IN THIS CHAPTER

» Finding errors in your JavaScript code

» Using JavaScript debuggers

» Working around errors

Chapter 5 Troubleshooting JavaScript Programs

A fact of life when working with any type of programming language is that there will always be errors as you develop your application code. Working with JavaScript is no different. There are plenty of opportunities for coding errors to cause all sorts of problems in your web applications. But don't worry — getting an error in your application isn't the end of the world. There are some simple tools at your disposal to help you find and fix those errors before your site visitors experience them. This chapter helps give you some ideas for what to do when errors occur as you develop your applications and offers tips for how to work your way through them.

Identifying Errors

Your web application may run into an error and it's fairly obvious that something went wrong — something that was supposed to happen didn't. Other times, however, program errors can be a little more subtle, such as altering the data in a way that's not obvious until you analyze the output. These types of errors are dangerous, because often you don't even know they're present until it's too late. It helps to be able to watch your JavaScript program and observe when the subtle coding errors occur.

The old-fashioned way of doing that was to insert `alert()` statements at strategic places in your code to watch variables as your code processes things. Just stick in the variable you want to monitor inside the `alert()` function to get a quick display of the value the variable contains at that point in the program:

```
alert(lastName);
```

That generates a lot of pop-up messages as you walk through your application, but it's a great strategy for helping watch what's going on "behind the scenes" in the code. This method is especially helpful with logic errors in the code — to detect when something isn't working quite the way you thought it would.

Yet another code troubleshooting method often used in the past was commenting out sections of code. JavaScript supports adding comment lines in the code to help with documenting what's going on. There are two types of comments that you can use in JavaScript. This is a one-line comment:

```
// Comments are fun!
```

This is a comment that spans more than one line in the code:

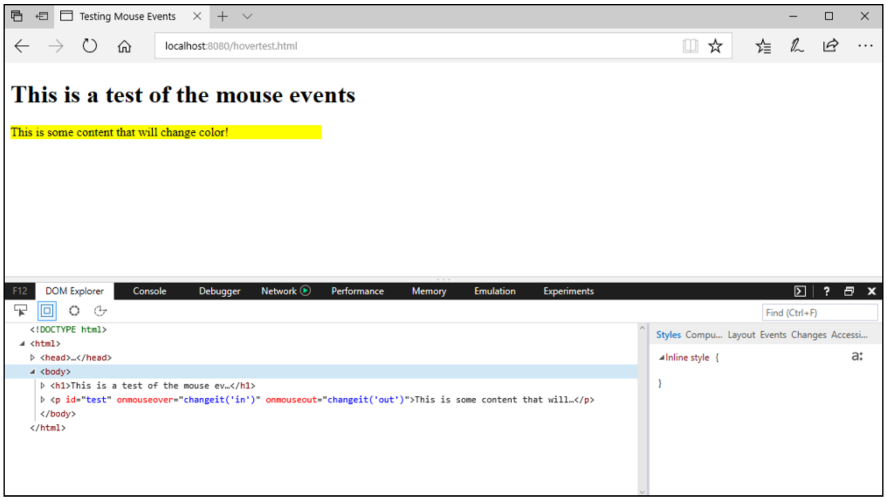
```
/* Comments allow you to document what is happening with your code. Comments are
   useful, but calling them fun is a bit of a stretch. */
```

When JavaScript sees the comment tags, it skips any text that's within the comment. While this is mainly intended to add commentary to your programs so you (or anyone else reading your code) know what code does what, it was also common to use this method to temporarily remove lines or entire blocks of code from the program without actually deleting them. Just place the JavaScript comment tags around the code you want to skip, and then run the program to watch how it works.

These are good troubleshooting methods, and they still come in handy at times, but today we have more sophisticated troubleshooting techniques at our fingertips.

Fortunately, all the popular web browsers today support JavaScript debuggers. A *debugger* is a program that points out program errors as they occur while you run the web application in the browser. Most debuggers also allow you to step through your JavaScript code one line at a time. This provides the opportunity to watch as each variable changes value, to help you track exactly where things are going awry.

All the main web browsers in use today either have a JavaScript debugger built in or easily added as a plug-in. It has become somewhat of a standard to launch the debugger tools by hitting the F12 key while viewing a web page. Figure 5-1 shows the IE Developer Tools section that appears.



This is a test of the mouse events

This is some content that will change color!

```
<!DOCTYPE html>
<html>
  <head>…</head>
  <body>
    <h1>This is a test of the mouse ev…</h1>
    <p id="test" onmouseover="changeit('in')" onmouseout="changeit('out')">This is some content that will…</p>
  </body>
</html>
```

```
Inline style {
}
```

FIGURE 5-1: The Microsoft Edge Developer Tools interface.

You can use the Developer Tools to help with your troubleshooting methods and quickly find (and fix) coding issues.

