Chapter 19

PHP

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.)

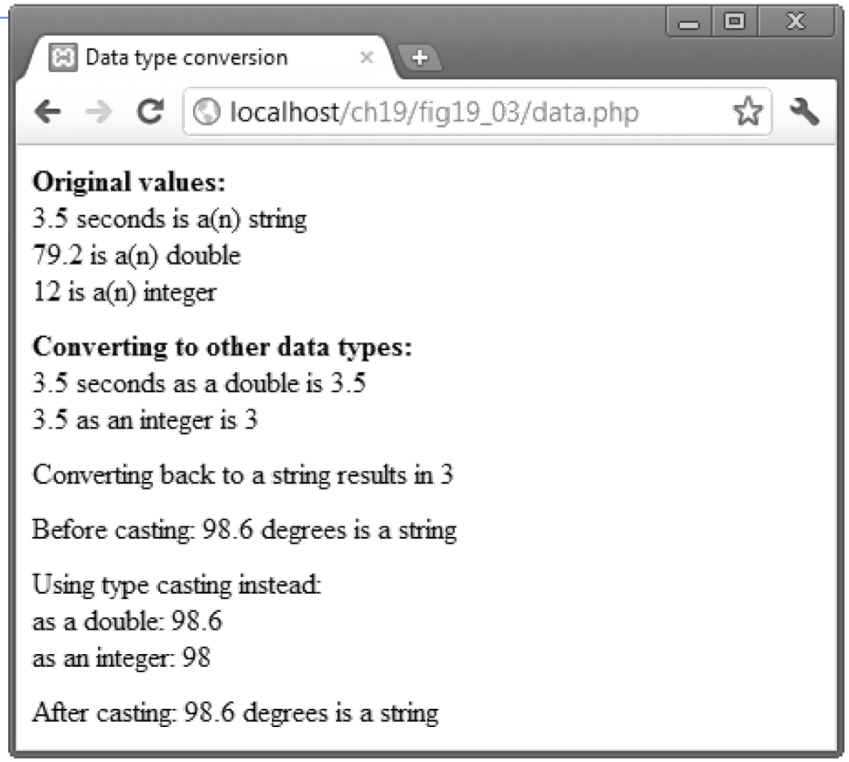

Fig. 19.3 | Data type conversion. (Part 4 of 4.)

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.

```
<!DOCTYPE html>

<!-- Fig. 19.4: operators.php -->
<!-- Using arithmetic operators. -->
<html>
   <head>
      <meta charset = "utf-8">
      <style type = "text/css">
         p { margin: 0; }
      </style>
      <title>Using arithmetic operators</title>
   </head>
   <body>
      <?php
         $a = 5;
         print( "<p>The value of variable a is $a</p>" );

         // define constant VALUE
         define( "VALUE", 5 );

         // add constant VALUE to variable $a
         $a = $a + VALUE;
         print( "<p>Variable a after adding constant VALUE is $a</p>" );

```

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.)

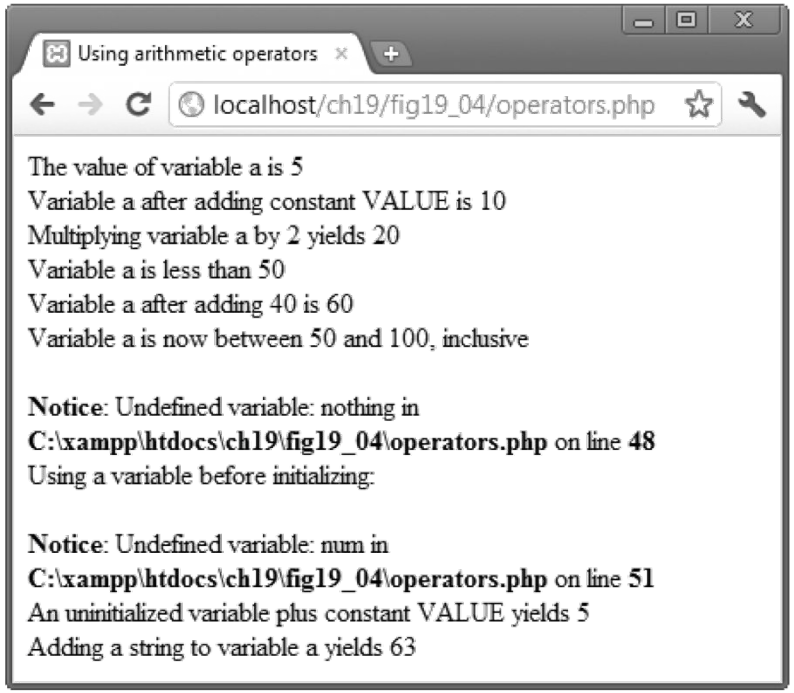

Fig. 19.4 | Using arithmetic operators. (Part 4 of 4.)

