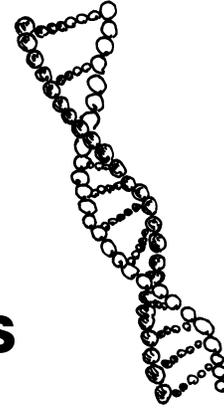


chapter 4

The Building Blocks: Data Types, Literals, Variables, and Constants



“One man’s constant is another man’s variable.”
—Alan Perlis

4.1 Data Types

A program can do many things, including perform calculations, sort names, prepare phone lists, display images, play chess, ad infinitum. To do anything, however, the program works with the data that is given to it. Data types specify what kind of data, such as numbers and characters, can be stored and manipulated within a program. PHP supports a number of fundamental basic data types, such as integers, floats, and strings. Basic data types are the simplest building blocks of a program. They are called scalars and can be assigned a single literal value such as a number, 5.7, or a string of characters, such as "hello", a date and time, or a boolean (true/false). See Figure 4.1.

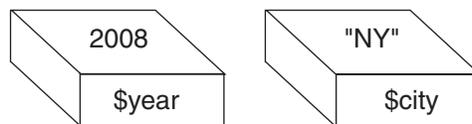


Figure 4.1 Scalars hold one value.

PHP also supports composite data types, such as arrays and objects. Composite data types represent a collection of data, rather than a single value (see Figure 4.2). The composite data types are discussed in Chapter 8, “Arrays,” and Chapter 17, “Objects.”

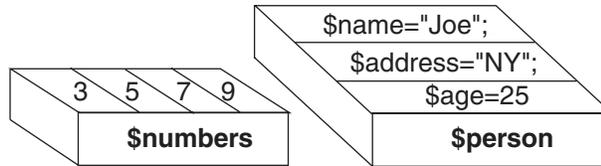


Figure 4.2 Arrays and objects hold multiple values.

The different types of data are commonly stored in variables. Examples of PHP variables are `$num = 5` or `$name = "John"` where variables `$num` and `$name` are assigned an integer and a string, respectively. Variables hold values that can change throughout the program, whereas once a constant is defined, its value does not change. `PHP_VERSION` and `PHP_OS` are examples of predefined PHP constants. The use of PHP variables and constants is addressed in “Variables” on page 70 and “Constants” on page 99 of this chapter.

PHP supports four core data types:

- Integer
- Float (also called double)
- String
- Boolean

In addition to the four core data types, there are four other special types:

- Null
- Array
- Object
- Resources

4.1.1 Numeric Literals

PHP supports both integers and floating-point numbers. See Example 4.1.

- **Integers**—Integers are whole numbers and do not contain a decimal point; for example, 123 and -6 . Integers can be expressed in decimal (base 10), octal (base 8), and hexadecimal (base 16), and are either positive or negative values.
- **Floating-point numbers**—Floating-point numbers, also called doubles or reals, are fractional numbers such as 123.56 or -2.5 . They must contain a decimal point or an exponent specifier, such as $1.3e-2$. The letter “e” can be either upper or lowercase.

PHP numbers can be very large (the size depends on your platform), but a precision of 14 decimal digits is a common value or $(\sim 1.8e^{308})$.

4.1 Data Types

61

EXAMPLE 4.1

```

12345      integer
23.45      float
.234E-2    float in scientific notation
.234e+3    float in scientific notation
0x456fff   integer in base 16, hexadecimal
0x456FFF   integer in base 16, hexadecimal
0777      integer in base 8, octal

```

EXAMPLE 4.2

```

<html>
<head><title>Printing Numbers</title>
</head>
<body bgcolor="lightblue">
<font face = "arial" size = '+1'>
<?php
    print "The positive integer is <em><b>" . 5623 . "
      .</b></em><br />";
    print "The negative integer is <em><b>" . -22 . "</b></em><br />";
    print "The floating point number is <em><b>" . 15.3 . "
      .</b></em><br />";
    print "The number in scientific notation is <em><b>" . 5e3 . "
      . </b></em><br />";
    print "\tThe string is: <em><b>I can't help you!</em>
      </b><br />";
?>
</body>
</html>

```

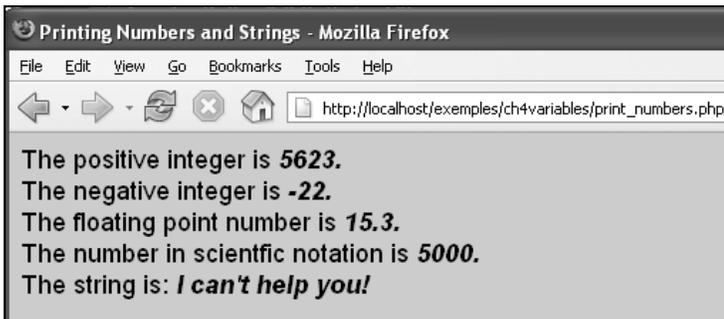


Figure 4.3 Output from Example 4.2.

4.1.2 String Literals and Quoting

We introduce strings in this chapter but Chapter 6, “Strings,” provides a more comprehensive coverage. String literals are a row of characters enclosed in either double or single quotes.¹ **The quotes must be matched.** If the string starts with a single quote, it must end with a matching single quote; likewise if it starts with a double quote, it must end with a double quote. If a string of characters is enclosed in single quotes, the characters are treated literally (each of the characters represents itself). We can say the single quotes are the democratic quotes: All characters are treated equally.

Double quotes do not treat all characters equally. If a string is enclosed in double quotes, most of the characters represent themselves, but dollar signs and backslashes have a special meaning as shown in the following examples.

Single quotes can hide double quotes, and double quotes can hide single quotes:²

```
"This is a string"
'This is another string'
"This is also 'a string'"
'This is "a string"'
```

An empty set of quotes is called the null string. If a number is enclosed in quotes, it is considered a string; for example, "5" is a string, whereas 5 is a number.

Strings are called constants or literals. The string value "hello" is called a string constant or literal. To change a string requires replacing it with another string.

Strings can contain escape sequences (a single character preceded with a backslash). Escape sequences cause a character to behave in a certain way; for example, a "\t" represents a tab and "\n" represents a newline. The backslash is also used for quoting a single character so that it will not be interpreted; for example, \\$5.00 where the dollar sign in PHP is used to represent variables rather than money. \\$5.00 could also be written as '\$5' because single quotes protect all characters from interpretation.

Here documents, also called *here-docs*, provide a way to create a block of text that simplifies writing strings containing lots of single quotes, double quotes, and variables (see Example 4.4).

EXAMPLE 4.3

```
<html>
  <head><title>Quotes</title></head>
  <body bgcolor="lightblue"><font size='+1'>
1   <?php
2     $name = "Nancy"; // Setting a PHP variable
     print "<ol>";
```

1. PHP always null-terminates strings internally and keeps track of the length of the string.
2. PHP recognizes editors that use straight quotes, such as vi or Notepad, but not editors that automatically transform straight quotes into curly quotes.

EXAMPLE 4.3 (CONTINUED)

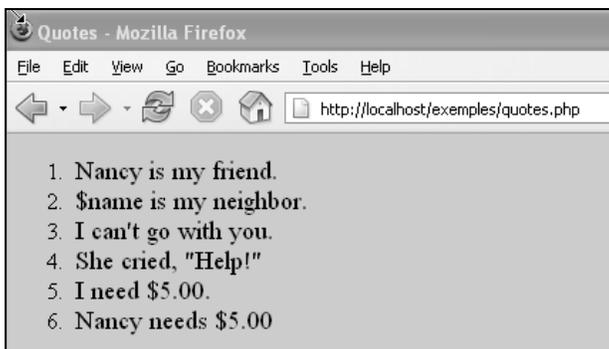
```

3     print "<li> $name is my friend.</li>"; // Double quotes
4     print '<li> $name is my neighbor.</li>'; // Single quotes
5     print "<li> I can't go with you.</li>"; // Nested quotes
6     print "<li> She cried, \"Help!\"</li>"; // Escaping quotes
7     print "<li> I need \$5.00.</li>"; // The backslash
                                     // quotes one character
8     print "<li> $name needs ". '$5.00 </li>'; // Nested quotes
     print "</ol>";
     ?>
     </body>
</html>

```

EXPLANATION

- 1 PHP program starts here.
- 2 `$name` is a PHP variable. It is assigned the string "Nancy". You will learn all about variables in the section "Variables" on page 70.
- 3 When a string is enclosed within double quotes, the PHP interpreter will substitute the variable with its value; for example, `$name` will be replaced with "Nancy".
- 4 When a string is enclosed in single quotes, all characters are treated as literals. Variable substitution will not occur.
- 5 Single quotes can be nested within double quotes and vice versa.
- 6 Quotes can be escaped with a backslash to make them literal characters within a string.
- 7 The dollar sign is escaped from PHP interpretation, that is, is treated as a literal character.
- 8 A string in double quotes is concatenated to a string in single quotes. Just as the backslash protects the dollar sign from interpretation, so do the single quotes. Remember, characters in single quotes are all treated as literals; that is, PHP does not consider any of the enclosed characters as special. See the output in Figure 4.4.

**Figure 4.4** Single and double quotes.

The Here Document—A Special Kind of Quoting. Here documents are a kind of quoting popular in a number of languages, such as JavaScript, Perl, Shell scripts, and so on. Here documents, also called *here-docs*, allow you to quote a large block of text within your script without using multiple print statements and quotes. The entire block of text is treated as though it is surrounded by double quotes. This can be useful if you have a large block of HTML within your PHP script interspersed with variables, quotes, and escape sequences.

Rules for a Here Document:

1. The user-defined delimiter word starts and terminates the here document. Text is inserted between the delimiters. The delimiter can contain letters, numbers, and the underscore character. The first letter must be a letter or an underscore. By convention, the delimiter should be in all uppercase letters to make it stand out from other words in your script. The delimiter is preceded by three < characters; for example, <<<DELIMITER

```
print <<<HERE_DOC_DELIMITER
    <text here>
    ...
    < more text>
    ...

HERE_DOC_DELIMITER
```

2. The delimiter cannot be surrounded by any spaces, comments, or other text. The final delimiter can optionally be terminated with a semicolon and must be on a line by itself.
3. All variable and escape sequences will be interpreted within the here document.

