19

- Introduction to PHP

OBJECTIVES

In this chapter you will:

- Manipulate data of various types.
- Use operators, arrays and control statements.
- Use regular expressions to search for text that matches a patterns.
- Construct programs that process form data.
- Store data on the client using cookies.
- Create programs that interact with MySQL databases.

19.1 Introduction

- PHP, or PHP: Hypertext Preprocessor, has become the most popular server-side scripting language for creating dynamic web pages.
- PHP is open source and platform independent—implementations exist for all major UNIX, Linux, Mac and Windows operating systems. PHP also supports a large number of databases.

19.2 A Simple PHP Program

- The power of the web resides not only in serving content to users, but also in responding to requests from users and generating web pages with dynamic content.
- PHP code is embedded directly into text-based documents, such as HTML, though these script segments are interpreted by a server *before* being delivered to the client.
- PHP script file names end with `.php`.
- In PHP, code is inserted between the scripting delimiters `<?php` and `?>`. PHP code can be placed anywhere in HTML5 markup, as long as the code is enclosed in these delimiters.

19.2 A Simple PHP Program (Cont.)

- Variables are preceded by a $ and are created the first time they're encountered.
- PHP statements terminate with a semicolon (;).
- Single-line comments which begin with two forward slashes (//) or a pound sign (#). Text to the right of the delimiter is ignored by the interpreter. Multiline comments begin with delimiter /* and end with delimiter */.
- When a variable is encountered inside a double-quoted ("") string, PHP interpolates the variable. In other words, PHP inserts the variable's value where the variable name appears in the string.
- All operations requiring PHP interpolation execute on the server before the HTML5 document is sent to the client.
- PHP variables are loosely typed—they can contain different types of data at different times.

Common Programming Error 19.1

Variable names in PHP are case sensitive. Failure to use the proper mixture of cases to refer to a variable will result in a logic error, since the script will create a new variable for any name it doesn't recognize as a previously used variable.

Common Programming Error 19.2

Forgetting to terminate a statement with a semicolon (;) is a syntax error.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.1: first.php -->
4  <!-- Simple PHP program. -->
5  <html>
6  <?php
7     $name = "Paul"; // declaration and initialization
8  ?><!-- end PHP script -->
9     <head>
10       <meta charset = "utf-8">
11       <title>Simple PHP document</title>
12    </head>
13    <body>
14       <!-- print variable name's value -->
15       <h1><?php print( "Welcome to PHP, $name!" ); ?></h1>
16    </body>
17 </html>
```

Fig. 19.1 | Simple PHP program. (Part 1 of 2.)

Fig. 19.1 | Simple PHP program. (Part 2 of 2.)

Type	Description
`int`, `integer`	Whole numbers (i.e., numbers without a decimal point).
`float`, `double`, `real`	Real numbers (i.e., numbers containing a decimal point).
`string`	Text enclosed in either single (`''`) or double (`""`) quotes. [*Note:* Using double quotes allows PHP to recognize more escape sequences.]
`bool`, `boolean`	`true` or `false`.
`array`	Group of elements.
`object`	Group of associated data and methods.
`resource`	An external source—usually information from a database.
`NULL`	No value.

Fig. 19.2 | PHP types.

19.3 Converting Between Data Types

- Type conversions can be performed using function `settype`. This function takes two arguments—a variable whose type is to be changed and the variable's new type.
- Variables are typed based on the values assigned to them.
- Function `gettype` returns the current type of its argument.
- Calling function `settype` can result in loss of data. For example, doubles are truncated when they are converted to integers.
- When converting from a string to a number, PHP uses the value of the number that appears at the beginning of the string. If no number appears at the beginning, the string evaluates to 0.

19.3 Converting Between Data Types

- Another option for conversion between types is casting (or type casting). Casting does not change a variable's content—it creates a temporary copy of a variable's value in memory.
- The concatenation operator (`.`) combines multiple strings.
- A `print` statement split over multiple lines prints all the data that is enclosed in its parentheses.

