

# 5

## Using Strings

As introduced in Chapter 2, “Variables,” a second category of variables used by PHP is strings—a collection of characters enclosed within either single or double quotation marks. A string variable may consist of a single letter, a word, a sentence, a paragraph, HTML code, or even a jumble of nonsensical letters, numbers, and symbols (which might represent a password). Strings may be the most common variable type used in PHP.

This chapter covers PHP’s most basic built-in functions and operators for manipulating string data, regardless of whether the string originates from a form or is first declared within the script. Some common techniques will be introduced—joining strings together, trimming strings, and encoding strings. Other uses for strings are illustrated in subsequent chapters.

---

### In This Chapter

Creating the HTML Form	92
Concatenating Strings	95
Handling Newlines	98
HTML and PHP	100
Encoding and Decoding Strings	103
Finding Substrings	107
Replacing Parts of a String	111
Review and Pursue	114

---

[http://www.w3schools.com/php/php\\_form\\_validation.asp](http://www.w3schools.com/php/php_form_validation.asp)

# Creating the HTML Form

As in Chapter 3, let's begin by creating an HTML form that sends different values—in the form of string variables—to a PHP script. The theoretical example being used is an online bulletin board or forum where users can post a message, their email address, and their first and last names <sup>A</sup>.

## To create the HTML form:

1. Begin a new HTML document in your text editor or IDE, to be named **posting.php** (Script 5.1):

```
<!DOCTYPE html>
```

```
<head>
```

```
  <meta charset=utf-8"/>
  <title>Forum Posting</title>
```

```
</head>
```

```
<body>
```

```
<!-- Script 5.1 - posting.php -->
<div><p>Please complete this form
→ to submit your posting:</p>
```

2. Create the initial **form** tag:

```
<form action="handle_post.php"
→ method="post">
```

This form will send its data to the **handle\_post.php** script and will use the POST method.



<sup>A</sup> This HTML form is the basis for most of the examples in this chapter.

**Script 5.1** This form sends string data to a PHP script.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <metacharset=utf-8"/>
5    <title>Forum Posting</title>
6  </head>
7
8  <body>
9    <!-- Script 5.1 - posting.html -->
10   <div><p>Please complete this form to
11     submit your posting:</p>
12   <form action="handle_post.php"
13     method="post">
14     <p>First Name: <input
15       type="text" name="first_name"
16       size="20"></p>
17     <p>Last Name: <input
18       type="text" name="last_name"
19       size="20"></p>
20     <p>Email Address: <input
21       type="email" name="email"
22       size="30"></p>
23     <p>Posting: <textarea name="posting"
24       rows="9" cols="30"></textarea></p>
25     <input type="submit"
26       name="submit" value="Send My
27       Posting">
28   </form>
29 </div>
30 </body>
31 </html>
```

3. Add inputs for the first name, last name, and email address:

```
<p>First Name: <input type="text"
→ name="first_name" size="20"></p>
<p>Last Name: <input type="text"
→ name="last_name" size="20"></p>
<p>Email Address: <input type=
→ "email" name="email"
→ size="30"></p>
```

These are all basic text input types, which were covered in Chapter 3.

Remember that the various inputs' name values should adhere to the rules of PHP variable names (no spaces; must not begin with a number; must consist only of letters, numbers, and the underscore).

4. Add an input for the posting:

```
<p>Posting: <textarea
→ name="posting" rows="9"
→ cols="30"></textarea></p>
```

The posting field is a **textarea**, which is a larger type of text input box.

5. Create a submit button and close the form:

```
<input type="submit" name="submit"
→ value="Send My Posting">
</form>
```

Every form must have a submit button (or a submit image).

6. Complete the HTML page:

```
</div>
</body>
</html>
```

*continues on next page*

7. Save the file as **posting.php**, place it in the appropriate directory on your PHP-enabled server, and view it in your Web browser **A**.

This is an HTML page, so it doesn't have to be on a PHP-enabled server in order for you to view it. But because it will eventually send data to a PHP script, it's best to go ahead and place the file on your server.

**TIP** Technically speaking, all form data, aside from uploaded files, is sent to the handling script as strings. This includes numeric data entered into text boxes, options selected from drop-down menus, checkbox or radio button values, and so forth. Even the form in Chapter 4, "Using Numbers," sent strings with numeric values to the handling script.

**TIP** Many forum systems written in PHP are freely available for your use. This book doesn't discuss how to fully develop one, but a multilingual forum is developed in my *PHP 6 and MySQL 5 for Dynamic Web Sites: Visual QuickPro Guide* (Peachpit Press, 2007).

**TIP** This book's Web site has a forum where readers can post questions and other readers (and the author) answer questions. You can find it at [www.LarryUllman.com/forum/list.php?30](http://www.LarryUllman.com/forum/list.php?30).

# Concatenating Strings

*Concatenation* is an unwieldy term but a useful concept. It refers to the appending of one item onto another. Specifically, in programming, you concatenate *strings*. The period (.) is the operator for performing this action, and it's used like so:

```
$s1 = 'Hello, ';  
$s2 = 'world!';  
$greeting = $s1 . $s2;
```

The end result of this concatenation is that the **\$greeting** variable has a value of *Hello, world!*.

Because of the way PHP deals with variables, the same effect could be accomplished using

```
$greeting = "$s1$s2";
```

This code works because PHP replaces variables within double quotation marks with their value. However, the formal method of using the period to concatenate strings is more commonly used and is recommended (it will be more obvious what's occurring in your code).

Another way of performing concatenation involves the *concatenation assignment operator*:

```
$greeting = 'Hello, ';  
$greeting .= 'world!';
```

This second line roughly means “assign to **\$greeting** its current value plus the concatenation of *world!*” The end result is **\$greeting** having the value *Hello, world!* once again.

The **posting.php** script sends several string variables to the **handle\_post.php** page. Of those variables, the first and last names could logically be concatenated. It's quite common, and even recommended, to take a user's first and last names as separate inputs, as this form does. On the other hand, it would be advantageous to be able to refer to the two together as one name. You'll write the PHP script with this in mind.

## To use concatenation:

1. Begin a new document in your text editor or IDE, to be named **handle\_post.php** (Script 5.2):

```
<!DOCTYPE html>

<html lang="en">
<head>

    <meta charset="utf-8">

    <title>Forum Posting</title>
</head>
<body>
```

2. Create the initial PHP tag, and address error management, if necessary:

```
<?php // Script 5.2 -
→ handle_post.php
```

If you don't have **display\_errors** enabled, or if **error\_reporting** is set to the wrong level, see Chapter 3 for the lines to include here to alter those settings.

3. Assign the form data to local variables:

```
$first_name = $_POST['first_name'];
$last_name = $_POST['last_name'];
$posting = $_POST['posting'];
```

The form uses the POST method, so all the form data will be available in **\$\_POST**.

This example doesn't have a line for the email address because you won't be using it yet, but you can replicate this code to reference that value as well.

4. Create a new **\$name** variable using concatenation:

```
$name = $first_name . ' ' .
→ $last_name;
```

**Script 5.2** This PHP script demonstrates *concatenation*, one of the most common manipulations of a string variable. Think of it as addition for strings.

```
1  <!DOCTYPE html">
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5
6      <title>Forum Posting</title>
7  </head>
8  <body>
9  <?php // Script 5.2 - handle_post.php
10 /* This script receives five values
    from posting.php
11 first_name, last_name, email, posting,
    submit */
12
13 // Address error management, if you
    want.
14
15 // Get the values from the $_POST
    array:
16 $first_name = $_POST['first_name'];
17 $last_name = $_POST['last_name'];
18 $posting = $_POST['posting'];
19
20 // Create a full name variable:
21 $name = $first_name . ' ' .
    $last_name;
22
23 // Print a message:
24 print "<div>Thank you, $name, for
    your posting:
25 <p>$posting</p></div>";
26
27 ?>
28 </body>
29 </html>
```

Please complete this form to submit your posting:

First Name:

Last Name:

Email Address:

Posting: 

This is my posting. It could be more original.

**A** The HTML form in use...

Thank you, Jeremy Messersmith, for your posting.

This is my posting. It could be more original.

**B** ...and the resulting PHP page.

**TIP** You can link as many strings as you want using concatenation. You can even join numbers to strings:

```
$new_string = $s1 . $s2 . $number;
```

This works because PHP is *weakly typed*, meaning that its variables aren't locked in to one particular format. Here, the `$number` variable will be turned into a string and appended to the value of the `$new_string` variable.

**TIP** Concatenation can be used in many ways, even when you're feeding arguments to a function. An uncommon but functional example would be

```
$text = nl2br($heading . $body);
```

The `nl2br()` function, first mentioned in Chapter 1, "Getting Started with PHP," will be discussed in detail next.

This act of concatenation takes two variables plus a space and joins them all together to create a new variable, called `$name`. Assuming you entered *Elliott* and *Smith* as the names, then `$name` would be equal to *Elliott Smith*.

5. Print out the message to the user:

```
print "<div>Thank you, $name, for  
→ your posting:  
<p>$posting</p></div>";
```

This message reports back to the user what was entered in the form.

6. Close the PHP section and complete the HTML page:

```
?>  
</body>  
</html>
```

7. Save your script as **handle\_post.php**, place it in the same directory as **posting.php** (on your PHP-enabled server), and test both the form and the script in your Web browser **A** and **B**.

As a reminder, you must load the form through a URL (*http://localhost*) so that, when the form is submitted, the handling PHP script is also run through a URL.

**TIP** If you used quotation marks of any kind in your form and saw extraneous slashes in the printed result, ~~see the sidebar “Magic Quotes” in Chapter 3 for an explanation of the cause and for the fix.~~

**TIP** As a reminder, it's important to understand the difference between single and double quotation marks in PHP. Characters within single quotation marks are treated literally; characters within double quotation marks are interpreted (for example, a variable's name will be replaced by its value). See Chapter 3 for a refresher.

# Handling Newlines

A common question beginning PHP developers have involves handling newlines in strings. The `textarea` form element allows a user to enter text over multiple lines by pressing Return/Enter. Each use of Return/Enter equates to a newline in the resulting string. These newlines work within a `textarea` but have no effect on a rendered PHP page **A** and **B**.

To create the equivalent of newlines in a rendered Web page, you use the break tag: `<br>`. Fortunately, PHP has the `nl2br()` function, which automatically converts newlines into break tags:

```
$var = nl2br($var);
```

Let's apply this function to `handle_post.php` so that the user's posting retains its formatting.

## To convert newlines to breaks:

1. Open `handle_post.php` (Script 5.2) in your text editor or IDE, if it is not already open.
2. Apply the `nl2br()` function when assigning a value to the `$posting` variable (Script 5.3):

```
$posting = nl2br($_POST['posting']);
```

Now `$posting` will be assigned the value of `$_POST['posting']`, with any newlines converted to HTML break tags.

Please complete this form to submit your posting:

First Name:

Last Name:

Email Address:

Here's one line.

Here's another line.

Here's a third line.

Posting:

**A** Newlines in form data like text areas...

Thank you, Rocky Votolato, for your posting:

Here's one line. Here's another line. Here's a third line.

**B** ...are not rendered by the Web browser.

Thank you, Rocky Votolato, for your posting:

Here's one line.

Here's another line.

Here's a third line.

**C** Now the same submitted data **A** is properly displayed over multiple lines in the Web browser.

```
<div>Thank you, Rocky Votolato, for your posting:
<p>Here's one line.

Here's another line.

Here's a third line.</p></div></body>
</html>
```

**D** The HTML source, corresponding to **B**, shows the effect that newlines have in the Web browser (i.e., they add spacing within the HTML source code).



**Script 5.3** When you use the `n12br()` function, newlines entered into the posting **textarea** are honored when displayed in the Web browser.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>Forum Posting</title>
6
7  </head>
8  <body>
9  <?php // Script 5.3 - handle_post.php #2
10 /* This script receives five values
   from posting.html:
11 first_name, last_name, email, posting,
   submit */
12
13 // Address error management, if you
   want.
14
15 // Get the values from the $_POST
   array:
16 $first_name = $_POST['first_name'];
17 $last_name = $_POST['last_name'];
18 $posting = n12br($_POST['posting']);
19
20 // Create a full name variable:
21 $name = $first_name . ' ' . $last_name;
22
23 // Print a message:
24 print "<div>Thank you, $name, for your
   posting:
25 <p>$posting</p></div>";
26
27 ?>
28 </body>
29 </html>
```

3. Save the file, place it in the same directory as **posting.php** (on your PHP-enabled server), and test again in your Web browser **C**.

**TIP** Newlines can also be inserted into strings by placing the newline character—`\n`—between double quotation marks.

**TIP** Other HTML tags, like paragraph tags, also affect spacing in the rendered Web page. You can turn newlines (or any character) into paragraph tags using a replace function, but the code for doing so is far more involved than just invoking `n12br()`.

**TIP** Newlines present in strings sent to the browser will have an effect, but only in the HTML source of the page **D**.

## HTML and PHP

As stated several times over by now, PHP is a server-side technology that's frequently used to send data to the Web browser. This data can be in the form of plain text, HTML code, or, more commonly, both.

In this chapter's primary example, data is entered in an HTML form and then printed back to the Web browser using PHP. A potential problem is that the user can enter HTML characters in the form, which can affect the resulting page's formatting **A** and **B**—or, worse, cause security problems.

You can use a couple of PHP functions to manipulate HTML tags within PHP string variables:

- **htmlspecialchars()** converts certain HTML tags into their *entity versions*.
- **htmlentities()** turns *all* HTML tags into their *entity versions*.
- **strip\_tags()** removes all HTML and PHP tags.

The first two functions turn an HTML tag (for example, `<span>`) into an entity version like `&lt;span&gt;`. The entity version appears in the output but isn't rendered. You might use either of these if you wanted to display code without enacting it. The third function, **strip\_tags()**, removes HTML and PHP tags entirely.

You ought to watch for special tags in user-provided data for two reasons. First, as already mentioned, submitted HTML would likely affect the rendered page (e.g., mess up a table, tweak the CSS, or just add formatting where there shouldn't be any). The second concern is more important. Because JavaScript is placed within HTML **script** tags, a malicious user could submit JavaScript that would be executed when it's redisplayed on the page **C**. This is how *cross-site scripting* (XSS) attacks are performed.

Please complete this form to submit your posting:

First Name:

Last Name:

Email Address:

Let's make an ordered list:

```
<ul>
<li>Something</li>
<li>Something Else</li>
<li>Something New</li>
</ul>
```

Posting:

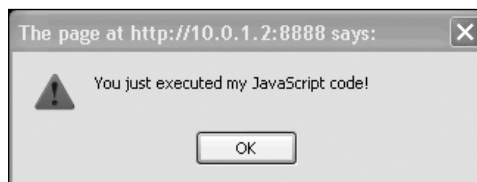
**A** If the user enters HTML code in the posting...