Working with Browser Developer Tools

The Developer tools interface used by all the main web browsers has many of the same features across all browsers. The interface contains seven different tabs:

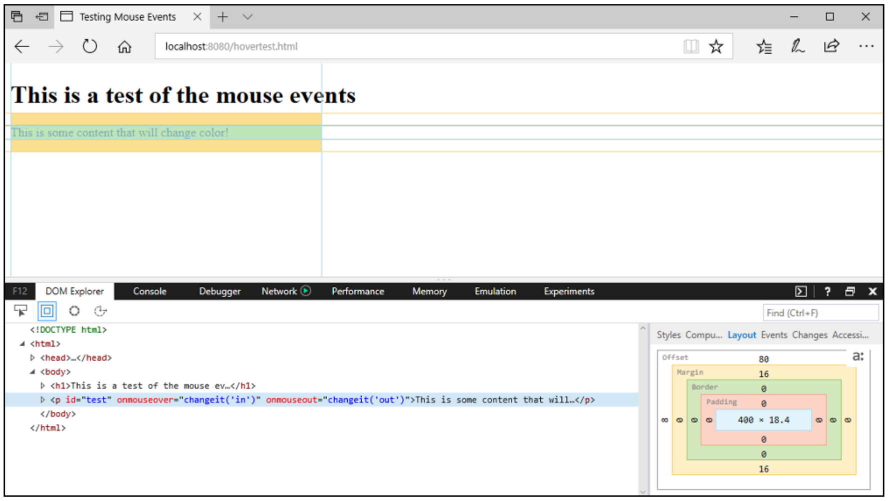
- **DOM Explorer:** Breaks down the web page elements into their Document Object Model (DOM) objects. This tool is great for exploring the DOM elements, especially if you need to isolate an element to reference in your JavaScript code.
- **Console:** Displays the JavaScript console, which logs error and warning messages caused by the JavaScript code in the web page, as well as any messages logged to the console directly from the JavaScript program.
- **Debugger:** A full-featured JavaScript debugger for troubleshooting JavaScript code line by line.
- **Network:** Displays network information about remote servers contacted to display the content on the web page.
- **Performance:** Profiles the central processing unit (CPU) utilization required while the JavaScript code in your web page runs.
- **Memory:** Profiles the memory utilization required while the JavaScript code in your web page runs.

» **Emulation:** In addition to the standard developer tools, the Internet Explorer and Edge browsers allow you to change the version of browser emulation used to display a web page. This allows you to view the web page as it would be seen in an older version of Internet Explorer, a great tool for developing web pages.

The following sections walk through the first three tools as they work in the Microsoft Internet Explorer and Edge browsers. Other browsers offer similar features but may require slightly different methods for using them. When you've learned how to use the tools in one browser, it's fairly easy to figure out how to use them in the others.

The DOM Explorer

The DOM Explorer disassembles the web page HTML5 code into the separate DOM elements that comprise the web page. It displays each element in a hierarchical tree structure, showing the general layout of the web page. Embedded elements are shown in the tree as child objects of the parent element, allowing you to collapse entire sections down to view a single level of the tree hierarchy at a time. Figure 5-2 demonstrates how this looks.



This is a test of the mouse events

This is some content that will change color!

```
<!DOCTYPE html>
◢ <html>
  ▷ <head>…</head>
  ◢ <body>
    ▷ <h1>This is a test of the mouse ev…</h1>
    ▷ <p id="test" onmouseover="changeit('in')" onmouseout="changeit('out')">This is some content that will…</p>
    </body>
</html>
```

FIGURE 5-2: Using the DOM Explorer to examine the HTML in a web page.

In some browsers, when you hover the mouse pointer over a DOM element, the DOM Explorer highlights the area of the web page the DOM element generates. This helps you identify which area on the web page is created by which HTML5 code. Unfortunately, Internet Explorer doesn't support this feature, but Edge does.

Instead of highlighting the elements in the web page, Internet Explorer displays a layout diagram for the element to the right side of the DOM Explorer. The highlighted areas use separate colors to show the element text area, the padding area around the text, the border area around the element, and the margin area defined for the element.

Inside each area is a number showing how the area is sized in the HTML5 code. What's even cooler is that you can click the number of an area to edit it directly in the DOM tree and then see how the change affects the layout of the elements on the web page. This is a great way to help visualize and experiment with your web page layout.

TIP

For position values that are calculated by the browser (such as percentages and em units), the DOM Explorer displays both the configured value, as well as the calculated value in pixels. This is yet another great tool for experimenting with layout structures.

The DOM Explorer also allows you to make edits directly to the HTML5 code for an element and then view how the changes affect the web page in real time. There are three ways to do that:

- Double-click directly on an element attribute in the DOM Explorer to change its value.
- Right-click an element and select Add Attribute to add a new attribute.
- Right-click an element and select Edit as HTML to edit the element manually.

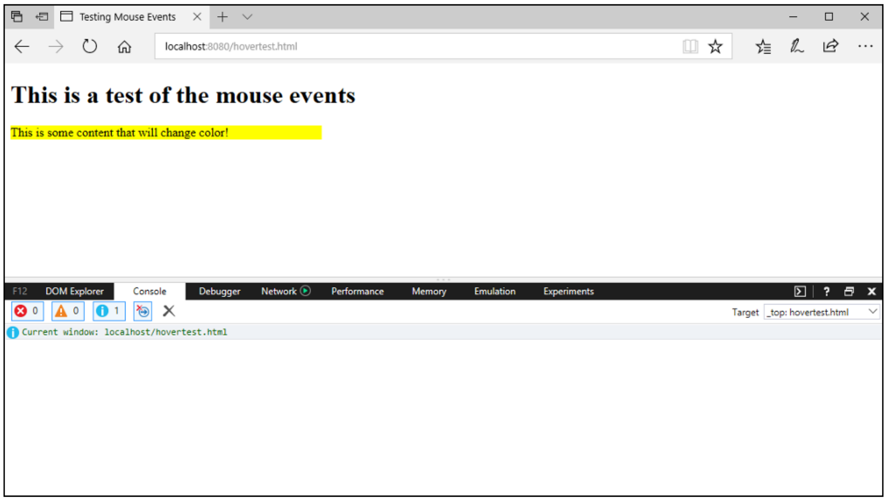
The DOM Explorer also tracks event handlers that your JavaScript code attaches to HTML5 events, allowing you to detect when an event handler is misapplied or didn't get applied at all.