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.)

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.)

```
<!DOCTYPE html>

<!-- Fig. 19.7: arrays.php -->
<!-- Array manipulation. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Array manipulation</title>
      <style type = "text/css">
         p    { margin: 0; }
         .head { margin-top: 10px; font-weight: bold; }
      </style>
   </head>
   <body>
      <?php
         // create array first
         print( "<p class = 'head'>Creating the first array</p>" );
         $first[ 0 ] = "zero";
         $first[ 1 ] = "one";
         $first[ 2 ] = "two";
         $first[] = "three";

```

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.)

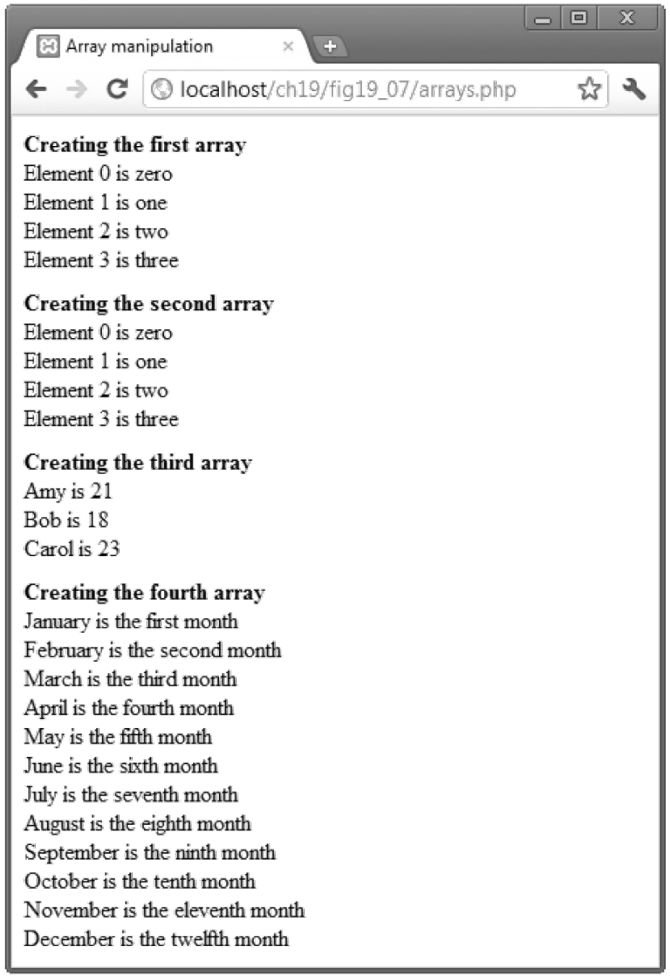

Fig. 19.7 | Array manipulation. (Part 4 of 4.)

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.)

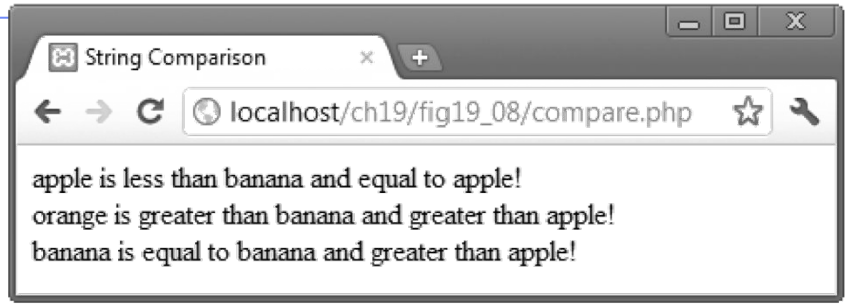

Fig. 19.8 | Using the string-comparison operators. (Part 3 of 3.)