```
<!DOCTYPE html>

<!-- Fig. 19.3: data.php -->
<!-- Data type conversion. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Data type conversion</title>
      <style type = "text/css">
         p      { margin: 0; }
         .head  { margin-top: 10px; font-weight: bold; }
         .space { margin-top: 10px; }
      </style>
   </head>
   <body>
      <?php
         // declare a string, double and integer
         $testString = "3.5 seconds";
         $testDouble = 79.2;
         $testInteger = 12;
      ?><!-- end PHP script -->

```

Fig. 19.3 | Data type conversion. (Part 1 of 4.)

```
23          <!-- print each variable's value and type -->
24          <p class = "head">Original values:</p>
25          <?php
26             print( "<p>$testString is a(n) " . gettype( $testString )
27                . "</p>" );
28             print( "<p>$testDouble is a(n) " . gettype( $testDouble )
29                . "</p>" );
30             print( "<p>$testInteger is a(n) " . gettype( $testInteger )
31                . "</p>" );
32          ?><!-- end PHP script -->
33          <p class = "head">Converting to other data types:</p>
34          <?php
35             // call function settype to convert variable
36             // testString to different data types
37             print( "<p>$testString " );
38             settype( $testString, "double" );
39             print( " as a double is $testString</p>" );
40             print( "<p>$testString " );
41             settype( $testString, "integer" );
42             print( " as an integer is $testString</p>" );
43             settype( $testString, "string" );
44             print( "<p class = 'space'>Converting back to a string results in
45                $testString</p>" );
46
```

Fig. 19.3 | Data type conversion. (Part 2 of 4.)

```
47          // use type casting to cast variables to a different type
48          $data = "98.6 degrees";
49          print( "<p class = 'space'>Before casting: $data is a " .
50             gettype( $data ) . "</p>" );
51          print( "<p class = 'space'>Using type casting instead:</p>
52             <p>as a double: " . (double) $data . "</p>" .
53             "<p>as an integer: " . (integer) $data . "</p>";
54          print( "<p class = 'space'>After casting: $data is a " .
55             gettype( $data ) . "</p>" );
56       ?><!-- end PHP script -->
57    </body>
58 </html>
```

Fig. 19.3 | Data type conversion. (Part 3 of 4.)

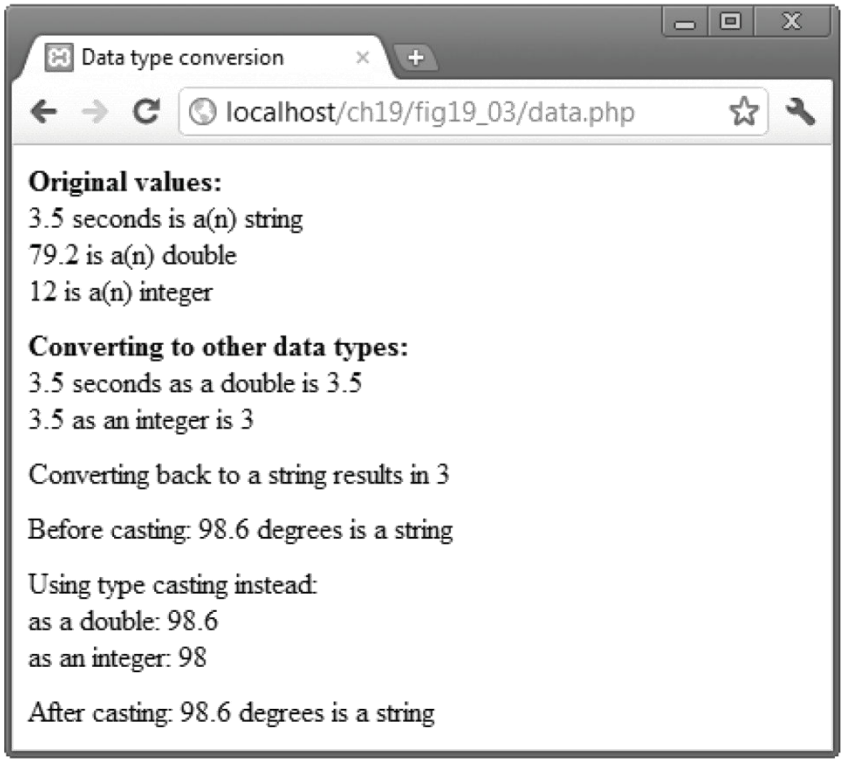

Fig. 19.3 | Data type conversion. (Part 4 of 4.)

Error-Prevention Tip 19.1

Function `print` can be used to display the value of a variable at a particular point during a program's execution. This is often helpful in debugging a script.

19.4 Arithmetic Operators

- Function `define` creates a named constant. It takes two arguments—the name and value of the constant. An optional third argument accepts a boolean value that specifies whether the constant is case insensitive—constants are case sensitive by default.
- Uninitialized variables have undefined values that evaluate differently, depending on the context. In a numeric context, it evaluates to 0. In contrast, when an undefined value is interpreted in a string context (e.g., $nothing), it evaluates to the string "undef".
- Keywords may not be used as function, method, class or namespace names.

Common Programming Error 19.3

Assigning a value to a constant after it's declared is a syntax error.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.4: operators.php -->
4  <!-- Using arithmetic operators. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <style type = "text/css">
9           p { margin: 0; }
10       </style>
11       <title>Using arithmetic operators</title>
12    </head>
13    <body>
14       <?php
15          $a = 5;
16          print( "<p>The value of variable a is $a</p>" );
17
18          // define constant VALUE
19          define( "VALUE", 5 );
20
21          // add constant VALUE to variable $a
22          $a = $a + VALUE;
23          print( "<p>Variable a after adding constant VALUE is $a</p>" );
24
```