The Console

The Console displays messages received by the browser from the loaded web page. There are three categories of messages that display in the Console:

- **Errors:** Issues that cause the web page to not load or perform correctly, such as invalid JavaScript code
- **Warnings:** Issues that allow the web page to load, but that may cause unexpected behavior
- **Information:** Any noncritical information provided by the web page

When you click the Console tab, you see the interface shown in Figure 5-3.



This is a test of the mouse events

This is some content that will change color!

```
Current window: localhost/hovertest.html
```

FIGURE 5-3: The Developer Tools Console window in Microsoft Edge.

The first three icons at the top allow you to filter the messages to hide or display the error (the red X), warning (the yellow triangle), or information (the blue circle) messages. They also show the count of each type of message generated since the last clear of the Console. You can clear out the messages by clicking the black X icon.

To watch the Console in action, let's work on an example with some bad JavaScript code and see what happens. Follow these steps:

1. **Open the** `hover.html` **file created in Book 3, Chapter 4.**
2. **In the** `changeit()` **function code, change the** `getElementById()` **functions to the incorrect** `getElementByid()` **name.**

   Note the lower-case *i* in the function name, a mistake that I make all too often on my own!
3. **Open the XAMPP Console and start the Apache web server.**
4. **Open Internet Explorer or Edge, and enter the following URL:**

   ```
   http://localhost:8080/hover.html
   ```
5. **Press F12.**

   The Developer Tools window appears.
6. **Click the Console tab.**
7. **Hover your mouse pointer over the p element content in the main web page, and watch the messages that appear in the Console area.**

While on the web page, all you see is that nothing happens, not much to help with why that was. However, the Console shows the error messages that identify exactly what went wrong. The misnamed `getElementByid()` functions generate an error in the Console each time the mouse events trigger. The error messages point you to the misnamed function and the line numbers they appear on in the code. This is a huge help in figuring out just what went wrong when a dynamic action doesn't work correctly in your web pages.

Besides watching the error and warning messages that the web page generates on its own, you can generate your own messages in the Console from your code. The `console.log()` JavaScript function allows you to send messages directly to the Console for viewing. Just add the line anywhere in your JavaScript code to display useful information to the Console.

For example, one method I often use when working with events is to add a `console.log()` function to identify each time an event is triggered in the JavaScript code:

```
function changeit(state) {
    if (state == "in") {
        console.log("mouseover triggered");
        document.getElementById("test").style.backgroundColor"red";
    } else if (state == "out") {
        console.log("mouseout triggered");
        document.getElementById("test").style.backgroundColor="yellow";
      }
   }
```

As the HTML5 code triggers each mouse event and passes control over to the JavaScript `changeit()` function, the `console.log()` functions run based on just which event triggered. Then you can just watch the Console area to tell just what's going on "behind the scenes" in your application!

WARNING

Adding `console.log()` functions to the code is a great troubleshooting technique, but be sure to remove them before taking your application live for site visitors! You don't want to needlessly clutter up their Console logs with troubleshooting data.

Below the Console window is a command line interface (CLI) that allows you to enter JavaScript code to run inside the web page. Just type the JavaScript code you want at the CLI prompt and press Enter or Return. You can use this to quickly test variable values or override variable values to see how they affect the program operation.

If you need to enter a long JavaScript statement (such as defining a function), click the double up-arrow icon at the far-right side of the CLI. This expands the CLI pane to display more lines of code. When you're ready to submit the code, click the green arrow to run it.

TIP

The Console CLI also allows you to copy and paste code into it. You can use the CLI to insert new functions, or test out additional code as the program is running. As you enter new code, the browser interprets it on the fly, at the current point in the application. If the application is paused by the Debugger tool, the code is executed at that point in the program.

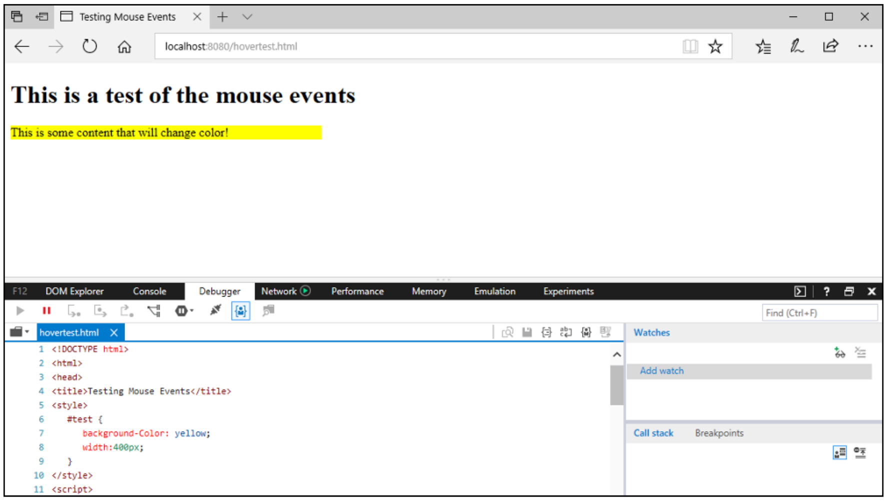
The Debugger

The Debugger allows you to watch your JavaScript code in action. This tool is a powerful way to step through the JavaScript code in the application one statement at a time and observe exactly what's going on. The Debugger allows you to pause the JavaScript code at any point in the program and view the following:

- The path that caused the program to get where it is
- The values of any variables that have been set by the code
- How variables change at each statement after that point

To cause the JavaScript program to pause in the Debugger you need to set one or more breakpoints in the code. The breakpoint signals to the browser to stop processing code and enter the Debugger window.