19.7 String Processing with Regular Expressions

- Text manipulation is usually done with regular expressions—a series of characters that serve as *pattern-matching* templates (or search criteria) in strings, text files and databases.
- Function preg_match uses regular expressions to search a string for a specified pattern using Perl-compatible regular expressions (PCRE).
- If a pattern is found, preg_match returns the length of the matched string—which evaluates to true in a boolean context.
- *Anything enclosed in single quotes in a print statement is not interpolated, unless the single quotes are nested in a double-quoted string literal.*
- Function preg_match takes two arguments—a regular-expression pattern to search for and the string to search.
- Function preg_match performs *case-insensitive pattern matches*.

```
<!DOCTYPE html>

<!-- Fig. 19.9: expression.php -->
<!-- Regular expressions. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Regular expressions</title>
      <style type = "text/css">
         p { margin: 0; }
      </style>
   </head>
   <body>
      <?php
         $search = "Now is the time";
         print( "<p>Test string is: '$search'</p>" );

         // call preg_match to search for pattern 'Now' in variable search
         if ( preg_match( "/Now/", $search ) )
            print( "<p>'Now' was found.</p>" );

         // search for pattern 'Now' in the beginning of the string
         if ( preg_match( "/^Now/", $search ) )
            print( "<p>'Now' found at beginning of the line.</p>" );

```

Fig. 19.9 | Regular expressions. (Part 1 of 3.)

```
26          // search for pattern 'Now' at the end of the string
27          if ( !preg_match( "/Now$/", $search ) )
28             print( "<p>'Now' was not found at the end of the line.</p>" );
29
30          // search for any word ending in 'ow'
31          if ( preg_match( "/\b([a-zA-Z]*ow)\b/i", $search, $match ) )
32             print( "<p>Word found ending in 'ow': " .
33                $match[ 1 ] . "</p>" );
34
35          // search for any words beginning with 't'
36          print( "<p>Words beginning with 't' found: " );
37
38          while ( preg_match( "/\b(t[[:alpha:]]+)\b/", $search, $match ) )
39          {
40             print( $match[ 1 ] . " " );
41
42             // remove the first occurrence of a word beginning
43             // with 't' to find other instances in the string
44             $search = preg_replace("/" . $match[ 1 ] . "/", "", $search);
45          } // end while
46
47          print( "</p>" );
48       ?><!-- end PHP script -->
49    </body>
50 </html>h
```

Fig. 19.9 | Regular expressions. (Part 2 of 3.)

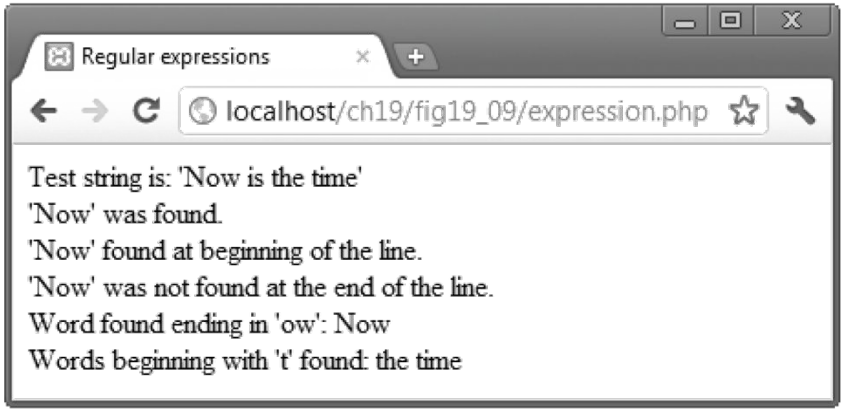

Fig. 19.9 | Regular expressions. (Part 3 of 3.)

19.7.2 Representing Patterns

- Regular expressions can include metacharacters such as ^, $ and . that specify patterns.
- For example, the caret (^) metacharacter matches the beginning of a string, while the dollar sign ($) matches the end of a string.
- The period (.) metacharacter matches any single character.
- Bracket expressions are lists of characters enclosed in square brackets ([]) that match any single character from the list.
- Ranges can be specified by supplying the beginning and the end of the range separated by a dash (-).

19.7.2 Representing Patterns

- The \b before and after the parentheses indicates the beginning and end of a word, respectively—in other words, we're attempting to match whole words.
- Quantifiers are used in regular expressions to denote how often a particular character or set of characters can appear in a match.

Quantifier	Matches
{*n*}	Exactly *n* times
{*m*,*n*}	Between *m* and *n* times, inclusive
{*n*,}	*n* or more times
+	One or more times (same as {1,})
*	Zero or more times (same as {0,})
?	Zero or one time (same as {0,1})

Fig. 19.10 | Some regular expression quantifiers.

