PHP

19.1 Introduction

- PHP, or PHP: Hypertext Preprocessor, has become the most popular server-side scripting language for creating dynamic web pages
- PHP was created by Rasmus Lerdorf to track users at his website
  - In 1995, Lerdorf released it as a package called the "Personal Home Page Tools"
  - Two years later, PHP 2 featured built-in database support and form handling
  - In 1997, PHP 3 was released after a substantial rewrite, which resulted in a large increase in performance and led to an explosion of PHP use

19.1 Introduction

- The release of PHP 4 featured the new Zend Engine from Zend, a PHP software company
  - This version was considerably faster and more powerful than its predecessor, further increasing PHP's popularity
- Currently, PHP 5 features the Zend Engine 2, which provides further speed increases, exception handling and a new object-oriented programming model

19.1 Introduction

- PHP is an open-source technology that's supported by a large community of users and developers
- PHP is *platform independent*—implementations exist for all major UNIX, Linux, Mac and Windows operating systems
- PHP also supports many databases, including MySQL
- After introducing the basics of the PHP scripting language, we discuss form processing and business logic, which are vital to e-commerce applications
- Next, we build a three-tier web application that queries a MySQL database

19.1 Introduction

- We also show how PHP can use cookies to store information on the client that can be retrieved during future visits to the website
- Finally, we revisit the form-processing example to demonstrate some of PHP's more dynamic capabilities
- Visual Studio Code
  - https://code.visualstudio.com/docs?dv=win
- PHP Tools for Visual Studio
  - https://visualstudiogallery.msdn.microsoft.com/6eb51f05-ef01-4513-ac83-4c5f50c95fb5

19.2 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
- Interactivity between the user and the server has become a crucial part of web functionality
  - Making PHP—a language written specifically for handling client requests
- PHP code is embedded directly into text-based documents, such as HTML
  - Though these script segments are interpreted by the server before being delivered to the client

19.2 Simple PHP Program

- PHP script file names end with .php
- Figure 19.1 presents a simple PHP script that displays a welcome message
- PHP code is inserted between the delimiters **<?php** and **?>** and can be placed anywhere in HTML markup
  - Declares variable **$name** and assigns it the string "Paul"
  - All variables are preceded by a **$** and are created the first time they're encountered by the PHP interpreter
  - PHP statements terminate with a semicolon (**;**)
  - http://www.w3schools.com/php/php_echo_print.asp
    - **echo** vs. **print**

```
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.)

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.

19.2 Simple PHP Program

- PHP variables are loosely typed—they can contain different types of data (e.g., integers, doubles or strings) at different times
  - http://www.w3schools.com/php/php_datatypes.asp

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

- Converting between different data types may be necessary when performing arithmetic operations with variables
- **gettype** returns the current type of its argument
  - **$testString = "3.5 seconds";**
  - **gettype($testString);**
- Type conversions can be performed using function **settype**
  - **settype($testString, "integer");**
  - Calling function settype can result in loss of data
- Type casting
  - **(double) $data**

19.3 Converting Between Data Types

- String Concatenation
  - The concatenation operator (.) combines multiple strings in the same print statement
  - A print statement may be split over multiple lines
    - All data that's enclosed in the parentheses and terminated by a semicolon
  - **print(" Using type casting instead: as a double: " . (double) $data . " "**

```
<!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.)

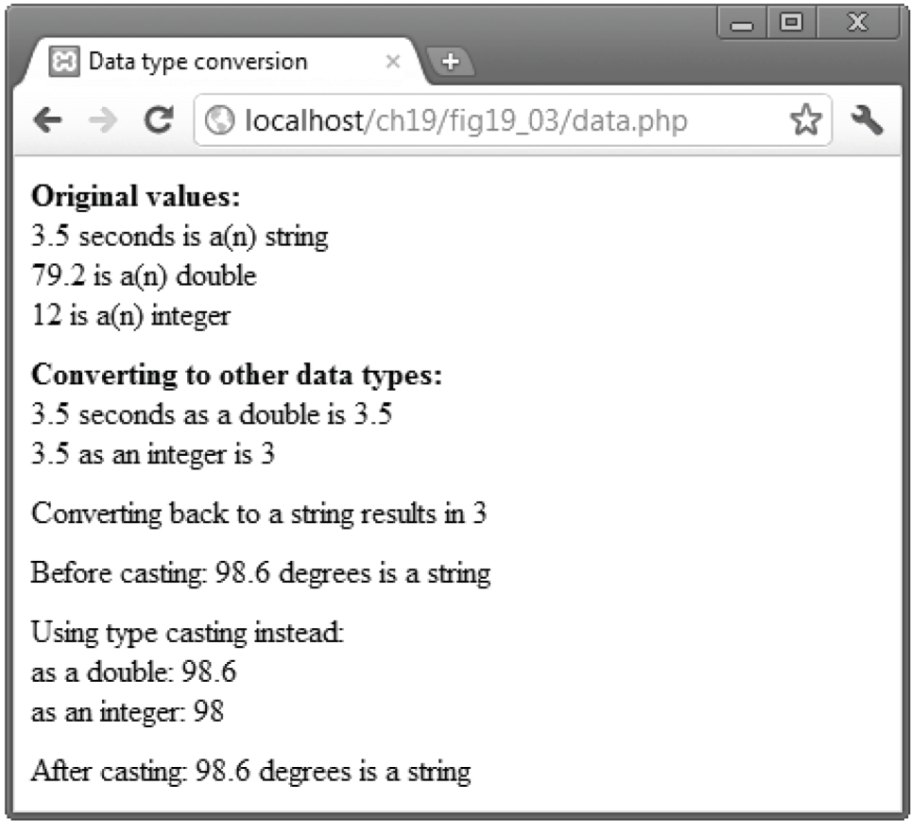


Fig. 19.3 | Data type conversion. (Part 4 of 4.)