When you open the Debugger tool window, you'll see different sections appear in the interface, as shown in Figure 5-4.



This is a test of the mouse events

This is some content that will change color!

```
<!DOCTYPE html>
<html>
<head>
<title>Testing Mouse Events</title>
<style>
   #test {
      background-Color: yellow;
      width:400px;
   }
</style>
<script>
```

FIGURE 5-4: The Debugger interface in the Microsoft Edge Developer Tools.

The Debugger interface has three main sections:

- **Script pane:** The script pane (on the left side) shows the web page HTML5 and JavaScript code. It indicates whether there are any breakpoints and, if the program is paused, where in the code it's paused.
- **Watch pane:** The watch pane (on the right side at the top) shows a list of variables that you're watching and their current values.
- **Call Stack and Breakpoints pane:** The Call Stack and Breakpoints pane (on the right side at the bottom) displays the chain of function calls that led to the current location in the code (the call stack), as well as the list of breakpoints set in the program code.

Each pane provides information about the running JavaScript program and what's going on each time the Debugger pauses the program to examine the code in a breakpoint.

There are three ways to set a breakpoint in your JavaScript program:

- Click next to the line number in the script pane of the statement where you want the program to pause.
- Use the icons in the Breakpoints pane to add an XML or event breakpoint. Event breakpoints pause the program when a specified event is triggered (such as when you click the mouse button).
- Add the `debugger` statement in your JavaScript code. Although this method is handy, it can also be very dangerous. If you use this method, don't forget to remove the debugger statements from your code before going live with site visitors.

TIP

Setting breakpoints inside the Debugger interface is the best method. Those breakpoints are only temporary — they go away when you close out your browser window.

When the Debugger pauses the program code at a breakpoint, you have a set of icons available above the script pane that control how the browser executes the code in debugger mode. Table 5-1 lists the icons that you can use.

TABLE 5-1 Debugger Control Icons

Icon	Description
Continue	Removes the code pause and continues with the next statement.
Break	Exits from the Debugger mode after the next statement.
Step Into	Proceeds to the next line of code. If the next line is a function, the Debugger follows into the function code.
Step Over	Proceeds to the next line of code. If the next line is a function, the Debugger runs the function code, but not in debug mode.
Step Out	Exits from the called function back to the main program.
Break on New Worker	Exits the Debugger when a new web page is created.
Exception Control	Sets how to handle exceptions as they're thrown in the code.
Show Next Statement	Lets you skip lines of code to execute in the program.
Run to Cursor	Resumes execution of the code until the line in the code where the cursor is located.
Set Next Statement	Lets you skip statements in a function without running them.

To go line-by-line through the JavaScript code, use the Step Into control icon. If you come to a JavaScript function in the code (such as the `getElementById()` function), clicking the Step Into control icon will follow the code into the JavaScript library that implements that function. This can get somewhat tedious at times, because some JavaScript functions require hundreds or even thousands of lines of code to implement, before you get back to your own code! To avoid that, use the Step Over control icon. The Step Over control feature runs through the JavaScript library code that implements the function, but then pauses again when control gets back to your code.

When you're done debugging the code, click the Continue control icon to return back to the normal operation of the program.

To watch the Debugger tool in action, follow these steps:

1. **Ensure that the Apache web server is running; if it isn't, start it from the XAMPP Console.**
2. **Open your Internet Explorer or Edge browser and enter the following URL:**

```
http://localhost:8080/hovertest.html
```

3. **Press F12.**

   The Developer Tools window appears.

4. **Click the Debugger tab.**

5. **In the script pane, click the line number for the first statement in the** `changeit()` **function.**

   This should be the following line:

   ```
   if (status == "in") {
   ```

6. **Observe what happens in the Breakpoint pane.**

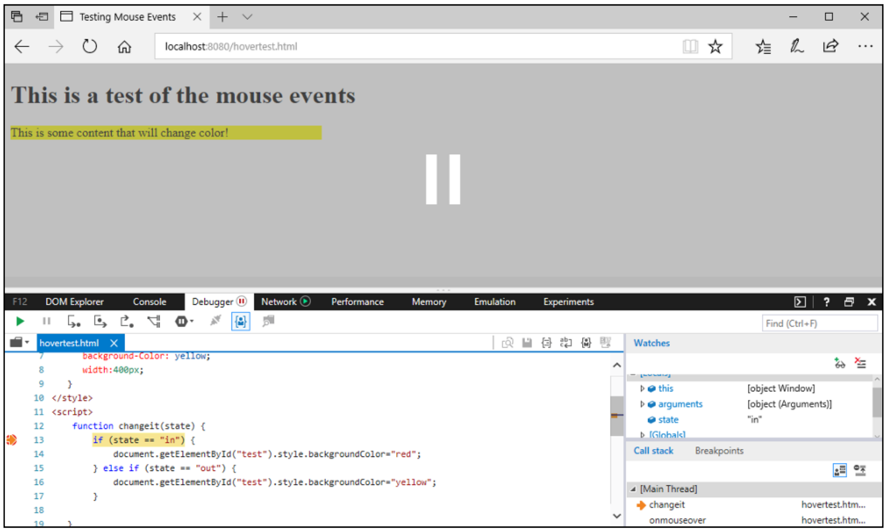
   A new breakpoint should be set, indicated by a red dot next to the line.

