

With dynamic web pages the core layout of the Web page can remain the same (where the logo, links, and so forth display), but the data presented constantly changes.

Dynamic Web pages use a scripting language embedded within the HTML code.

The scripting language produces code that can change with each visit to the Web page.

# 2

## Variables

In the previous chapter, you learned how to use PHP to send simple text and HTML to a Web browser—in other words, something for which you don't need PHP at all! Don't worry, though; this book will teach you how to use **print** in conjunction with other PHP features to do great and useful things with your Web site.

To make the leap from creating simple, static pages to dynamic Web applications and interactive Web sites, you need *variables*. Understanding what variables are, the types that a language supports, and how to use them is critical.

This chapter discusses the fundamentals of variables used in PHP, and later chapters cover the different types in greater detail. If you've never dealt with variables before, this chapter will be a good introduction. If you're familiar with the concept, then you should be able to work through this chapter with ease.

---

### In This Chapter

What Are Variables?	32
Variable Syntax	36
Types of Variables	38
Variable Values	41
Understanding Quotation Marks	45
Review and Pursue	48

---

There are some differences between **echo** and **print** statement:

**echo** - can output one or more strings

**print** - can only output one string

**Tip:** **echo** is marginally faster compared to **print** as **echo** does not return any value.

[http://www.w3schools.com/php/php\\_echo\\_print.asp](http://www.w3schools.com/php/php_echo_print.asp)

## What Are Variables?

A *variable* is a container for data. Once data has been stored in a variable (or, stated more accurately, once a variable has been assigned a value), that data can be altered, printed to the Web browser, saved to a database, emailed, and so forth.

Variables in PHP are, by their nature, flexible: You can put data into a variable, retrieve that data from it (without affecting the value of the variable), put new data in, and continue this cycle as long as necessary. But variables in PHP are largely temporary: *Most only exist*—that is, they only have a value—for the duration of the script's execution on the server. Once the execution passes the final closing PHP tag, those variables cease to exist. Furthermore, after users click a link or submit a form, they are taken to a new page that may have an entirely separate set of variables.

Before getting too deep into the discussion of variables, let's write a quick script that reveals some of PHP's *predefined* variables. These are variables that PHP automatically creates when a script runs. Over the course of the book you'll be introduced to many different predefined variables. For this particular example, let's look at the predefined **\$\_SERVER** variable. It contains lots of information about the computer on which PHP is running.

The **print\_r()** function offers an easy way to display any variable's value:

```
print_r($variable_name);
```

Just provide the name of the variable you'd like to inspect as a single argument to the **print\_r()** function (you'll learn more about a variable's syntax throughout this chapter).

**\$\_SERVER** is a Superglobal Variable (predefined variable). They are accessible everywhere in a PHP program.

**\$\_SERVER** contains information such as headers, paths, and script locations.

**Script 2.1** The `print_r()` function is called to show the values stored in the `$_SERVER` predefined variable.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">
4 <head>
5 <meta http-equiv="Content-Type"
  content="text/html; charset=utf-8"/>
6 <title>Predefined Variables</title>
7 </head>
8 <body>
9 <pre>
10 <?php // Script 2.1 - predefined.php
11
12 // Show the value of the $_SERVER
  variable:
13 print_r($_SERVER);
14
15 ?>
16 </pre>
17 </body>
18 </html>
```

**<pre>** tag defines preformatted text. It preserves both spaces and line breaks.

**print\_r** is used to return an array in a human readable format in PHP.

## To print PHP's predefined variables:

1. Create a new PHP script in your text editor or IDE, to be named **predefined.php** (Script 2.1).
2. Create the initial HTML tags:

```
<!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=utf-8"/>
  <title>Predefined Variables
  → </title>
</head>
<body>
<pre>
```

This code repeats the XHTML template created in the preceding chapter. Within the body of the page, the **<pre>** tags are being used to make the generated PHP information more legible. Without using the **<pre>** tags, the **print\_r()** function's output would be quite messy.

3. Add the PHP code:

