appropriate PHP extension, the script will not execute properly.

- Place the file in the proper directory of your Web server.

If you are running PHP on your own computer (presumably after following the installation directions in Appendix A, "Installation"), you just need to move, copy, or save the file to a specific folder on your computer. Check the documentation for your particular Web server to identify the correct directory, if you don't already know what it is.

If you are running PHP on a hosted server (i.e., on a remote computer), you'll need to use an FTP application to upload the file to the proper directory. Your hosting company will provide you with access and the other necessary information.

- Run `first.php` in your Web browser (**Figure 1.1**).

Because PHP scripts need to be parsed by the server, you *absolutely must* access them via the URL. You cannot simply open them in your Web browser as you would a file in other applications.

If you are running PHP on your own computer, you'll need to go to something like `http://localhost/first.php`, `http://127.0.0.1/first.php`, or

`http://localhost/~<user>/first.php` (on Mac OS X, using your actual username for `<user>`). If you are using a Web host, you'll need to use `http://your-domain-name/first.php` (e. g., `http://www.example.com/first.php`).

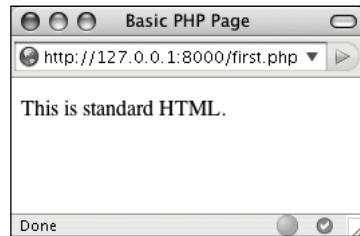


Figure 1.1 While it seems like any other (simple) HTML page, this is in fact a PHP script and the basis for the rest of the examples in the book.

7. If you don't see results like those in Figure 1.1, start debugging.

Part of learning any programming language is mastering debugging. It's a sometimes-painful but absolutely necessary process. With this first example, if you don't see a simple, but perfectly valid, Web page, follow these steps:

1. Confirm that you have a working PHP installation (see Appendix A for testing instructions).
2. Make sure that you are running the script through a URL. The address in the Web browser must begin with `http://`. If it starts with `file://`, that's the problem (**Figure 1.2**).
3. If you get a file not found (or similar) error, you've likely put the file in the wrong directory or mistyped the file's name (either when saving it or in your Web browser).

If you've gone through all this and are still having problems, turn to the book's corresponding forum (www.DMCInsights.com/phorum/list.php?20).

✓ Tips

- To find more information about HTML and XHTML, check out Elizabeth Castro's excellent book *HTML, XHTML, and CSS, Sixth Edition: Visual QuickStart Guide*, (Peachpit Press, 2006) or search the Web.
- There are actually three different pairs of PHP tags. Besides the formal (`<?php` and `?>`), there are the short tags (`<? and ?>`), and the script style (`<script language="php">` and `</script>`). This last style is rarely used, and the formal style is recommended.
- Because I am running PHP on my own computer, you will sometimes see URLs like `http://127.0.0.1:8000/first.php` in this book's figures. The important thing is that I'm running these scripts via `http://`; don't let the rest of the URL confuse you.
- You can embed multiple sections of PHP code within a single HTML document (i.e., you can go in and out of the two languages). You'll see examples of this throughout the book.

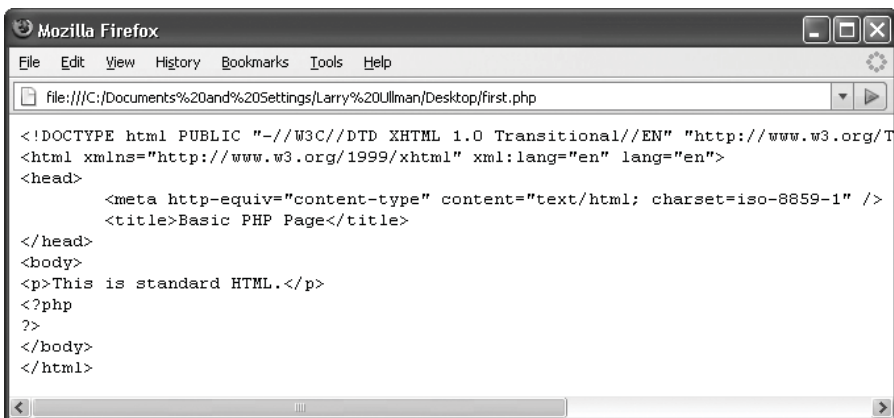


Figure 1.2 If you see the actual PHP code (in this case, the tags) in the Web browser, this means that the PHP Web server is not running the code for one reason or another.

Sending Data to the Web Browser

To create dynamic Web sites with PHP, you must know how to send data to the Web browser. PHP has a number of built-in functions for this purpose, the most common being `echo()` and `print()`. I personally tend to favor `echo()`:

```
echo 'Hello, world!';
echo "What's new?";
```

You could use `print()` instead, if you prefer:

```
print "Hello, world!";
print "What's new?";
```

As you can see from these examples, you can use either single or double quotation marks (but there is a distinction between the two types of quotation marks, which will be made clear by the chapter's end). The first quotation mark after the function name indicates the start of the message to be printed. The next matching quotation mark (i.e., the next quotation mark of the same kind as the opening mark) indicates the end of the message to be printed.

Along with learning how to send data to the Web browser, you should also notice that in PHP all statements (a line of executed code, in layman's terms) must end with a semicolon. Also, PHP is case-insensitive when it comes to function names, so `ECHO()`, `echo()`, `eChO()`, and so forth will all work. The all-lowercase version is easiest to type, of course.

Needing an Escape

As you might discover, one of the complications with sending data to the Web involves printing single and double quotation marks. Either of the following will cause errors:

```
echo "She said, "How are you?"";
echo 'I'm just ducky.';
```

There are two solutions to this problem. First, use single quotation marks when printing a double quotation mark and vice versa:

```
echo 'She said, "How are you?";
echo "I'm just ducky.";
```

Or, you can *escape* the problematic character by preceding it with a backslash:

```
echo "She said, \"How are you?\"";
print 'I\'m just ducky.';
```

As escaped quotation mark will merely be printed like any other character. Understanding how to use the backslash to escape a character is an important concept, and one that will be covered in more depth at the end of the chapter.

Script 1.3 Using `print()` or `echo()`, PHP can send data to the Web browser (see Figure 1.3).