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

- PHP provides several arithmetic operators
- Declare variable $a and assigns to it the value 5
  - **$a = 5;**
- Call a function define to create a named constant
  - **define("VALUE", 5);**
- Function define takes two arguments—the name and value of the constant
- An optional third argument accepts a bool value that specifies whether the constant is case insensitive
  - Constants are case sensitive by default

```
<!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.)

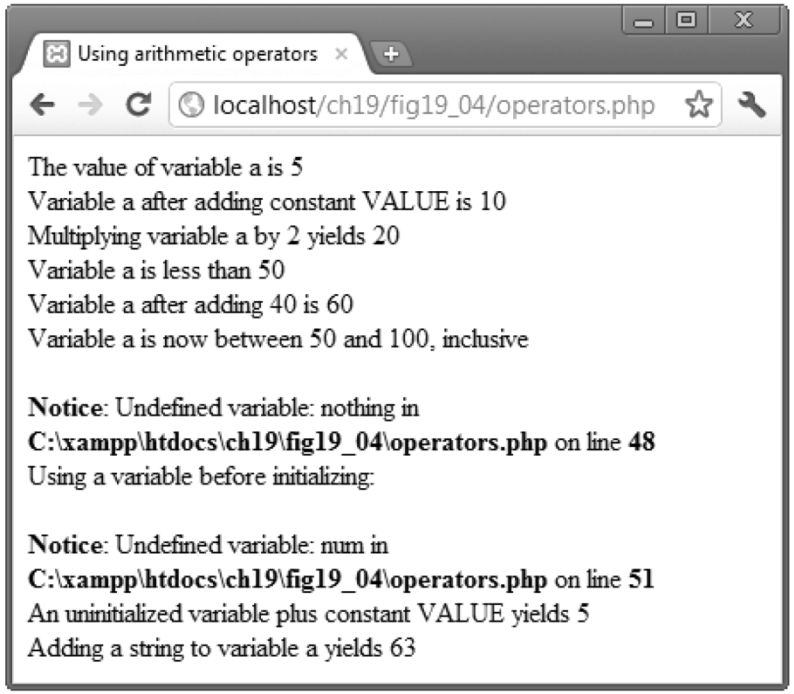


Fig. 19.4 | Using arithmetic operators. (Part 4 of 4.)

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.

Common Programming Error 19.3

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

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 (**[]**)
  - **$first[0] = "zero";**
- If a value is assigned to an array element of an array that does not exist, then the array is created

19.5 Initializing and Manipulating Arrays

- Likewise, assigning a value to an element where the index is omitted appends a new element to the end of the array
  - **$first[2] = "two";**
  - **$first[] = "three";**
  - http://www.w3schools.com/php/php_arrays.asp
- The **for** statement prints each element's value
  - http://www.w3schools.com/php/php_looping_for.asp
- Function **count** returns the total number of elements in the array
- In this example, the for statement terminates when the counter (**$i**) is equal to the number of array elements

19.5 Initializing and Manipulating Arrays

- In addition to integer indices, arrays can have float or nonnumeric indices
- An array with noninteger indices is called an **associative array**
  - **$third["Amy"] = 21;**
  - **$third["Bob"] = 18;**
  - **$third["Carol"] = 23;**
- 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

19.5 Initializing and Manipulating Arrays

- 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
- Function **next** moves the internal pointer to the next element and returns the element
  - **for (reset($third); $element = key($third); next($third))**

```
print( "<p>$element is $third[$element]</p>" );
```

```
<!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.)

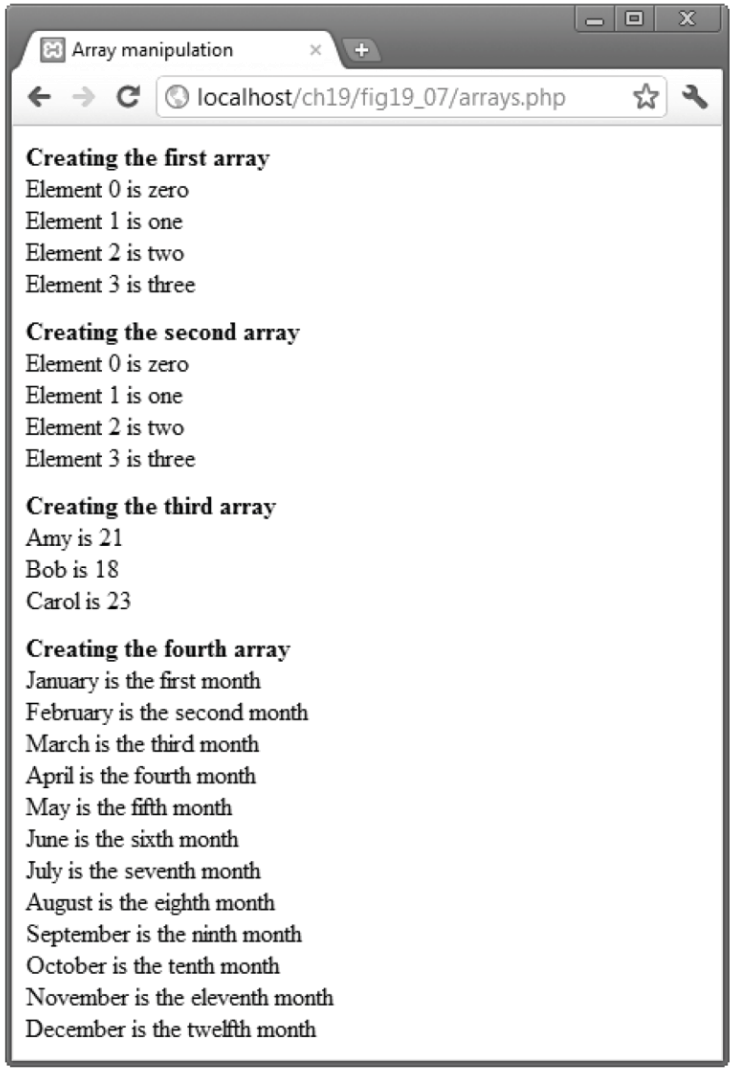


Fig. 19.7 | Array manipulation. (Part 4 of 4.)