Thank you, Damien Rice, for your posting:

Let's make an ordered list:

- Something
- Something Else
- Something New

**B** ...it's rendered by the Web browser when reprinted.



**C** Displaying HTML submitted by a user in the Web browser can have terrible consequences, such as the execution of JavaScript.

**htmlspecialchars** renders content safe for display in HTML.

**htmlentities** does everything that **htmlspecialchars** but goes further and also encodes accent character sybols into their html entities.

[http://www.w3schools.com/Php/func\\_string\\_htmlspecialchars.asp](http://www.w3schools.com/Php/func_string_htmlspecialchars.asp)

**Script 5.4** This version of the PHP script addresses HTML tags in two different ways.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5
6      <title>Forum Posting</title>
7  </head>
8  <body>
9      <?php // Script 5.4 - handle_post.php
10     /* This script receives five values
11     from posting.html:
12     first_name, last_name, email, posting,
13     submit */
14
15     // Address error management, if you
16     want.
17
18     // Get the values from the $_POST
19     array:
20     $first_name = $_POST['first_name'];
21     $last_name = $_POST['last_name'];
22     $posting = nl2br($_POST['posting']);
23
24     // Create a full name variable:
25     $name = $first_name . ' ' . $last_name;
26
27     // Adjust for HTML tags:
28     $html_post = htmlentities($_POST
29     ['posting']);
30     $strip_post = strip_tags($_POST
31     ['posting']);
32
33     // Print a message:
34     print "<div>Thank you, $name, for
35     your posting:
36     <p>Original: $posting</p>
37     <p>Entity: $html_post</p>
38     <p>Stripped: $strip_post</p></div>";
39
40     ?>
41 </body>
42 </html>
```

To see the impact these functions have, this next rewrite of **handle\_post.php** will use them each and display the respective results.

## To address HTML in PHP:

1. Open **handle\_post.php** (Script 5.3) in your text editor or IDE, if it is not already open.
2. Before the **print** line, add (Script 5.4):

```
$html_post = htmlentities
→ ($_POST['posting']);
$strip_post = strip_tags
→ ($_POST['posting']);
```

To clarify the difference between how these two functions work, apply them both to the posting text, creating two new variables in the process. Refer to **\$\_POST['posting']** here and not **\$posting** because **\$posting** already reflects the application of the **nl2br()** function, which means that break tags may have been introduced that were not explicitly entered by the user.

3. Alter the **print** statement to read as follows:

```
print "<div>Thank you, $name, for
→ your posting:
<p>Original: $posting</p>
<p>Entity: $html_post</p>
<p>Stripped: $strip_post</p></div>";
```

To highlight the different results, print out the three different versions of the posting text. First is the original posting as it was entered, after being run through **nl2br()**. Next is the **htmlentities()** version of the posting, which will show the HTML tags without rendering them. Finally, the **strip\_tags()** version will be printed; it doesn't include any HTML (or PHP) tags.

*continues on next page*

4. Save the file, place it in the same directory as `posting.php` (on your PHP-enabled server), and test it again in your Web browser **D** and **E**.

If you view the HTML source code of the resulting PHP page **F**, you'll also see the effect that applying these functions has.

**TIP** For security purposes, it's almost always a good idea to use `htmlspecialchars()`, `htmlspecialchars()`, or `strip_tags()` to any user-provided data that's being printed to the Web browser. I don't do so through the course of this book only to minimize clutter.

**TIP** The `html_entity_decode()` function does just the opposite of `htmlspecialchars()`, turning HTML entities into their respective HTML code.

**TIP** Another useful function for outputting strings in the Web browser is `wordwrap()`. This function wraps a string to a certain number of characters.

**TIP** To turn newlines into breaks while still removing any HTML or PHP tags, apply `nl2br()` after `strip_tags()`:

```
$posting =  
nl2br(strip_tags($_POST['posting']));
```

In that line, the `strip_tags()` function will be called first, and its result will be sent to the `nl2br()` function.

Please complete this form to submit your posting:

First Name:

Last Name:

Email Address:

Posting: 

I don't understand why it says  
<em>something</em>.

**D** The HTML characters entered as part of a posting will now be addressed by PHP.

Thank you, Laura Burhenn, for your posting:

Original: I don't understand why it says *something*.

Entity: I don't understand why it says <em>something</em>.

Stripped: I don't understand why it says something.

**E** The resulting PHP page shows the original post as it would look if printed without modification, the effect of `htmlspecialchars()`, and the effect of `strip_tags()`.