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN" "http://www.w3.org/
  TR/xhtml1/DTD/xhtml1-transitional.dtd">

2  <html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">

3  <head>

4  <meta http-equiv="content-type" content=
  "text/html; charset=iso-8859-1" />

5  <title>Using Echo()</title>

6  </head>

7  <body>

8  <p>This is standard HTML.</p>

9  <?php

10 echo 'This was generated using PHP!';

11 ?>

12 </body>

13 </html>

```

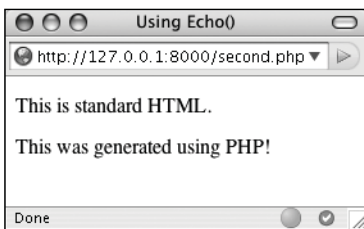


Figure 1.3 The results still aren't glamorous, but this page was in part dynamically generated by PHP.

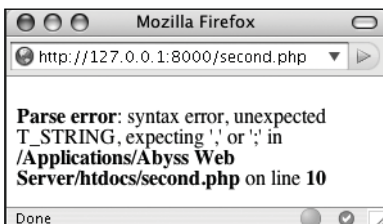


Figure 1.4 This may be the first of many parse errors you see as a PHP programmer (this one is caused by an un-escaped quotation mark).

To send data to the Web browser:

1. Open `first.php` (refer to Script 1.2) in your text editor or IDE.
2. Between the PHP tags (lines 9 and 10), add a simple message (**Script 1.3**).
`echo 'This was generated using
→ PHP!';`
3. If you want, change the page title to better describe this page (line 5).
`<title>Using Echo()</title>`

This change only affects the browser window's title bar.

4. Save the file as `second.php`, place it in your Web directory, and test it in your Web browser (**Figure 1.3**).

5. If necessary, debug the script.

If you see a parse error instead of your message (see **Figure 1.4**), check that you have both opened and closed your quotation marks and escaped any problematic characters (see the sidebar). Also be certain to conclude each statement with a semicolon.

continues on next page

If you see an entirely blank page, this is probably for one of two reasons:

- ▲ There is a problem with your HTML. Test this by viewing the source of your page and looking for HTML problems there (**Figure 1.5**).
- ▲ An error occurred, but `display_errors` is turned off in your PHP configuration, so nothing is shown. In this case, see the section in Appendix A on how to configure PHP so that you can turn `display_errors` back on.

✓ Tips

- Technically, `echo()` and `print()` are language constructs, not functions. That being said, don't be flummoxed as I continue to call them "functions" for convenience. Also, I include the parentheses when referring to functions—say `echo()`, not just `echo`—to help distinguish them from variables and other parts of PHP. This is just my own little convention.

- You can, and often will, use `echo()` and `print()` to send HTML code to the Web browser, like so (**Figure 1.6**):

```
echo '<p>Hello, <b>world</b>!</p>';
```

- `Echo()` and `print()` can both be used to print text over multiple lines:

```
echo 'This sentence is  
printed over two lines.';
```

What happens in this case is that the return (created by pressing Enter or Return) becomes part of the printed message, which isn't terminated until the closing single quotation mark. The net result will be the "printing" of the return in the HTML source code (**Figure 1.7**). This will not have an effect on the generated page (**Figure 1.8**).

For more on this, see the sidebar "Understanding White Space."

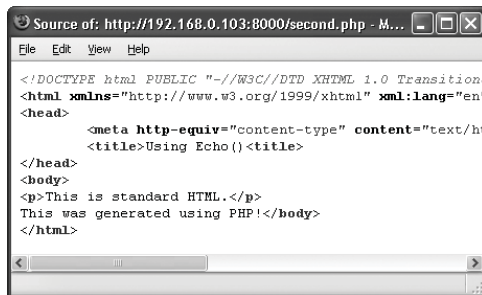


Figure 1.5 One possible cause of a blank PHP page is a simple HTML error, like the closing title tag here (it's missing the slash).

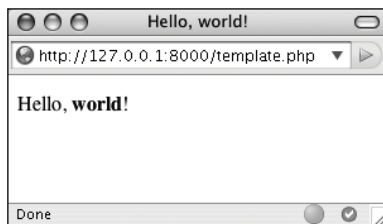


Figure 1.6 PHP can send HTML code (like the formatting here) as well as simple text (see Figure 1.3) to the Web browser.

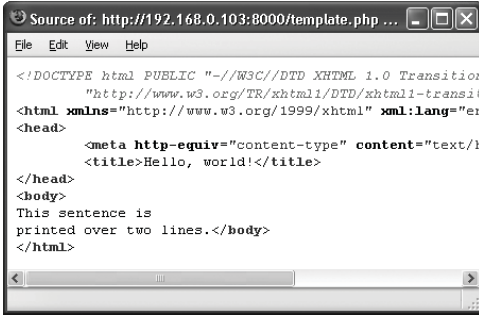


Figure 1.7 Printing text and HTML over multiple PHP lines will generate HTML source code that also extends over multiple lines. Note that extraneous white spacing in the HTML source will not affect the look of a page (see Figure 1.8) but can make the source easier to review.



Figure 1.8 The return in the HTML source (Figure 1.7) has no effect on the rendered result. The only way to alter the spacing of a displayed Web page is to use HTML tags (like `
` and `<p>`).

Understanding White Space

With PHP you send data (like HTML tags and text) to the Web browser, which will, in turn, render that data as the Web page the end user sees. Thus, what you are doing with PHP is creating the *HTML source* of a Web page. With this in mind, there are three areas of notable *white space* (extra spaces, tabs, and blank lines): in your PHP scripts, in your HTML source, and in the rendered Web page.

PHP is generally white space insensitive, meaning that you can space out your code however you want to make your scripts more legible. HTML is also generally white space insensitive. Specifically, the only white space in HTML that affects the rendered page is a single space (multiple spaces still get rendered as one). If your HTML source has text on multiple lines, that doesn't mean it'll appear on multiple lines in the rendered page (see Figures 1.7 and 1.8).

To alter the spacing in a rendered Web page, use the HTML tags `
` (line break, `
` in older HTML standards) and `<p>` (paragraph). To alter the spacing of the HTML source created with PHP, you can

- ◆ Use `echo()` or `print()` over the course of several lines.
- or*
- ◆ Print the newline character (`\n`) within double quotation marks.

Writing Comments

Creating executable PHP code is only a part of the programming process (admittedly, it's the most important part). A secondary but still crucial aspect to any programming endeavor involves documenting your code.

In HTML you can add comments using special tags:

```
<!-- Comment goes here. -->
```

HTML comments are viewable in the source (Figure 1.9) but do not appear in the rendered page.

PHP comments are different in that they aren't sent to the Web browser at all, meaning they won't be viewable to the end user, even when looking at the HTML source.

PHP supports three comment types. The first uses the pound or number symbol (#):

```
# This is a comment.
```

The second uses two slashes:

```
// This is also a comment.
```

Both of these cause PHP to ignore everything that follows until the end of the line (when you press Return or Enter). Thus, these two comments are for single lines only. They are also often used to place a comment on the same line as some PHP code:

```
print 'Hello!'; // Say hello.
```

A third style allows comments to run over multiple lines:

```
/* This is a longer comment  
that spans two lines. */
```

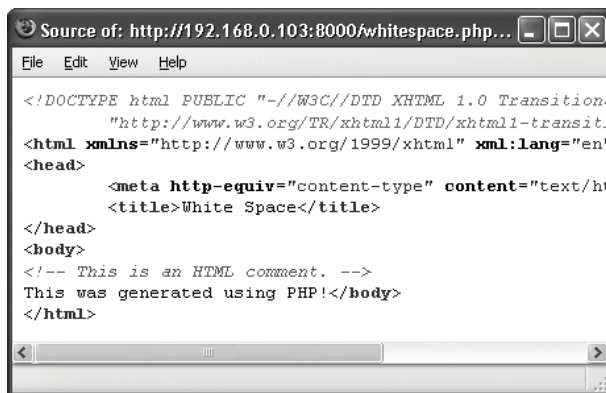


Figure 1.9 HTML comments appear in the browser's source code but not in the rendered Web page.

Script 1.4 These basic comments demonstrate the three syntaxes you can use in PHP.

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.0 Transitional//EN" "http://www.w3.org/
    TR/xhtml1/DTD/xhtml1-transitional.dtd">
2  <html xmlns="http://www.w3.org/1999/
    xhtml" xml:lang="en" lang="en">
3  <head>
4      <meta http-equiv="content-type" content=
        "text/html; charset=iso-8859-1" />
5      <title>Comments</title>
6  </head>
7  <body>
8  <?php
9
10     # Created August 27, 2007
11     # Created by Larry E. Ullman
12     # This script does nothing much.
13
14     echo '<p>This is a line of text.<br />This
        is another line of text.</p>';
15
16     /*
17     echo 'This line will not be executed.';
18     */
19
20     echo "<p>Now I'm done.</p>"; // End of PHP
        code.
21
22     ?>
23 </body>
24 </html>

```

To comment your scripts:

1. Begin a new PHP document in your text editor or IDE, starting with the initial HTML (**Script 1.4**).

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//EN"
→ "http://www.w3.org/TR/xhtml1/DTD/
→ xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/
→ xhtml" xml:lang="en" lang="en">

<head>

    <meta http-equiv="content-type"
        content="text/html; charset=iso-
        8859-1" />

    <title>Comments</title>

</head>

<body>

```

2. Add the initial PHP tag and write your first comments.

```

<?php

# Created August 26, 2007

# Created by Larry E. Ullman

# This script does nothing much.

```

One of the first comments each script should contain is an introductory block that lists creation date, modification date, creator, creator's contact information, purpose of the script, and so on. Some people suggest that the shell-style comments (#) stand out more in a script and are therefore best for this kind of notation.

3. Send some HTML to the Web browser.

```

echo '<p>This is a line of text.
→ <br />This is another line of
→ text.</p>';

```

continues on next page

It doesn't matter what you do here, just so the Web browser has something to display. For the sake of variety, I'll have the `echo()` statement print some HTML tags, including a line break (`
`) to add some spacing to the generated HTML page.

4. Use the multiline comments to comment out a second `echo()` statement.

```
/*
echo 'This line will not be
→ executed.';
*/
```

By surrounding any block of PHP code with `/*` and `*/`, you can render that code inert without having to delete it from your script. By later removing the comment tags, you can reactivate that section of PHP code.

5. Add a final comment after a third `echo()` statement.

```
echo "<p>Now I'm done.</p>"; // End
→ of PHP code.
```

This last (superfluous) comment shows how to place one at the end of a line, a common practice. Note that I used double quotation marks to surround the message, as single quotation marks would conflict with the apostrophe (see the “Needing an Escape” sidebar, earlier in the chapter).

6. Close the PHP section and complete the HTML page.

```
?>
</body>
</html>
```

7. Save the file as `comments.php`, place it in your Web directory, and test it in your Web browser (**Figure 1.10**).

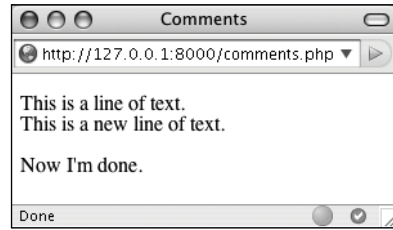


Figure 1.10 The PHP comments in Script 1.4 don't appear in the Web page or the HTML source (Figure 1.11).

8. If you're the curious type, check the source code in your Web browser to confirm that the PHP comments do not appear there (**Figure 1.11**).

✓ Tips

- You shouldn't nest (place one inside another) multiline comments (`/* */`). Doing so will cause problems.
- Any of the PHP comments can be used at the end of a line (say, after a function call):

```
echo 'Howdy'; /* Say 'Howdy' */
```

Although this is allowed, it's far less common.

- It's nearly impossible to over-comment your scripts. Always err on the side of writing too many comments as you code. That being said, in the interest of saving space, the scripts in this book will not be as well documented as I would suggest they should be.
- It's also important that as you change a script you keep the comments up-to-date and accurate. There's nothing more confusing than a comment that says one thing when the code really does something else.

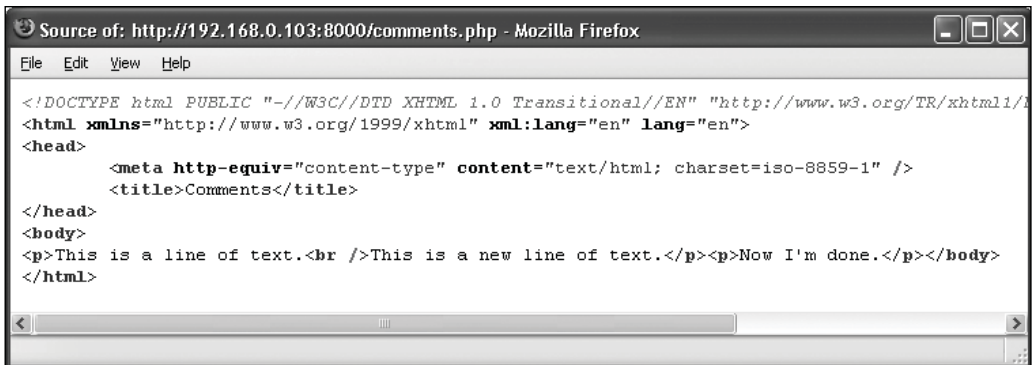


Figure 1.11 The PHP comments from Script 1.4 are nowhere to be seen in the client's browser.

What Are Variables?

Variables are containers used to temporarily store values. These values can be numbers, text, or much more complex data. PHP has eight types of variables. These include four scalar (single-valued) types—*Boolean* (`TRUE` or `FALSE`), *integer*, *floating point* (decimals), and *strings* (characters); two nonscalar (multivalued)—*arrays* and *objects*; plus *resources* (which you'll see when interacting with databases) and *NULL* (which is a special type that has no value).

Regardless of what type you are creating, all variables in PHP follow certain syntactical rules:

- ◆ A variable's name—also called its *identifier*—must start with a dollar sign (\$), for example, `$name`.
- ◆ The variable's name can contain a combination of strings, numbers, and the underscore, for example, `$my_report1`.
- ◆ The first character after the dollar sign must be either a letter or an underscore (it cannot be a number).
- ◆ Variable names in PHP are case-sensitive. This is a *very* important rule. It means that `$name` and `$Name` are entirely different variables.

To begin working with variables, let's make use of several predefined variables whose values are automatically established when a PHP script is run. Before getting into this script, there are two more things you should know. First, variables can be assigned values using the equals sign (=), also called the *assignment operator*. Second, variables can be printed without quotation marks:

```
print $some_var;
```

Script 1.5 This script prints three of PHP's many predefined variables.

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN" "http://www.w3.org/
  TR/xhtml1/DTD/xhtml1-transitional.dtd">

2  <html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">

3  <head>

4    <meta http-equiv="content-type" content=
    "text/html; charset=iso-8859-1" />

5    <title>Predefined Variables</title>

6  </head>

7  <body>

8    <?php # Script 1.5 - predefined.php

9

10   // Create a shorthand version of the
    variable names:

11   $file = $_SERVER['SCRIPT_FILENAME'];

12   $user = $_SERVER['HTTP_USER_AGENT'];

13   $server = $_SERVER['SERVER_
    SOFTWARE'];

14

15   // Print the name of this script:

16   echo "<p>You are running the file:<br
    /><b>$file</b>.</p>\n";

17

18   // Print the user's information:

19   echo "<p>You are viewing this page using:
    <br /><b>$user</b></p>\n";

20

21   // Print the server's information:

22   echo "<p>This server is running:<br /><b>
    $server</b>.</p>\n";

23

24   ?>

25 </body>

26 </html>

```

Or variables can be printed within double quotation marks:

```
print "Hello, $name";
```

You cannot print variables within single quotation marks:

```
print 'Hello, $name'; // Won't work!
```

To use variables:

1. Begin a new PHP document in your text editor or IDE, starting with the initial HTML (**Script 1.5**).

```

<!DOCTYPE html PUBLIC "-//W3C//
  DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">

<head>

  <meta http-equiv="content-type"
    → content="text/html; charset=
    → iso-8859-1" />

  <title>Predefined Variables</
    → title>

</head>

<body>

```

2. Add your opening PHP tag and your first comment.

```
<?php # Script 1.5 - predefined.php
```

From here on out, my scripts will no longer comment on the creator, creation date, and so forth, although you should continue to document your scripts thoroughly. I will, however, make a comment listing the script number and filename for ease of cross-referencing (both in

continues on next page

the book and when you download them from the book's supporting Web site, www.DMCInsights.com/phpmysql3/).

3. Create a shorthand version of the first variable to be used in this script.

```
$file = $_SERVER['SCRIPT_FILENAME'];
```

This script will use three variables, each of which comes from the larger and predefined `$_SERVER` variable. `$_SERVER` refers to a mass of server-related information. The first variable the script uses is `$_SERVER['SCRIPT_FILENAME']`. This variable stores the full path and name of the script being run (for example, `C:\Program Files\Apache\htdocs\predefined.php`).

The value stored in `$_SERVER['SCRIPT_FILENAME']` will be assigned to the new variable `$file`. Creating new variables with shorter names and then assigning them values from `$_SERVER` will make it easier to refer to the variables when printing them. (It also gets around some other issues you'll learn about in due time.)

4. Create a shorthand version of the other two variables.

```
$user = $_SERVER['HTTP_USER_AGENT'];
```

```
$server = $_SERVER['SERVER_
→ SOFTWARE'];
```

`$_SERVER['HTTP_USER_AGENT']` represents the Web browser and operating system of the user accessing the script. This value is assigned to `$user`.

`$_SERVER['SERVER_SOFTWARE']` represents the Web application on the server that's

running PHP (e.g., Apache, Abyss, Xitami, IIS). This is the program that must be installed (see Appendix A) in order to run PHP scripts on that computer.

5. Print out the name of the script being run.

```
echo "<p>You are running the file:
→ <br /><b>$file</b>.</p>\n";
```

The first variable to be printed is `$file`. Notice that this variable must be printed out within double quotation marks and that I also make use of the PHP newline (`\n`), which will add a line break in the generated HTML source. Some basic HTML tags—paragraph and bold—are added to give the generated page some flair.

6. Print out the information of the user accessing the script.

```
echo "<p>You are viewing this page
→ using:<br /><b>$user</b></p>\n";
```

This line prints the second variable, `$user`. To repeat what's said in the fourth step, `$user` correlates to `$_SERVER['HTTP_USER_AGENT']` and refers to the operating system, browser type, and browser version being used to access the Web page.

7. Print out the server information.

```
echo "<p>This server is running:<br
→ /><b>$server</b>.</p>\n";
```

8. Complete the HTML and PHP code.

```
?>
</body>
</html>
```

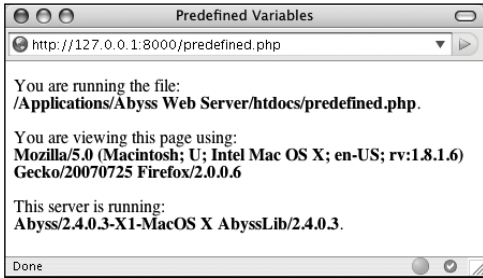


Figure 1.12 The predefined.php script reports back to the viewer information about the script, the Web browser being used to view it, and the server itself.

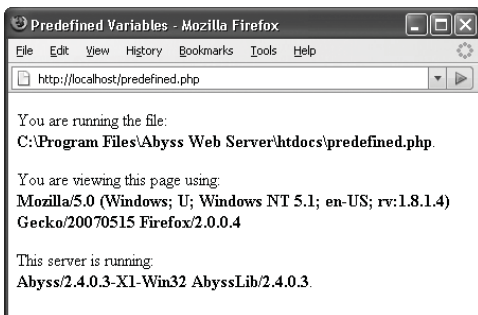


Figure 1.13 This is the book's first truly dynamic script, in that the Web page changes depending upon the server running it and the Web browser viewing it (compare with Figure 1.12).

9. Save your file as `predefined.php`, place it in your Web directory, and test it in your Web browser (**Figure 1.12**).

✓ Tips

- If you have problems with this, or any other script, turn to the book's corresponding Web forum (www.DMCInsights.com/phorum/) for assistance.
- If possible, run this script using a different Web browser and/or on another server (**Figure 1.13**).
- The most important consideration when creating variables is to use a consistent naming scheme. In this book you'll see that I use all-lowercase letters for my variable names, with underscores separating words (`$first_name`). Some programmers prefer to use capitalization instead: `$FirstName`.
- PHP is very casual in how it treats variables, meaning that you don't need to initialize them (set an immediate value) or declare them (set a specific type), and you can convert a variable among the many types without problem.

Introducing Strings

The first variable type to delve into is *strings*. A string is merely a quoted chunk of characters: letters, numbers, spaces, punctuation, and so forth. These are all strings:

- ◆ 'Tobias'
- ◆ "In watermelon sugar"
- ◆ '100'
- ◆ 'August 2, 2006'

To make a string variable, assign a string value to a valid variable name:

```
$first_name = 'Tobias';  
$today = 'August 2, 2006';
```

When creating strings, you can use either single or double quotation marks to encapsulate the characters, just as you would when printing text. Likewise, you must use the same type of quotation mark for the beginning and the end of the string. If that same mark appears within the string, it must be escaped:

```
$var = "Define \"platitude\", please.";
```

To print out the value of a string, use either `echo()` or `print()`:

```
echo $first_name;
```

To print the value of string within a context, use double quotation marks:

```
echo "Hello, $first_name";
```

You've already worked with strings once—when using the predefined variables in the preceding section. In this next example, you'll create and use new strings.