7. **Click Add Watch in the Watches pane.**

8. **In the text box that appears, type** state**, to watch the** `state` **variable that's used in the** `changeit()` **function.**

9. **Reload the** `hovertest.html` **page in the browser window and then hover the mouse pointer over the p element section on the web page.**

   When you hover the mouse pointer over the p element, that triggers the `onmouseover` event, which calls the `changeit()` JavaScript function. The Debugger detects the breakpoint that you set and pauses the program execution, as shown in Figure 5-5.



```
        background-Color: yellow;
        width:400px;
    }
</style>
<script>
    function changeit(state) {
        if (state == "in") {
            document.getElementById("test").style.backgroundColor="red";
        } else if (state == "out") {
            document.getElementById("test").style.backgroundColor="yellow";
        }

    }
```

this	[object Window]
arguments	[object (Arguments)]
state	"in"
[Globals]	

[Main Thread]

changeit	hovertest.htm...
onmouseover	hovertest.htm...

FIGURE 5-5: Pausing the code at a breakpoint in the Debugger.

Notice the information you now have available at your fingertips. The orange arrow in the Script pane indicates the statement at which the debugger is paused. In the Watches pane, you can now see the `state` variable's value as the program enters into the `changeit()` function. In the Call Stack and Breakpoints pane, you can see just how the program got to the `changeit()` function. It shows that the main program thread triggered an `onmouseout` event, and it's currently at the `changeit()` function. In the Script pane, you see a pointer that shows just where in the code things stopped.

Follow these steps to continue on with the debugging process:

1. **Click the Step Into icon to move on to the next line of code.**

   If you kept the `console.log()` statement in the code from the previous example, it'll take you into the JavaScript library to run that function. If you prefer to avoid doing that, click the Step Over icon.

2. **Continue clicking the Step Into icon to walk your way through the `changeit()` function code in the web page.**

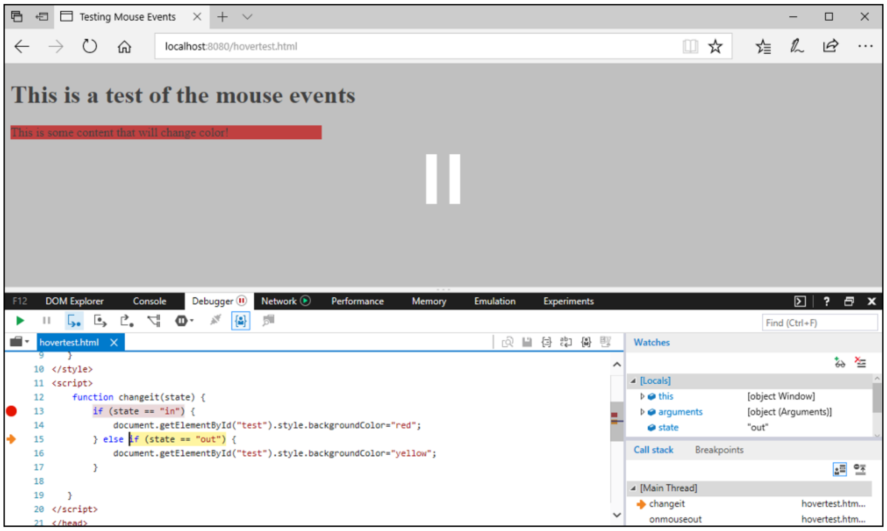
   Eventually the pointer will return to the p element defined in the `hovertest.html` file.

3. **Click the Continue icon in the controls.**

   The Debugger will again stop at the `changeit()` function. This is because it detected that the mouse pointer is no longer in the p element section, so the `onmouseout` event triggered.

4. **Note the value of the `state` variable.**

   It should now be set to `out`, as shown in Figure 5-6.



```
    }
</style>
<script>
   function changeit(state) {
      if (state == "in") {
         document.getElementById("test").style.backgroundColor="red";
      } else if (state == "out") {
         document.getElementById("test").style.backgroundColor="yellow";
      }

   }
</script>
</head>
```

[Locals]	
this	[object Window]
arguments	[object (Arguments)]
state	"out"

[Main Thread]	
changeit	hovertest.htm...
onmouseout	hovertest.htm...

FIGURE 5-6: Stopping the Debugger later on in the code.

5. **Click the Step Into icon, and watch how the `if...else` statement in the code evaluates the state variable and jumps to the `else` section of the code.**
6. **Click the Continue icon to return the program back to running normally.**
7. **Disable the breakpoint that you previously set by clicking the check box for the breakpoint in the Breakpoints pane.**
8. **Run the program again and watch what happens.**

   Now the Debugger won't stop at the breakpoint.
9. **Close the browser window to end the test.**

With just a few simple commands, you have a full-fledged method of debugging your dynamic web applications. That makes developing your web applications a much easier task.

Working Around Errors

There may be times in your application where you don't want things to come to a grinding halt just because of some type of error in the program code. Often JavaScript programs rely on data supplied by the site visitor, and you wouldn't want an invalid data entry to cause your program to crash.

One method to prevent that is to intercept errors before they make it to the browser and cause problems. This process is called "catching the errors." With catching the errors, the program detects when something is amiss and provides some alternate code for the browser to run, bypassing the normal code that would have produced the fatal error.

You do this in JavaScript with the `try...catch` statement. The `try...catch` statement consists of two code blocks — the `try` code block to run and monitor for errors, and the `catch` code block to run in case any errors are detected in the `try` code block. Here's the format for the `try...catch` statement:

```
try {
    code to test
} catch (err) {
    code to run if test fails
}
```

The `catch()` function takes one parameter — a variable to place an `Error` object that JavaScript generates to describe the error that occurred. The `Error` object has two properties:

- `name`: Returns the name of the error type
- `message`: A string message describing the error in more detail

The error name identifies the type of error that occurred in the `try` code block. There are six different error types supported in JavaScript, shown in Table 5-2.