```
<?php // Script 2.1 - predefined.php
print_r($_SERVER);
?>
```

The PHP code contains just one function call. The function should be provided with the name of a variable.

*continues on next page*

In this example, the variable is `$_SERVER`, which is special in PHP. `$_SERVER` stores all sorts of data about the server: its name and operating system, the name of the current user, information about the Web server application (Apache, Abyss, IIS, etc.), and more. It also reflects the PHP script being executed: its name, where it's stored on the server, and so forth.

Note that you must type `$_SERVER` exactly as it is here, in all uppercase letters.

4. Complete the HTML page:

```
</pre>
</body>
</html>
```

5. Save the file as **predefined.php**, upload it to your server (or save it to the appropriate directory on your computer), and test it in your Web browser **A**.

Once again, remember that you must run all PHP scripts through a URL (i.e., `http://something`).

Script 2.1 (predefined.php) should NOT be posted on your external web server (nor the school's). Only view it locally. For security reasons, don't upload it for public viewing.

```
Array
(
    [HTTP_HOST] => phpvqs4:8088
    [HTTP_USER_AGENT] => Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6;
    [HTTP_ACCEPT] => text/html,application/xhtml+xml,application/xml;q=0
    [HTTP_ACCEPT_LANGUAGE] => en-us,en;q=0.5
    [HTTP_ACCEPT_ENCODING] => gzip,deflate
    [HTTP_ACCEPT_CHARSET] => ISO-8859-1,utf-8;q=0.7,*;q=0.7
    [HTTP_KEEP_ALIVE] => 115
    [HTTP_CONNECTION] => keep-alive
    [PATH] => /usr/bin:/bin:/usr/sbin:/sbin
    [SERVER_SIGNATURE] =>
    [SERVER_SOFTWARE] => Apache
    [SERVER_NAME] => phpvqs4
    [SERVER_ADDR] => 127.0.0.1
    [SERVER_PORT] => 8088
    [REMOTE_ADDR] => 127.0.0.1
    [DOCUMENT_ROOT] => /Users/larryullman/Sites/phpvqs4
    [SERVER_ADMIN] => you@example.com
    [SCRIPT_FILENAME] => /Users/larryullman/Sites/phpvqs4/predefined.php
    [REMOTE_PORT] => 51149
    [GATEWAY_INTERFACE] => CGI/1.1
    [SERVER_PROTOCOL] => HTTP/1.1
    [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
    [REQUEST_URI] => /predefined.php
    [SCRIPT_NAME] => /predefined.php
    [PHP_SELF] => /predefined.php
    [REQUEST_TIME] => 1269831848
    [argv] => Array
        (
        )
    [argc] => 0
)
```

**A** The `$_SERVER` variable, as printed out by this script, is a master list of values pertaining to the server and the PHP script.

6. If possible, transfer the file to another computer or server running PHP and execute the script in your Web browser again **B**.

**TIP** Printing out the value of any variable as you've done here is one of the greatest debugging tools. Scripts often don't work as you expect them to because one or more variables do not have the values you assume they should, so confirming their actual values is extremely helpful.

**TIP** If you don't use the HTML `<pre></pre>` tags, the result will be like the mess in **C**.

```
Array
(
    [HTTP_HOST] => larryullman.com
    [HTTP_USER_AGENT] => Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_5; en-us
    [HTTP_ACCEPT] => application/xml,application/xhtml+xml,text/html;q=0.9,text/
    [HTTP_ACCEPT_LANGUAGE] => en-us
    [HTTP_ACCEPT_ENCODING] => gzip, deflate
    [HTTP_CONNECTION] => keep-alive
    [PATH] => /sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin
    [SERVER_SIGNATURE] =>
    Apache/2.0.63 (CentOS) Server at larryullman.com Port 80