Script 1.6 String variables are created and their values sent to the Web browser in this introductory script.

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN" "http://www.w3.org/
  TR/xhtml1/DTD/xhtml1-transitional.dtd">

2  <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">

3  <head>

4  <meta http-equiv="content-type" content=
  "text/html; charset=iso-8859-1" />

5  <title>Strings</title>

6  </head>

7  <body>

8  <?php # Script 1.6 - strings.php

9

10 // Create the variables:

11 $first_name = 'Haruki';

12 $last_name = 'Murakami';

13 $book = 'Kafka on the Shore';

14

15 //Print the values:

16 echo "<p>The book <em>$book</em> was
  written by $first_name $last_name.</p>";

17

18 ?>

19 </body>

20 </html>

```

To use strings:

1. Begin a new PHP document in your text editor or IDE, starting with the initial HTML and including the opening PHP tag (**Script 1.6**).

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//EN"
→ "http://www.w3.org/TR/xhtml1/DTD/
→ xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/
→ xhtml" xml:lang="en" lang="en">

<head>

  <meta http-equiv="content-type"
  → content="text/html; charset=
  → iso-8859-1" />

  <title>Strings</title>

</head>

<body>

  <?php # Script 1.6 - strings.php

```

2. Within the PHP tags, create three variables.

```

$first_name = 'Haruki';
$last_name = 'Murakami';
$book = 'Kafka on the Shore';

```

This rudimentary example creates `$first_name`, `$last_name`, and `$book` variables that will then be printed out in a message.

3. Add an `echo()` statement.

```

echo "<p>The book <em>$book</em>
→ was written by $first_name
→ $last_name.</p>";

```

continues on next page

All this script does is print a statement of authorship based upon three established variables. A little HTML formatting (the emphasis on the book's title) is thrown in to make it more attractive. Remember to use double quotation marks here for the variable values to be printed out appropriately (more on the importance of double quotation marks at the chapter's end).

4. Complete the HTML and PHP code.

```
?>
</body>
</html>
```

5. Save the file as `strings.php`, place it in your Web directory, and test it in your Web browser (**Figure 1.14**).
6. If desired, change the values of the three variables, save the file, and run the script again (**Figure 1.15**).

✓ Tips

- If you assign another value to an existing variable (say `$book`), the new value will overwrite the old one. For example:


```
$book = 'High Fidelity';
$book = 'The Corrections';
/* $book now has a value of
'The Corrections'. */
```
- PHP has no set limits on how big a string can be. It's theoretically possible that you'll be limited by the resources of the server, but it's doubtful that you'll ever encounter such a problem.

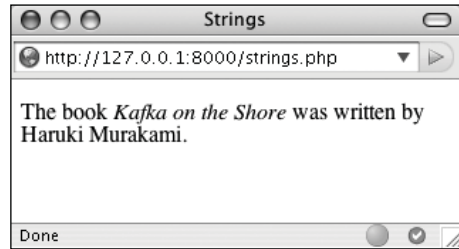


Figure 1.14 The resulting Web page is based upon printing out the values of three variables.



Figure 1.15 The output of the script is changed by altering the variables in it.

Script 1.7 Concatenation gives you the ability to easily manipulate strings, like creating an author's name from the combination of their first and last names.

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN" "http://www.w3.org/
  TR/xhtml1/DTD/xhtml1-transitional.dtd">

2  <html
  xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">

3  <head>

4  <meta http-equiv="content-type" content=
  "text/html; charset=iso-8859-1" />

5  <title>Concatenation</title>

6  </head>

7  <body>

8  <?php # Script 1.7 - concat.php

9

10 // Create the variables:

11 $first_name = 'Melissa';

12 $last_name = 'Bank';

13 $author = $first_name . ' ' . $last_name;

14

15 $book = 'The Girls\' Guide to Hunting and
  Fishing';

16

17 //Print the values:

18 echo "<p>The book <em>$book</em> was
  written by $author.</p>";

19

20 ?>

21 </body>

22 </html>

```

Concatenating Strings

Concatenation is like addition for strings, whereby characters are added to the end of the string. It's performed using the *concatenation operator*, which is the period (.):

```

$city= 'Seattle';

$state = 'Washington';

$address = $city . $state;

```

The `$address` variable now has the value *SeattleWashington*, which almost achieves the desired result (*Seattle, Washington*). To improve upon this, you could write

```
$address = $city . ', ' . $state;
```

so that a comma and a space are added to the mix.

Concatenation works with strings or numbers. Either of these statements will produce the same result (*Seattle, Washington 98101*):

```

$address = $city . ', ' . $state .
    ' 98101';

$address = $city . ', ' . $state .
    ' ' . 98101;

```

Let's modify `strings.php` to use this new operator.

To use concatenation:

1. Open `strings.php` (refer to Script 1.6) in your text editor or IDE.
2. After you've established the `$first_name` and `$last_name` variables (lines 11 and 12), add this line (**Script 1.7**):


```
$author = $first_name . ' ' .
    $last_name;
```

continues on next page

As a demonstration of concatenation, a new variable—`$author`—will be created as the concatenation of two existing strings and a space in between.

3. Change the `echo()` statement to use this new variable.

```
echo "<p>The book <em>$book</em> was  
→ written by $author.</p>";
```

Since the two variables have been turned into one, the `echo()` statement should be altered accordingly.

4. If desired, change the HTML page title and the values of the first name, last name, and book variables.
5. Save the file as `concat.php`, place it in your Web directory, and test it in your Web browser (**Figure 1.16**).

✓ Tips

- PHP has a slew of useful string-specific functions, which you'll see over the course of this book. For example, to calculate how long a string is (how many characters it contains), use `strlen()`:

```
$num = strlen('some string');
```
- You can have PHP convert the case of strings with: `strtolower()`, which makes it entirely lowercase; `strtoupper()`, which makes it entirely uppercase; `ucfirst()`, which capitalizes the first character; and `ucwords()`, which capitalizes the first character of every word.

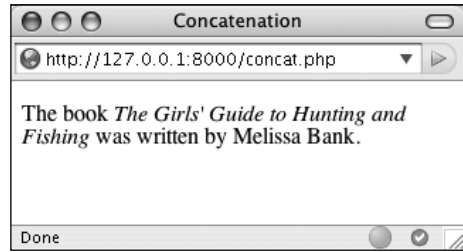


Figure 1.16 In this revised script, the end result of concatenation is not apparent to the user (compare with Figures 1.14 and 1.15).

- If you are merely concatenating one value to another, you can use the *concatenation assignment operator* (`.=`). The following are equivalent:

```
$title = $title . $subtitle;  
$title .= $subtitle;
```
- The initial example in this section could be rewritten using either