```
<p>Original: I don't understand why it says <em>something</em>.</p>  
<p>Entity: I don't understand why it says &lt;em&gt;something&lt;/em&gt;.</p>  
<p>Stripped: I don't understand why it says something.</p></div></body>
```

**F** The HTML source for the content displayed in **E**.

**Script 5.5** This script encodes two variables before adding them to a link. Then the values can be successfully passed to another page.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD
  XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="Content-Type"
6     content="text/html; charset=utf-8"/>
7   <title>Forum Posting</title>
8 </head>
9 <body>
10 <?php // Script 5.5 - handle_post.php
11 /* This script receives five values
12    from posting.html:
13    first_name, last_name, email, posting,
14    submit */
15
16 // Address error management, if you
17 want.
18
19 // Get the values from the $_POST
20 array:
21 $first_name = $_POST['first_name'];
22 $last_name = $_POST['last_name'];
23 $posting = nl2br($_POST['posting']);
24
25 // Create a full name variable:
26 $name = $first_name . ' ' . $last_name;
27
28 // Print a message:
29 print "<div>Thank you, $name, for your
30 posting:
31 <p>$posting</p></div>";
32
33 // Make a link to another page:
34 $name = urlencode($name);
35 $email = urlencode($_POST['email']);
36 print "<p>Click <a href=\"  thanks.php?
37 name=$name&email=$email  \">>here
38 </a> to continue.</p>";
39
40 ?>
41 </body>
42 </html>
```

## Encoding and Decoding Strings

At the end of Chapter 3, the section “Manually Sending Data to a Page” demonstrated how to use the thinking behind the GET form method to send data to a page. In that example, instead of using an actual form, data was appended to the URL, making it available to the receiving script. I was careful to say that only single words could be passed this way, without spaces or punctuation. But what if you want to pass several words as one variable value or use special characters?

To safely pass any value to a PHP script through the URL, apply the `urlencode()` function. As its name implies, this function takes a string and *encodes* it (changes its format) so that it can properly be passed as part of a URL. Among other things, the function replaces spaces with plus signs (+) and translates special characters (for example, the apostrophe) into less problematic versions. To use this function, you might code

```
$string = urlencode($string);
```

To demonstrate one application of `urlencode()`, let's update the `handle_post.php` page so that it also creates a link that passes the user's name and email address to a third page.

### To use `urlencode()`:

1. Open `handle_post.php` (Script 5.4) in your text editor or IDE, if it is not already open.
2. Delete the `htmlentities()` and `strip_tags()` lines added in the previous set of steps (Script 5.5).

*continues on next page*

3. Revert to the older version of the **print** invocation:

```
print "<div>Thank you, $name, for  
→ your posting:  
<p>$posting</p></div>";
```

4. After the **print** statement, add the following:

```
$name = urlencode($name);  
$email = urlencode($_POST['email']);
```

This script will pass these two variables to a second page. In order for it to do so, they must both be encoded.

Because the script has not previously referred to or used the **\$email** variable, the second line both retrieves the email value from the **\$\_POST** array and encodes it in one step. This is the same as having these two separate lines:

```
$email = $_POST['email'];  
$email = urlencode($email);
```

5. Add another **print** statement that creates the link:

```
print "<p>Click <a href=\"thanks.  
→ php?name=$name&email=$email\">  
→ here</a> to continue.</p>";
```

The primary purpose of this **print** statement is to create an HTML link in the Web page, the source code of which would be something like

```
<a href="thanks.php?name=Larry+  
→ Ullman&email=larry%40example.  
→ com">here</a>
```

To accomplish this, begin by hard-coding most of the HTML and then include the appropriate variable names. Because the HTML code requires that the URL for the link be in double quotation marks—and the **print** statement already uses double quotation marks—you must escape them (by preceding them with backslashes) in order for them to be printed.

Please complete this form to submit your posting:

First Name:

Last Name:

Email Address:

Nothing like a piano cover of  
Radiohead or Elliott Smith!

Posting:

**A** Another use of the form.

Thank you, Christopher O'Reilly, for your posting:

Nothing like a piano cover of Radiohead or Elliott Smith!

Click [here](#) to continue.

**B** The handling script now displays a link to another page.

```
<p>Click <a href="thanks.php?name=Christopher+O%27Reilly&email=chris.oreilly%40example.com">here</a>
```

**C** The HTML source code of the page **B** shows the dynamically generated link.

6. Save the file, place it in the proper directory of your PHP-enabled server, and test it again in your Web browser **A** and **B**.

Note that clicking the link will result in a server error, as the **thanks.php** script hasn't yet been written.

7. View the HTML source code of the handling page to see the resulting link in the HTML code **C**.

**TIP** Values sent directly from a form are automatically URL-encoded prior to being sent and decoded upon arrival at the receiving script. You only need the `urlencode()` function to manually encode data (as in the example).

**TIP** The `urldecode()` function does just the opposite of `urlencode()`—it takes an encoded URL and turns it back into a standard form. You'll use it less frequently, though, as PHP will automatically decode most values it receives.

*continues on next page*

**TIP** Since you can use concatenation with functions, the new `print` statement could be written as follows:

```
print 'Click <a href="thanks.php?
→ name=' . $name . '&email=' .
→ $email . '">here</a> to continue.';
```

This method has two added benefits over the original approach. First, it uses single quotation marks to start and stop the statement, meaning you don't need to escape the double quotation marks. Second, the variables used are more obvious—they aren't buried in a lot of other code.

**TIP** You do not need to encode numeric PHP values in order to use them in a URL, as they do not contain problematic characters. That being said, it won't hurt to encode them either.