EXAMPLE 4.4

```
1 <?php
2     $bgcolor="darkblue";
3     $tablecolor = "yellow";
4     print <<<MY_BOUNDARY
5     <html><head><title>heredoc</title></head>
6     <body bgcolor="$bgcolor">
7     <table border="1" bgcolor=$tablecolor>
8         <tr><th>Author</th><th>Book</th></tr>
9         <tr>
10            <td>Marcel Proust</td>
11            <td>Remembrance of Things Past</td>
12        </tr>
```

EXAMPLE 4.4 (CONTINUED)

```

        <tr>
            <td>Charles Dickens</td>
            <td>Tale of Two Cities</td>
        </tr>
    </table>
</body>
7 </html>
8 MY_BOUNDARY;
  ?>

```

EXPLANATION

- 1 PHP starts here.
- 2 Two scalar variables are defined.
- 3 This is the *here-doc*. The user-defined terminator, `MY_BOUNDARY`, is prepended with `<<<`. There can be no space after the terminator; otherwise an error like this will be displayed: *Parse error: syntax error, unexpected T_SL in c:\wamp\www\examples\ch4variables\heredoc.php on line 4*
- 4 All of the HTML document is embedded in the here document. The HTML will be sent to the browser as is. Any PHP code embedded within the HTML tags will be handled by the PHP interpreter.
- 5 The value of the variable, `$bgcolor`, will be assigned as the background color of the page.
- 6 An HTML table is started here. The value of the variable, `$tablecolor`, will be assigned as the background color of the table cells.
- 7 The HTML document ends here, inside the *here-doc*.
- 8 The user-defined terminator, `MY_BOUNDARY`, marks the end of the here document. There can be no spaces surrounding the terminator. The semicolon is optional.

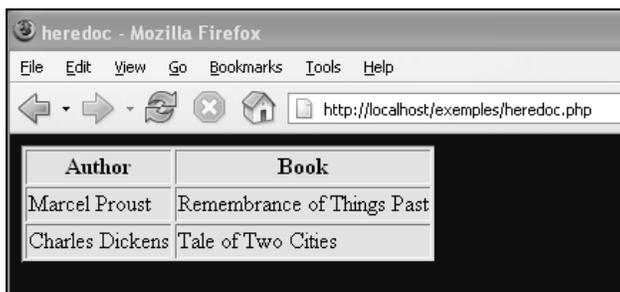


Figure 4.5 Here document output.

Escape Sequences. Escape sequences consist of a backslash followed by a single character. When enclosed in double quotes, the backslash causes the interpretation of the next character to “escape” from its normal ASCII code and to represent something else (see Table 4.1). To display the escape sequences in your browser, the HTML `<pre>` tag can be used (see Example 4.5); otherwise, the escape sequences placed within your PHP script will not be interpreted.

Table 4.1 Escape Sequences

<i>Escape Sequence</i>	<i>What It Represents</i>
<code>\'</code>	Single quotation mark
<code>\"</code>	Double quotation
<code>\t</code>	Tab
<code>\n</code>	Newline
<code>\r</code>	Return/line feed
<code>\\$</code>	A literal dollar sign
<code>\\</code>	Backslash
<code>\70</code>	Represents the octal value
<code>\x05</code>	Represents the hexadecimal character

EXAMPLE 4.5

```

<html><head><title>Escape Sequences</title></head>
<body bgcolor="orange">
<b>
1  <pre>
2  <?php
3      print "\t\tTwo tabs are \\t\\t, and two newlines are
        \\n\\n.\n\n";
4      print "\tThe escaped octal numbers represent ASCII
        \101\102\103.\n";
        print "\tThe escaped hexadecimal numbers represent ASCII
        \x25\x26.\n";
5      print '\tWith single quotes, backslash sequences are not
        interpreted.\n';
        ?>
</pre>
</b>
</body>
</html>

```

EXPLANATION

- 1 Because this file will be displayed in a browser window, the HTML `<pre>` tags are used to retain spaces and tabs. If you run PHP at the command line, the escape sequences will be interpreted.
- 2 The PHP program starts here with its opening tag.
- 3 The escape sequences must be enclosed in double quotes. The sequences for tab (`\t`) and newline (`\n`) characters produce tabs and newlines. If a backslash is prepended with another backslash, then the backslash is treated as a literal.
- 4 In this example, by preceding an octal or hexadecimal number with a backslash, its ASCII equivalent is displayed.
- 5 If a string is enclosed in single quotes, escape sequences are ignored. See the output in Figure 4.6.

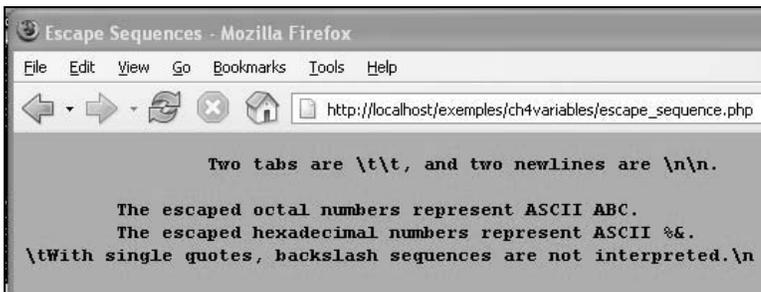


Figure 4.6 Escape sequences and the `<pre>` tag.

```

$ php escape_sequence.php
<html><head><title>Escape Sequences</title></head>
<body bgcolor="orange">
<b>
<pre>
    Two tabs are \t\t, and two newlines are \n\n.
    The escaped octal numbers represent ASCII ABC.
    The escaped hexadecimal numbers represent ASCII %&.
\tWith single quotes, backslash sequences are not interpreted.\n</pre>
</b>
</body>
</html>

```

Figure 4.7 Escape sequences at the command line.

4.1.3 Boolean Literals

Boolean literals (introduced in PHP 4) are logical values that have only one of two values, *true* or *false*, both **case insensitive**. You can think of the values as *yes* or *no*, *on* or *off*, or *1* or *0*. They are used to test whether a condition is true or false. When using numeric comparison and equality operators, the value *true* evaluates to 1 and *false* evaluates to the empty string (see Figure 4.8).

```
$answer1 = true;
or
if ($answer2 == false) { do something; }
```

EXAMPLE 4.6

```
<?php
if ( 0 == False && "" == FALSE) {
    print "zero and null are <em>>false.</em><br /> ";}
if ( 1 == True && "abc" == true) {
    print "1 and \"abc\" are both <em>>true.</em><br /> "; }
?>
```

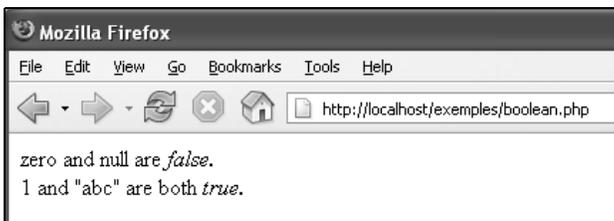


Figure 4.8 True and false.

4.1.4 Special Data Types

Null. NULL represents “no value,” meaning “nothing,” not even an empty string or zero. It is a type of NULL. An uninitialized variable contains the value NULL. A variable can be assigned the value NULL, and if a variable has been unset, it is considered to be NULL.

Resource. A resource is a special variable, holding a reference to an external resource such as a database object or file handler. Resources are created and used by special functions. File and database resources are defined by the PHP interpreter and are only accessible by functions provided by the interpreter (see Chapter 11, “Files and Directories,” and Chapter 15, “PHP and MySQL Integration”).

The `gettype()` Function. The `gettype()` built-in function returns a string to identify the data type of its argument. The argument might be a variable, string, keyword, and so on. You can use the `gettype()` function to check whether or not a variable has been defined because if there is no value associated with the variable, the `gettype()` function returns `NULL` (see Figure 4.9).

Strings returned from the `gettype()` function include the following:

```
"boolean" (since PHP 4)
"integer"
"double" (for historical reasons "double" is returned in case of a float,
and not simply "float")
"string"
"array"
"object"
"resource" (since PHP 4)
"NULL" (since PHP 4)
```

FORMAT

```
string gettype ( mixed var )
```

Examples:

```
$type=gettype(54.6); // Returns "float"
print gettype("yes"); // Returns and prints "string"
```

EXAMPLE 4.7

```
<html>
<head><title>Getting the Data Type with gettype()</title>
</head><body bgcolor="lightblue">
<font face = "arial" size = '+1'>
<pre>
<?php
    print "Type <b>5623</b> is: " . gettype(5623) . ".\n";
    print "Type <b>-22</b> is: " . gettype(-22) . ".\n";
    print "Type <b>15.3</b> is: " . gettype(15.3) . ".\n";
    print "Type <b>5e3</b> is: " . gettype(5e3) . ".\n";
    print "Type <b>\\"Hi\"</b> is: " . gettype("Hi") . ".\n";
    print "Type <b>>true</b> is: " . gettype(true) . ".\n";
    print "Type <b>>false</b> is: " . gettype(false) . ".\n";
    print "Type <b>>null</b> is: " . gettype(null) . ".\n";
    print "Type <b>\$nothing</b> is: " . gettype($nothing) . ".\n";
?>
</body>
</html>
```

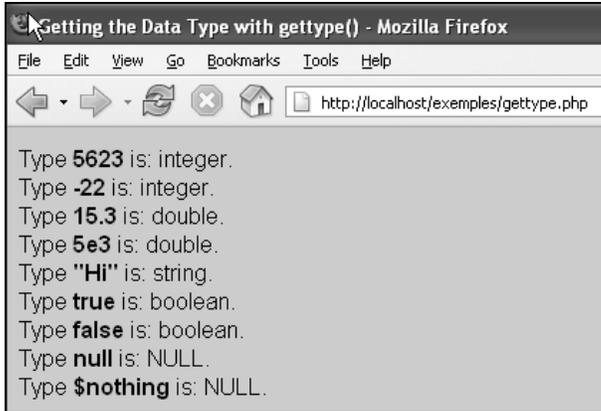


Figure 4.9 PHP data types. Output from Example 4.7.