```
$address = "$city, $state";  
or  
$address = $city;  
$address .= ', ';  
$address .= $state;
```

Introducing Numbers

In introducing variables, I was explicit in stating that PHP has both integer and floating-point (decimal) number types. In my experience, though, these two types can be classified under the generic title *numbers* without losing any valuable distinction (for the most part). Valid number-type variables in PHP can be anything like

- ◆ 8
- ◆ 3.14
- ◆ 10980843985
- ◆ -4.2398508
- ◆ 4.4e2

Notice that these values are never quoted—in which case they'd be strings with numeric values—nor do they include commas to indicate thousands. Also, a number is assumed to be positive unless it is preceded by the minus sign (-).

Along with the standard arithmetic operators you can use on numbers (**Table 1.1**), there are dozens of functions. Two common ones are `round()` and `number_format()`.

The former rounds a decimal to the nearest integer:

```
$n = 3.14;
$n = round($n); // 3
```

It can also round to a specified number of decimal places:

```
$n = 3.142857;
$n = round($n, 3); // 3.143
```

The `number_format()` function turns a number into the more commonly written version, grouped into thousands using commas:

```
$n = 20943;
$n = number_format($n); // 20,943
```

This function can also set a specified number of decimal points:

```
$n = 20943;
$n = number_format($n, 2); // 20,943.00
```

To practice with numbers, let's write a mock-up script that performs the calculations one might use in an e-commerce shopping cart.

TABLE 1.1 The standard mathematical operators.

Arithmetic Operators	
OPERATOR	MEANING
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

To use numbers:

1. Begin a new PHP document in your text editor or IDE (**Script 1.8**).

```
<!DOCTYPE html PUBLIC "-//W3C//
    DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/
    xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/
    xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="content-type"
        → content="text/html; charset=
        → iso-8859-1" />
    <title>Numbers</title>
</head>
<body>
<?php # Script 1.8 - numbers.php
```

2. Establish the requisite variables.

```
$quantity = 30;
$price = 119.95;
$taxrate = .05;
```

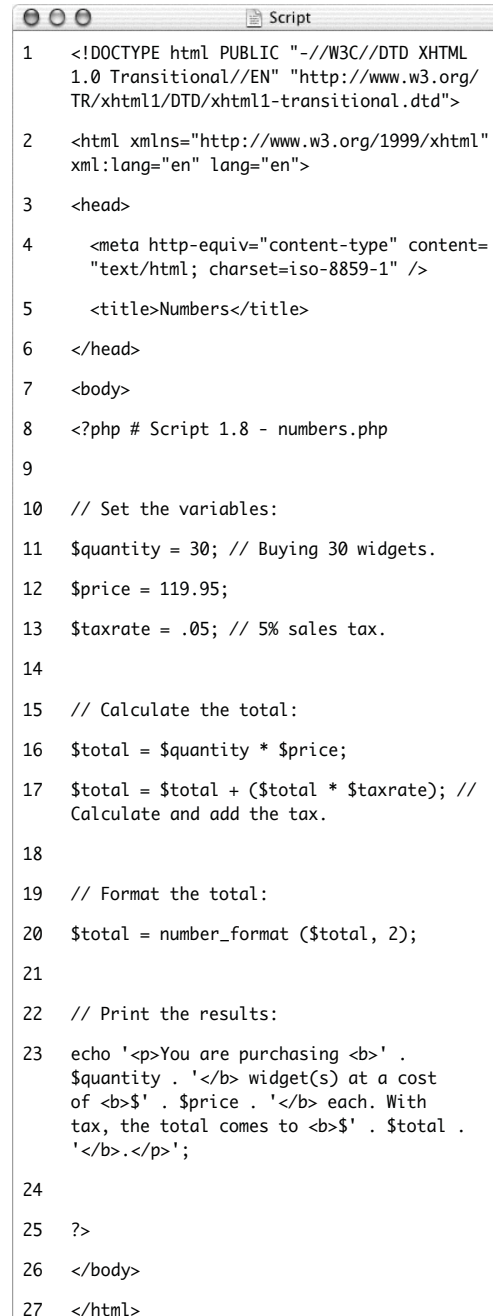
This script will use three hard-coded variables upon which calculations will be made. Later in the book, you'll see how these values can be dynamically determined (i.e., by user interaction with an HTML form).

3. Perform the calculations.

```
$total = $quantity * $price;
$total = $total + ($total * $taxrate);
```

The first line establishes the order total as the number of widgets purchased multiplied by the price of each widget.

Script 1.8 The numbers.php script demonstrates basic mathematical calculations, like those used in an e-commerce application.



```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN" "http://www.w3.org/
  TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">
3 <head>
4     <meta http-equiv="content-type" content=
      "text/html; charset=iso-8859-1" />
5     <title>Numbers</title>
6 </head>
7 <body>
8 <?php # Script 1.8 - numbers.php
9
10 // Set the variables:
11 $quantity = 30; // Buying 30 widgets.
12 $price = 119.95;
13 $taxrate = .05; // 5% sales tax.
14
15 // Calculate the total:
16 $total = $quantity * $price;
17 $total = $total + ($total * $taxrate); //
  Calculate and add the tax.
18
19 // Format the total:
20 $total = number_format ($total, 2);
21
22 // Print the results:
23 echo '<p>You are purchasing <b>' .
  $quantity . '</b> widget(s) at a cost
  of <b>$' . $price . '</b> each. With
  tax, the total comes to <b>$' . $total .
  '</b>.</p>';
24
25 ?>
26 </body>
27 </html>
```

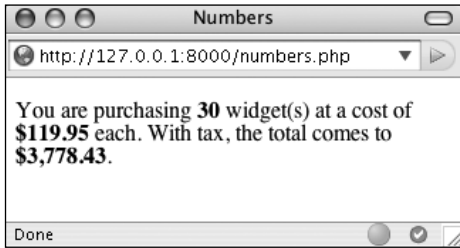


Figure 1.17 The numbers PHP page (Script 1.8) performs calculations based upon set values.



Figure 1.18 To change the generated Web page, alter any or all of the three variables (compare with Figure 1.17).

The second line then adds the amount of tax to the total (calculated by multiplying the tax rate by the total).

4. Format the total.

```
$total = number_format ($total, 2);
```

The `number_format()` function will group the total into thousands and round it to two decimal places. This will make the display more appropriate to the end user.

5. Print the results.

```
echo '<p>You are purchasing <b>' .  
→ $quantity . '</b> widget(s) at a cost  
→ of <b>$' . $price . '</b> each. With  
→ tax, the total comes to <b>$' .  
→ $total . '</b>.</p>';
```

The last step in the script is to print out the results. To use a combination of HTML, printed dollar signs, and variables, the `echo()` statement uses both single-quoted text and concatenated variables. You could also put this all within a double-quoted string (as in previous examples), but when PHP encounters, for example, at a cost of `$price` in the `echo()` statement, the double dollar sign would cause problems. You'll see an alternative solution in the last example of this chapter.

6. Complete the PHP code and the HTML page.