    [SERVER_SOFTWARE] => Apache/2.0.63 (CentOS)
    [SERVER_NAME] => larryullman.com
    [SERVER_ADDR] => 207.58.187.78
    [SERVER_PORT] => 80
    [REMOTE_ADDR] => 71.59.97.51
    [DOCUMENT_ROOT] => httpdocs
    [SERVER_ADMIN] => Larry@DMCinsights.com
    [SCRIPT_FILENAME] => httpdocs/predefined.php
    [REMOTE_PORT] => 44766
    [GATEWAY_INTERFACE] => CGI/1.1
    [SERVER_PROTOCOL] => HTTP/1.1
    [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
    [REQUEST_URI] => /predefined.php
    [SCRIPT_NAME] => /predefined.php
    [PHP_SELF] => /predefined.php
    [REQUEST_TIME] => 1289832083
)
```

**B** With the `predefined.php` page, different servers will generate different results (compare with Figure **A**).

```
Array ( [HTTP_HOST] => phpvqs4:8888 [HTTP_USER_AGENT] => Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv:1.9.2.12) Gecko
/*;q=0.8 [HTTP_ACCEPT_LANGUAGE] => en-us,en;q=0.5 [HTTP_ACCEPT_ENCODING] => gzip,deflate [HTTP_ACCEPT_CHARSET] => I
[HTTP_CACHE_CONTROL] => max-age=0 [PATH] => /usr/bin:/bin:/usr/sbin:/sbin [SERVER_SIGNATURE] => [SERVER_SOFTWARE] => Ap
[REMOTE_ADDR] => 127.0.0.1 [DOCUMENT_ROOT] => /Users/larryullman/Sites/phpvqs4 [SERVER_ADMIN] => you@example.com [SCRIP
[GATEWAY_INTERFACE] => CGI/1.1 [SERVER_PROTOCOL] => HTTP/1.1 [REQUEST_METHOD] => GET [QUERY_STRING] => [REQUE
[REQUEST_TIME] => 1289832176 [argv] => Array ( ) [argc] => 0 )
```

**C** With large, complex variables such as `$SERVER`, not using the HTML preformatting tags with `print_r()` creates an incomprehensible mess (compare to Figures **A** and **B**).

## Variable Syntax

Now that you've had a quick dip in the variable pool, it's time to investigate the subject further. In the preceding example, the script reported upon PHP's predefined `$_SERVER` variable. You can also create your own variables, once you understand the proper syntax. To create appropriate variable names, you must follow these rules:

- All variable names must be preceded by a dollar sign (`$`).
- Following the dollar sign, the variable name must begin with either a letter (A–Z, a–z) or an underscore (`_`). It can't begin with a number.
- The rest of the variable name can contain any combination of letters, underscores, and numbers.
- You may not use spaces within the name of a variable. (Instead, the underscore is commonly used to separate words.)
- Each variable must have a unique name.
- Variable names are *case-sensitive*! Consequently, `$variable` and `$Variable` are two different constructs, and it would be a bad idea to use two variables with such similar names.

This last point is perhaps the most important: variable names in PHP are case-sensitive. Using the wrong letter case is a very common cause of bugs. (If you used, for example, `$_server` or `$_Server` in the previous script, you'd see either an error message or nothing at all **A**.)

A variable is a symbolic representation of a value. You can think of it as a symbol that refers to something and it will make more sense once you actually start using them.

As its name suggests, it can change over time or vary. It has a variable value because it can point to different values.

One of the most important features of a programming language like PHP is the use of variables to store values that might change or aren't known in advance. You use variables all the time, without even realizing it.

For example, what's your name? My name's Marianne, but my dog's name is Molly. The word name is a real-life variable.

Name always remains the same, but its value changes.