4.2 Variables

4.2.1 Definition and Assignment

Variables are fundamental to all programming languages. They are data items that represent a memory storage location in the computer. Variables are containers that hold data such as numbers and strings. In PHP programs there are three types of variables:

1. Predefined variables
2. User-defined variables
3. Form variables related to names in an HTML form

Variables have a *name*, a *type*, and a *value*.

```
$num = 5;           // name: "$num", value: 5, type: numeric
$friend = "Peter"; // name: "$friend", value: "Peter", type: string
$x = true;         // name: "$x", value: true, type: boolean
```

The values assigned to variables can change throughout the run of a program whereas constants, also called literals, remain fixed.

PHP variables can be assigned different types of data, including:

- Numeric
- String
- Boolean
- Objects
- Arrays

Computer programming languages like C++ and Java require that you specify the type of data you are going to store in a variable when you declare it. For example, if you are going to assign an integer to a variable, you would have to say something like:

```
int n = 5;
```

and if you were assigning a floating-point number:

```
float x = 44.5;
```

Languages that require that you specify a data type are called “strongly typed” languages. PHP, conversely, is a dynamically, or loosely typed, language, meaning that you do not have to specify the data type of a variable. In fact, doing so will produce an error. With PHP you would simply say:

```
$n = 5;
$x = 44.5;
```

and PHP will figure out what type of data is being stored in `$n` and `$x`.

4.2.2 Valid Names

Variable names consist of any number of letters (an underscore counts as a letter) and digits. The first letter must be a letter or an underscore (see Table 4.2). Variable names are case sensitive, so `Name`, `name`, and `NAme` are all different variable names.

Table 4.2 Valid and Invalid Variable Name Examples

Valid Variable Names	Invalid Variable Names
<code>\$name1</code>	<code>\$10names</code>
<code>\$price_tag</code>	<code>box.front</code>
<code>\$_abc</code>	<code>\$name#last</code>
<code>\$Abc_22</code>	<code>A-23</code>
<code>\$A23</code>	<code>\$5</code>

4.2.3 Declaring and Initializing Variables

Variables are normally declared before they are used. PHP variables can be declared in a script, come from an HTML form, from the query string attached to the script’s URL, from cookies, from the server, or from the server’s environment. Variable names are explicitly preceded by a `$`. You can assign a value to the variable (or initialize a variable) when you declare it, but it is not mandatory.

FORMAT

```
$variable_name = value;    initialized
$variable_name;          uninitialized, value is null
```

To declare a variable called `firstname`, you could say:

```
$first_name="Ellie";
```

You can declare multiple variables on the same line by separating each declaration with a semicolon. For example, you could say:

```
$first_name; $middle_name; $last_name;
```

Double, Single, and Backquotes in Assignment Statements. When assigning a value to a variable, if the value is a string, then the string can be enclosed in either single or double quotes; if the value is returned from a function, then the function is not enclosed in quotes; and if the value is returned from a system command (see “Execution Operators” on page 143), then the command is enclosed in backquotes:

```
$name = "Marko";           // Assign a string
$city = 'San Francisco';  // Assign a string
$now = date("m/d/Y");     // Assign output of a function
$dirlist = `ls -l`;       // Assign output of a UNIX/Linux system command
$dirlist = `dir /D/L`     // Assign a Windows system command
```

EXAMPLE 4.8

```
<html>
<head><title>Variables</title></head>
<body bgcolor="lightblue">
<font face = "arial" size='+1'>
<?php
1   $name="Joe Shmoe";
2   $age=25.4;
3   $now=date("m/d/Y");
4   $nothing;
5   echo "$name is $age years old.<br />";
6   echo '$nothing contains the value of ',gettype($nothing),
   ".<br />";
7   echo "Today is $now<br />";
?>
</font>
</body>
</html>
```

EXPLANATION

- 1 The variable called `$name` is defined and initialized within the string value "Joe Shmoe". The string can be enclosed in either single or double quotes.
- 2 The variable called `$age` is assigned the floating-point value, 25.4. When assigning a number, the value is not quoted.
- 3 The variable called `$now` is assigned the return value of the built-in `date()` function. The function is not enclosed in quotes or it will not be executed. Its arguments, "m/d/Y", must be a string value, and are enclosed in quotes.
- 4 The variable `$nothing` is not assigned an initial value; it will have the value `NULL`.
- 5 The string is enclosed in double quotes. The floating-point value of `$age` is evaluated within the string.
- 6 The `gettype()` function tells us that the type of `$nothing` is `NULL`; that is, it has no value.
- 7 The output of the PHP built-in `date()` function was assigned to `$now` and is printed (see Figure 4.10).

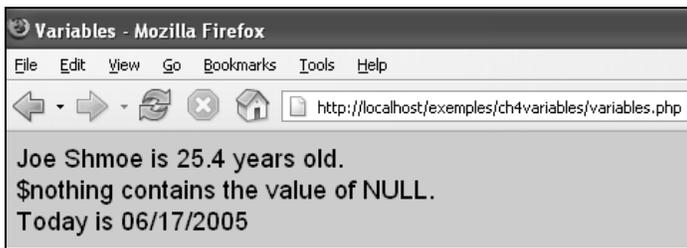


Figure 4.10 With or without quotes. Output from Example 4.8.

EXAMPLE 4.9

```

<html><head><title>Backticks</title></head>
<body bgcolor="lightgreen">
  <b>
  <pre>
  <?php
1     $month=`cal 7 2005`; // UNIX command
2     echo "$month<br />";
  >
  </b>
  </pre>
</body>
</html>

```

EXPLANATION

- 1 The UNIX/Linux `cal` command and its arguments are enclosed in backquotes (also called backticks). In PHP the backquotes are actually operators (see “Execution Operators” on page 143). The command is executed by the operating system. Its output will be assigned to the variable, `$month`.
- 2 The PHP code is embedded within HTML `<pre>` tags to allow the calendar, `$month`, to be displayed in its natural format (see Figure 4.11).

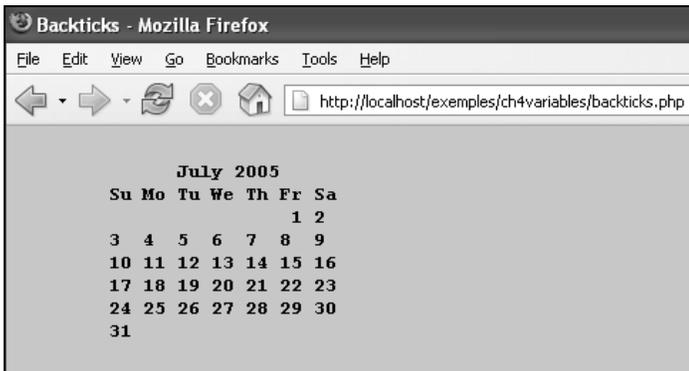


Figure 4.11
Backquotes and UNIX.
Output from Example 4.9.

EXAMPLE 4.10

```

<html><head><title>Backticks for Windows Command</title></head>
<body bgcolor="66cccc">
<b>
<pre>
<?php
1     $today = `date /T`; // Windows command
2     echo "Today is $today<br />";
    ?>
</b>
</pre>
</body>
</html>
    
```

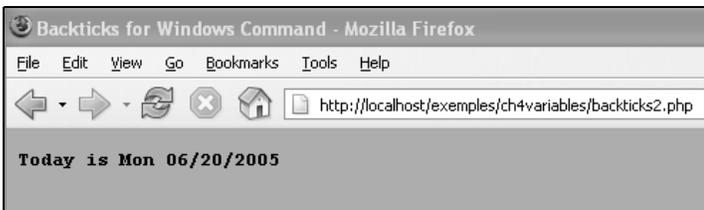


Figure 4.12
Backquotes and Windows.
Output from Example 4.10.

4.2.4 Displaying Variables

The print and echo Constructs. So far, we have seen examples using both `print` and `echo` to display data. These language constructs can be used interchangeably. The only essential difference between `echo()` and `print()` is that `echo` allows multiple, comma-separated arguments, and `print` doesn't. Neither require parentheses around their arguments because technically they are not functions, but special built-in constructs. (In fact, arguments given to `echo` must not be enclosed within parentheses.) To print formatted strings, see `printf` and `sprintf` in Chapter 6, "Strings."

Consider the following. Three variables are declared:

```
$name = "Tom";
$state = "New York";
$salary = 80000;
```

`echo()` can take a comma-separated list of string arguments:

```
echo $name, $state, $salary;
```

`print()` takes one string argument:

```
print $name;
```

However, the concatenation operator can be used to print multiple strings or strings containing multiple variables:

```
print $name . $state . $salary;
echo $name . $state . $salary;
```

or all of the variables can be enclosed in double quotes:

```
print "$name $state $salary<br />";
echo "$name $state $salary<br />";
```

If a variable is enclosed in double quotes, it will be evaluated and its value displayed. If enclosed in single quotes, variables will not be evaluated. With single quotes, what you see is what you get. Like all other characters enclosed within single quotes, the `$` is treated as a literal character.

The following strings are enclosed in single quotes:

```
echo '$name lives in $state and earns $salary.';
$name lives in $state and earns $salary.
```

```
print '$name lives in $state and earns $salary.';
$name lives in $state and earns $salary.
```

76 Chapter 4 • The Building Blocks: Data Types, Literals, Variables, and Constants

The same strings are enclosed in double quotes:

```
echo "$name lives in $state and earns \$salary.";
Tom lives in New York and earns $80000.
```

```
print "$name lives in $state and earns \$salary.";
Tom lives in New York and earns $80000.
```

Shortcut Tags. There are several shortcuts you can use to embed PHP within the HTML portion of your file, but to use these shortcuts, you must make a change in the `php.ini` file. (If you don't know where to find the `php.ini` file you are using, look at the output of the built-in `phpinfo()` function where you will find the correct path to the file.) Use caution: The PHP developers set this directive to “off” for security reasons.