```
?>  
  
</body>  
  
</html>
```

7. Save the file as `numbers.php`, place it in your Web directory, and test it in your Web browser (**Figure 1.17**).

8. If desired, change the initial three variables and rerun the script (**Figure 1.18**).

continues on next page

✓ Tips

- PHP supports a maximum integer of around two billion on most platforms. With numbers larger than that, PHP will automatically use a floating-point type.
- When dealing with arithmetic, the issue of precedence arises (the order in which complex calculations are made). While the PHP manual and other sources tend to list out the hierarchy of precedence, I find programming to be safer and more legible when I group clauses in parentheses to force the execution order (see line 17 of Script 1.8).
- Computers are notoriously poor at dealing with decimals. For example, the number 2.0 may actually be stored as 1.99999. Most of the time this won't be a problem, but in cases where mathematical precision is paramount, rely on integers, not decimals. The PHP manual has information on this subject, as well as alternative functions for improving computational accuracy.
- Many of the mathematical operators also have a corresponding assignment operator, letting you create a shorthand for assigning values. This line,

```
$total = $total + ($total *  
$taxrate);
```

could be rewritten as

```
$total += ($total * $taxrate);
```
- If you set a `$price` value without using two decimals (e.g., 119.9 or 34), you would want to apply `number_format()` to `$price` before printing it.

Introducing Constants

Constants, like variables, are used to temporarily store a value, but otherwise, constants and variables differ in many ways. For starters, to create a constant, you use the `define()` function instead of the assignment operator (`=`):

```
define ('NAME', 'value');
```

Notice that, as a rule of thumb, constants are named using all capitals, although this is not required. Most importantly, constants do not use the initial dollar sign as variables do (because constants are not variables).

A constant can only be assigned a scalar value, like a string or a number. And unlike variables, a constant's value cannot be changed.

To access a constant's value, like when you want to print it, you cannot put the constant within quotation marks:

```
echo "Hello, USERNAME"; // Won't work!
```

With that code, PHP would literally print *Hello, USERNAME* and not the value of the `USERNAME` constant (because there's no indication that `USERNAME` is anything other than literal text). Instead, either print the constant by itself:

```
echo 'Hello, ';
```

```
echo USERNAME;
```

or use the concatenation operator:

```
echo 'Hello, ' . USERNAME;
```

PHP runs with several predefined constants, much like the predefined variables used earlier in the chapter. These include `PHP_VERSION` (the version of PHP running) and `PHP_OS` (the operating system of the server).

To use constants:

1. Begin a new PHP document in your text editor or IDE (**Script 1.9**).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
→ "http://www.w3.org/TR/xhtml1/
→ DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/
→ xhtml" xml:lang="en" lang="en">

<head>

  <meta http-equiv="content-type"
  → content="text/html; charset=
  → iso-8859-1" />

  <title>Constants</title>

</head>

<body>

<?php # Script 1.9 - constants.php
```

2. Create a new date constant.

```
define ('TODAY', August 28, 2007');

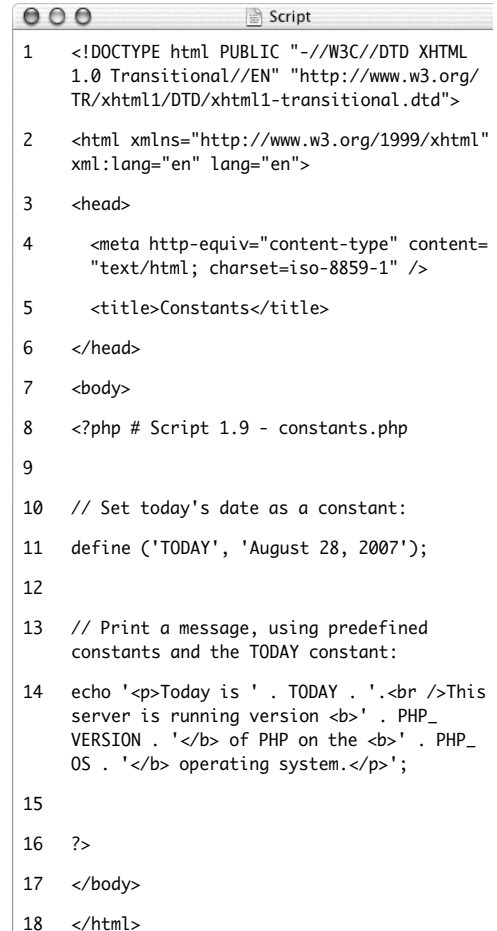
An admittedly trivial use of constants,
but this example will illustrate the point.
In Chapter 8, "Using PHP with MySQL,"
you'll see how to use constants to store
your database access information.
```

3. Print out the date, the PHP version, and operating system information.

```
echo '<p>Today is ' . TODAY . ' .<br
→ />This server is running version
→ <b>' . PHP_VERSION . ' </b> of PHP
→ on the <b>' . PHP_OS . ' </b>
→ operating system.</p>';
```

Since constants cannot be printed within quotation marks, use the concatenation operator to create the echo() statement.

Script 1.9 Constants are another temporary storage tool you can use in PHP, distinct from variables.



```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN" "http://www.w3.org/
  TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">
3 <head>
4   <meta http-equiv="content-type" content=
    "text/html; charset=iso-8859-1" />
5   <title>Constants</title>
6 </head>
7 <body>
8 <?php # Script 1.9 - constants.php
9
10 // Set today's date as a constant:
11 define ('TODAY', 'August 28, 2007');
12
13 // Print a message, using predefined
  constants and the TODAY constant:
14 echo '<p>Today is ' . TODAY . ' .<br />This
  server is running version <b>' . PHP_
  VERSION . ' </b> of PHP on the <b>' . PHP_
  OS . ' </b> operating system.</p>';
15
16 ?>
17 </body>
18 </html>
```

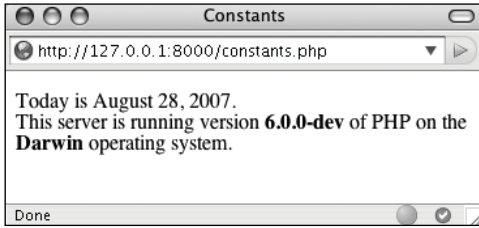


Figure 1.19 By making use of PHP's constants, you can learn more about your PHP setup.



Figure 1.20 Running the same script (refer to Script 1.9) on different servers garners different results.

4. Complete the PHP code and the HTML page.

```
?>
```

```
</body>
```

```
</html>
```

5. Save the file as `constants.php`, place it in your Web directory, and test it in your Web browser (**Figure 1.19**).

✓ Tips

- If possible, run this script on another PHP-enabled server (**Figure 1.20**).
- In Chapter 11, “Cookies and Sessions,” you’ll learn about another constant, `SID` (which stands for *session ID*).