```
Notice: Undefined variable: _server in /Users/larryullman/s
```

**A** Misspelling a variable's name, including its case, will create und

**Script 2.2** Properly documenting the purposes of variables, along with using meaningful names, is a hallmark of a professional programmer.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="Content-Type"
6     content="text/html; charset=utf-8"/>
7   <title>Variables and Comments</title>
8 </head>
9 <body>
10  <?php // Script 2.2
11    // Define my variables....
12
13    $year = 2011; // The current year.
14    $june_avg = 88; // The average
15    temperature for the month of June.
16    $page_title = 'Weather Reports';
17    // A title for the page.
18
19    // ... and so forth.
20    print gettype($year) . '<br>';
21  </body>
22 </html>
```

**TABLE 2.1** Valid Variables in PHP

Name
\$first_name
\$person
\$address1
\$_SERVER

**TABLE 2.2** Invalid Variables in PHP

Name	Reason
\$first name	Has a space
\$first.name	Has a period
first_name	Does not begin with \$
\$1address	A number cannot follow \$

To help minimize bugs, I recommend the following policies:

- Always use all lowercase variable names.
- Make your variable names descriptive (e.g., `$first_name` is better than `$fn`).
- Use comments to indicate the purpose of variables (**Script 2.2**), redundant as that may seem.
- Above all, be consistent with whatever naming convention you choose!

**Table 2.1** lists some sample valid variables; **Table 2.2** lists some invalid variables and the rules they violate.

**TIP** Unlike some other languages, PHP generally doesn't require you to *declare* or *initialize* a variable prior to use. In other words, you *can* refer to variables without first defining them. But it's best not to do that; I try to write my scripts so that every variable is defined or validated before use.

**TIP** There are two main variable naming conventions, determined by how you delineate words. These are the so-called *camel-hump* or *camel-case* (named because of the way capital letters break up the word—for example, `$FirstName`) and *underscore* (`$first_name`) styles. This book uses the latter convention.

`print gettype($year) . '<br>';`  
The `gettype()` function is used to get the data type of a variable.

## Types of Variables

This book covers the three main PHP variable types: *numbers*, *strings*, and *arrays*. I'll introduce them quickly here, and later chapters will discuss them in more detail:

- Chapter 4, "Using Numbers"
- Chapter 5, "Using Strings"
- Chapter 7, "Using Arrays"

A fourth variable type, *objects*, is introduced in Appendix B, "Resources and Next Steps," but isn't covered in this book. That particular subject is just too advanced for a beginner's guide—in fact, basic coverage of the subject in my *PHP 5 Advanced: Visual QuickPro Guide* (Peachpit Press, 2007) requires over 150 pages!

### Numbers

Technically speaking, PHP breaks numbers into two types: *integers* and *floating-point* (also known as *double-precision floating-point* or *doubles*). Due to the lax way PHP handles variables, it won't affect your programming to group the two categories of numbers into one all-inclusive membership. Still, let's briefly discuss the differences between the two, for clarity's sake.

The first type of numbers—integers—are the same as whole numbers. They can be positive or negative but include neither fractions nor decimals. Numbers that use a decimal point (even something like 1.0) are floating-point numbers. You must also use floating-point numbers to refer to fractions, because the only way to express a fraction in PHP is to convert it to its decimal equivalent. Hence,  $1\frac{1}{4}$  is written as 1.25. Table 2.3 lists some sample valid numbers and their formal type; Table 2.4 lists invalid numbers and the rules they violate.

**TIP** As you'll soon see, you can quote invalid numbers to turn them into valid strings.

**TABLE 2.3** Valid Numbers in PHP

Number	Type
1	Integer
1.0	Floating-point
1972	Integer
19.72	Floating-point
-1	Integer
-1.0	Floating-point

**TABLE 2.4** Invalid Numbers in PHP

Number	Reason
1/3	Contains a slash
1996a	Contains a letter
08.02.06	Contains multiple decimals

Floating point numbers are more commonly known as decimal numbers. That is numbers that have a decimals in them followed by a number of significant digits 2.75 is an example of a floating point number.

Programming divides numbers into these two types. Integers and floating point, and the reason why is because computers store integers and floating points in different ways in memory.

One important point to note, is just that you can't divide by zero. It's an illegal operation, and you'll get a warning if you try and divide by zero.