From the `php.ini` file:

```
; Allow the <? tag. Otherwise, only <?php and <script> tags are recognized.
; NOTE: Using short tags should be avoided when developing applications or
; libraries that are meant for redistribution, or deployment on PHP
; servers which are not under your control, because short tags may not
; be supported on the target server. For portable, redistributable code,
; be sure not to use short tags.
short_open_tag = Off
```

<-- Turn this “On” to make short tags work

Instead of using the `print()` or `echo()` functions to output the value of variables, they can be nested within HTML code by using `<?=` and `?>` shortcut tags where they will automatically be evaluated and printed. (Note: There can be no space between the question mark and the equal sign.) All of the following formats are acceptable:

FORMAT

```
<?= expression ?>
<?= $color ?>
```

```
<? echo statement; ?>
<? echo $color; ?>
```

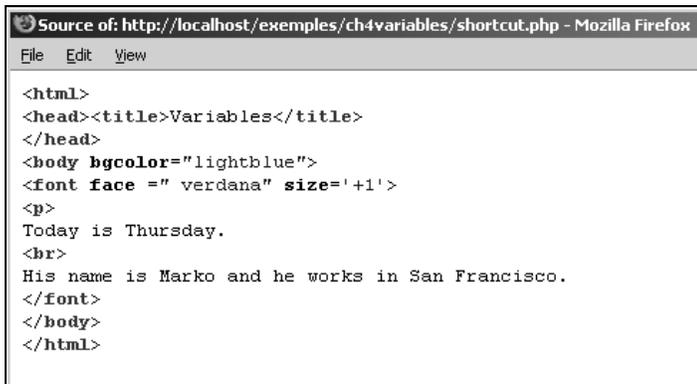
```
You have chosen a <?= $color ?> paint for your canvas.
```

EXAMPLE 4.11

```
<html>
<head><title>Variables</title></head>
<body bgcolor="lightblue">
<?php
1     $name = "Marko";
2     $city = "San Francisco";
    ?>
<font face = "verdana" size='+1'>
<p>
3 Today is <?=date("l")?>. // same as <?php echo date("l"); ?>
<br />
4 His name is <?=$name?> and he works in <?=$city?>.
</font>
</body>
</html>
```

EXPLANATION

- 1, 2 Two variables are assigned the string values, "Marko" and "San Francisco".
- 3, 4 The PHP shortcut tag is embedded within the HTML tags. PHP will evaluate the expression within the shortcut tags and print their values. The resulting HTML code contains the result of the evaluation as shown when viewing the browser's source (see Figure 4.13). In this example, the built-in `date()` function with a "l" option will return the day of the week. The variables, `$name` and `$city` are evaluated and placed within the HTML code.



```
Source of: http://localhost/examples/ch4variables/shortcut.php - Mozilla Firefox
File Edit View
<html>
<head><title>Variables</title>
</head>
<body bgcolor="lightblue">
<font face = " verdana" size='+1'>
<p>
Today is Thursday.
<br>
His name is Marko and he works in San Francisco.
</font>
</body>
</html>
```

Figure 4.13 HTML source page viewed in the browser.

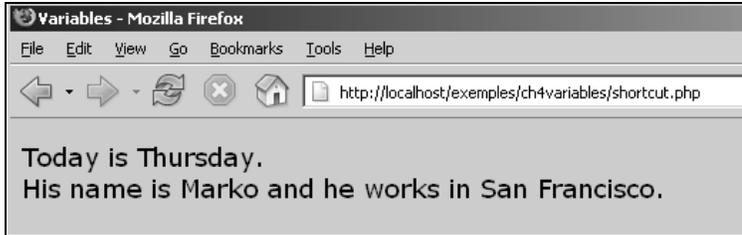


Figure 4.14 Using shortcut tags. The output from Example 4.11.

4.2.5 Variables and Mixed Data Types

Remember, strongly typed languages like C++ and Java require that you specify the type of data you are going to store in a variable when you declare it, but PHP is loosely typed. It doesn't expect or allow you to specify the data type when declaring a variable. You can assign a string to a variable and later assign a numeric value. PHP doesn't care and at runtime, the PHP interpreter will convert the data to the correct type. In Example 4.12, consider the following variable, initialized to the floating-point value of 5.5. In each successive statement, PHP will convert the type to the proper data type (see Table 4.3).

Table 4.3 How PHP Converts Data Types

<i>Variable Assignment</i>	<i>Conversion</i>
<code>\$item = 5.5;</code>	Assigned a float
<code>\$item = 44;</code>	Converted to integer
<code>\$item = "Today was bummer";</code>	Converted to string
<code>\$item = true;</code>	Converted to boolean
<code>\$item = NULL;</code>	Converted to the null value

Example 4.12 demonstrates data type conversion. The `gettype` built-in function is used to display the data types after PHP has converted the data to the correct type. See Figure 4.15 for the output.

EXAMPLE 4.12

```

<html><head><title>Type Conversion</title></head>
<body bgcolor="pink"><font size="+1">
<?php
1 $item = 5.5;
   print "$item is a " . gettype($item) . "<br />";
2 $item = 44;
   print "$item is a " . gettype($item) . "<br />";

```

EXAMPLE 4.12 (CONTINUED)

```

3  $item = "Today was a bummer!";
   print "\"$item\" is a " . gettype($item) . "<br />";
4  $item = true;
   print "$item is a " . gettype($item) . "<br />";
5  $item = NULL;
   print "$item is a " . gettype($item) . "<br />";
?>
</body>
</html>

```



Figure 4.15 Mixing data types. Output from Example 4.12.

Type Casting. Like C and Java, PHP provides a method to force the conversion of one type of value to another using the cast operator (see Chapter 5, “Operators”).

4.2.6 Concatenation and Variables

To concatenate variables and strings together on the same line, the dot (.) is used. The dot is an operator because it operates on the expression on either side of it (each called an operand). In expressions involving numeric and string values with the dot operator, PHP converts numeric values to strings. For example, consider the following statements:

```

// returns "The temperature is 87"
$temp = "The temperature is " . 87;
// returns "25 days till Christmas"
$message = 25 . " days till Christmas";

```

EXAMPLE 4.13

```

<html>
<head><title>Concatenation</title></head>
<body bgcolor="ccff66">
<b>
<?php
1   $n = 5 . " cats";
2   $years = 9;
   print "The $n " . "lived " . $years * 5 .
3   " years. <br />";
4   echo "He owns ", $years. $n , ".<br />";
   ?>
</b>
</body>
</html>

```

EXPLANATION

- 1 Variable `$n` is assigned a number concatenated to a string. The number is converted to a string and the two strings are joined together as one string by using the concatenation operator, resulting in the string "5 cats".
- 2 Variable `$years` is assigned the number 9.
- 3 The concatenation operator joins all the expressions into one string to be displayed. The `print()` function can only take one argument.
- 4 The `echo` statement takes a list of comma-separated arguments, which causes the values of `$years` and `$n` to be displayed just as with the concatenation operator. See Figure 4.16 for the output.

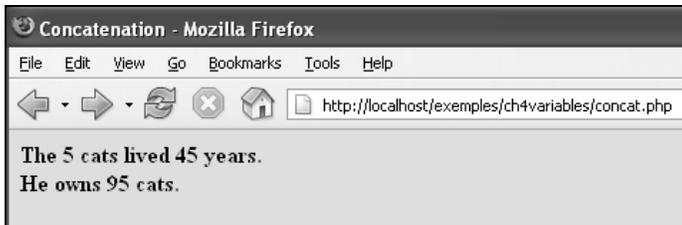


Figure 4.16 Concatenation.

4.2.7 References

Another way to assign a value to a variable is to create a reference (PHP 4). A reference is when one variable is an alias or pointer to another variable; that is, they point to the same underlying data. Changing one variable automatically changes the other. This might be useful in speeding things up when using large arrays and objects, but for now, we will not need to use references.

4.2 Variables

81

To assign by reference, prepend an ampersand (&) to the beginning of the old variable that will be assigned to the new variable; for example, `$ref = &$old;`. See Example 4.14 and its output in Figure 4.17.

EXAMPLE 4.14

```
<html><head><title>References</title></head>
<body bgcolor="yellow"><font size="+1">
<?php
1 $husband = "Honey"; // Assign the value "Honey" to $husband
2 $son = &$husband; // Assign a reference to $son.
                        // Now $husband is a reference or alias
                        // for $husband. They reference the same data.
3 print "His wife calls him $husband, and his Mom calls him $son.
<br />";
4 $son = "Lazy"; // Assign a new value to $son;
                        // $husband gets the same value
5 print "Now his wife and mother call him $son, $husband man.<br />";
?>
</body>
</html>
```

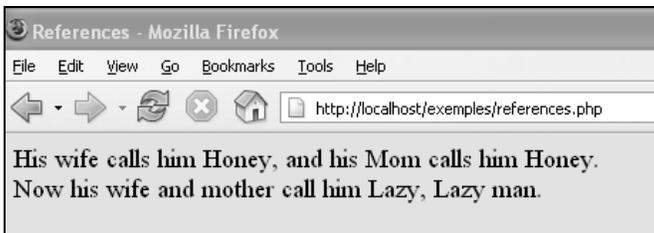


Figure 4.17 References. Output from Example 4.14.

One important thing to note is that only named variables can be assigned by reference, as shown in Example 4.15.

EXAMPLE 4.15

```
<?php
$age = 26;
$old = &$age; // This is a valid assignment.
$old = &(26 + 7); // Invalid; references an unnamed expression.
?>
```

4.2.8 Variable Variables (Dynamic Variables)

A variable variable is also called a dynamic variable. It is a variable whose name is stored in another variable. By using two dollar signs, the variable variable can access the value of the original variable. Consider the following example:

```
$pet = "Bozo";  
$clown = "pet"; // A variable is assigned the name of another variable  
echo $clown;    // prints "pet"  
echo ${$clown}; // prints Bozo
```

Dynamic variables are useful when you are dealing with variables that all contain a similar name such as form variables. Curly braces are used to ensure that the PHP parser will evaluate the dollar signs properly; that is, `$clown` will be evaluated first, and the first dollar sign removed, resulting in `${pet}`, and finally `$pet` will be evaluated to "Bozo". Example 4.16 demonstrates how variable variables can be used dynamically to change the color of a font. Output is shown in Figure 4.18.