19.6 String Comparisons

- Many string-processing tasks can be accomplished by using the equality and comparison operators
- Function **strcmp** compares two strings
  - **strcmp($fruits[$i], "banana")**
- The function returns
  - -1 if the first string alphabetically precedes the second string
  - 0 if the strings are equal
  - 1 if the first string alphabetically follows the second
- Relational operators (==, !=, <, <=, > and >=) can also be used to compare strings

```
<!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.)

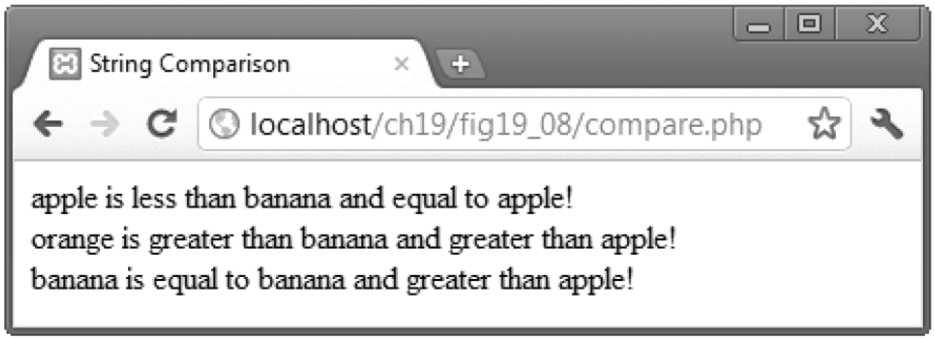


Fig. 19.8 | Using the string-comparison operators. (Part 3 of 3.)

19.7 String Processing with Regular Expressions

- PHP can process text easily and efficiently, enabling straightforward searching, substitution, extraction and concatenation of strings
- 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)
  - **preg_match("/Now/", $search);**

19.7 String Processing with Regular Expressions

- Regular expressions can include metacharacters
  - The caret (^) metacharacter matches the beginning of a string
  - The dollar sign ($) matches the end of a string
  - The period (.) metacharacter matches any single character except newlines
- 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 (**-**)
  - **[a-zA-Z]**

19.7 String Processing with Regular Expressions

- 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
- The expression [a-zA-Z]*ow inside the parentheses represents any word ending in ow
- The quantifier ***** matches the preceding pattern zero or more times
- To perform *case insensitive pattern matches* you simply place the letter **i** after the regular-expression pattern
  - **"/\b([a-zA-Z]*ow)\b/i"**

19.7 String Processing with Regular Expressions

- The pattern uses the character class **[[:alpha:]]** to recognize any letter
  - This is equivalent to the **[a-zA-Z]**
  - **/\b(t[[:alpha:]]+)\b/I**

  matches any word beginning with the character t followed by one or more letters

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.

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.

```
<!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.)

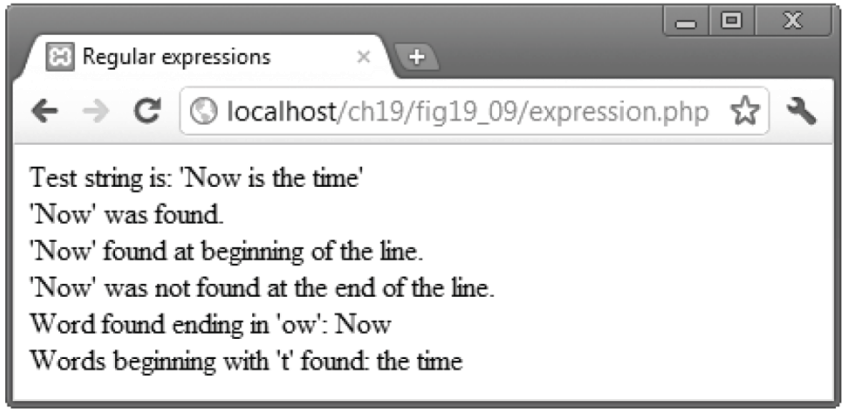


Fig. 19.9 | Regular expressions. (Part 3 of 3.)

19.8 Form Processing and Business Logic

- Knowledge of a client's execution environment is useful to system administrators who want to access client-specific information such as the client's web browser, the server name or the data sent to the server by the client
- One way to obtain this data is by using a **superglobal array**
  - Are associative arrays predefined by PHP that hold variables acquired from user input, the environment or the web server, and are accessible in any variable scope

Variable name	Description
`$_SERVER`	Data about the currently running server.
`$_ENV`	Data about the client's environment.
`$_GET`	Data sent to the server by a `get` request.
`$_POST`	Data sent to the server by a `post` request.
`$_COOKIE`	Data contained in cookies on the client's computer.
`$GLOBALS`	Array containing all global variables.

Fig. 19.12 | Some useful superglobal arrays.

19.8 Form Processing and Business Logic

- Superglobal arrays are useful for verifying user input
- The arrays **$_GET** and **$_POST** retrieve information sent to the server by HTTP **get** and **post** requests, respectively
  - For a script to have access to this data when it loads another page
  - For example, if data entered by a user into a form is posted to a script, the **$_POST** array will contain all of this information in the new script
  - Thus, any information entered into the form can be accessed easily from a confirmation page, or a page that verifies whether fields have been entered correctly

19.8 Form Processing and Business Logic

- Forms enable web pages to collect data from users and send it to a web server for processing
  - Such capabilities allow users to purchase products, request information, send and receive web-based e-mail, create profiles in online networking services and take advantage of various other online services
- The form's action attribute indicates that when the user clicks the Register button, the form data will be posted to form.php for processing
  - **<form method = "post" action = "form.php">**

19.8 Form Processing and Business Logic

- Scripts located on the web server's machine can access the form data sent as part of the request
  - We assign a unique name (e.g., email) to each of the form's controls
  - When Register is clicked, each field's name and value are sent to the web server
  - Script form.php accesses the value for each field through the superglobal array **$_POST**, which contains key/value pairs corresponding to name–value pairs for variables submitted through the form

19.8 Form Processing and Business Logic

- The superglobal array **$_GET** would contain these key–value pairs if the form had been submitted using the HTTP **get** method
- In general, **get** is not as secure as post, because it appends the information directly to the URL, which is visible to the user

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.13: form.html -->
4  <!-- HTML form for gathering user input. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Sample Form</title>
9        <style type = "text/css">
10          label  { width: 5em; float: left; }
11       </style>
12    </head>
13    <body>
14       <h1>Registration Form</h1>
15       <p>Please fill in all fields and click Register.</p>
16
17       <!-- post form data to form.php -->
18       <form method = "post" action = "form.php">
19          <h2>User Information</h2>
20
21          <!-- create four text boxes for user input -->
22          <div><label>First name:</label>
23             <input type = "text" name = "fname"></div>
24          <div><label>Last name:</label>
```

Fig. 19.13 | HTML5 form for gathering user input. (Part 1 of 4.)