TABLE 5-2 **JavaScript Error Types**

Error	Description
`EvalError`	An `eval()` function has produced an error.
`RangeError`	A value out of range has occurred.
`ReferenceError`	An invalid location was referenced in the code.
`SyntaxError`	Invalid JavaScript code was detected.
`TypeError`	An invalid data type was used.
`URIError`	An error in the encode`URI()` function was detected.

Besides automatically detecting errors, you can create your own custom error checks and messages by using the `throw` statement inside the `try` code block:

```
try {
    if (value < 1000) throw "The value is too small";
    if (value > 10000) throw "The value is too large";
} catch (err) {
    alert(err);
}
```

The string assigned to the `throw` statement is displayed as part of the `Error` object. To demonstrate using the `try...catch` method to your JavaScript code, let's work out a simple exercise. Follow these steps to create the demo:

1. **Open your favorite text editor, program editor, or IDE package.**
2. **Type the following code:**

```
<!DOCTYPE html>
<html>
<head>
```

```
<title>Catching Errors Test</title>
<style>
    fieldset {
        width: 450px;
    }

    label, input {
        margin: 10px;
    }
</style>
<script>
function calculate() {
    var games, scores, array, total, average, output;
    games = document.getElementById("games").value;
    scores = document.getElementById("scores").value;
    array = scores.split(',');
    total = 0;
    for(i = 0; i < array.length; i++)  {
        total = total + parseInt(array[i]);
    }
    try {
      if (games == 0) {
        throw "Please enter a valid number of games";
      } else if (games == "") {
        throw "Please enter a valid number of games";
      } else if (isNaN(games)) {
        throw "Please enter a valid number of games";
      }
      average = total / games;
      output = "The average is " + average;
      document.getElementById("result").innerHTML = output;
    } catch (err) {
      document.getElementById("result").innerHTML = err;
    }
}
</script>
</head>
<body>
<fieldset>
<legend>Bowling Calculator</legend>
<label>Enter number of games bowled</label>
<input type="text" id="games" size="3"><br>
<label>Enter scores, separated by commas</label>
<input type="text" id="scores" size="20"><br>
<button onclick="calculate()">
```

```
Calculate average
</button>
<p id="result"></p>
</fieldset>
</body>
</html>
```

3. **Save the file as `catchtest.html` in the `DocumentRoot` folder of the Apache web server.**

   That's `c:\xampp\htdocs` for XAMPP on Windows, or `/Applications/XAMPP/htdocs` for XAMPP on macOS.

4. **Open the XAMPP Control Panel and start the Apache web server.**
5. **Open your browser, and enter the following URL:**

   ```
   http://localhost:8080/catchtest.html
   ```

   You may need to use a different port in the URL for your web server.

6. **Enter** 3 **for the number of games and** 100,105,100 **for the scores.**
7. **Click the Calculate Average button to view the results.**
8. **Change the number of games to an invalid value — enter** 0 **or some text, or just leave the field empty.**
9. **Click the Calculate Average button and see what happens.**

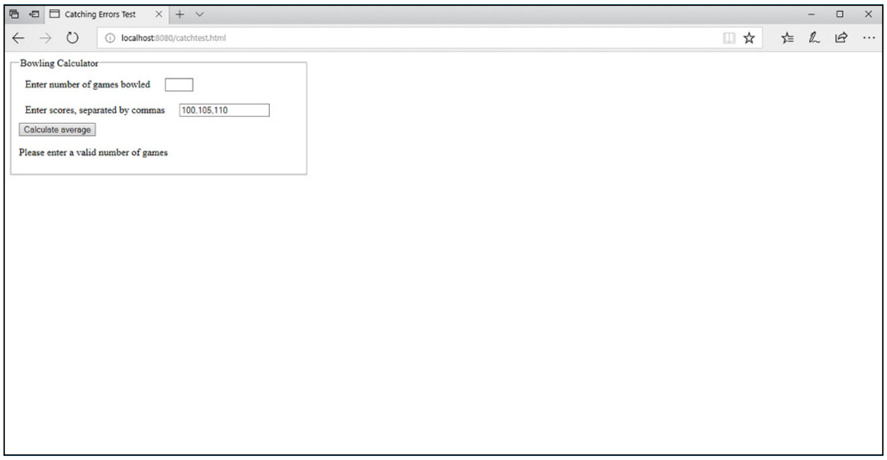
   You should see the text from the appropriate `throw` statement that caught the error appear.

10. **Close the browser, and shut down the Apache web server.**

When you enter an invalid value for the number of games, the `if...then` condition checks will detect it, and use the `throw` statement to intercept the error and trigger the `catch` code block, displaying the message defined in the `throw` statement (see Figure 5-7).

There's one more piece to the `try...catch` statement that you can use. The `finally` statement allows you to enter a block of code that gets executed no matter what happens in the try code block:

```
try {
    code to test
} catch (error) {
     code to run if errors
} finally {
    final code always runs
}
```



Bowling Calculator

Enter number of games bowled

Enter scores, separated by commas 100,105,110

Calculate average

Please enter a valid number of games

FIGURE 5-7: Catching an invalid data entry using the `try...catch` statement.

Any code you place in the `finally` code block runs at all times. If the code in the `try` code block is successful, the JavaScript interpreter jumps to the `finally` code block and runs that code. If the code in the `try` code block fails, the JavaScript interpreter runs the code in the `catch` code block, and then runs the code in the `finally` code block. This is a good way to have "cleanup" code for the function.