EXAMPLE 4.16

```
<html>  
<head><title>Variable Variables</title></head>  
<body bgcolor="669966">  
<font face = "arial" size="+1">  
<?php  
1   $color1 = "red";  
   $color2 = "blue";  
   $color3 = "yellow";  
2   for($count = 1; $count <= 3; $count++){  
3       $primary = "color" . $count; // Variable variable  
4       print "<font color=${$primary}>";  
       echo "The value stored in $primary: ${$primary}<br />";  
5   }  
?>  
</font>  
</body>  
</html>
```

4.2 Variables

EXPLANATION

- 1 Three variables are defined and assigned colors. Notice the variable names only differ by the number appended to the name, 1, 2, and 3.
- 2 Although we haven't yet discussed loops, this is the best way to illustrate the use of variable variables (or dynamic variables). The initial value in the loop, `$count`, is set to 1. If the value of `$count` is less than 3 (`$count < 3`), then control goes to line 3. After the closing curly brace is reached on line 5, control will go back to the top of the loop and the value of `$count` will be incremented by 1. If `$count` is less than 3, the process repeats, and when `$count` reaches 3, the loop terminates.
- 3 The first time through the loop, the value of `$count` is appended to the string "color" resulting in `color1`. The value, "color1", is then assigned to the variable, `$primary`, so that `$primary = color1;`.
- 4 PHP expands `${$primary}`^a as follows:
 - `${color1}` First evaluate `$primary` within the curly braces.
 - `$color1` Remove the braces and now evaluate `$color1` resulting in "red".
 The color of the font and the text will be red. Next time through the loop, the count will go up by one (`$count = 2`) and the `$color2` will be "blue", and finally `$color3` will be "yellow".
- 5 See "The for Loop" on page 235 for an example of how to make use of dynamic variables with forms.

- a. The curly braces are required. If you omit them, the variable `$$primary` will be evaluated as `$color1` but not "red".

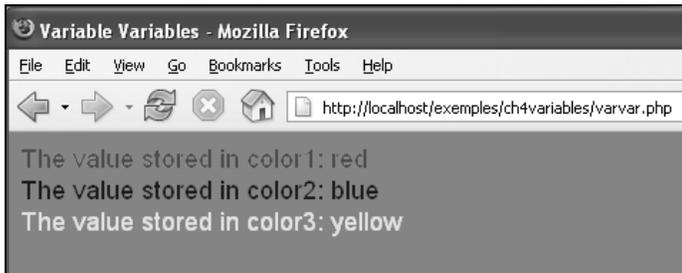


Figure 4.18 Dynamic variables. Output from Example 4.16.

4.2.9 Scope of Variables

Scope refers to where a variable is available within a script. Scope is important because it prevents important variables from being accidentally modified in some other part of the program and thus changing the way the program behaves. PHP has specific rules to control the visibility of variables. A local variable is one that exists only within a function. A global variable is available anywhere in the script other than from within functions. (See Chapter 9, “User-Defined Functions,” for creating global variables within functions.) For the most part all PHP variables have a single scope, global.

Local Variables. Variables created within a function are not available to the rest of the script. They are local to the function and disappear (go out of scope) when the function exits. If you have the same name for a variable within a function as in the main program, modifying the variable in the function will not affect the one outside the function (see Chapter 9, “User-Defined Functions”). Likewise, the function does not have access to variables created outside of the function. Most of the variables we create will be visible in the script or function in which they are declared. See Chapter 9, “User-Defined Functions,” to get a full understanding of scope, including local variables, globals, superglobals, and static.

Global and Environment Variables. Superglobal variables (see Table 4.4) are accessible everywhere within a script and within functions. They are special variables provided by PHP to help you manage HTML forms, cookies, sessions, and files, and to get information about your environment and server.

Table 4.4 Some Superglobal Variables

Name	Meaning
<code>\$GLOBALS</code>	An array of all global variables
<code>\$_SERVER</code>	Contains server variables (e.g., <code>REMOTE_ADDR</code>)
<code>\$_GET</code>	Contains form variables sent through GET method
<code>\$_POST</code>	Contains form variables sent through POST method
<code>\$_COOKIE</code>	Contains HTTP cookie variables
<code>\$_FILES</code>	Contains variables provided to the script via HTTP post file uploads
<code>\$_ENV</code>	Contains the environment variables
<code>\$_REQUEST</code>	A merge of the GET variables, POST variables, and cookie variables
<code>\$_SESSION</code>	Contains HTTP variables registered by the session module

4.2.10 Managing Variables

You might want to find out if a variable has been declared, you might want to delete one that has been set, or check to see if one that is set is not empty or is a string, number, scalar, and so on. PHP provides a number of functions (see Table 4.5) to help you manage variables.

Table 4.5 Functions for Managing Variables

Function	What It Returns
<code>isset()</code>	True if variable has been set.
<code>empty()</code>	True if variable is empty: <code>" "</code> (an empty string) <code>"0"</code> (a string) <code>0</code> (an integer)
<code>is_bool()</code>	True if variable is boolean; that is, contains <code>TRUE</code> or <code>FALSE</code> .
<code>is_callable()</code>	True if variable is assigned the name of a function or an object.
<code>is_double()</code> , <code>is_float()</code> , <code>is_real()</code>	True if variable is a floating-point number (e.g., <code>5.67</code> or <code>.45</code>).
<code>is_int</code> , <code>is_integer</code> , <code>is_long</code>	True if a variable is assigned a whole number.
<code>is_null()</code>	True if a variable was assigned the <code>NULL</code> value.
<code>is_numeric()</code>	True if the variable was assigned a numeric string value or a number.
<code>is_object()</code>	True if the variable is an object.
<code>is_resource()</code>	True if the variable is a resource.
<code>is_scalar()</code>	True if the value was assigned a single value, such as a number (e.g., <code>"555"</code> a string, or a boolean, but not an array or object).
<code>is_string()</code>	True if a variable is a string of text (e.g., <code>"hello"</code>).
<code>unset()</code>	Unsets or destroys a list of values.

The `isset()` Function. The `isset()` function returns true if a variable has been set and false if it hasn't. If the variable has been set to `NULL` or has no value, it returns false. If you want to see if a variable has been set to `NULL`, use the `is_null()` function. To ensure that a variable has an initial value, the `isset()` function can be used to set a default value. See Examples 4.17 and 4.18.

FORMAT

```
bool isset ( variable, variable, variable .... );
```

Example:

```
$set = isset( $name );      // returns true or false
print isset($a, $b, $c);   // prints 1 or nothing
```

EXAMPLE 4.17

```
<html><head><title>Testing Variables</title></head>
<body bgcolor="#66CC66">
The <b>isset()</b> function returns a boolean value. <br />
If one or more variables exist and have a value, true is returned;
otherwise false.
<font face="verdana" size="+1">
<p />
<?php
1   $first_name="John"; $middle_name=" "; $last_name="Doe";
2   $age;
3   $state=NULL;
4   print 'isset($first_name,$middle_name,$last_name) : ' .
      isset($first_name,$last_name) . "<br />";
5   print 'isset($age) : ' . isset($age) . "<br />";
   print 'isset($city) : ' . isset($city) . "<br />";
   print 'isset($state) : ' . isset($state) . "<br />";
?>
</body>
</html>
```

EXPLANATION

- 1 Three variables are assigned string values; `$middle_name` is assigned the empty string, a set of double quotes containing no text.
- 2 `$age` is declared but has not been assigned any value yet, which evaluates to `NULL`.
- 3 `$state` is declared and assigned the value `NULL`, which means it has no value.
- 4 The `isset()` function returns true if a variable has been set and given a non-null value. In this case, all three variables have a value, even the variable assigned an empty string. If true, 1 is displayed; if false, 0 or nothing is displayed.
- 5 Because `$age` was not given any value, it is implied to be null, and `isset()` returns false. `$city` was never even declared, and `$state` was assigned `NULL`; `isset()` returns false. If you want to check explicitly for the `NULL` value (case insensitive), use the built-in `is_null()` function (see Figure 4.19).

4.2 Variables

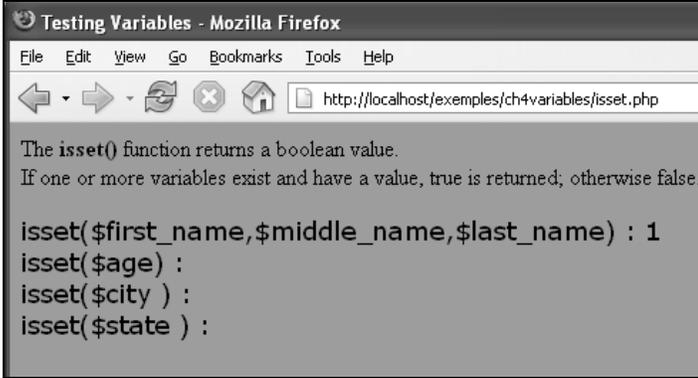


Figure 4.19 Is the variable set? Output from Example 4.17.

EXAMPLE 4.18

```

<html><head><title>Give a Variable a Default Value</title></head>
<body bgcolor="66C68">
<font face="verdana" size="+1">
<p>
<?php
1     if ( ! isset($temp) ) { $temp = 68 ; } // Sets a default value
2     echo "The default temperature is $temp degrees.<br />";
?>
</p>
</body>
</html>

```

EXPLANATION

- 1 The `isset()` function returns true if `$temp` has been set. The `!` operator, the unary “not” operator, reverses the boolean result. The expression reads, “if `$temp` is not set, define it with a value of 68.”
- 2 The `echo` statement displays the default value in the browser (see Figure 4.20).



Figure 4.20 Setting a default value. Output from Example 4.18.

The empty() Function. The `empty()` function returns true if a variable does not exist, or exists and has been assigned one of the following: an empty string " ", 0 as a number, "0" as a string, NULL, or no value at all. Example 4.19 demonstrates the use of the `empty()` function.