Single vs. Double Quotation Marks

In PHP it's important to understand how single quotation marks differ from double quotation marks. With `echo()` and `print()`, or when assigning values to strings, you can use either, as in the examples uses so far. But there is a key difference between the two types of quotation marks and when you should use which. I've introduced this difference already, but it's an important enough concept to merit more discussion.

In PHP, values enclosed within single quotation marks will be treated literally, whereas those within double quotation marks will be interpreted. In other words, placing variables and special characters (**Table 1.2**) within double quotes will result in their represented values printed, not their literal values. For example, assume that you have

```
$var = 'test';
```

The code `echo "var is equal to $var";` will print out *var is equal to test*, whereas the code `echo 'var is equal to $var';` will print out *var is equal to \$var*. Using an escaped dollar sign, the code `echo "\$var is equal to $var";` will print out *\$var is equal to test*, whereas the code `echo '\$var is equal to $var';` will print out *\\$var is equal to \$var*.

As these examples should illustrate, double quotation marks will replace a variable's name (`$var`) with its value (*test*) and a special character's code (`\$`) with its represented value (*\$*). Single quotes will always display exactly what you type, except for the escaped single quote (`\'`) and the escaped backslash (`\\`), which are printed as a single quotation mark and a single backslash, respectively.

As another example of how the two quotation marks differ, let's modify the `numbers.php` script as an experiment.

TABLE 1.2 These characters have special meanings when used within double quotation marks.

Escape Sequences	
CODE	MEANING
\"	Double quotation mark
\'	Single quotation mark
\\	Backslash
\n	Newline
\r	Carriage return
\t	Tab
\\$	Dollar sign

Script 1.10 This, the final script in the chapter, demonstrates the differences between using single and double quotation marks.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN" "http://www.w3.org/
  TR/xhtml1/DTD/xhtml1-transitional.dtd">

2 <html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">

3 <head>

4   <meta http-equiv="content-type" content=
     "text/html; charset=iso-8859-1" />

5   <title>Quotation Marks</title>

6 </head>

7 <body>

8 <?php # Script 1.10 - quotes.php

9

10 // Set the variables:

11 $quantity = 30; // Buying 30 widgets.

12 $price = 119.95;

13 $taxrate = .05; // 5% sales tax.

14

15 // Calculate the total.

16 $total = $quantity * $price;

17 $total = $total + ($total * $taxrate); //
  Calculate and add the tax.

18

19 // Format the total:

20 $total = number_format ($total, 2);

21

22 // Print the results using double quotation
  marks:

23 echo '<h3>Using double quotation
  marks:</h3>';

24 echo "<p>You are purchasing <b>$quantity
  </b> widget(s) at a cost of <b>\$price</b>
  each. With tax, the total comes to <b>\$
  $total</b>.</p>\n";

25

26 // Print the results using single quotation
  marks:

27 echo '<h3>Using single quotation
  marks:</h3>';
  
```

(script continues)

To use single and double quotation marks:

1. Open `numbers.php` (refer to Script 1.8) in your text editor or IDE.
2. Delete the existing `echo()` statement (**Script 1.10**).
3. Print a caption and then rewrite the original `echo()` statement using double quotation marks.

```
echo '<h3>Using double quotation
→ marks:</h3>';
```

```
echo "<p>You are purchasing <b>$
→ quantity</b> widget(s) at a cost
→ of <b>\$price</b> each. With tax,
→ the total comes to <b>\$total</
→ b>.</p>\n";
```

In the original script, the results were printed using single quotation marks and concatenation. The same result can be achieved using double quotation marks. When using double quotation marks, the variables can be placed within the string. There is one catch, though: trying to print a dollar amount as `$12.34` (where `12.34` comes from a variable) would suggest that you would code `$$var`. That will not work; instead, escape the initial dollar sign, resulting in `\$var`, as you see

continues on next page

Script 1.10 continued

```

28 echo '<p>You are purchasing <b>$quantity
  </b> widget(s) at a cost of <b>\$price</b>
  each. With tax, the total comes to
  <b>\$total</b>.</p>\n';

29

30 ?>

31 </body>

32 </html>
  
```

twice in this code. The first dollar sign will be printed, and the second becomes the start of the variable name.

4. Repeat the `echo()` statements, this time using single quotation marks.

```
echo '<h3>Using single quotation
marks:</h3>';
```

```
echo '<p>You are purchasing <b>$
→ quantity</b> widget(s) at a cost
→ of <b>\$price</b> each. With tax,
→ the total comes to <b>\$total
→ </b>.</p>\n';
```

This `echo()` statement is used to highlight the difference between using single or double quotation marks. It will not work as desired, and the resulting page will show you exactly what does happen instead.

5. If you want, change the page's title.
6. Save the file as `quotes.php`, place it in your Web directory, and test it in your Web browser (**Figure 1.21**).
7. View the source of the Web page to see how using the newline character (`\n`) within each quotation mark type also differs.

You should see that when you place the newline character within double quotation marks it creates a newline in the HTML source. When placed within single quotation marks, the literal characters `\` and `n` are printed instead.

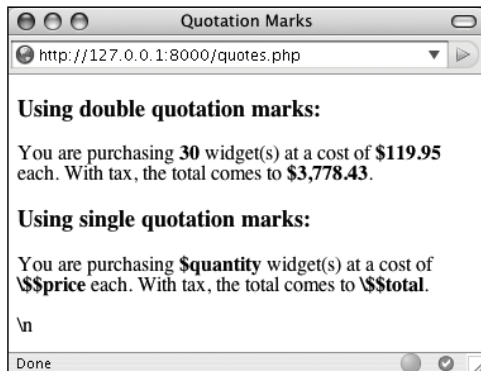


Figure 1.21 These results demonstrate when and how you'd use one type of quotation mark as opposed to the other. If you're still unclear as to the difference between the types, use double quotation marks and you're less likely to have problems.

✓ Tips

- Because PHP will attempt to find variables within double quotation marks, using single quotation marks is theoretically faster. If you need to print the value of a variable, though, you *must* use double quotation marks.
- As valid HTML often includes a lot of double-quoted attributes, it's often easiest to use single quotation marks when printing HTML with PHP:

```
echo '<table width="80%" border="0"
→ cellspacing="2" cellpadding="3"
→ align="center">';
```

If you were to print out this HTML using double quotation marks, you would have to escape all of the double quotation marks in the string:

```
echo "<table width=\"80%\" border=\\
→ \"0\" cellspacing=\"2\" cellpadding
→ =\"3\" align=\"center\">";
```

PROGRAMMING WITH PHP

2

Now that you have the fundamentals of the PHP scripting language down, it's time to build on those basics and start truly programming. In this chapter you'll begin creating more elaborate scripts while still learning some of the standard constructs, functions, and syntax of the language.

You'll begin by creating an HTML form, then learning how you can use PHP to handle the submitted values. From there, the chapter covers conditionals and the remaining operators (Chapter 1, "Introduction to PHP" presented the assignment, concatenation, and mathematical operators), arrays (another variable type), and one last language construct, loops.