```
25             <input type = "text" name = "lname"></div>
26          <div><label>Email:</label>
27             <input type = "text" name = "email"></div>
28          <div><label>Phone:</label>
29             <input type = "text" name = "phone"
30                placeholder = "(555) 555-5555"></div>
31          </div>
32
33          <h2>Publications</h2>
34          <p>Which book would you like information about?</p>
35
36          <!-- create drop-down list containing book names -->
37          <select name = "book">
38             <option>Internet and WWW How to Program</option>
39             <option>C++ How to Program</option>
40             <option>Java How to Program</option>
41             <option>Visual Basic How to Program</option>
42          </select>
43
44          <h2>Operating System</h2>
45          <p>Which operating system do you use?</p>
46
47          <!-- create five radio buttons -->
48          <p><input type = "radio" name = "os" value = "Windows"
49                checked>Windows
```

Fig. 19.13 | HTML5 form for gathering user input. (Part 2 of 4.)

```
50                <input type = "radio" name = "os" value = "Mac OS X">Mac OS X
51                <input type = "radio" name = "os" value = "Linux">Linux
52                <input type = "radio" name = "os" value = "Other">Other</p>
53
54             <!-- create a submit button -->
55             <p><input type = "submit" name = "submit" value = "Register"></p>
56          </form>
57       </body>
58    </html>
```

Fig. 19.13 | HTML5 form for gathering user input. (Part 3 of 4.)

The form is filled out with an incorrect phone number

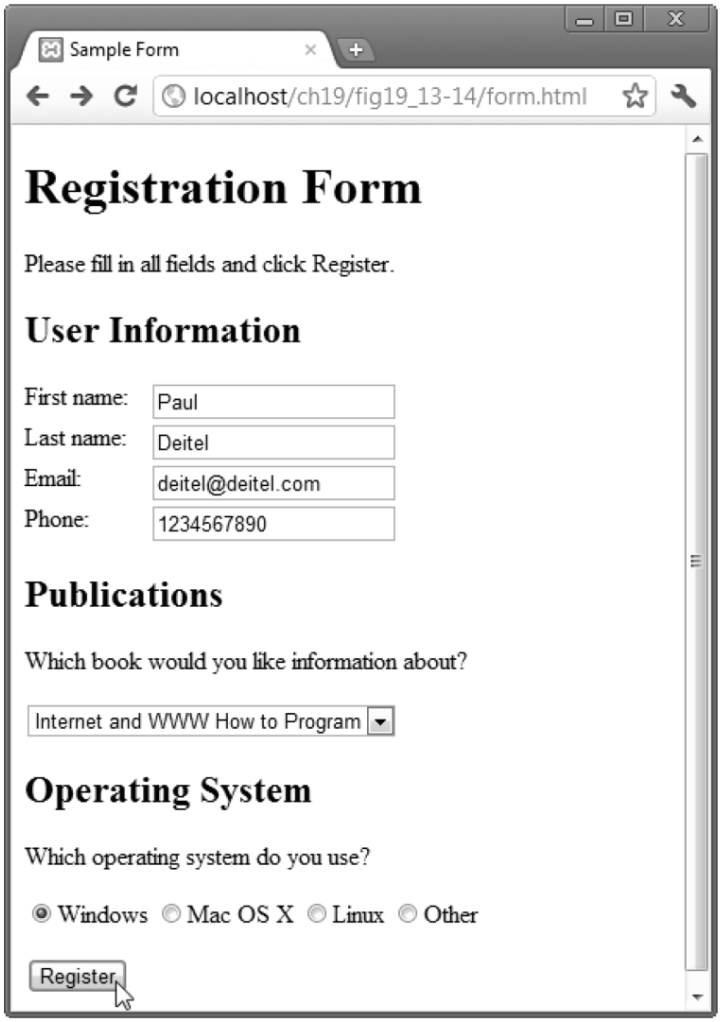


Fig. 19.13 | HTML5 form for gathering user input. (Part 4 of 4.)

Good Programming Practice 19.1

Use meaningful HTML5 object names for `input` fields. This makes PHP scripts that retrieve `form` data easier to understand.

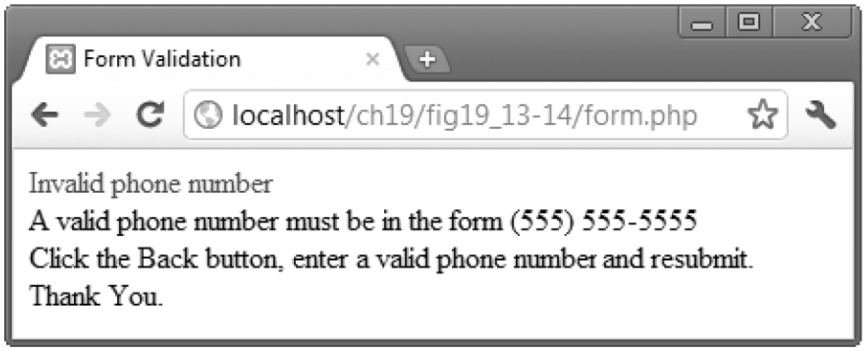
```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.14: form.php -->
4  <!-- Process information sent from form.html. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Form Validation</title>
9        <style type = "text/css">
10          p       { margin: 0px; }
11          .error  { color: red }
12          p.head  { font-weight: bold; margin-top: 10px; }
13       </style>
14    </head>
15    <body>
16       <?php
17          // determine whether phone number is valid and print
18          // an error message if not
19          if (!preg_match( "/^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$/",
20             $_POST["phone"]))
21          {
```

Fig. 19.14 | Process information sent from `form.html`. (Part 1 of 3.)