Fig. 19.4 | Using arithmetic operators. (Part 1 of 4.)

```
25          // multiply variable $a by 2
26          $a *= 2;
27          print( "<p>Multiplying variable a by 2 yields $a</p>" );
28
29          // test if variable $a is less than 50
30          if ( $a < 50 )
31             print( "<p>Variable a is less than 50</p>" );
32
33          // add 40 to variable $a
34          $a += 40;
35          print( "<p>Variable a after adding 40 is $a</p>" );
36
37          // test if variable $a is 50 or less
38          if ( $a < 51 )
39             print( "<p>Variable a is still 50 or less</p>" );
40          elseif ( $a < 101 ) // $a >= 51 and <= 100
41             print( "<p>Variable a is now between 50 and 100,
42                inclusive</p>" );
43          else // $a > 100
44             print( "<p>Variable a is now greater than 100</p>" );
45
46          // print an uninitialized variable
47          print( "<p>Using a variable before initializing:
48             $nothing</p>" ); // nothing evaluates to ""
49
```

Fig. 19.4 | Using arithmetic operators. (Part 2 of 4.)

```
50          // add constant VALUE to an uninitialized variable
51          $test = $num + VALUE; // num evaluates to 0
52          print( "<p>An uninitialized variable plus constant
53             VALUE yields $test</p>" );
54
55          // add a string to an integer
56          $str = "3 dollars";
57          $a += $str;
58          print( "<p>Adding a string to variable a yields $a</p>" );
59       ?><!-- end PHP script -->
60    </body>
61 </html>
```

Fig. 19.4 | Using arithmetic operators. (Part 3 of 4.)