4 PHP

Contents at a Glance

IN THIS CHAPTER

» Understanding PHP

» Using PHP in HTML5 programs

» Storing data in PHP programs

» Including PHP code in multiple programs

Chapter 1 Understanding PHP Basics

Welcome to the PHP minibook! If you've been following along through the previous minibooks, you've seen how to create web page content using HTML5, how to style and position it using CSS3, and how to add some dynamic features to your web pages using JavaScript. This minibook examines the next piece to dynamic web applications — using a server-side programming language to make your web applications even more dynamic. As the title of the book suggests, the server-side programming language that I discuss is PHP, one of the most popular server-side programming languages in use on the Internet today!

Seeing the Benefits of PHP

So far, you've already seen that JavaScript is a popular client-side programming language and that it has the ability to change the content and style of a web page dynamically. One question you may be asking is, "Why do I need a server-side programming language, too?" This section examines what your web applications will gain by adding PHP to the mix and what you can do when you incorporate PHP code in your applications.

A centralized programming language

One of the downsides to using a client-side programming language is that your code is dependent on how each individual browser runs it. Great strides have been made in the standardization of JavaScript, but each browser still has its own set of quirks when running JavaScript code, as well as its own set of libraries that offers different features, making it impossible to know just how your JavaScript code will run in all situations.

Unlike that environment, server-side PHP programs run on the same server that hosts your web pages, so every site visitor who accesses your web pages runs the PHP code on the same server, using the same set of library features. You know exactly how your application code will run and exactly what it will produce for all your website visitors.

Another added benefit of using PHP code in your web pages is the ability to control the actual PHP server itself. Because all the PHP code in your web pages runs from the same location, you can customize the feature settings in the PHP server to your specific environment. This allows you to utilize just the libraries you need or set memory usage just how you want, giving you some control over the performance of your web applications.

TIP

Book 1, Chapter 2, shows some of the configuration settings available in the PHP server and how you can change them to customize your PHP environment.

Centralized data management

These days, data rules the world. Just about every web application requires some type of data to run. Whether it's displaying news stories, posting blog entries, or just tracking your bowling team scores, you need some type of data to use in your dynamic web application.

When you use data, you need some method for storing it. A *content management system* (CMS) provides an interface to track data in a single repository, allowing you to create, read, update, and delete data records freely. The CMS package is often installed as part of the web server environment and often utilizes a database server that specializes in quickly storing and retrieving data records.

By using PHP, you can access the data in your CMS package directly from the server. That usually means faster response times, as opposed to your individual site visitors accessing the CMS server from their locations. It also means more control over how your application accesses and displays the data. The only data your site visitors can see is what your application presents to them. All your CMS access information stays hidden on the server — none of the code to access the data is downloaded to the client browsers. This is a also huge benefit for security reasons.

Understanding How to Use PHP

After you decide to incorporate PHP into your web applications, you need to know just how to do that. This section walks through the basics of adding PHP code to your web pages and how to get output from your PHP programs to appear in your web pages as they display in your site visitors' browsers.

Embedding PHP code

Just as with JavaScript, you embed PHP code directly into the HTML5 code that creates the web page. As you can probably guess, you need a way to identify the embedded PHP code, and that method is to use tags.

There are actually four different ways to tag PHP code in the HTML5 document. The most common method is to use the special `<?php` and `?>` tag combination. Just place the PHP code you need to embed between the opening `<?php` tag and the closing `?>` tag, like this:

```
<!DOCTYPE html>
<html>
<body>
<?php
     php code
?>
</body>
</html>
```

You can place the PHP tags anywhere in the HTML5 code — they don't need to be in the body element. You can have as many HTML5 elements that you need outside the PHP code area to provide supporting content on the web page, but you can't place HTML5 elements inside the PHP code area. Only PHP code can reside inside the PHP code area.

The `<?php` tag is the most common way to identify PHP code, but it's not the only way. Another method is to use the `<script>` HTML5 tag:

```
<!DOCTYPE html>
<html>
<body>
<script language="php">
    php code
</script>
</body>
</html>
```

This looks very similar to what you use to embed JavaScript code into HTML5 code, which could be good or bad. Just remember to include the `language` attribute in the tag and identify the code as PHP code. Using the same `<script>` tags to embed both JavaScript and PHP code can be a bit confusing, which is why the `<?php` tag has become so popular.

The third type of PHP tag is called the *short open tag.* It uses `<?` as the opening tag, instead of the full `<?php` tag. For this tag method to work, though, the PHP server must have the `short_open_tag` setting enabled in its configuration file. The short open tag saves some typing, but it can get confusing as you look through the program code.

Finally, the fourth type of PHP tag is the `<%` opening tag. This is called the *ASP style tag* because this is the same tag used when programming with the Microsoft ASP.NET family of server-side programming languages. If you're already comfortable with using ASP.NET programming, you can use this style of tag for PHP coding as well. Similar to the short open tag, you must enable the `asp_tags` setting in the PHP server configuration file to use this method.

Identifying PHP pages

Because PHP is a server-side programming language, the PHP processor that runs the PHP code is located on the server — usually the same physical server as the web server. To process the embedded PHP code, your web page must pass your HTML5 document to the PHP server on its way to the site visitor who requested it.

The web server must be able to detect when a web page contains embedded PHP code and when it doesn't. If the web page contains PHP code, the web server must pass the entire HTML5 document to the PHP server for processing. We don't want the web server to pass all HTML5 documents to the PHP server, because that would slow down processing web pages that don't contain embedded PHP code. The web server must know when it has to send an HTML5 file to the PHP server for processing. You control that by using file extensions.