19.7.3 Finding Matches

- The optional third argument to function preg _match is an array that stores matches to each parenthetical statement of the regular expression.
- The first element stores the string matched for the entire pattern, and the remaining elements are indexed from left to right.
- To find multiple instances of a given pattern, we must make multiple calls to preg_match, and remove matched instances before calling the function again by using a function such as preg_replace.

19.7.4 Character Classes

- Character classes are enclosed by the delimiters [: and :].
- When this expression is placed in another set of brackets, it is a regular expression matching all of the characters in the class.
- A bracketed expression containing two or more adjacent character classes in the class delimiters represents those character sets combined.

Character class	Description
`alnum`	Alphanumeric characters (i.e., letters [`a-zA-Z`] or digits [`0-9`])
`alpha`	Word characters (i.e., letters [`a-zA-Z`])
`digit`	Digits
`space`	White space
`lower`	Lowercase letters
`upper`	Uppercase letters

Fig. 19.11 | Some regular expression character classes.

19.7.5 Finding Multiple Instances of a Pattern

- Function preg_replace takes three arguments—
  - the pattern to match,
  - a string to replace the matched string and
  - the string to search. The modified string is returned.

19.10 Using Cookies

- A cookie is a piece of information that's stored by a server in a text file on a client's computer to maintain information about the client during and between browsing sessions.
- *A server can access only the cookies that it has placed on the client.*
- Function setcookie takes the name of the cookie to be set as the first argument, followed by the value to be stored in the cookie.
- The optional third argument indicates the expiration date of the cookie.
- *If no expiration date is specified, the cookie lasts only until the end of the current session*—that is, when the user closes the browser. This type of cookie is known as a session cookie, while one with an expiration date is a persistent cookie.

19.10 Using Cookies (Cont.)

- If only the name argument is passed to function setcookie, the cookie is deleted from the client's computer.
- Cookies defined in function setcookie are sent to the client at the same time as the information in the HTTP header; therefore, setcookie needs to be called *before* any other output
- PHP creates the superglobal array $_COOKIE, which contains all the cookie values indexed by their names, similar to the values stored in array $_POST when an HTML5 form is posted

```
<!DOCTYPE html>

<!-- Fig. 19.17: cookies.html -->
<!-- Gathering data to be written as a cookie. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Writing a cookie to the client computer</title>
      <style type = "text/css">
         label { width: 7em; float: left; }
      </style>
   </head>
   <body>
      <h2>Click Write Cookie to save your cookie data.</h2>
      <form method = "post" action = "cookies.php">
         <div><label>Name:</label>
            <input type = "text" name = "name"><div>
         <div><label>Height:</label>
            <input type = "text" name = "height"></div>
         <div><label>Favorite Color:</label>
            <input type = "text" name = "Color"></div>
         <p><input type = "submit" value = "Write Cookie">
      </form>
   </body>
</html>
```

Fig. 19.17 | Gathering data to be written as a cookie. (Part 1 of 2.)

Fig. 19.17 | Gathering data to be written as a cookie. (Part 2 of 2.)

```
1  <!-- Fig. 19.18: cookies.php -->
2  <!-- Writing a cookie to the client. -->
3  <?php
4     define( "FIVE_DAYS", 60 * 60 * 24 * 5 ); // define constant
5
6     // write each form field's value to a cookie and set the
7     // cookie's expiration date
8     setcookie( "name", $_POST["name"], time() + FIVE_DAYS );
9     setcookie( "height", $_POST["height"], time() + FIVE_DAYS );
10    setcookie( "color", $_POST["color"], time() + FIVE_DAYS );
11 ?><!-- end PHP script -->
12
13 <!DOCTYPE html>
14
15 <html>
16    <head>
17       <meta charset = "utf-8">
18       <title>Cookie Saved</title>
19       <style type = "text/css">
20          p { margin: 0px; }
21       </style>
22    </head>
```

Fig. 19.18 | Writing a cookie to the client. (Part 1 of 3.)

```
23      <body>
24         <p>The cookie has been set with the following data:</p>
25
26         <!-- print each form field's value -->
27         <p>Name: <?php print( $Name ) ?></p>
28         <p>Height: <?php print( $Height ) ?></p>
29         <p>Favorite Color:
30            <span style = "color: <?php print( "$Color" ) ?> ">
31            <?php print( "$Color" ) ?></span></p>
32         <p>Click <a href = "readCookies.php">here</a>
33            to read the saved cookie.</p>
34      </body>
35   </html>
```

Fig. 19.18 | Writing a cookie to the client. (Part 2 of 3.)