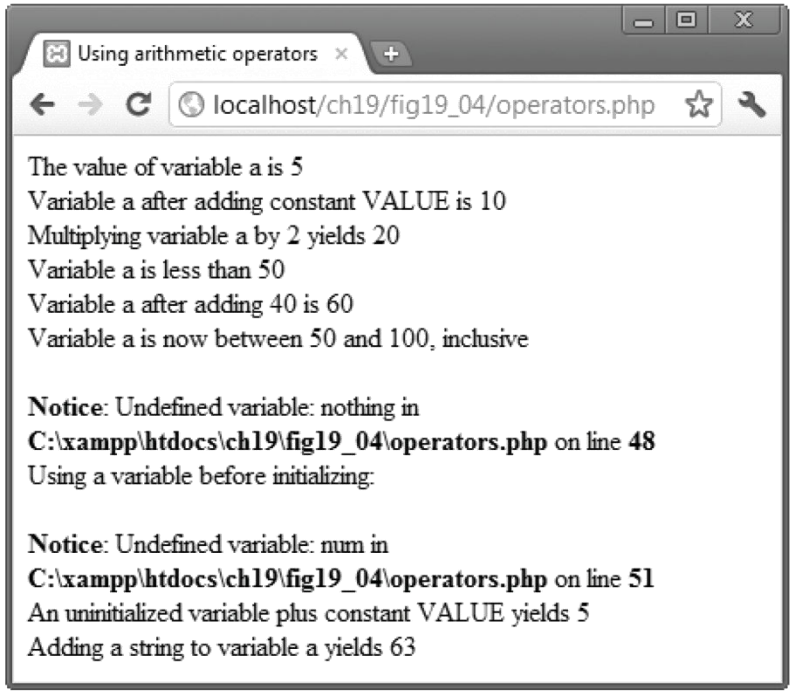

Fig. 19.4 | Using arithmetic operators. (Part 4 of 4.)

Error-Prevention Tip 19.2

Initialize variables before they're used to avoid subtle errors. For example, multiplying a number by an uninitialized variable results in 0.

PHP keywords				
abstract	and	array	as	break
case	catch	class	clone	const
continue	declare	default	do	else
elseif	enddeclare	endfor	endforeach	endif
endswitch	endwhile	extends	final	for
foreach	function	global	goto	if
implements	interface	instanceof	namespace	new
or	private	protected	public	static
switch	throw	try	use	var
while	xor			

Fig. 19.5 | PHP keywords.

Operator	Type	Associativity
`new` `clone`	constructor copy an object	none
`[]`	subscript	left to right
`++` `--`	increment decrement	none

`~`
`-`
`@`
`(`*type*`)`
	bitwise not
	unary negative
	error control
	cast
		right to left

`instanceof`		none
`!`	not	right to left

`*`
`/`
`%`
	multiplication
	division
	modulus
		left to right

Fig. 19.6 | PHP operator precedence and associativity. (Part 1 of 5.)

Operator	Type	Associativity
+	addition	left to right
-	subtraction	
.	concatenation	
<<	bitwise shift left	left to right
>>	bitwise shift right	
<	less than	none
>	greater than	
<=	less than or equal	
>=	greater than or equal	
==	equal	none
!=	not equal	
===	identical	
!==	not identical	
&	bitwise AND	left to right
^	bitwise XOR	left to right

Fig. 19.6 | PHP operator precedence and associativity. (Part 2 of 5.)

Operator	Type	Associativity
|	bitwise OR	left to right
&&	logical AND	left to right
||	logical OR	left to right
?:	ternary conditional	left to right

Fig. 19.6 | PHP operator precedence and associativity. (Part 3 of 5.)

Operator	Type	Associativity
=	assignment	right to left
+=	addition assignment	
-=	subtraction assignment	
*=	multiplication assignment	
/=	division assignment	
%=	modulus assignment	
&=	bitwise AND assignment	
|=	bitwise OR assignment	
^=	bitwise exclusive OR assignment	
.=	concatenation assignment	
<<=	bitwise shift left assignment	
>>=	bitwise shift right assignment	
=>	assign value to a named key	
and	logical AND	left to right
xor	exclusive OR	left to right
or	logical OR	left to right

Fig. 19.6 | PHP operator precedence and associativity. (Part 4 of 5.)

Operator	Type	Associativity
,	list	left to right

Fig. 19.6 | PHP operator precedence and associativity. (Part 5 of 5.)

19.5 Initializing and Manipulating Arrays

- PHP provides the capability to store data in arrays. Arrays are divided into elements that behave as individual variables. Array names, like other variables, begin with the $ symbol.
- Individual array elements are accessed by following the array's variable name with an index enclosed in square brackets ([]).
- *If a value is assigned to an array element of an array that does not exist, then the array is created.* Likewise, assigning a value to an element where the index is omitted appends a new element to the end of the array.
- Function count returns the total number of elements in the array.
- Function array creates an array that contains the arguments passed to it. The first item in the argument list is stored as the first array element (index 0), the second item is stored as the second array element and so on.