## Strings

A string is any number of characters enclosed within a pair of either single (') or double (") quotation marks. Strings can contain any combination of letters, numbers, symbols, and spaces. Strings can also contain variables.

Here are examples of valid string values:

```
"Hello, world!"
"Hello, $first_name!"
"1/3"
'Hello, world! How are you today?'
"08.02.06"
"1996"
```

That last one is an *empty string*—a string that contains no characters.

In short, to create a string, just wrap something within quotation marks. There are cases, however, where you may run into problems. For example:

```
"I said, "How are you?""
```

This string will be tricky and I hinted at the same problem in Chapter 1, “Getting Started with PHP,” with respect to printing HTML code. When PHP hits the second quotation mark in the above, it assumes the string ends there; the continuing text (*How...*) causes an error. To use a quotation mark within a string you can *escape* the quotation mark by putting a backslash (\) before it:

```
"I said, \"How are you?\""
```

The backslash tells PHP to treat each escaped quotation mark as part of the *value* of the string, rather than using it as the string’s opening or closing indicators.

You can similarly circumvent this problem by using different quotation mark types:

```
'I said, "How are you?'"
"I said, 'How are you?'"
```

**TIP** Notice that “1996” converts an integer into a string, simply by placing the number within quotes. Essentially, the string contains the characters 1996, whereas the number (a nonquoted value) would be equal to 1996. It’s a fine distinction, and one that won’t matter in your code, because you can perform mathematical calculations with the string 1996 just as you can with the number.

**TIP** Chapter 1 also demonstrated how to create a new line by printing the \n character within double quotation marks. Although escaping a quotation mark prints the quotation mark, escaping an *n* prints a new line, escaping an *r* creates a carriage return, and escaping a *t* creates a tab.

**TIP** Understanding strings, variables, and the single and double quotation marks is critical to programming with PHP. For this reason, a section at the end of this chapter is dedicated to the subject.

Placing a pair of quotes together with nothing between them results in what is called an empty string. It is commonly used for erasing or initializing the value of a string variable.

```
$name = '';
```

## Arrays

Arrays are covered more thoroughly in Chapter 7, but let's look at them briefly here. Whereas a string or a number contains a single value (both are said to be *scalar*), an array can have more than one value assigned to it. You can think of an array as a list or table of values: you can put multiple strings and/or numbers into one array.

Arrays use *keys* to create and retrieve the values they store. The resulting structure—a list of key-value pairs—looks similar to a two-column spreadsheet. Unlike arrays in other programming languages, the array structure in PHP is so flexible that it can use either numbers or strings for both the keys and the values. The array doesn't even need to be consistent in this respect. (All of this will make more sense in Chapter 7, when you start working with specific examples.)

PHP has two different types of arrays, based on the format of the keys. If the array uses numbers for the keys (Table 2.5), it's called an *indexed* array. If it uses strings for the keys (Table 2.6), it's an *associative* array. In either case, the values in the array can be of any variable type (string, number, and so on).

**TIP** The array's key is also called its *index*. You'll see these two terms used interchangeably.

**TIP** An array can, and frequently will, contain other arrays, creating what is called a *multi-dimensional* array.

**TIP** What PHP calls an *associative array* is called a *hash* in Perl and Ruby, among other languages.

TABLE 2.5 Indexed Array

Key	Value
0	Don
1	Betty
2	Roger
3	Jane

TABLE 2.6 Associative Array

Key	Value
VT	Vermont
NH	New Hampshire
IA	Iowa
PA	Pennsylvania

Scalar can only hold one piece of data at a time.

```
Number is 1  
String is Hello, world!
```

**A** The result of printing the values of two variables.

```
_SERVER is Array
```

**B** Using the `print` statement on a complex variable type, such as an array, will not have the results you desire.

## Variable Values

To assign a value to a variable, regardless of the variable type, you use the equals sign (`=`). Therefore, the equals sign is called the *assignment operator*, because it assigns the value on the right to the variable on the left. For example:

```
$number = 1;  
$floating_number = 1.2;  
$string = "Hello, world!";
```

As each of these lines represents a complete statement (i.e., an executable action), they each conclude with a semicolon.