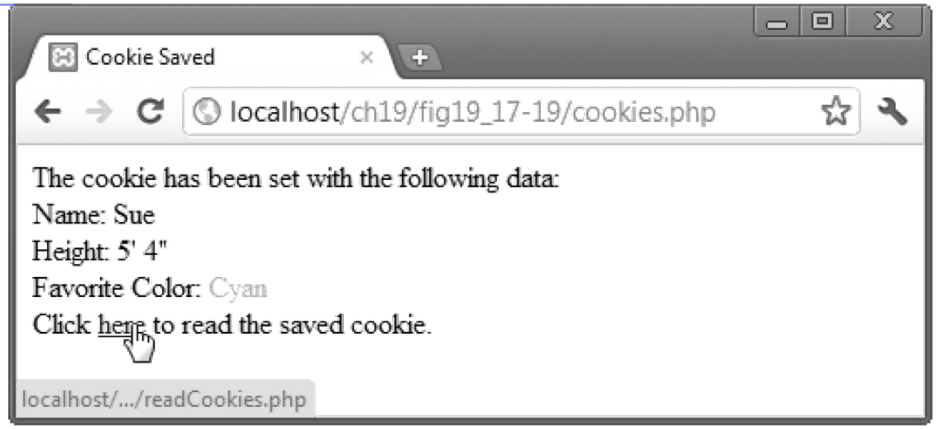

Fig. 19.18 | Writing a cookie to the client. (Part 3 of 3.)

19.11 Dynamic Content

- The isset function determines whether the $_POST array contains keys representing the various form fields.
- The notation *$$variable* specifies a variable variable, which allows the code to reference variables dynamically.
- You can use this expression to obtain the value of the variable whose name is equal to the value of $variable.
- The function mysql_real_escape_string inserts a backslash (\) before any special characters in the passed string.

```
<!DOCTYPE html>

<!-- Fig. 19.20: dynamicForm.php -->
<!-- Dynamic form. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Registration Form</title>
      <style type = "text/css">
         p       { margin: 0px; }
         .error  { color: red }
         p.head  { font-weight: bold; margin-top: 10px; }
         label   { width: 5em; float: left; }
      </style>
   </head>
   <body>
      <?php
         // variables used in script
         $fname = isset($_POST[ "fname" ]) ? $_POST[ "fname" ] : "";
         $lname = isset($_POST[ "lname" ]) ? $_POST[ "lname" ] : "";
         $email = isset($_POST[ "email" ]) ? $_POST[ "email" ] : "";
         $phone = isset($_POST[ "phone" ]) ? $_POST[ "phone" ] : "";
         $book = isset($_POST[ "book" ]) ? $_POST[ "book" ] : "";
         $os = isset($_POST[ "os" ]) ? $_POST[ "os" ] : "";
```

Fig. 19.20 | Dynamic form. (Part 1 of 10.)

```
25          $iserror = false;
26          $formerrors =
27             array( "fnameerror" => false, "lnameerror" => false,
28                "emailerror" => false, "phoneerror" => false );
29
30          // array of book titles
31          $booklist = array( "Internet and WWW How to Program",
32             "C++ How to Program", "Java How to Program",
33             "Visual Basic How to Program" );
34
35          // array of possible operating systems
36          $systemlist = array( "Windows", "Mac OS X", "Linux", "Other" );
37
38          // array of name values for the text input fields
39          $inputlist = array( "fname" => "First Name",
40             "lname" => "Last Name", "email" => "Email",
41             "phone" => "Phone" );
42
```

Fig. 19.20 | Dynamic form. (Part 2 of 10.)

```
43          // ensure that all fields have been filled in correctly
44          if ( isset( $_POST["submit"] ) )
45          {
46             if ( $fname == "" )
47             {
48                $formerrors[ "fnameerror" ] = true;
49                $iserror = true;
50             } // end if
51
52             if ( $lname == "" )
53             {
54                $formerrors[ "lnameerror" ] = true;
55                $iserror = true;
56             } // end if
57
58             if ( $email == "" )
59             {
60                $formerrors[ "emailerror" ] = true;
61                $iserror = true;
62             } // end if
63
```