```
22              print( "<p class = 'error'>Invalid phone number</p>
23                 <p>A valid phone number must be in the form
24                 (555) 555-5555</p><p>Click the Back button,
25                 enter a valid phone number and resubmit.</p>
26                 <p>Thank You.</p></body></html>" );
27              die(); // terminate script execution
28           }
29        ?><!-- end PHP script -->
30        <p>Hi <?php print( $_POST["fname"] ); ?>. Thank you for
31           completing the survey. You have been added to the
32           <?php print( $_POST["book"] ); ?>mailing list.</p>
33        <p class = "head">The following information has been saved
34           in our database:</p>
35        <p>Name: <?php print( $_POST["fname"] );
36           print( $_POST["lname"] ); ?></p>
37        <p>Email: <?php print( "$email" ); ?></p>
38        <p>Phone: <?php print( "$phone" ); ?></p>
39        <p>OS: <?php print( $_POST["os"] ); ?></p>
40        <p class = "head">This is only a sample form.
41           You have not been added to a mailing list.</p>
42     </body>
43  </html>
```

Fig. 19.14 | Process information sent from `form.html`. (Part 2 of 3.)

a) Submitting the form in Fig. 19.13 redirects the user to `form.php`, which gives appropriate instructions if the phone number is in an incorrect format



b) The results of form.php after the user submits the form in Fig. 19.13 with a phone number in a valid format

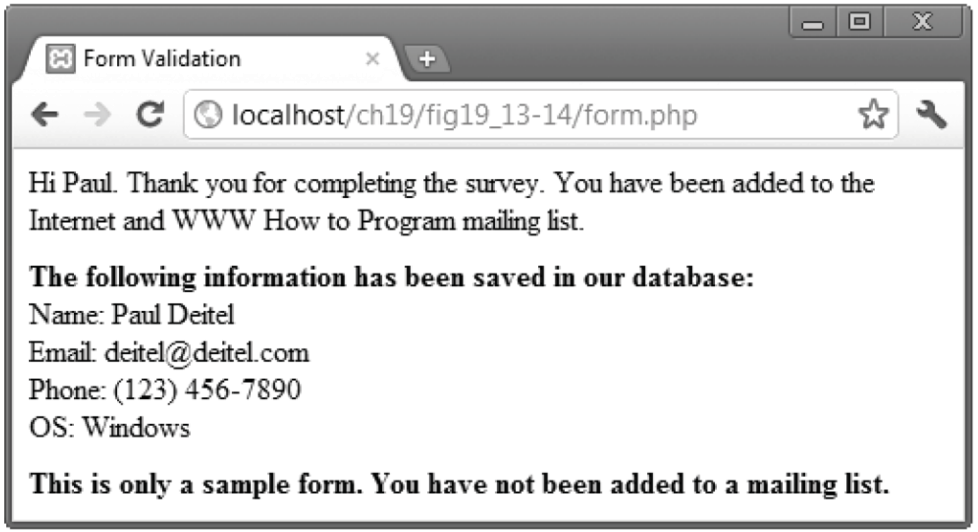


Fig. 19.14 | Process information sent from `form.html`. (Part 3 of 3.)

Software Engineering Observation 19.1

Use business logic to ensure that invalid information is not stored in databases. Validate important or sensitive form data on the server, since JavaScript may be disabled by the client. Some data, such as passwords, must always be validated on the server side.

19.9 Reading from a Database

- PHP offers built-in support for many databases
- Our database examples use MySQL
- The example in this section uses a Products database
- The user selects the name of a column in the database and submits the form
- A PHP script then builds a SQL SELECT query, queries the database to obtain the column's data and outputs the data in an HTML5 document that's displayed in the user's web browser

19.9 Reading from a Database

- An HTML5 form, specifying that the data submitted from
- the form will be sent to the script database.php in a post request
- A select box to the form sets the name of the select box to select and set its default
- selection to *. Submitting * specifies that *all* rows and columns are to be retrieved from the database
- Each of the database's column names is set as an option in the select box

19.9 Reading from a Database

- Script database.php builds a SQL query with the posted field name then queries the MySQL database
- Call function **mysql_connect** to connect to the MySQL database
  - **$database = mysql_connect("localhost","iw3htp", "password");**
  - Pass three arguments—the server's hostname, a username and a password
  - Returns a **database handle**—a representation of PHP's connection to the database—which we assign to variable **$database**

19.9 Reading from a Database

- Function **mysql_select_db** to select and open the database to be queried (in this case, products)
  - **mysql_select_db("products", $database);**
  - The function returns true on success or false on failure
  - We call **die** if the database cannot be opened
- Function **mysql_query** specifying the query string and the database to query
  - **mysql_query($query, $database);**
  - If the query fails, the function returns false

19.9 Reading from a Database

- Function **mysql_error** returns any error strings from the database
  - **die(mysql_error() . "</body></html>");**
- If the query succeeds, **mysql_query** returns a resource containing the query result, which we assign to variable **$result**
- Once we've stored the data in $result, we call **mysql_close** to close the connection to the database
  - **mysql_close($database);**
- Function **mysql_query** can also execute SQL statements such as INSERT or DELETE that do not return results

19.9 Reading from a Database

- Show each record in the *result set* and construct an HTML5 table containing the results
- The loop's condition calls the **mysql_fetch_row** function to return an array containing the values for each column in the current row of the query result ($result)
  - **$row = mysql_fetch_row($result);**
- The array is stored in variable **$row**
- The **foreach** statement takes the name of the array (**$row**), iterates through each index value of the array and stores the value in variable **$value**

19.9 Reading from a Database

- Each element of the array is then printed as an individual cell
- When the result has no more rows, false is returned by function **mysql_fetch_row**, which terminates the loop
- After all the rows in the result have been displayed, the table's closing tag is written
- Display the number of rows in **$result** by calling **mysql_num_rows** with **$result** as an argument
  - **mysql_num_rows($result);**

```
<!DOCTYPE html>

<!-- Fig. 19.15: data.html -->
<!-- Form to query a MySQL database. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Sample Database Query</title>
   </head>
   <body>
      <h1>Querying a MySQL database.</h1>
      <form method = "post" action = "database.php">
         <p>Select a field to display:
            <!-- add a select box containing options -->
            <!-- for SELECT query -->
            <select name = "select">
               <option selected>*</option>
               <option>ID</option>
               <option>Title</option>
               <option>Category</option>
               <option>ISBN</option>
            </select></p>
```

Fig. 19.15 | Form to query a MySQL database. (Part 1 of 2.)

```
23            <p><input type = "submit" value = "Send Query"></p>
24         </form>
25      </body>
26   </html>
```

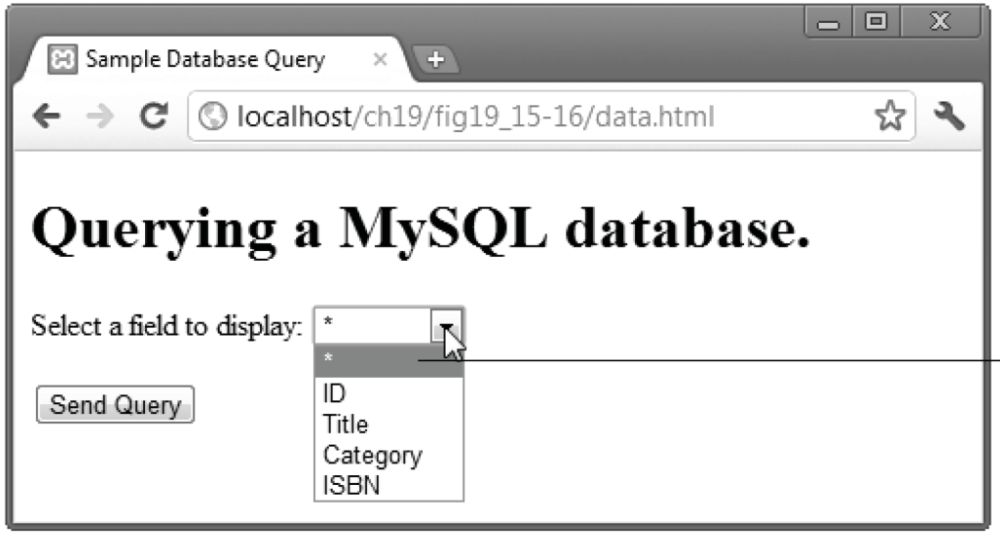


Fig. 19.15 | Form to query a MySQL database. (Part 2 of 2.)

