PHP Functions

- Besides the built-in PHP functions, we can create our own functions.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.
- A user defined function declaration starts with the word "function":

```
function functionName() {
    code to be executed;}
```

- A function name can start with a letter or underscore (not a number).
- Function names are NOT case-sensitive.

```
<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // call the function
?>
```

```php
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

Output:

Jani Refsnes.
Hege Refsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.

```php
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight
<br>";
}

setHeight(350);
setHeight(); // will use the default value
of 50
setHeight(135);
setHeight(80);
?>
```

PHP – Sort Functions For Arrays

- sort() – sort arrays in ascending order
- rsort() – sort arrays in descending order
- asort() – sort associative arrays in ascending order, according to the value
- ksort() – sort associative arrays in ascending order, according to the key
- arsort() – sort associative arrays in descending order, according to the value
- krsort() – sort associative arrays in descending order, according to the key

Sort Array in Ascending Order – sort()

- The following example sorts the elements of the $cars array in ascending alphabetical order:

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);

$clength = count($cars);
for($x = 0; $x < $clength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
?>

</body>
</html>
```

Sort Array in Ascending Order – sort()

- The following example sorts the elements of the $numbers array in ascending numerical order:

```
<!DOCTYPE html>
<html>
<body>

<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);

$arrlength = count($numbers);
for($x = 0; $x < $arrlength; $x++) {
    echo $numbers[$x];
    echo "<br>";
}
?>

</body>
</html>
```

Sort Array in Descending Order – rsort()

- The following example sorts the elements of the $cars array in descending alphabetical order:

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);

$clength = count($cars);
for($x = 0; $x < $clength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
?>

</body>
</html>
```

Sort Array in Descending Order – rsort()

- The following example sorts the elements of the $numbers array in descending numerical order:

```
<!DOCTYPE html>
<html>
<body>

<?php
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers);

$arrlength = count($numbers);
for($x = 0; $x < $arrlength; $x++) {
    echo $numbers[$x];
    echo "<br>";
}
?>

</body>
</html>
```

Sort Array (Ascending Order), According to Value – asort()

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>

</body>
</html>
```

Sort Array (Ascending Order), According to Key – ksort()

- The following example sorts an associative array in ascending order, according to the key:

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>

</body>
</html>
```

Sort Array (Descending Order), According to Value – arsort()

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>

</body>
</html>
```

Sort Array (Descending Order), According to Key – krsort()

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>

</body>
</html>
```

Formatting User Input/Output

- addslashes()
- stripslashes()
- strip_tags()
- htmlspecialchars()

addslashes()

- returns a string with backslashes in front of predefined characters (Demo).
- The predefined characters are:
  - single quote (')
  - double quote (")
  - backslash (\)
  - NULL
- **Tip:** This function can be used to prepare a string for storage in a database and database queries.

stripslashes()

removes backslashes added by the addslashes() function.

```
<!DOCTYPE html>
<html>
<body>

<?php
echo stripslashes("Who\'s Peter Griffin?");
?>

</body>
</html>
```

Output:

Who's Peter Griffin?

strip_tags()

Strips a string from HTML, XML, and PHP tags.

(Demo)

htmlspecialchars()

- The htmlspecialchars() function converts some predefined characters to HTML entities.

- The predefined characters are:
  - & (ampersand) becomes &
  - " (double quote) becomes "
  - ' (single quote) becomes '
  - < (less than) becomes <
  - > (greater than) becomes >

  - (Demo)

PHP 5 Include Files

- The include statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.
- Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.
- It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include statement.
- Syntax: `include 'filename';`

- Assume we have a standard footer file called "footer.php", that looks like this:

```
<?php
  echo "<p>Copyright &copy; 1999-" . date("Y")
  . " W3Schools.com</p>";
  ?>
```

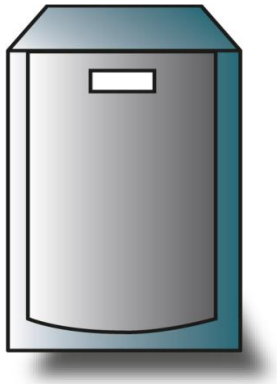
- To include the footer file in a page, use the include statement (Demo, Demo2):