Fig. 19.20 | Dynamic form. (Part 3 of 10.)

```
64          if ( !preg_match( "/^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$/",
65             $phone ) )
66          {
67             $formerrors[ "phoneerror" ] = true;
68             $iserror = true;
69          } // end if
70
71          if ( !$iserror )
72          {
73             // build INSERT query
74             $query = "INSERT INTO contacts " .
75                "( LastName, FirstName, Email, Phone, Book, OS ) " .
76                "VALUES ( '$lname', '$fname', '$email', " .
77                "'" . mysql_real_escape_string( $phone ) .
78                "', '$book', '$os' )";
79
80             // Connect to MySQL
81             if ( !( $database = mysql_connect( "localhost",
82                "iw3htp", "password" ) ) )
83                die( "<p>Could not connect to database</p>" );
84
85             // open MailingList database
86             if ( !mysql_select_db( "MailingList", $database ) )
87                die( "<p>Could not open MailingList database</p>" );
88
```

Fig. 19.20 | Dynamic form. (Part 4 of 10.)

```
89                // execute query in MailingList database
90                if ( !( $result = mysql_query( $query, $database ) ) )
91                {
92                   print( "<p>Could not execute query!</p>" );
93                   die( mysql_error() );
94                } // end if
95
96                mysql_close( $database );
97
98                print( "<p>Hi $fname. Thank you for completing the survey.
99                      You have been added to the $book mailing list.</p>
100                  <p class = 'head'>The following information has been
101                     saved in our database:</p>
102                  <p>Name: $fname $lname</p>
103                  <p>Email: $email</p>
104                  <p>Phone: $phone</p>
105                  <p>OS: $os</p>
106                  <p><a href = 'formDatabase.php'>Click here to view
107                     entire database.</a></p>
108                  <p class = 'head'>This is only a sample form.
109                     You have not been added to a mailing list.</p>
110                  </body></html>" );
111               die(); // finish the page
112            } // end if
113         } // end if
```

Fig. 19.20 | Dynamic form. (Part 5 of 10.)

```
114
115         print( "<h1>Sample Registration Form</h1>
116            <p>Please fill in all fields and click Register.</p>" );
117
118         if ( $iserror )
119         {
120            print( "<p class = 'error'>Fields with * need to be filled
121               in properly.</p>" );
122         } // end if
123
124         print( "<!-- post form data to dynamicForm.php -->
125            <form method = 'post' action = 'dynamicForm.php'>
126            <h2>User Information</h2>
127
128            <!-- create four text boxes for user input -->" );
129         foreach ( $inputlist as $inputname => $inputalt )
130         {
131            print( "<div><label>$inputalt:</label><input type = 'text'
132               name = '$inputname' value = '" . $$inputname . "'>" );
133
134            if ( $formerrors[ ( $inputname )."error" ] == true )
135               print( "<span class = 'error'>*</span>" );
136
137            print( "</div>" );
138         } // end foreach
```

Fig. 19.20 | Dynamic form. (Part 6 of 10.)

```
139
140          if ( $formerrors[ "phoneerror" ] )
141             print( "<p class = 'error'>Must be in the form
142                (555)555-5555" );
143
144          print( "<h2>Publications</h2>
145             <p>Which book would you like information about?</p>
146
147             <!-- create drop-down list containing book names -->
148             <select name = 'book'>" );
149
150          foreach ( $booklist as $currbook )
151          {
152             print( "<option" .
153                ($currbook == $book ? " selected>" : ">") .
154                $currbook . "</option>" );
155          } // end foreach
156
157          print( "</select>
158             <h2>Operating System</h2>
159             <p>Which operating system do you use?</p>
160
161             <!-- create five radio buttons -->" );
162
163          $counter = 0;
```

Fig. 19.20 | Dynamic form. (Part 7 of 10.)

```
164
165          foreach ( $systemlist as $currsystem )
166          {
167             print( "<input type = 'radio' name = 'os'
168                value = '$currsystem' " );
169
170             if ( ( !$os && $counter == 0 ) || ( $currsystem == $os ) )
171                print( "checked" );
172
173             print( ">$currsystem" );
174             ++$counter;
175          } // end foreach
176
177          print( "<!-- create a submit button -->
178             <p class = 'head'><input type = 'submit' name = 'submit'
179             value = 'Register'></p></form></body></html>" );
180    ?><!-- end PHP script -->
```

Fig. 19.20 | Dynamic form. (Part 8 of 10.)