19.5 Initializing and Manipulating Arrays (Cont.)

- Arrays with nonnumeric indices are called associative arrays.
- You can create an associative array using the operator =>, where the value to the left of the operator is the array index and the value to the right is the element's value.
- PHP provides functions for iterating through the elements of an array.
- Each array has a built-in internal pointer, which points to the array element currently being referenced.
- Function `reset` sets the internal pointer to the first array element. Function `key` returns the index of the element currently referenced by the internal pointer, and function `next` moves the internal pointer to the next element.

19.5 Initializing and Manipulating Arrays (Cont.)

- The foreach statement, designed for iterating through arrays, starts with the array to iterate through, followed by the keyword as, followed by two variables—the first is assigned the index of the element and the second is assigned the value of that index's element. (If only one variable is listed after as, it is assigned the value of the array element.)

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.7: arrays.php -->
4  <!-- Array manipulation. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Array manipulation</title>
9        <style type = "text/css">
10          p    { margin: 0; }
11          .head { margin-top: 10px; font-weight: bold; }
12       </style>
13    </head>
14    <body>
15       <?php
16          // create array first
17          print( "<p class = 'head'>Creating the first array</p>" );
18          $first[ 0 ] = "zero";
19          $first[ 1 ] = "one";
20          $first[ 2 ] = "two";
21          $first[] = "three";
22
```

Fig. 19.7 | Array manipulation. (Part 1 of 4.)

```
23          // print each element's index and value
24          for ( $i = 0; $i < count( $first ); ++$i )
25             print( "Element $i is $first[$i]</p>" );
26
27          print( "<p class = 'head'>Creating the second array</p>" );
28
29          // call function array to create array second
30          $second = array( "zero", "one", "two", "three" );
31
32          for ( $i = 0; $i < count( $second ); ++$i )
33             print( "Element $i is $second[$i]</p>" );
34
35          print( "<p class = 'head'>Creating the third array</p>" );
36
37          // assign values to entries using nonnumeric indices
38          $third[ "Amy" ] = 21;
39          $third[ "Bob" ] = 18;
40          $third[ "Carol" ] = 23;
41
42          // iterate through the array elements and print each
43          // element's name and value
44          for ( reset( $third ); $element = key( $third ); next( $third ) )
45             print( "<p>$element is $third[$element]</p>" );
46
47          print( "<p class = 'head'>Creating the fourth array</p>" );
```

Fig. 19.7 | Array manipulation. (Part 2 of 4.)