```
<html>
  <body>
  <h1>Welcome to my home page!</h1>
  <p>Some text.</p>
  <p>Some more text.</p>
  <?php include 'footer.php';?>
</body></html>
```

Form Processing: How forms work

User requests a particular URL

XHTML Page supplied with Form



Web Server

User fills in form and submits.
Another URL is requested and the
Form data is sent to this page either in
URL or as a separate piece of data.

User

XHTML Response

PHP for Forms

- HTML Forms are used to select different kinds of user input.
- Set the form action attribute to
  - <form action=" echo htmlspecialchars($_SERVER["PHP_SELF"]); " method="post"> - or
  - <form action="script.php" method="post">;
- Make sure that you name each form field that you want to process as these names will be available to the processing script as variables
  - <input type="text" name="inputtext">
  - The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

19.8 Form Processing and Business Logic

- Superglobal arrays 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.
- The arrays $_GET and $_POST retrieve information sent to the server by HTTP get and post requests, respectively.
- These are superglobals, which means that they are always accessible, regardless of scope – and you can access them from any function, class or file without having to do anything special.

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.

19.8.2 Using PHP to Process HTML5 Forms

- Using `method = "post"` appends form data to the browser request that contains the protocol and the requested resource's URL. Scripts located on the web server's machine can access the form data sent as part of the request.

- Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

Reminder: Using GET

- Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters.
- GET may be used for sending non-sensitive data.

19.4 Form Processing and Business Logic (Cont.)

- We escape the normal meaning of a character in a string by preceding it with the backslash character (\).
- Function `die` *terminates* script execution. The function's optional argument is a string, which is printed as the script exits.

```
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

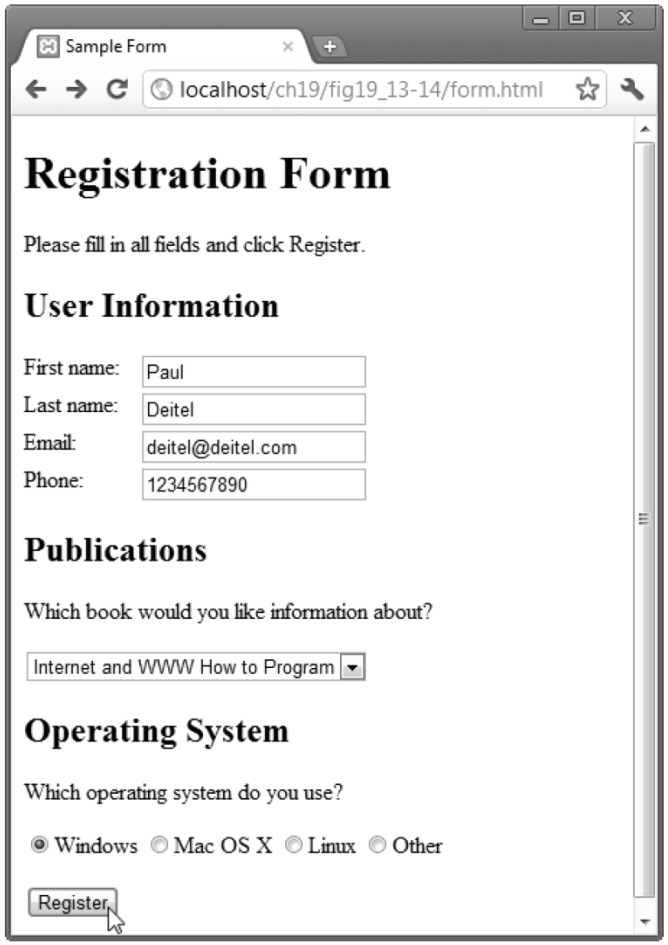


Fig. 19.13 | HTML5 form for gathering user input. (Part 4 of 4.)