a) Registration form after it was submitted with a missing field and an incorrectly formatted phone number

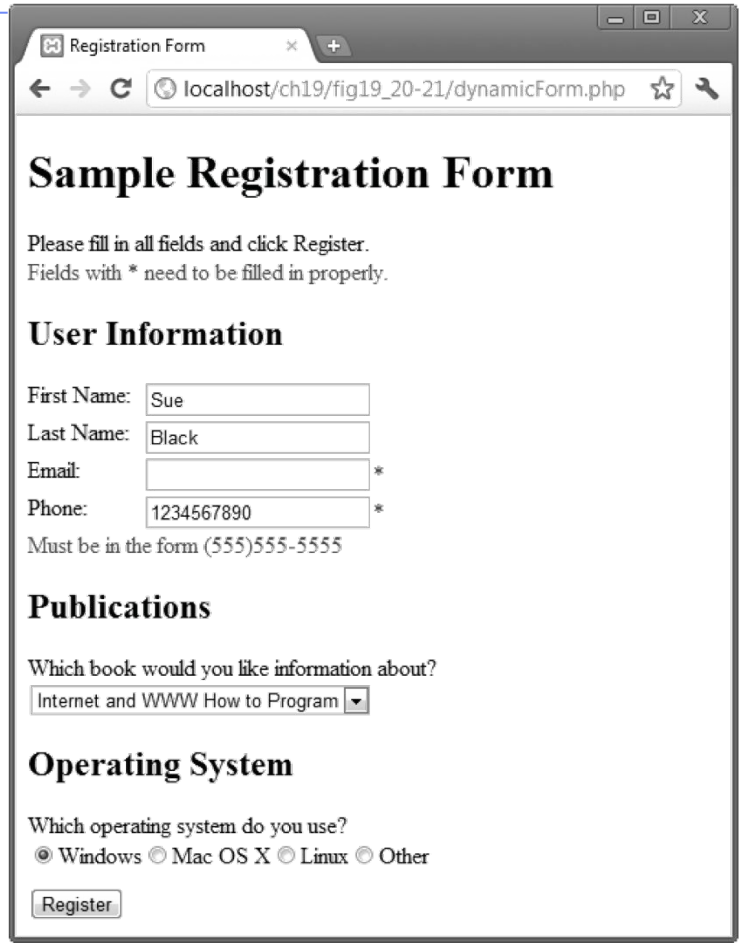

Fig. 19.20 | Dynamic form. (Part 9 of 10.)

b) Confirmation page displayed after the user properly fills in the form and the information is stored in the database

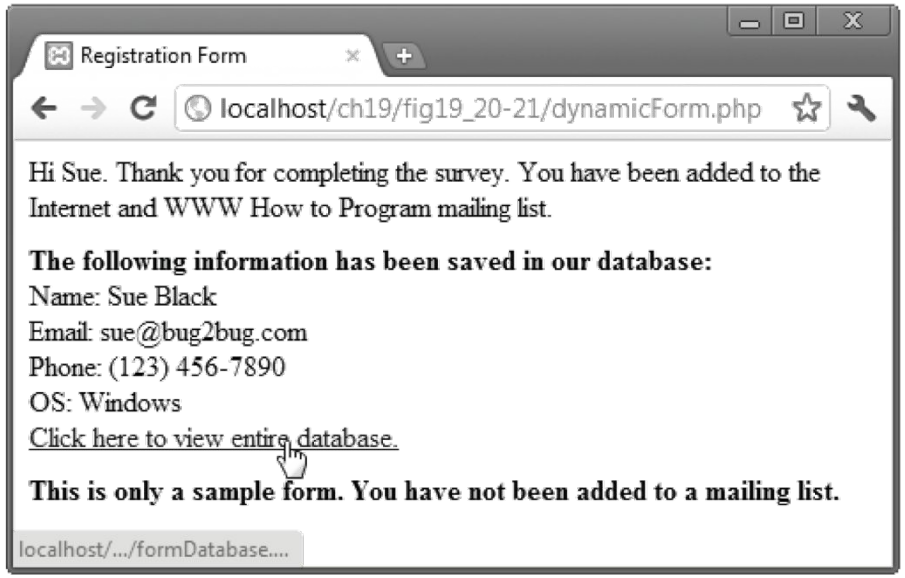

Fig. 19.20 | Dynamic form. (Part 10 of 10.)

```
<!DOCTYPE html>

<!-- Fig. 19.21: formDatabase.php -->
<!-- Displaying the MailingList database. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Search Results</title>
      <style type = "text/css">
         table  { background-color: lightblue;
                  border: 1px solid gray;
                  border-collapse: collapse; }
         th, td { padding: 5px; border: 1px solid gray; }
         tr:nth-child(even) { background-color: white; }
         tr:first-child { background-color: lightgreen; }
      </style>
   </head>
   <body>
      <?php
         // build SELECT query
         $query = "SELECT * FROM contacts";