```
48
49          // call function array to create array fourth using
50          // string indices
51          $fourth = array(
52             "January"   => "first",   "February" => "second",
53             "March"     => "third",   "April"    => "fourth",
54             "May"       => "fifth",   "June"     => "sixth",
55             "July"      => "seventh", "August"   => "eighth",
56             "September" => "ninth",   "October"  => "tenth",
57             "November"  => "eleventh","December" => "twelfth" );
58
59          // print each element's name and value
60          foreach ( $fourth as $element => $value )
61             print( "<p>$element is the $value month</p>" );
62       ?><!-- end PHP script -->
63    </body>
64 </html>
```

Fig. 19.7 | Array manipulation. (Part 3 of 4.)

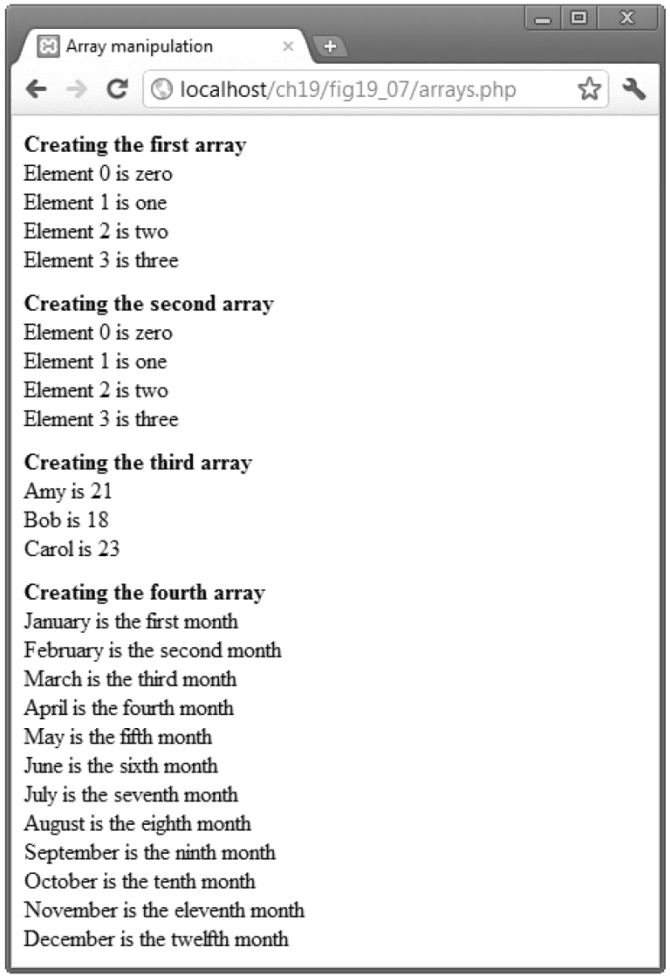

Fig. 19.7 | Array manipulation. (Part 4 of 4.)

19.6 String Comparisons

- Many string-processing tasks can be accomplished using the equality and relational operators (==, !=, <, <=, > and >=).
- Function `strcmp` compares two strings. The function returns -1 if the first string alphabetically precedes the second string, 0 if the strings are equal, and 1 if the first string alphabetically follows the second.

```
<!DOCTYPE html>

<!-- Fig. 19.8: compare.php -->
<!-- Using the string-comparison operators. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>String Comparison</title>
      <style type = "text/css">
         p { margin: 0; }
      </style>
   </head>
   <body>
      <?php
         // create array fruits
         $fruits = array( "apple", "orange", "banana" );

         // iterate through each array element
         for ( $i = 0; $i < count( $fruits ); ++$i )
         {
            // call function strcmp to compare the array element
            // to string "banana"
            if ( strcmp( $fruits[ $i ], "banana" ) < 0 )
               print( "<p>" . $fruits[ $i ] . " is less than banana " );
```

Fig. 19.8 | Using the string-comparison operators. (Part 1 of 3.)

```
25              elseif ( strcmp( $fruits[ $i ], "banana" ) > 0 )
26                 print( "<p>" . $fruits[ $i ] . " is greater than banana ");
27              else
28                 print( "<p>" . $fruits[ $i ] . " is equal to banana " );
29
30              // use relational operators to compare each element
31              // to string "apple"
32              if ( $fruits[ $i ] < "apple" )
33                 print( "and less than apple!</p>" );
34              elseif ( $fruits[ $i ] > "apple" )
35                 print( "and greater than apple!</p>" );
36              elseif ( $fruits[ $i ] == "apple" )
37                 print( "and equal to apple!</p>" );
38           } // end for
39        ?><!-- end PHP script -->
40     </body>
41  </html>
```

Fig. 19.8 | Using the string-comparison operators. (Part 2 of 3.)

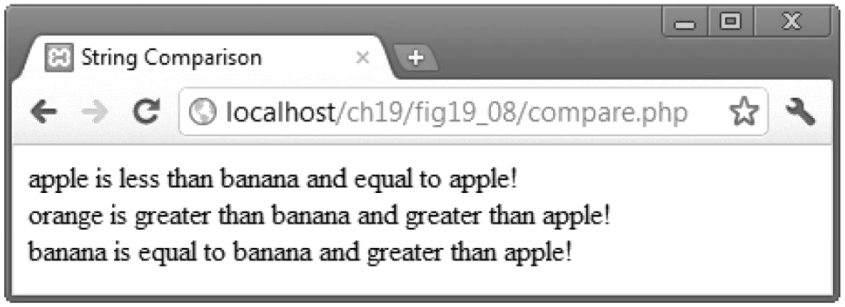

Fig. 19.8 | Using the string-comparison operators. (Part 3 of 3.)