FORMAT

```
boolean empty ( variable );
```

Example:

```
if ( empty($result) ){
print "\$result either doesn't exist or is empty\n";
```

EXAMPLE 4.19

```
<html><head><title>Testing Variables</title></head>
<body bgcolor="66C66">
The <b>empty()</b> function returns a boolean value. <br />
If a variable doesn't exist or is assigned the empty string,
0, <br />or "0", NULL, or hasn't been assigned any value;
returns true, otherwise false.
<font face="verdana" size="+1">
<p>

<?php
1   $first_name=""; $last_name=" ";
2   $age=0;
3   $salary="0";
4   $state=NULL;
   print 'empty($first_name) : ' . empty($first_name) . "<br />";
   print 'empty($last_name) : ' . empty($last_name) . "<br />";
   print 'empty($age) : ' . empty($age) . "<br />";
   print 'empty($salary) : ' . empty($salary) . "<br />";
   print 'empty($state) : ' . empty($state) . "<br />";
?>

</p>
</body>
</html>
```

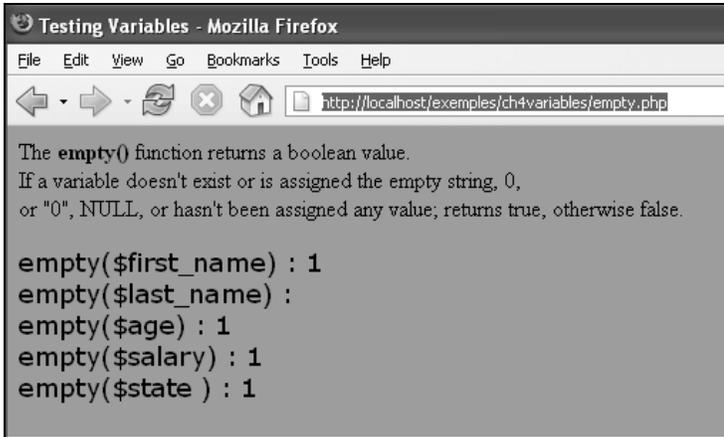


Figure 4.21 Is the variable empty? Output from Example 4.19.

The unset() Function. The `unset()` function (technically a language construct) unsets or destroys a given variable. It can take a varied number of arguments and behaves a little differently within functions (see Chapter 9, “User-Defined Functions”). As of PHP 4, it has no return value and is considered a statement.

FORMAT

```
void unset ( mixed var [, mixed var [, mixed ...]] )
```

Example:

```
unset($a, $b); // unsets the variables
```

EXAMPLE 4.20

```
<html><head><title>Testing Variables</title></head>
<body bgcolor="66C66">
The <b>unset()</b> function destroys a variable. <br />
<font face="verdana" size="+1">
<p>
<?php
1     $first_name="John";
     $last_name="Doe";
     $age=35;
2     unset($first_name, $last_name);
     print 'After unset() was used, isset($first_name,$last_name) '.
3     isset($first_name,$last_name). "returns false.<br />";
?>
</p>
</body>
</html>
```

EXPLANATION

- 1 Two scalar variables are declared and assigned string values.
- 2 The built-in `unset()` function will destroy the variables listed as arguments.
- 3 The `isset()` function returns true if a variable exists, and false if it doesn't.

**Figure 4.22** Destroying variables.

4.2.11 Introduction to Form Variables

Now we are starting to get into what makes PHP so popular. As we mentioned in the introduction to this book, PHP was designed as a Web-based programming language to create dynamic Web content, to gather and manipulate information submitted by HTML forms. For now, because we are talking about variables, we will examine a simple form and how PHP collects and stores the form information in variables. Chapter 10, “More on PHP Forms,” provides a comprehensive discussion on HTML forms and introduces the special global arrays used to process them in your PHP scripts.

The `php.ini` File and `register_globals`. Before getting started, there are some issues to be aware of based on the version of PHP you are using. The PHP initialization file, called `php.ini`, contains a directive called `register_globals`. Older versions of PHP (prior to 4.2.0) had this directive turned to “On” as the default, allowing PHP to create simple variables from form data. Since then, `register_globals` has been set to “Off” to avoid potential security breaches. If using PHP 5, you will have to turn this feature on before PHP can directly assign form input to simple global variables, or the data must be extracted programatically. We discuss both ways to do this in the following section. The next excerpt is taken from the PHP 5 `php.ini` file, showing the line where `register_globals` is set. The default is “Off” and you should really try to adhere to this setting.

From the `php.ini` file:

```
; You should do your best to write your scripts so that they do not require
; register_globals to be on; Using form variables as globals can easily lead
; to possible security problems, if the code is not very well thought of.
register_globals = Off
```

If you do not set `register_globals` to “On,” add the following line to your PHP program:

```
extract($_REQUEST);
```

The `$_REQUEST` superglobal array contains the information submitted to the server from the HTML form. After extracting this information, PHP will create simple variables corresponding to the form data as shown in Example 4.24. In Chapter 10, “More on PHP Forms,” all aspects of extracting form data are discussed in detail. For now, assume `register_globals` is set to “On.”

How PHP Handles Form Input. For each HTML form parameter, PHP creates a global variable by the same name and makes it available to your script. For example, consider this HTML input type for two text fields:

```
<input type="text" name="your_name">
<input type="text" name="your_phone">
```

If you have a text field named “`your_name`”, PHP will create a variable called `$your_name`. And if you have another text field named “`your_phone`”, PHP will in turn, create a variable called `$your_phone`. The values assigned to the PHP variables are the same values the user entered in the HTML text fields when filling out the form.

Example 4.21 illustrates a simple HTML form consisting of two fields. The form starts with the opening `<form>` tag. The `ACTION` attribute of the form tag is assigned the name of the PHP script that will handle the form input: `<form ACTION="php script">`.

After the user fills out the form (see Figure 4.25) and presses the submit button, the values that he or she typed in the text boxes will be sent to the PHP script (see Example 4.22). The browser knows *where* to send the data based on the `ACTION` attribute of the `<form>` tag, but it also needs to know *how* to send the form data to the server. The *how*, or method, is also an attribute of the `<form>` tag, called the `METHOD` attribute. There are two popular HTTP methods used to send the form information to the server—the `GET` method (default) and the `POST` method. Because the `GET` method is the default, you don’t have to explicitly assign it as an attribute. The browser just assumes that is the method you are using. The `GET` method tells the browser to send a URL-encoded string, called the query string, to the server. It attaches this encoded query string to the end of the URL in the browser’s location box, prepended with a `?`. It is the method used when doing searches or handling static pages and `GET` query strings are limited in size (see Figure 4.23).

If using the `POST` method, the `METHOD` attribute must be added to the HTML `<form>` tag `METHOD="POST"` (case insensitive). With the `POST` method, the browser sends an encoded message body in the HTTP header to the server so that it doesn't appear in the URL. It is not limited in size, but it can't be bookmarked or reloaded, and does not appear in the browser's history (see Figure 4.24).

When PHP gets the form input from the server, it takes care of decoding the query string or message body and assigning the respective input values to PHP variables as shown in Example 4.21. (For a complete discussion of the differences between the `GET` and `POST` methods, see <http://www.cs.tut.fi/~jkorpela/forms/methods.html>.)

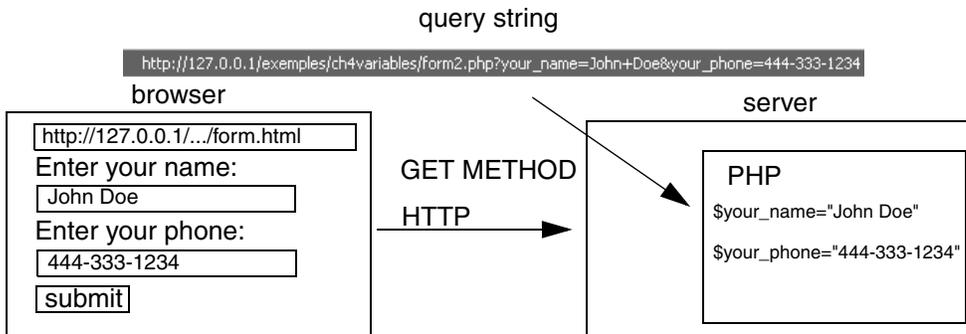


Figure 4.23 The `GET` method sends form input in the URL.

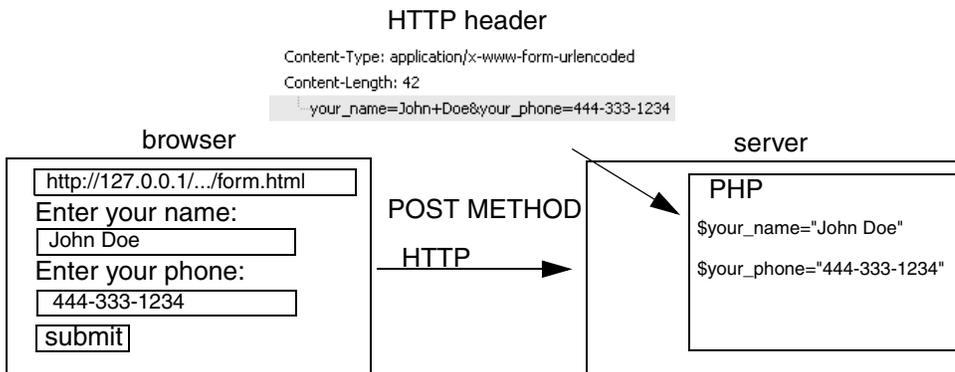


Figure 4.24 The `POST` method sends form input in an HTTP header.