Good Programming Practice 19.1

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

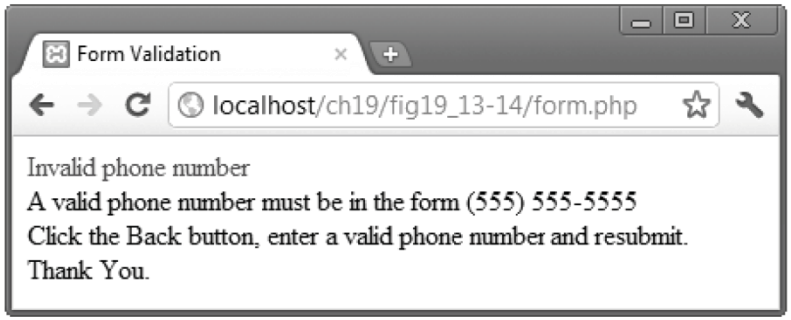
```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.14: form.php -->
4  <!-- Process information sent from form.html. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Form Validation</title>
9        <style type = "text/css">
10          p       { margin: 0px; }
11          .error  { 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

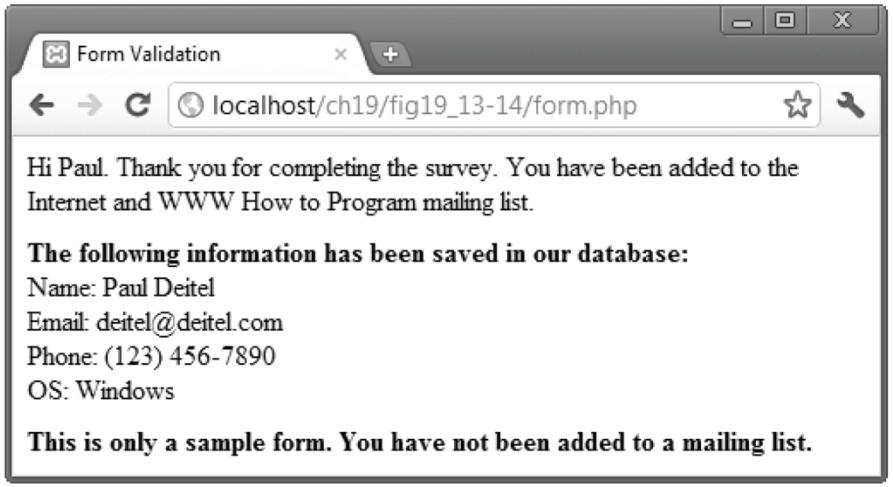


Fig. 19.14 | Process information sent from `form.html`. (Part 3 of 3.)

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

- Function `mysql_connect` connects to the MySQL database. It takes three arguments—
  - the server's hostname
  - a username
  - a password

  and returns a database handle—a representation of PHP's connection to the database, or `false` if the connection fails.
- Function `mysql_select_db` selects and opens the database to be queried.
- The function returns `true` on success or `false` on failure.

19.9 Reading from a Database (Cont.)

- To query the database, we call function `mysql_query`, specifying the query string and the database to query.
- This returns a resource containing the result of the query, or false if the query fails.
- It can also execute SQL statements such as `INSERT` or `DELETE` that do not return results.
- The `mysql_error` function returns any error strings from the database.
- `mysql_close` closes the connection to the database specified in its argument.
- The `mysql_fetch_row` function returns an array containing the values for each column in the current row of the query result (`$result`).

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.15: data.html -->
4  <!-- Form to query a MySQL database. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Sample Database Query</title>
9     </head>
10    <body>
11       <h1>Querying a MySQL database.</h1>
12       <form method = "post" action = "database.php">
13          <p>Select a field to display:
14             <!-- add a select box containing options -->
15             <!-- for SELECT query -->
16             <select name = "select">
17                <option selected>*</option>
18                <option>ID</option>
19                <option>Title</option>
20                <option>Category</option>
21                <option>ISBN</option>
22             </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>
```