**TIP** At the end of the chapter you'll be prompted to create `thanks.php`, which greets the user by name and email address **D**.

Thank you, Christopher O'Reilly.  
We will contact you at `chris.oreilly@example.com`.

**D** The third page in this process—to be created by you at the end of the chapter—prints a message based on values it receives in the URL.

## Encrypting and Decrypting Strings

Frequently, in order to protect data, programmers *encrypt* it—alter its state by transforming it to a form that's more difficult, if not impossible, to discern. Passwords are an example of a value you might want to encrypt. Depending on the level of security you want to establish, usernames, email addresses, and phone numbers are likely candidates for encryption, too.

You can use the `crypt()` function to encrypt data, but be aware that no decryption option is available (it's known as *one-way* encryption). So, a password may be encrypted using it and then stored, but the decrypted value of the password can never be determined. Using this function in a Web application, you might encrypt a user's password upon registration; then, when the user logged in, the password they entered at that time would also be encrypted, and the two protected versions of the password would be compared. The syntax for using `crypt()` is

```
$data = crypt($data);
```

A second encryption function is `mcrypt_encrypt()`, which can be decrypted using the appropriately named `mcrypt_decrypt()` function. Unfortunately, in order for you to be able to use these two functions, the Mcrypt extension must be installed with the PHP module. Its usage and syntax is also more complex (I discuss it in my *PHP 5 Advanced: Visual QuickPro Guide* [Peachpit Press, 2007]).

If the data is being stored in a database, you can also use functions built into the database application (for example, MySQL, PostgreSQL, Oracle, or SQL Server) to perform encryption and decryption. Depending on the technology you're using, it most likely provides both one- and two-way encryption tools.



## Comparing Strings

To compare two strings, you can always use the equality operator, which you'll learn about in the next chapter. Otherwise, you can use these functions:

- **strcmp()** indicates how two strings compare by returning a whole number.
- **strnatcmp()** is similar but linguistically more precise.

These also have case-insensitive companions, **strcasecmp()** and **strnatcasecmp()**.

To see if a substring is contained within another string (i.e., to find a needle in a haystack), you'll use these functions:

- **strstr()** returns the haystack from the first occurrence of a needle to the end.
- **stripos()** searches through a haystack and returns the numeric location of a particular needle.

Both of these functions also have a case-insensitive alternative: **stristr()** and **stripos()**, respectively. Each of these functions is normally used in a conditional to test whether the substring was found.

## Finding Substrings

PHP has a few functions you can use to pull apart strings, search through them, and perform comparisons. Although these functions are normally used with conditionals, discussed in Chapter 6, “Control Structures,” they are important enough that they'll be introduced here; later chapters will use them more formally.

Earlier in this chapter you learned how to join strings using concatenation. Along with making larger strings out of smaller pieces, PHP easily lets you extract subsections from a string. The trick to using any method to pull out a subsection of a string is that you must know something about the string itself in order to know how to break it up.

The **strtok()** function creates a substring, referred to as a *token*, from a larger string by using a predetermined separator (such as a comma or a space). For example, if you have users enter their full name in one field (presumably with their first and last names separated by a space), you can pull out their first name with this code:

```
$first = strtok($_POST['name'], ' ');
```

That line tells PHP to extract everything from the beginning of **\$\_POST['name']** until it finds a blank space.

If you have users enter their full name in the format *Surname, First*, you can find their surname by writing

```
$last = strtok($_POST['name'], ', ');
```

A second way to pull out sections of a string is by referring to the *indexed position* of the characters within the string. The indexed position of a string is the numerical location of a character, counting from the beginning. However, PHP—like most programming languages—begins all indexes with the number 0. For example, to index the string *Larry*, you begin with the L at position 0, followed by *a* at 1, *r* at 2, the second *r* at 3, and *y* at 4. Even though the string length of *Larry* is 5, its index goes from 0 to 4 (i.e., indexes always go from 0 to the string's length minus 1).

With this in mind, you can call on the **substr()** function to create a substring based on the index position of the substring's characters:

```
$sub = substr($string, 0, 10);
```

The first argument is the master string from which the substring will be derived. Second, indicate where the substring begins, as its indexed position (0 means that you want to start with the first character). Third, from that starting point, state how many characters the substring should contain (10). If the master string does not have that many characters in it, the resulting substring will end with the end of the master string. This argument is optional; if omitted, the substring will also go until the end of the master string.