To print out the value of a variable, you can use the `print` function:

```
print $number;  
print $string;
```

If you want to print a variable's value within a context, you can place the variable's name in the printed string, as long as you use double quotation marks **A**:

```
print "Number is $number";  
print "String is $string";
```

Using `print` in this way works for the scalar (single-valued) variable types—numbers and strings. For complex variable types—arrays and objects—you cannot just use `print` **B**:

```
print "_SERVER is $_SERVER";
```

As you've already seen, `print_r()` can handle these nonscalar types, and you'll learn other approaches later in the book.

*continues on next page*

Whether you're dealing with scalar or nonscalar variables, don't forget that printing out their values is an excellent debugging technique when you're having problems with a script!

Because variable types aren't locked in (PHP is referred to as a *weakly typed* language), they can be changed on the fly:

```
$variable = 1;
$variable = "Greetings";
```

If you were to print the value of `$variable` now, the result would be *Greetings*. The following script better demonstrates the concept of assigning values to variables and then accessing those values.

### To assign values to and access variables:

1. Create a new PHP script in your text editor or IDE, to be named `variables.php` (Script 2.3).
2. Create the initial HTML tags:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
→ 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=utf-8"/>
    <title>Variables</title>
</head>
<body>
```

**Script 2.3** Some basic variables are defined and their values printed by this script.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/
    xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml"
    xml:lang="en" lang="en">
4  <head>
5      <meta http-equiv="Content-Type"
        content="text/html; charset=utf-8"/>
6      <title>Variables</title>
7  </head>
8  <body>
9  <?php // Script 2.3 - variables.php
10
11  // An address:
12  $street = "100 Main Street";
13  $city = "State College";
14  $state = "PA";
15  $zip = 16801;
16
17  // Print the address:
18  print "<p>The address is:<br />$street
    <br />$city $state $zip<p>";
19
20  ?>
21 </body>
22 </html>
```

## HTML5 <br> tag

### NOTE:

\$zip data type should actually be a string. For two reasons, most zip codes now have nine digits and a special character ( - ). Use number data types when you need to perform a mathematical operation.

3. Begin the PHP code:

```
<?php // Script 2.3 - variables.php
```

4. Define some number and string variables:

```
$street = "100 Main Street";  
$city = "State College";  
$state = "PA";  
$zip = 16801;
```

These lines create four different variables of both string and number types. The strings are defined using quotation marks, and each variable name follows the syntactical naming rules.

Remember that each statement must conclude with a semicolon and that the variable names are case-sensitive.

5. Print out the values of the variables within some context:

```
print "<p>The address is:<br />  
→ $street <br />$city $state  
→ $zip</p>";
```

Here a single **print** statement can access all the variables. The entire string to be printed (consisting of text, HTML tags, and variables) is enclosed within double quotation marks. The HTML **<br />** tags make the text flow over multiple lines in the browser window (remember, the extra space and slash in the break tag are there for sake of XHTML compliance).

*continues on next page*

6. Complete the PHP section and the HTML page:

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

7. Save the file as **variables.php**, upload it to your server (or save it to the appropriate directory on your computer), and test it in your Web browser **C**.

**TIP** If you see a parse error **D** when you run this script, you probably either omitted a semicolon or have an imbalance in your quotation marks.

**TIP** If one of the variable's values isn't printed out or you see an *Undefined variable* error **E**, you most likely failed to spell a variable name the same way twice.

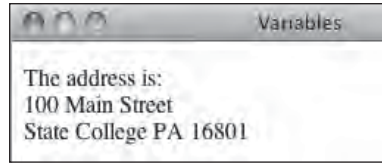
**TIP** If you see a blank page, you most likely have an error but PHP's `display_errors` configuration is set to off. See Chapter 3, "HTML Forms and PHP," for details.