```

Fig. 19.21 | Displaying the `MailingList` database. (Part 1 of 4.)

```
23          // Connect to MySQL
24          if ( !( $database = mysql_connect( "localhost",
25             "iw3htp", "password" ) ) )
26             die( "<p>Could not connect to database</p></body></html>" );
27
28          // open MailingList database
29          if ( !mysql_select_db( "MailingList", $database ) )
30             die( "<p>Could not open MailingList database</p>
31                </body></html>" );
32
33          // query MailingList database
34          if ( !( $result = mysql_query( $query, $database ) ) )
35          {
36             print( "<p>Could not execute query!</p>" );
37             die( mysql_error() . "</body></html>" );
38          } // end if
39       ?><!-- end PHP script -->
40
41       <h1>Mailing List Contacts</h1>
42       <table>
43          <caption>Contacts stored in the database</caption>
44          <tr>
45             <th>ID</th>
46             <th>Last Name</th>
47             <th>First Name</th>
```

Fig. 19.21 | Displaying the `MailingList` database. (Part 2 of 4.)

```
48                <th>E-mail Address</th>
49                <th>Phone Number</th>
50                <th>Book</th>
51                <th>Operating System</th>
52             </tr>
53             <?php
54                // fetch each record in result set
55                for ( $counter = 0; $row = mysql_fetch_row( $result );
56                   ++$counter )
57                {
58                   // build table to display results
59                   print( "<tr>" );
60
61                   foreach ( $row as $key => $value )
62                      print( "<td>$value</td>" );
63
64                   print( "</tr>" );
65                } // end for
66
67                mysql_close( $database );
68             ?><!-- end PHP script -->
69          </table>
70       </body>
71    </html>
```

Fig. 19.21 | Displaying the `MailingList` database. (Part 3 of 4.)

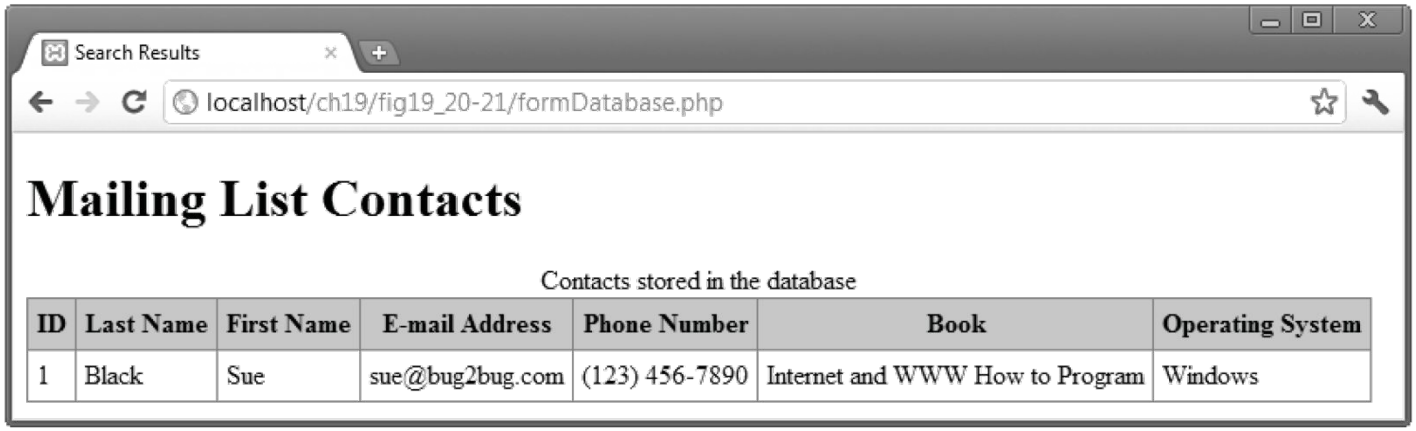

Fig. 19.21 | Displaying the `MailingList` database. (Part 4 of 4.)