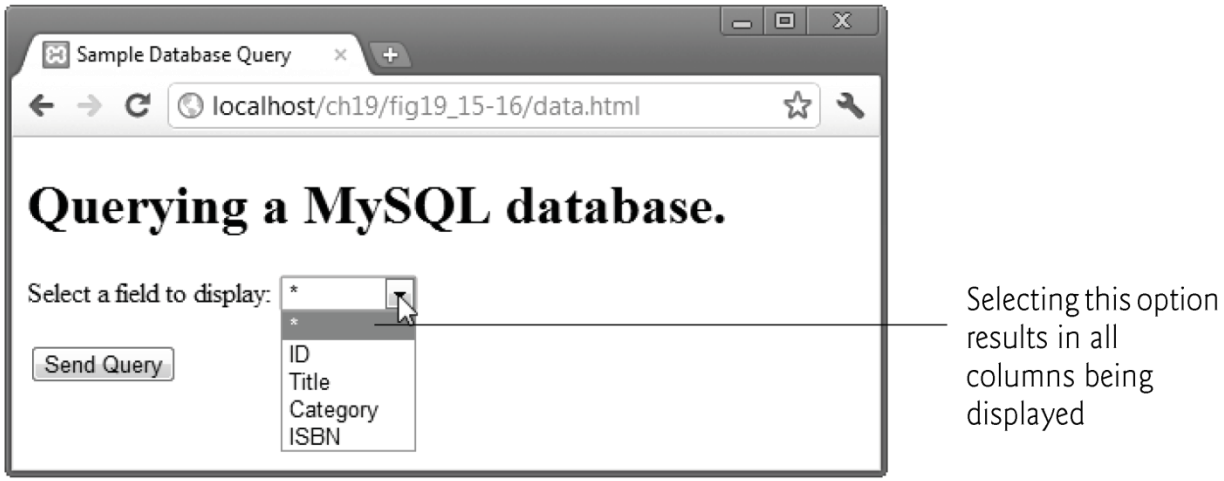


Fig. 19.15 | Form to query a MySQL database. (Part 2 of 2.)

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.16: database.php -->
4  <!-- Querying a database and displaying the results. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Search Results</title>
9     <style type = "text/css">
10          body  { font-family: sans-serif;
11                  background-color: lightyellow; }
12          table { background-color: lightblue;
13                  border-collapse: collapse;
14                  border: 1px solid gray; }
15          td    { padding: 5px; }
16          tr:nth-child(odd) {
17                  background-color: white; }
18       </style>
19    </head>
20    <body>
21       <?php
22          $select = $_POST["select"]; // creates variable $select
23
```

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.)

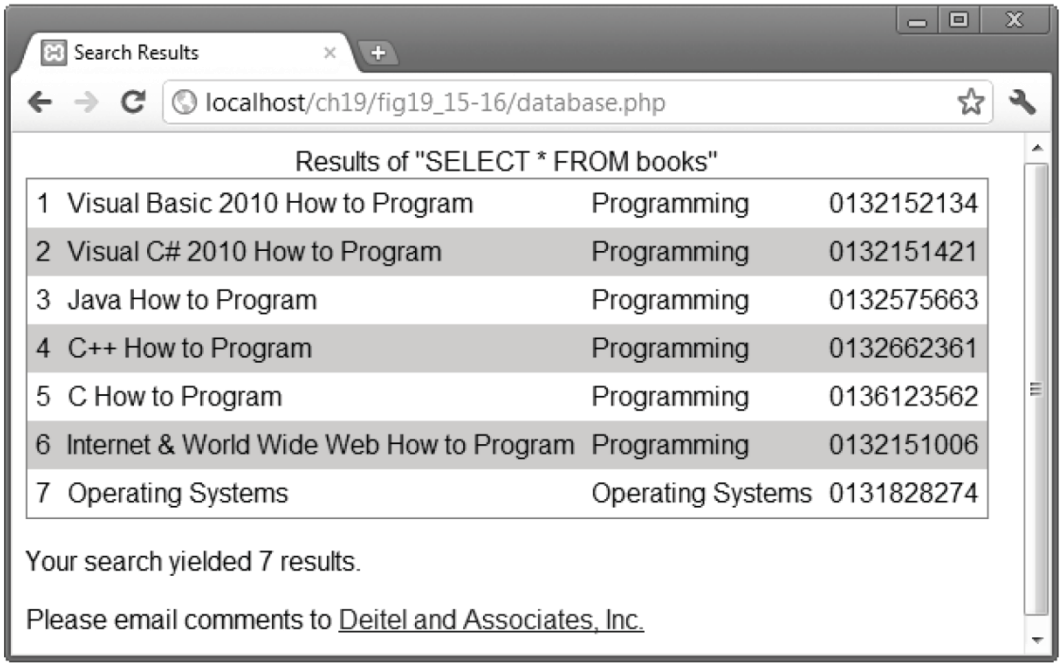


Fig. 19.16 | Querying a database and displaying the results. (Part 4 of 4.)