We need to make sure our `php.ini` configuration file allows us to display errors.

■ **Make sure `display_errors` is on.**

This is a basic PHP configuration setting (discussed in Appendix A). You can confirm this setting by executing the `phpinfo()` function (just use your browser to search for **display\_errors** in the resulting page). For security reasons, PHP may not be set to display the errors that occur. If that's the case, you'll end up seeing blank pages when problems occur. To debug most problems, you'll need to see the errors, so turn this setting on while you're learning. Read page 29.

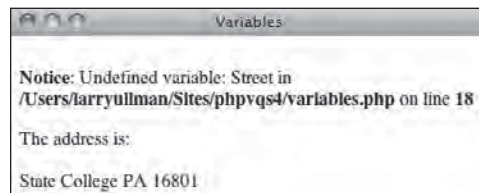
`<?php phpinfo(); ?>` - Read page 8



**C** Some variables are assigned values, and then printed within a context.

**Parse error:** syntax error, unexpected T\_VARIABLE in /Users/larryullman/Sites/phpvqs4/variables.php on line 15

**D** Parse errors are the most common type of PHP error, as you'll discover. They're frequently caused by missing semicolons or an imbalance of quotation marks or parentheses.



**E** The *Undefined variable* error indicates that you used a variable with no value (it hasn't been defined). This can happen with misspellings and capitalization inconsistencies.

**Script 2.4** This script simply demonstrates how the type of quotation mark you use with variables affects the end result.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="content-type"
6     content="text/html; charset=utf-8" />
7   <title>Quotes</title>
8 </head>
9 <body>
10 <?php // Script 2.4 - quotes.php
11 // Single or double quotation marks
  won't matter here:
12 $first_name = 'Larry';
13 $last_name = "Ullman";
14
15 // Single or double quotation marks DOES
  matter here:
16 $name1 = '$first_name $last_name';
17 $name2 = "$first_name $last_name";
18
19 // Single or double quotation marks DOES
  matter here:
20 print "<h1>Double Quotes</h1><p>name1
  is $name1 <br />
21 name2 is $name2</p>";
22
23 print '<h1>Single Quotes</h1><p>name1
  is $name1 <br />
24 name2 is $name2</p>';
25
26 ?>
27 </body>
28 </html>
```

Single quotes don't require as much resources because the PHP processor doesn't look in the string to see if there are variables that it needs to parse and replace. In keeping consistent with PHP coding conventions, the general rule of thumb to follow is to always use single quotes to define your strings.

## Understanding Quotation Marks

Now that you know the basics of variables and how to create them, I want to make sure you completely understand how to properly use quotation marks. PHP, like most programming languages, allows you to use both double (") and single (') quotation marks—but they give vastly different results. It's critical that you comprehend the distinction, so the next example will run tests using both types.

**The rule to remember is: Items within single quotation marks are treated literally; items within double quotation marks are extrapolated.** This means that within double quotation marks, a variable's name is replaced with its value, as in Script 2.3, but the same is not true for single quotation marks.

This rule applies anywhere in PHP you might use quotation marks, including uses of the **print** function and the assignment of values to string variables. An example is the best way to demonstrate this critical concept.

### To use quotation marks:

1. Begin a new PHP script in your text editor or IDE, to be named **quotes.php** (Script 2.4).
2. Create the initial HTML tags:

```
<!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">
```

*continues on next page*

```

<head>
  <meta http-equiv="content-type"
    → content="text/html;
    → charset=utf-8" />
  <title>Quotes</title>
</head>
<body>

```

3. Begin the PHP code:

```
<?php // Script 2.4 - quotes.php
```

4. Create two string variables:

```

$first_name = 'Larry';
$last_name = "Ullman";

```

It doesn't matter whether you use single or double quotation marks for these two variables, as each string can be treated literally. However, if you're using your own name here (and feel free to do so) and it contains an apostrophe, you'll need to either use double quotation marks or escape the apostrophe within single quotation marks:

```

$last_name = "O'Toole";
$last_name = 'O\'Toole';

```

5. Create two different *name* variables, using the existing first- and last-name variables:

```

$name1 = '$first_name $last_name';
$name2 = "$first_name
→ $last_name";

```

In these lines it makes a huge difference which quotation marks you use. The **\$name1** variable is now literally equal to *\$first\_name \$last\_name*, because no extrapolation occurs. Conversely, **\$name2** is equal to *Larry Ullman*, presumably the intended result.

6. Print out the variables using both types of quotation marks:

```

print "<h1>Double Quotes
→ </h1><p>name1 is $name1 <br />
name2 is $name2</p>";
print '<h1>Single Quotes
→ </h1><p>name1 is $name1 <br />
name2 is $name2</p>';

```

Again, the quotation marks make all the difference here. The first **print** statement, using double quotation marks, prints out the values of the **\$name1** and **\$name2** variables, whereas the second, using single quotation marks, prints out *\$name1* and *\$name2* literally.

The HTML in the **print** statements makes them more legible in the browser, and each statement is executed over two lines, which is perfectly acceptable.

7. Complete the PHP section and the HTML page:

```

?>
</body>
</html>

```





**A** The different quotation marks (single versus double) dictate whether the variable's name or value is printed.

You should almost never have to escape quotes in a string, because you can just alternate your quoting style.  
Here is an interesting short article to read:  
<http://techtalk.virendrachandak.com/php-double-quotes-vs-single-quotes/#axzz2sgxRQyCr>

8. Save the file as **quotes.php**, upload it to your server (or save it to the appropriate directory on your computer), and test it in your Web browser **A**.

**TIP** If you're still confused about the distinction between the two types of quotation marks, stick with double quotation marks and you'll be safer.

**TIP** Arguably, using single quotation marks when you can is marginally preferable, as PHP won't need to search the strings looking for variables. But, at best, this is a minor point.

**TIP** The shortcuts for creating newlines (`\n`), carriage returns (`\r`), and tabs (`\t`) must also be used within double quotation marks to have the desired effect.

**TIP** Remember that you don't always need to use quotation marks at all. When assigning a numeric value or when only printing a variable, you can omit them:

```
$num = 2;
print $num;
```

How does using single versus double quotation marks differ in outputting strings?

Single quotes always display exactly what you type (other than escaped quotes and backslashes); double quotes, not necessarily.

Values enclosed in single quotation marks will be treated literally, whereas those within double quotation marks will be interpreted. Placing variables and special characters within double quotes will result in their represented values printed, not their literal values.

Echo "the answer is \$var"; will see \$var replaced by its value in the Web browser.

## Review and Pursue

If you have any problems with the review questions or the pursue prompts, turn to the book's supporting forum ([www.LarryUllman.com/forum/](http://www.LarryUllman.com/forum/)).

### Review

- What kind of variable is `$_SERVER` an example of?
- What character must all variables begin with?
- What characters can be used first in a variable's name? What other characters can be used in a variable's name, after the first character?
- Are variable names case-sensitive or case-insensitive?
- What does it mean to say that a variable is *scalar*? What are examples of scalar variable types? What is an example of a nonscalar variable type?
- What is the assignment operator?
- What great debugging technique—with respect to variables—was introduced in this chapter?
- What is the difference between using single or double quotation marks?

### Pursue

- Create another PHP script that defines some variables and prints their values. Try using variables of different scalar types.
- Create a PHP script that prints the value of some variables within some HTML. More sophisticated practice might involve using PHP and variables to create a link or image tag.

In PHP, variable names are case-sensitive.

The basic assignment operator is `=`. It does NOT mean "equal to". It actually means that the left operand gets set to the value of the expression on the right.

Values enclosed in single quotation marks will be treated literally, whereas those within double quotation marks will be interpreted. Placing variables and special characters within double quotes will result in their represented values printed, not their literal values.