```
<!DOCTYPE html>

<!-- Fig. 19.16: database.php -->
<!-- Querying a database and displaying the results. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Search Results</title>
   <style type = "text/css">
         body  { font-family: sans-serif;
                 background-color: lightyellow; }
         table { background-color: lightblue;
                 border-collapse: collapse;
                 border: 1px solid gray; }
         td    { padding: 5px; }
         tr:nth-child(odd) {
                 background-color: white; }
      </style>
   </head>
   <body>
      <?php
         $select = $_POST["select"]; // creates variable $select

```

Fig. 19.16 | Querying a database and displaying the results. (Part 1 of 4.)

```
24            // build SELECT query
25            $query = "SELECT " . $select . " FROM books";
26
27            // Connect to MySQL
28            if ( !( $database = mysql_connect( "localhost",
29               "iw3htp", "password" ) ) )
30               die( "Could not connect to database </body></html>" );
31
32            // open Products database
33            if ( !mysql_select_db( "products", $database ) )
34               die( "Could not open products database </body></html>" );
35
36            // query Products database
37            if ( !( $result = mysql_query( $query, $database ) ) )
38            {
39               print( "<p>Could not execute query!</p>" );
40               die( mysql_error() . "</body></html>" );
41            } // end if
42
43            mysql_close( $database );
44         ?><!-- end PHP script -->
```

Fig. 19.16 | Querying a database and displaying the results. (Part 2 of 4.)

```
45          <table>
46             <caption>Results of "SELECT <?php print( "$select" ) ?>
47                FROM books"</caption>
48             <?php
49                // fetch each record in result set
50                while ( $row = mysql_fetch_row( $result ) )
51                {
52                   // build table to display results
53                   print( "<tr>" );
54
55                   foreach ( $row as $key => $value )
56                      print( "<td>$value</td>" );
57
58                   print( "</tr>" );
59                } // end while
60             ?><!-- end PHP script -->
61          </table>
62          <p>Your search yielded
63             <?php print( mysql_num_rows( $result ) ) ?> results.</p>
64          <p>Please email comments to <a href = "mailto:deitel@deitel.com">
65                Deitel and Associates, Inc.</a></p>
66       </body>
67    </html>
```

Fig. 19.16 | Querying a database and displaying the results. (Part 3 of 4.)

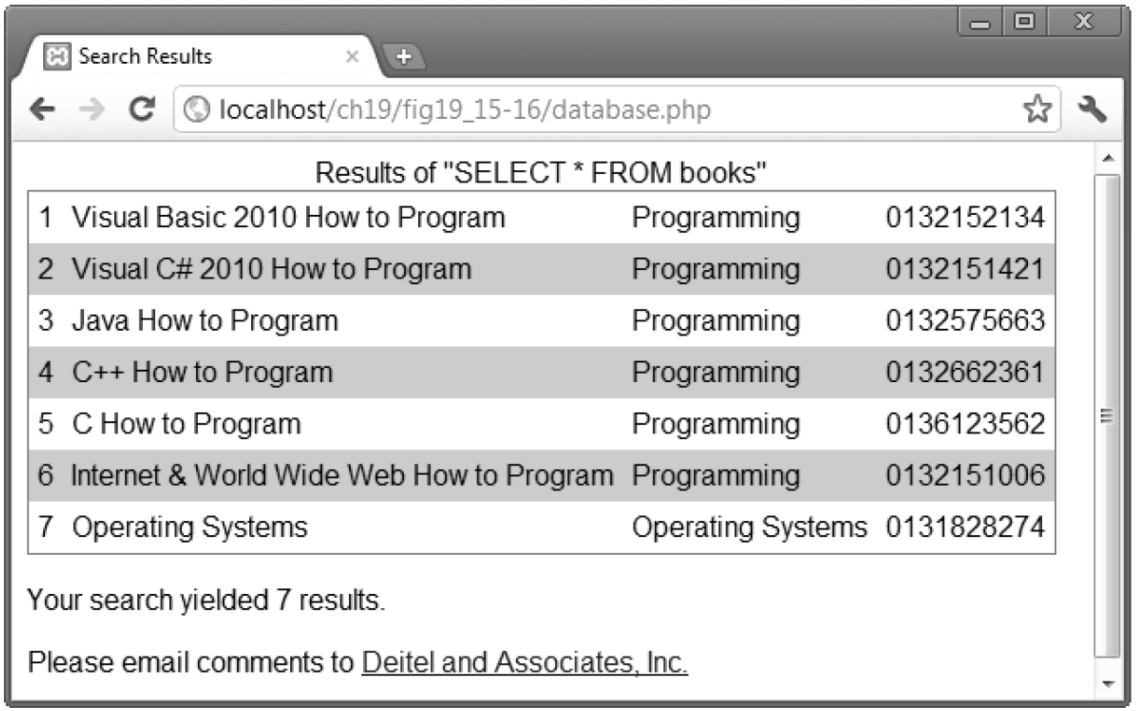


Fig. 19.16 | Querying a database and displaying the results. (Part 4 of 4.)

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 website can store a cookie on a client's computer to record user preferences and other information that the website can retrieve during the client's subsequent visits
  - For example, a website can use cookies to store clients' zip codes, so that it can provide weather reports and news updates tailored to the user's region
- Websites also can use cookies to track information about client activity

19.10 Using Cookies

- Analysis of information collected via cookies can reveal the popularity of websites or products
  - Marketers can use cookies to determine the effectiveness of advertising campaigns
  - Websites store cookies on users' hard drives, which raises issues regarding security and privacy
- Websites should not store critical information, such as credit card numbers or passwords, in cookies
  - Because cookies are typically stored in text files that any program can read
- Several cookie features address security and privacy concerns

19.10 Using Cookies

- A server can access only the cookies that it has placed on the client
  - For example, a web application running on www.deitel.com cannot access cookies that the website www.pearson.com has placed on the client's computer
- A cookie also has an *expiration date*, after which the web browser deletes it
- Users who are concerned about the privacy and security implications of cookies can disable cookies in their browsers
  - Disabling cookies can make it difficult or impossible for the user to interact with websites that rely on cookies to function properly

19.10 Using Cookies

- The information stored in a cookie is sent back to the web server from which it originated whenever the user requests a web page from that particular server
- The web server can send the client HTML5 output that reflects the preferences or information that's stored in the cookie
- Function **setcookie** to set the cookies to the values posted from cookies.html
  - **setcookie("name", $_POST["name"], time() + FIVE_DAYS);**
  - http://www.w3schools.com/php/php_cookies.asp