EXAMPLE 4.21

```
<html>
<head>
<title>Simple HTML Form</title>
</head>
<body bgcolor="lightblue"><font size="+1">
1   <form action="http://localhost/exemples/ch4variables/
      form_example.php" />
      <p>
      please enter your name: <br />
2   <input type="text" size=30 name="your_name" />
      <br />
3   please enter your phone number: <br />
      <input type="text" size=30 name="your_phone" />
      <br />
4   <input type=submit value="submit" />
5   </form>
</body>
</html>
```

EXPLANATION

- 1 The HTML `<form>` tag starts the form. The URL of the script that will handle the form data is assigned to the `action` attribute. The “method” on how the data will be transmitted is assigned to the `method` attribute. Because the `GET` method is the default method for transmitting data to the server, you do not have to explicitly assign it as an attribute. This example is using the `GET` method.
- 2, 3 The HTML input type is a text box that is set to hold 50 characters. One is named “your_name” and the other is named “your_phone”.
- 4 After the user enters his or her name and phone number in the respective text boxes, and presses the submit button (see Figure 4.25), the browser will collect and URL encode the input data, then send it to the server. The server will hand it to the PHP script listed (see Example 4.22) in the `action` attribute on line 1. When PHP gets the input data, it will decode it, and create a variable called `$your_name` (the name of the first text box) and a variable called `$your_phone` (the name of the second text box) and give it the values that were entered in the form by the user.
- 5 This `</form>` tag ends the HTML form.



Figure 4.25 The HTML form has been filled out by a user.

EXAMPLE 4.22

(The PHP Script)

```
<?php
extract($_REQUEST);
1 print "Your phone number is $your_name. <br />";
  print "Your phone number is $your_phone.";
?>
```

or in the HTML document use the PHP shortcut tags:

```
<html><head><title>Testing Variables</title></head>
<body>
2 Your phone number is <?=$your_name?> and your phone number is
  <?=$your_phone?>
</body></html>
```

EXPLANATION

- 1 The browser bundles up the input data, encodes it, and attaches it as a query string to the URL as:

```
?http://localhost/exemples/ch4variables/form_example.php?
your_name=Samual+B.+Johnson+Jr.&your_phone=222-444-8888
```

PHP decodes the query string; that is, it removes the + signs and & and any other encoding characters, and then creates global variables, called `$your_name` and `$your_phone`, and assigns values based on the user input. In this example, the values of the variables are printed as part of the PHP script.

- 2 You can also use the shortcut tags within the HTML document to display the value of the PHP variables. The output is displayed in the browser, as shown in Figure 4.26.



Figure 4.26 Output from the PHP script in Example 4.22. The input data is appended to the URL after the ?.

Extracting the Data by Request. In the previous example, we used the default `GET` method to send form input to the server. We also assumed the `register_globals` in the `php.ini` file was turned “On,” allowing PHP to create simple global variables with the form data. In the following example, we assume that `register_globals` is turned “Off” (the recommended and default setting) and that the method is `POST`, the more commonly used method when working with form input, as shown in Figure 4.24. This method sends the form input as a message body in the HTTP header to the server and does not append the data to the URL in the browser. Even though the input data is sent using a different method, it is received in the same URL-encoded format. When `register_globals` is turned off, PHP provides special variables, called arrays, to store the form information. Because we do not cover arrays until Chapter 8, “Arrays,” we will create simple variables to hold input data from the form, but first must explicitly extract the data from a special global array called `$_REQUEST`. This special array contains all of the input data for both `GET` and `POST` methods, and once it is extracted will be assigned to PHP variables with the same name as was given to the corresponding input devices in the HTML form.

EXAMPLE 4.23

(The HTML Form Source)

```
<html>
<head>
<title>First HTML Form</title>
</head>
<body bgcolor="lightblue"><font size="+1">
1   <form action="/phpforms/form1.php" method="POST">
   <p>
   Please enter your name: <br />
2   <input type="text" size=50 name="your_name">
   <p>
   Please enter your phone: <br />
3   <input type="text" size=50 name="your_phone">
   <p>
   Please enter your email address:<br />
4   <input type="text" size=50 name="your_email_addr">
   <p>
```

EXAMPLE 4.23 (CONTINUED)

```
5     <input type=submit value="submit">
      <input type=reset value="clear">
6     </form>
      <hr>
      </body>
      </html>
```

EXPLANATION

- 1 The HTML `<form>` tag starts the form. The URL of the script that will handle the form data is assigned to the `action` attribute. The “method” on how the data will be transmitted is assigned to the `method` attribute. The `POST` method is used here. This is the most common method for processing forms. The form input is sent in an HTTP header to the server.
- 2, 3, 4 The input devices are three text boxes for accepting user input. The name attribute is assigned the names of the respective boxes, `your_name`, `your_phone`, and `your_email` (see Figure 4.27). These same names will be used as variable names in the PHP program, `/phpforms/form1.php`, listed in the forms action attribute.
- 5 When the user presses the submit button, the form input is encoded and sent to the server. The form input will not be visible in the URL as it is with the `GET` method.
- 6 This marks the end of the form.

**Figure 4.27** Data has been entered into the HTML form.

EXAMPLE 4.24

(The PHP program)

```

<html><head><title>Processing First Form</title>
</head>
<body bgcolor = "lightgreen"><font size="+1">
<h2>Here is the form input:</h2>
<?php
1  extract($_REQUEST, EXTR_SKIP); // Extracting the form input
   print "Welcome to PHP $your_name<br />"; // register_globals
                                       // is off
   print "Can I call you at $your_phone<br />";
   print "Is it ok to send you email at $your_email_addr<br />";
?>
</body>
</html>

```

EXPLANATION

- 1 If the `register_globals` directive in the `php.ini` file is set to "Off," the built-in PHP `extract()` function can be used to get the form input stored in `$_REQUEST`, an array that contains input received from both GET and POST methods. The `extract()` function will convert the input into variables of the same name as the input devices in the HTML file. The `EXTR_SKIP` flag ensures that if there is a collision, that is, you have already defined a variable with the that name somewhere in your PHP program, it won't be overwritten.
- 2 The variables `$your_name`, `$your_phone`, and `$your_email_addr` were created by the `extract()` function and named after the text boxes originally named in the HTML form. The output is displayed in the browser, as in Figure 4.28.



Figure 4.28
After PHP processes the input data from the form.

Predefined Variables. PHP provides a number of predefined variables (see Table 4.6 and Figure 4.29), some that are not fully documented because they depend on which server is running, its configuration, and so on. Some are defined in the `php.ini` file. These variables describe the environment, server, browser, version number, configuration file, and so on.

Table 4.6 Predefined Variables^a

Variable	What It Does
<code>AUTH_TYPE</code>	If running the Apache server as a module, this is set to the authentication type.
<code>DOCUMENT_ROOT</code>	The full path of the Web's document root, normally where HTML pages are stored and defined in the server's configuration file.
<code>HTTP_USER_AGENT</code>	Identifies the type of Web browser to the server when it requests a file.
<code>HTTP_REFERER</code>	The full URL of the page that contained the link to this page. Of course if there isn't a referring page, this variable would not exist.
<code>REMOTE_ADDRESS</code>	The remote IP address of the client machine that requested the page.

- a. See the full list of predefined variables at http://www.phpfreaks.com/PHP_Reference/Predefined-Variables/8.php

There many more predefined variables; which ones are set depends on your PHP configuration. The function `phpinfo()` can be used to retrieve built-in variables that have been set.

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

4.3 Constants



PHP Variables

Variable	Value
_SERVER["COMSPEC"]	C:\WINDOWS\system32\cmd.exe
_SERVER["DOCUMENT_ROOT"]	c:/wamp/www
_SERVER["HTTP_ACCEPT"]	text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
_SERVER["HTTP_ACCEPT_CHARSET"]	ISO-8859-1,utf-8;q=0.7,*;q=0.7
_SERVER["HTTP_ACCEPT_ENCODING"]	gzip,deflate
_SERVER["HTTP_ACCEPT_LANGUAGE"]	en-us,en;q=0.5
_SERVER["HTTP_CONNECTION"]	keep-alive
_SERVER["HTTP_HOST"]	localhost
_SERVER["HTTP_KEEP_ALIVE"]	300
_SERVER["HTTP_USER_AGENT"]	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.5) Gecko/20041107 Firefox/1.0 (ax)
_SERVER["PATH"]	C:\Perl\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\system32\WBEM;C:\WINDOWS\system32\Program Files\Adaptec Shared\System;C:\Program Files\QuickTime\QTSystem\
_SERVER["REMOTE_ADDR"]	127.0.0.1
_SERVER["REMOTE_PORT"]	4061
_SERVER["SCRIPT_FILENAME"]	c:/wamp/www/first.php
_SERVER["SERVER_ADDR"]	127.0.0.1
_SERVER["SERVER_ADMIN"]	webmaster@localhost
_SERVER["SERVER_NAME"]	localhost
_SERVER["SERVER_PORT"]	80
_SERVER["SERVER_SIGNATURE"]	<ADDRESS>Apache/1.3.33 Server at localhost Port 80</ADDRESS>
_SERVER["SERVER_SOFTWARE"]	Apache/1.3.33 (Win32) PHP/5.0.3
_SERVER["SystemRoot"]	C:\WINDOWS
_SERVER["WINDIR"]	C:\WINDOWS
_SERVER["GATEWAY_INTERFACE"]	CGI/1.1
_SERVER["SERVER_PROTOCOL"]	HTTP/1.1

Figure 4.29 PHP variables (partial output from the phpinfo() function).

4.3 Constants

“The only thing constant in life is change.”
 —Francois de la Rouchefoucauld, French classical author

Some real-world constants, such as pi, the speed of light, the number of inches in a foot, and the value of midnight, are values that don't change. PHP not only provides its own predefined constants but lets you create your own. Using constants makes it easy to write and maintain your programs.

4.3.1 What Is a Constant?

Unlike variables, a constant is a value that, once set, cannot be changed or unset during the execution of your script. An example of a constant is the value of pi or the version of PHP you are using. Constants are very useful because they are visible throughout a

program (global in scope) and their values don't change; for example, a constant might be defined for the document root of your server, the name of your site, or the title, author, and copyright year of this book. Once defined, those values are fixed.

You can define constants at the top of your program or in another file that can be included in your script. (See the `require()` and `include()` functions discussed in “Managing Content with Include Files” on page 487.) Later if a constant value needs to be modified, once you change its value in the program, then when the program is executed, the new value will be reflected wherever the constant is used throughout the program, thus facilitating program maintenance.