You can also use negative numbers to count backward from the end of the string:

```
$string = 'ardvark';  
$sub = substr($string, -3, 3); // ark
```

The second line says that three characters should be returned starting at the third character from the end. With that particular example, you can again omit the third argument and have the same result:

```
$sub = substr($string, -3); // ark
```

**Script 5.6** This version of `handle_post.php` counts the number of words in the posting and trims the displayed posting down to just the first 50 characters.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5
6      <title>Forum Posting</title>
7  </head>
8  <body>
9  <?php // Script 5.6 - handle_post.php
10 /* This script receives five values
   from posting.html:
11 first_name, last_name, email, posting,
   submit */
12
13 // Address error management, if you
   want.
14
15 // Get the values from the $_POST
   array:
16 $first_name = $_POST['first_name'];
17 $last_name = $_POST['last_name'];
18 $posting = nl2br($_POST['posting']);
19
20 // Create a full name variable:
21 $name = $first_name . ' ' . $last_name;
22
23 // Get a word count:
24 $words = str_word_count($posting);
25
26 // Get a snippet of the posting:
27 $posting = substr($posting, 0, 50);
28
29 // Print a message:
30 print "<div>Thank you, $name, for
   your posting:
31 <p>$posting...</p>
32 <p>($words words)</p></div>";
33
34 ?>
35 </body>
36 </html>
```

To see how many characters are in a string, use `strlen()`:

```
print strlen('Hello, world!'); // 13
```

The count will include spaces and punctuation. To see how many *words* are in a string, use `str_word_count()`. This function, along with `substr()`, will be used in this next revision of the `handle_post.php` script.

### To create substrings:

1. Open `handle_post.php` (Script 5.5) in your text editor or IDE, if it is not already open.
2. Before the `print` statement, add the following (Script 5.6):

```
$words = str_word_count($posting);
```

This version of the script will do two new things with the user's posting. One will be to display the number of words it contains. That information is gathered here and assigned to the `$words` variable.

3. On the next line (also before the `print` statement), add

```
$posting = substr($posting, 0, 50);
```

The second new thing this script will do is limit the displayed posting to its first 50 characters. You might use this, for example, if one page shows the beginning of a post, then a link takes the user to the full posting. To implement this limit, the `substr()` function is called.

*continues on next page*

- There are two changes here. First, ellipses are added after the posting to indicate that this is just part of the whole posting. Then, within another paragraph, the number of words is printed.

- I'm referring specifically to the code added in the previous incarnation of the script, linking to **thanks.php**.

- A** and **B**.

**TIP** If you want to check whether a string matches a certain format—for example, to see if it’s a valid email address—you need to use *regular expressions*. Regular expressions are an advanced concept in which you define patterns and then see if a value fits the mold. See the PHP manual or my book *PHP 6 and MySQL 5 for Dynamic Web Sites: Visual QuickPro Guide* (Peachpit Press, 2007).

	This is a longer post. This is a longer post. This is a longer post. This is a longer post. This is a longer post. This is a longer post. This is a longer post. This is a longer post.
Posting:	

- A** Postings longer than 50 characters...

Thank you, Regina Spektor, for your posting:  
This is a longer post. This is a longer post. This...  
(60 words)

- B** ...will be cut short. The word count is also displayed.

## Adjusting String Case

A handful of PHP functions are used to change the case of a string's letters:

- **ucfirst()** capitalizes the first letter of the string.
- **ucwords()** capitalizes the first letter of words in a string.
- **strtoupper()** makes an entire string uppercase.
- **strtolower()** makes an entire string lowercase.


Due to the variance in people's names around the globe, there's no flawless way to automatically format names with PHP (or any programming language). In fact, I would be hesitant to alter the case of user-supplied data unless you have good cause to do so.

## Replacing Parts of a String

Instead of just finding substrings within a string, as the previous section discusses, you might find that you need to *replace substrings* with new values. You can do so using the **str\_ireplace()** function:

```
$string = str_ireplace($needle,  
→ $replacement, $haystack);
```

This function replaces every occurrence of **\$needle** found in **\$haystack** with **\$replacement**. For example:

```
$me = 'Larry E. Ullman';  
$me = str_ireplace('E.', 'Edward',  
→ $me);  i - means case insensitive
```

The **\$me** variable now has a value of *Larry Edward Ullman*.

That function performs a *case-insensitive* search. To be more restrictive, you can perform a *case-sensitive* search using **str\_replace()**. In this next script, **str\_ireplace()** will be used to eliminate “bad words” in submitted text.

There's one last string-related function I want to discuss: **trim()**. This function removes any white space—spaces, newlines, and tabs—from the beginning and end of a string. It's quite common for extra spaces to be added to a string variable, either because a user enters information carelessly or due to sloppy HTML code. For purposes of clarity, data integrity, and Web design, it's worth your while to delete those spaces from the strings before you use them. Extra spaces sent to the Web browser could make the page appear oddly, and those sent to a database or cookie could have unfortunate consequences at a later date (for example, if a password has a superfluous space, it might not match when it's entered without the space).

The **trim()** function automatically strips away any extra spaces from both the beginning and the end of a string (but not the middle). The format for using **trim()** is as follows:

```
$string = ' extra space before and  
→ after text '  
$string = trim($string);  
// $string is now equal to 'extra  
→ space before and after text'.
```

### To use **str\_ireplace()** and **trim()**:

1. Open **handle\_post.php** (Script 5.6) in your text editor or IDE, if it is not already open.
2. Apply **trim()** to the form data (Script 5.7):

```
$first_name =  
→ trim($_POST['first_name']);  
$last_name =  
→ trim($_POST['last_name']);  
$posting = trim($_POST['posting']);
```

Just in case the incoming data has extraneous white space at its beginning or end, the **trim()** function is applied.

3. Remove the use of **substr()**:  

```
$posting = substr($posting, 0, 50);
```

You'll want to see the entire posting for this example, so remove this invocation of **substr()**.
4. Before the **print** statement, add  

```
$posting = str_ireplace('badword',  
→ 'XXXXX', $posting);
```

This specific example flags the use of a bad word in a posting by crossing it out. Rather than an actual curse word, the code uses **badword**. (You can use whatever you want, of course.)

**Script 5.7** This final version of the handling script applies the **trim()** function and then replaces uses of **badword** with a bunch of Xs.