```
<!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.)

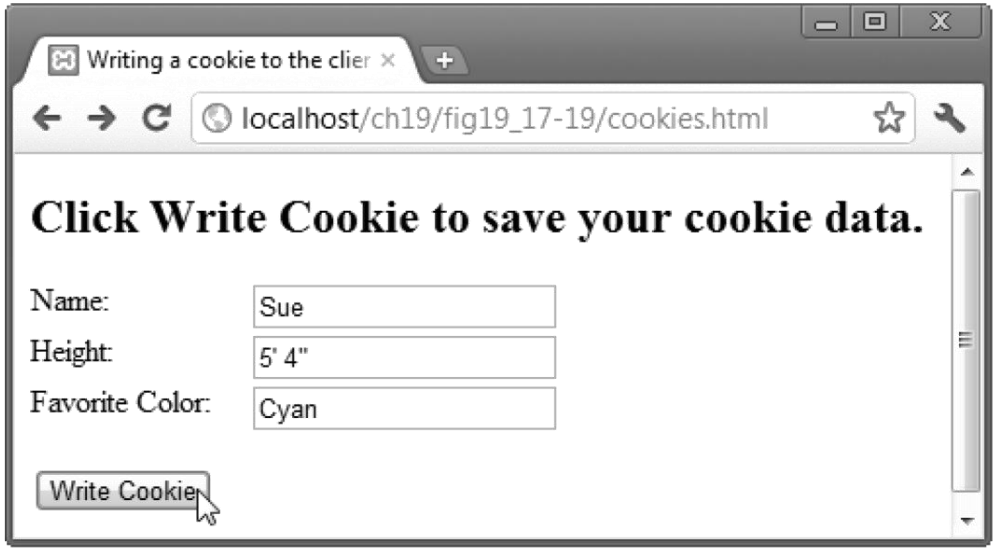


Fig. 19.17 | Gathering data to be written as a cookie. (Part 2 of 2.)

Software Engineering Observation 19.2

Some clients do not accept cookies. When a client declines a cookie, the browser application normally informs the user that the site may not function correctly without cookies enabled.

Software Engineering Observation 19.3

Cookies should not be used to store e-mail addresses, passwords or private data on a client's computer.

```
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.)

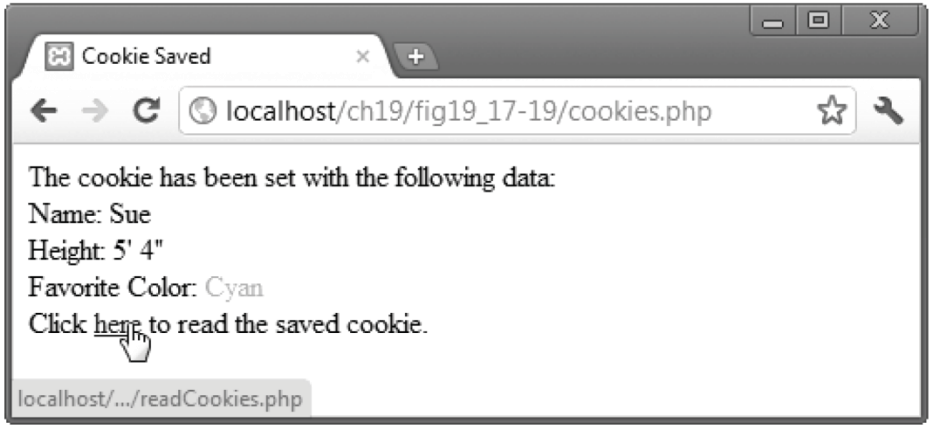


Fig. 19.18 | Writing a cookie to the client. (Part 3 of 3.)

19.10 Using Cookies

- 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
- Iterate through the **$_COOKIE** array using a **foreach** statement, printing out the name and value of each cookie in a paragraph
- The **foreach** statement takes the name of the array (**$_COOKIE**) and iterates through each index value of the array (**$key**)
  - In this case, the index values are the names of the cookies

19.10 Using Cookies

- Each element is then stored in variable **$value**, and these values become the individual cells of the table
- Try closing your browser and revisiting readCookies.php to confirm that the cookie has persisted

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.19: readCookies.php -->
4  <!-- Displaying the cookie's contents. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Read Cookies</title>
9        <style type = "text/css">
10          p { margin: 0px; }
11       </style>
12    </head>
13    <body>
14       <p>The following data is saved in a cookie on your computer.</p>
15       <?php
16          // iterate through array $_COOKIE and print
17          // name and value of each cookie
18          foreach ($_COOKIE as $key => $value )
19             print( "<p>$key: $value</p>" );
20       ?><!-- end PHP script -->
21    </body>
22 </html>
```

Fig. 19.19 | Displaying the cookie's contents. (Part 1 of 2.)

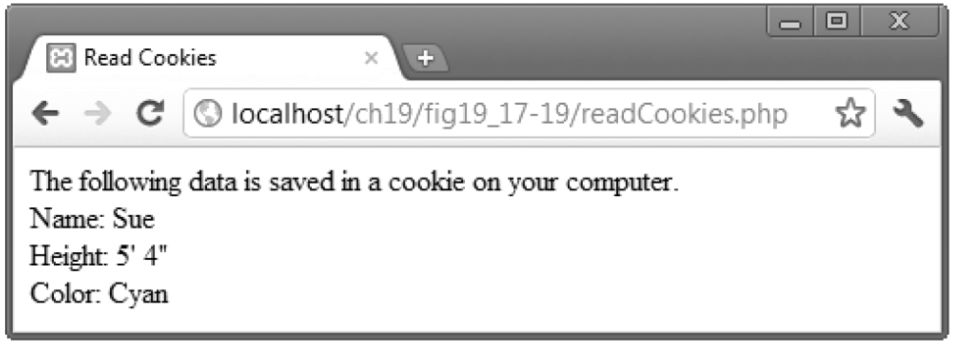


Fig. 19.19 | Displaying the cookie's contents. (Part 2 of 2.)