4.3.2 Creating Constants with the `define()` Function

PHP constants are defined as words, and by convention, capitalized. Like variables, they are case sensitive and consist of uppercase and lowercase letters, numbers, and the underscore. Like variables, they cannot start with a number.

Unlike variables, constants are *not* preceded by a dollar sign and are *not* interpreted when placed within quotes.

Constants are global in scope, meaning they are available for use anywhere in a PHP script.

The only way that you can create a constant is with the PHP built-in `define()` function. Only a single, scalar value can be assigned to a constant, including strings, integers, floats, and booleans.

The `define()` function creates a named constant. The first argument is the name of the constant and the second argument is the value that will be assigned to it. Constants are normally case sensitive, but you can use an optional third argument of `TRUE` to turn off case sensitivity.

FORMAT

```
bool define ( string name, mixed value [, bool case_insensitive] )
```

Example:

```
// defines document root
define( 'DOC_ROOT', '/http://artemis/~ellie/public_html' );
// defines the include folder
define( 'INCLUDES', DOC_ROOT.'../includes' );
```

EXAMPLE 4.25

```
<?php
1  define('ISBN', "0-13-140162-9");
2  define('TITLE', "JavaScript by Example" );
3  if ( defined('ISBN') and defined('TITLE')){
4      print ISBN . "<br />";
      print TITLE . "<br />";
5  define('TITLE', "PHP by Example"); // Can't change TITLE, and
                                     // can't redefine it.
6  print TITLE;
?>
```

EXPLANATION

- Two constants are defined, `ISBN` and `TITLE`, the first argument to the function. The second argument is the value being assigned to each of the constants. Once set, the only way to change a constant is to redefine it with the `define()` function.
- The `define()` function returns `TRUE` if the named constant has been defined. The expression reads, “if the constant `ISBN` and the constant `TITLE` have both been defined, proceed to line 3.”
- Notice that the constants are not quoted. If they are quoted, their values will not be printed, but just the words `ISBN` and `TITLE`.
- You cannot redefine a constant like this. If you want to modify the value, you must go back into the program and change the original definition on line 2.
- The constant `TITLE` was unaffected by line 5. By definition, a constant cannot be changed or unset. The output of this program is shown in Figure 4.30.

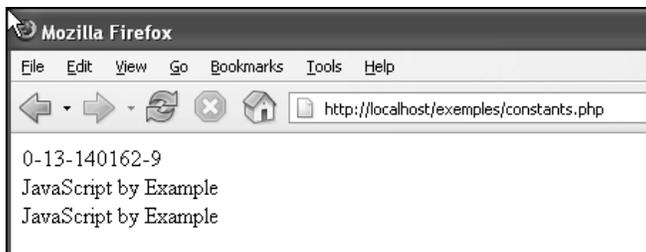


Figure 4.30 User-defined constants. Output from Example 4.25.

The `defined()` function checks whether a constant has been set. It returns `TRUE` if the constant has been defined; otherwise, `FALSE`.

4.3.3 The constant() Function

The `constant()` function returns that value of a constant. This function can be helpful if you don't know the name of the constant because its name was stored in a variable or was returned from a function.

FORMAT

```
mixed constant ( string name )
```

Example:

```
define (ISBN, "0-13-140162-9");
$value=constant(ISBN); // Returns 0-13-140162-9
```

4.3.4 Predefined and “Magic” Constants

PHP comes with a number of predefined constants shown in Table 4.6. They provide information that doesn't change such as the name of the script file, the version of PHP and the operating system, and so on.

There are five predefined constants called magic constants (see Table 4.7). These are constants that change depending on how they are used in a program. They cannot be enclosed in quotes and are not case sensitive. The name of the constant is enclosed in *two* underscores on both sides.

Table 4.7 Magic Constants

Name	Description
<code>__LINE__</code>	The current line number of the file.
<code>__FILE__</code>	The full path and filename of the file. If used inside an include, the name of the included file is returned.
<code>__FUNCTION__</code>	The function name (added in PHP 4.3.0). As of PHP 5 this constant returns the function name as it was declared (case sensitive). In PHP 4 its value is always lowercased.
<code>__CLASS__</code>	The class name (added in PHP 4.3.0). As of PHP 5 this constant returns the class name as it was declared (case sensitive). In PHP 4 its value is always lowercased.
<code>__METHOD__</code>	The class method name (added in PHP 5.0.0). The method name is returned as it was declared (case sensitive).

PHP has several special built-in constants described in Table 4.8.

Table 4.8 Built-In Constants

Name	Description
PHP_VERSION	The version of the PHP parser currently running
PHP_OS	The operating system of the server on which the PHP parser is running
PHP_OS	The name of the operating system on which the PHP parser is executing; e.g., Linux
TRUE	A true value.
FALSE	A false value.

The script in Example 4.26 shows how the predefined constants can be used to give information to the browser. Its output is displayed in Figure 4.31.

EXAMPLE 4.26

```
<?php
// Using PHP built-in constants
echo "PHP version = " . PHP_VERSION . "<br />";
echo "Server operating system = " . PHP_OS . "<br />";
echo "Current file name= " . __FILE__ . "<br />";
echo "Current line number= " . __LINE__ . "<br />";
echo "TRUE = " . TRUE . "<br />";
echo "false = " . FALSE . "<br />";
?>
```

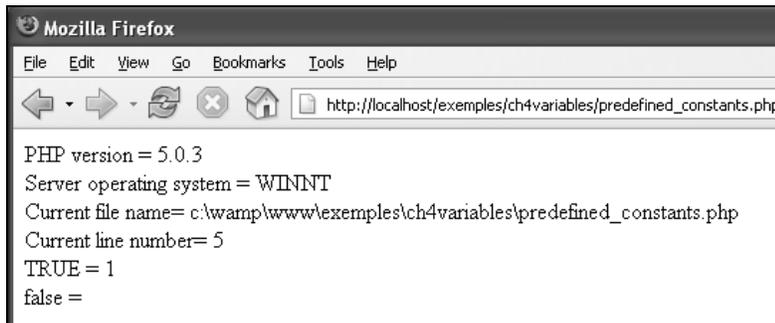


Figure 4.31 Predefined constants. Output from Example 4.26.

4.4 Chapter Summary

4.4.1 What You Should Know

Now that you have finished this chapter you should be able to answer the following questions:

1. What are the PHP basic data types?
2. What is the `gettype()` function?
3. What is the difference between a *scalar* and a *composite* data type?
4. What is the difference between a *variable* and a *constant*?
5. When do you need double quotes? Single quotes?
6. How can you see a backslash interpreted in the browser?
7. How do you concatenate two strings?
8. Why would you use a *here-doc*?
9. What data type is represented by true or false? Are true and false case sensitive?
10. What is `NULL`?
11. Is `"$_over-out"` a valid variable name? Why or why not?
12. Is it mandatory to initialize a variable?
13. What function can you use to tell if a variable exists?
14. How do you get rid of a variable?
15. What is meant by *scope*?
16. What is the function of the `register_globals` directive? In what file is it located? Is it on or off in your version of PHP?
17. What are *form* variables?
18. What is the difference between the `GET` and `POST` methods?
19. What is the value of `$_REQUEST`?

20. How do you create a constant?
21. Why are constants useful?
22. What is a “magic constant”?

4.4.2 What’s Next?

Another important chapter basic to all programming languages, Chapter 5, “Operators,” covers PHP’s rich set of operators and how to use them to manipulate data; for example, how to perform arithmetic on numbers, compare strings and numbers, test equality, combine expressions, and test them with logical operators, bitwise operations, and more.

CHAPTER 4 LAB

1. Create a string variable to contain *one* string that will be printed in bold text on two lines as:

“Ouch! That’s not nice,” snickered Mrs. O’Connell.
“You mustn’t do that, Mr. O’Connell.”

2. What does the following code print?

```
print ("What a <b>perfect</b> day.");
print ("3" + 2);
print ("5 dogs" + "6 cats" . "10 birds");
print ("<pre>\t\tIt's been real!\n</pre>");
print ("\t\tLater, dude.\n");
```

3. Create four variables that contain an integer, float, string, and boolean value. Print the value of each variable and its data type in an HTML table. Use `gettype()`.
4. In Exercise 1 you created the following PHP output. Now we will rewrite this script to include user-defined variables. Where you see `< >` in the example, input your variable values.

Print the output in the browser. Can you format the output so that a dollar sign appears before the money values and format the numbers with a precision of two places to the right of the decimal point (e.g., \$410.00)? Hint: See <http://www.htmlite.com/php011.php>.

Check to see if the variables are set (`isset()`) before displaying them.

Set variables as follows;

```
$sales    = 190000;
$rent     = 25000;
$salary   = 37500;
$supplies = 410;
$total    = $rent + $salary + $supplies; // Addition
$operating_income = $sales - $exp_total; // Subtraction
$net_income = $operating_income * 0.60; // Multiplication
```

Book Store Operating Costs

=====

Sales: <print variable values here>

Expenses:

Rent: < >

Salary:

Supplies:

Total: < >

Operating income: < >

Income after taxes (net): < >

=====

5. Use the shortcut PHP tags, `<?= ?>`, within the HTML document to display the variables in the previous exercise. (Check the `php.ini` file to see if shortcut tags are allowed and if set to “Off”, turn them “On”.)
6. Create an HTML form that contains three text boxes, one for the user’s name, one for his cell phone number, and one for his e-mail address. Write a PHP script to display the output.
7. Rewrite Exercise 4 so that the user enters input into an HTML form, and write a PHP script to process the form and display the output as it did in Exercise 4.
8. Write a PHP script that displays the values entered into the form fields. Add a constant to the following script that will define a `COPY_RIGHT` constant containing your `SITE_NAME` with the copyright symbol appended (concatenated) to it. Display the constants and their corresponding values in an HTML table. Hint: See <http://www.desilva.biz/php/constants.html>.

```
<?php>
// Define your site name, since it does NOT change
// anywhere within your script.
define( 'SITE_NAME', 'Your site' );

// Define the current year, possibly to use in your copyright
// statement or for 'date' calculations.
define( 'THIS_YEAR', date('Y') );
?>
```