```
1 <!DOCTYPE html>  
2 <html lang="en">  
3 <head>  
4     <meta charset="utf-8">  
5  
6     <title>Forum Posting</title>  
7 </head>  
8 <body>  
9 <?php // Script 5.7 - handle_post.php  
10 /* This script receives five values  
   from posting.html:  
11 first_name, last_name, email, posting,  
   submit */  
12  
13 // Address error management, if you  
   want.  
14  
15 // Get the values from the $_POST  
   array.  
16 // Strip away extra spaces using  
   trim():  
17 $first_name = trim($_POST  
   ['first_name']);  
18 $last_name = trim($_POST  
   ['last_name']);  
19 $posting = trim($_POST['posting']);  
20  
21 // Create a full name variable:  
22 $name = $first_name . ' ' . $last_name;  
23  
24 // Get a word count:  
25 $words = str_word_count($posting);  
26  
27 // Take out the bad words:  
28 $posting = str_ireplace('badword',  
   'XXXXX', $posting);  
29  
30 // Print a message:  
31 print "<div>Thank you, $name, for your  
   posting:  
32 <p>$posting</p>  
33 <p>($words words)</p></div>";  
34  
35 ?>  
36 </body>  
37 </html>
```

Please complete this form to submit your posting:

First Name:

Last Name:

Email Address:

I feel like using a BADWORD in my post!

Posting:

**A** If a user enters a word you'd prefer they not use...

Thank you, Bad Poster, for your posting:

I feel like using a XXXXXX in my post!

(9 words)

**B** ...you can have PHP replace it.

If you'd like to catch many bad words, you can use multiple lines, like so:

```
$posting = str_ireplace
→ ('badword1', 'XXXXX', $posting);
$posting = str_ireplace
→ ('badword2', 'XXXXX', $posting);
$posting = str_ireplace
→ ('badword3', 'XXXXX', $posting);
```

5. Update the **print** statement so that it no longer uses the ellipses:

```
print "<div>Thank you, $name, for
→ your posting:
<p>$posting</p>
<p>($words words)</p></div>";
```

6. Save the file, place it in the proper directory of your PHP-enabled server, and test again in your Web browser **A** and **B**.

**TIP** The `str_ireplace()` function will even catch bad words in context. For example, if you entered *I feel like using badwords*, the result would be *I feel like using XXXXXs*.

**TIP** The `str_ireplace()` function can also take an array of needle terms, an array of replacement terms, and even an array as the haystack. Because you may not know what an array is yet, this technique isn't demonstrated here.

**TIP** If you need to trim excess spaces from the beginning or the end of a string but not both, PHP breaks the `trim()` function into two more specific functions: `rtrim()` removes spaces found at the end of a string variable (on its right side), and `ltrim()` handles those at the beginning (its left). They're both used just like `trim()`:

```
$string = rtrim($string);
$string = ltrim($string);
```

# Review and Pursue

If you have any problems with the review questions or the pursue prompts, turn to the book's supporting forum ([www.LarryUllman.com/forum/](http://www.LarryUllman.com/forum/)).

## Review

- How do you create a string?
- What are the differences between using *single* and *double* quotation marks?
- What is the *concatenation* operator? What is the *concatenation assignment* operator?
- What is the impact of having a newline in a string printed to the browser? How do you convert a newline character to a break tag?
- What problems can occur when HTML is entered into form elements whose values will later be printed back to the Web browser? What steps can be taken to sanctify submitted form data?
- What function makes data safe to pass in a URL?
- How do you escape problematic characters within a string? What happens if you do not escape them?
- The characters in a string are indexed beginning at what number?
- What does the `trim()` function do?

## Pursue

- Look up the PHP manual page for one of the new functions mentioned in this chapter. Use the links on that page to examine a couple of other string-related functions PHP has.
- Check out the PHP manual page specifically for the `substr()` function. Read the other examples found on that page to get a better sense of how `substr()` can be used.
- Write the `thanks.php` script that goes along with Script 5.5. If you need help, revisit the `hello.php` script from Chapter 3 (Script 3.7).
- Rewrite the `print` statement in the final version of `handle_post.php` (Script 5.7), so that it uses single quotation marks and concatenation instead of double quotation marks.
- Create another HTML form for taking string values. Then create the PHP script that receives the form data, addresses any HTML or PHP code, manipulates the data in some way, and prints out the results.