19.11 Dynamic Content

- PHP can dynamically change the HTML5 it outputs based on a user's input
- The form is created using a series of loops, arrays and conditionals
- We add error checking to each of the text input fields and inform the user of invalid entries on the form itself, rather than on an error page
- If an error exists, the script maintains the previously submitted values in each form element
- Finally, after the form has been successfully completed, we store the input from the user in a MySQL database

```
<!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

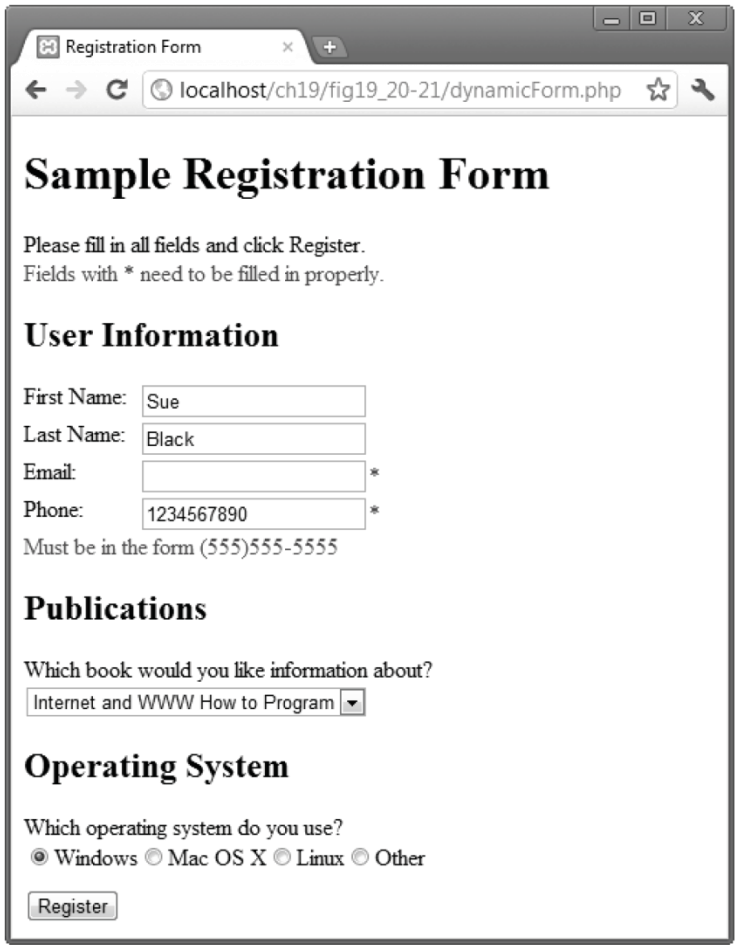


Fig. 19.20 | Dynamic form. (Part 9 of 10.)

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

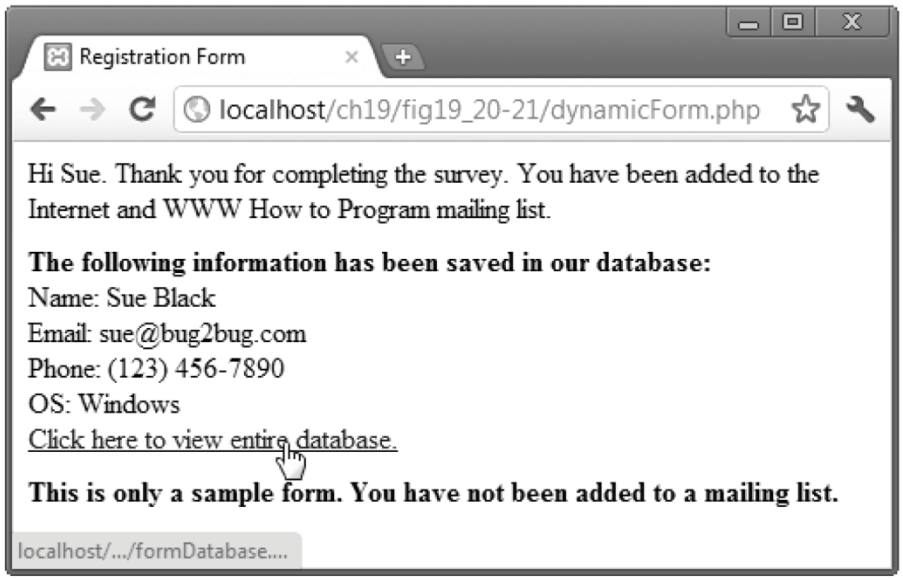


Fig. 19.20 | Dynamic form. (Part 10 of 10.)

19.11 Dynamic Content

- Displaying the database's contents
  - The script in Fig. 19.21 displays the contents of the MailingList database using the same techniques that we showed in Fig. 19.16
- In line 132 the value of **$$inputname** is assigned to the text field's value attribute
  - **print(" <label>$inputalt:</label><input type = 'text' name = '$inputname' value = '" . $$inputname . "'>");**
  - If the form has not yet been submitted, this will be the empty string ""
  - The notation **$$variable** specifies a variable variable, which allows the code to reference variables dynamically

19.11 Dynamic Content

- You can use this expression to obtain the value of the variable whose name is equal to the value of **$variable**
- PHP first determines the value of **$variable**, then appends this value to the leading $ to form the identifier of the variable you wish to reference dynamically
  - The expression $$variable can also be written as ${$variable}
- lines 129–138, we use **$$inputname** to reference the value of each form-field variable
- During the iteration of the loop, **$inputname** contains the name of one of the text input elements, such as "email“
- PHP replaces **$inputname** in the expression **$$inputname** with the string representing that element’s name forming the expression **${"email"}**

```
<!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.)

19.12 Web Resources

- The Deitel PHP Resource Center contains links to some of the best PHP information on the web
  - http://www.deitel.com/PHP/
- w3schools
  - http://www.w3schools.com/php/default.asp