When the Apache web server has the PHP module installed, there's a directive in the main `httpd.conf` configuration file identifying PHP programs that need to be sent to the PHP server for processing. That directive looks like this:

```
AddHandler application/x-httpd-php .php
```

This tells the Apache web server to send any files that site visitors request that end with the `.php` file extension to the PHP server. This way, you can identify any web pages that contain embedded PHP code by using the `.php` file extension instead of the standard `.html` file extension. Figure 1-1 shows this process.

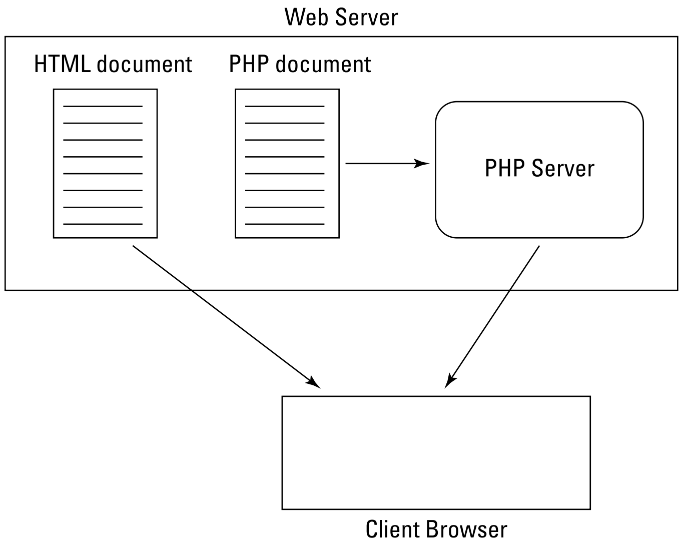


FIGURE 1-1: Processing PHP code in a web page.

Using the correct file extension for PHP files is crucial, because if you embed any PHP code into a file with an `.html` file extension, the PHP code won't get processed; instead, it will appear on the web page as text.

WARNING

When working with PHP code, you must run the web page through the web server so that it gets processed by the PHP server. You can't just double-click a `.php` file to open it in your browser — you must open your browser and use the `http://` URL to access the file via the web server.

Displaying output

As the PHP server reads the code in the file that the web server sends it, it passes any HTML5 code directly on to the client browser that requested the file and processes any PHP code embedded in the document. As it processes the PHP code, you'll want to be able to dynamically add content to the web page (after all, that's what you're here for). You do that using the `echo` statement.

The `echo` statement injects text into the HTML5 data stream that's sent to the client browser. The data appears to the client browser just as if it came from the HTML5 document — it has no idea that the PHP server dynamically generated the content.

To use the `echo` statement, you just specify the string value that you want to insert into the HTML5 output:

```
echo "this is my output";
```

In PHP, function names are not case sensitive, but it's fairly standard convention to use lowercase for function names. Also in PHP, all statements must end with a semicolon. If you forget the semicolon, you'll generate a parse error from the PHP processor.

Follow these steps to test out embedding PHP in an HTML5 document:

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
2. **Type the following code:**

```
<!DOCTYPE html>
<html>
<body>
<h1>This is a test of PHP code</h1>
<?php
    echo "<p>This text was dynamically generated!</p>";
?>
<h1>This is the end of the test</h1>
</body>
</html>
```

3. **Save the file as `phptest.php` in the `DocumentRoot` folder of your web server.**

   For XAMPP on Windows, use the `c:\xampp\htdocs` folder; for XAMPP on macOS, use `/Applications/XAMPP/htdocs`.

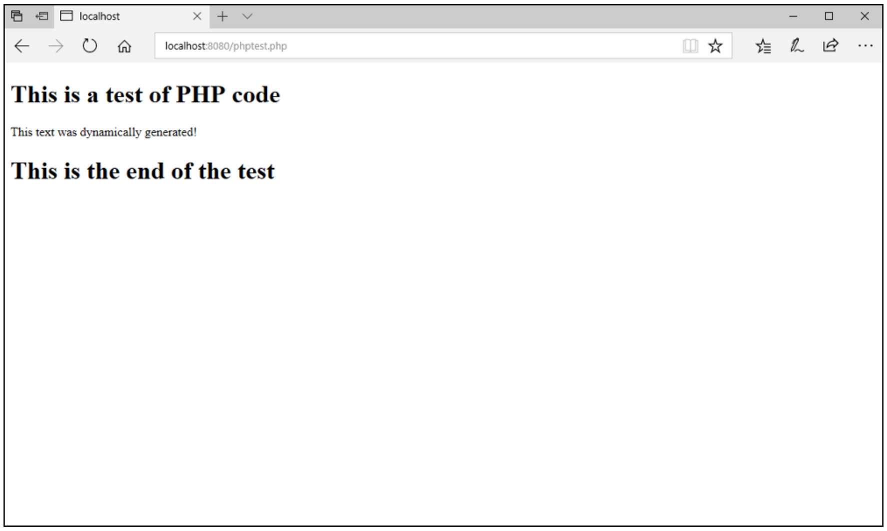
4. **Open the XAMPP Control Panel and start the Apache web server.**
5. **Open your browser and enter the following URL:**

```
http://localhost:8080/phptest.php
```

   You may need to use a different TCP port based on your Apache web server setting.

6. **Close your browser when you're done.**

When you run the `phptest.php` file in your browser, the web page should appear as shown in Figure 1-2.



This is a test of PHP code

This text was dynamically generated!

This is the end of the test

FIGURE 1-2: Output generated by the phptest.php program.

The p element section appears just as if you had typed it directly in the HTML5 code. The PHP server injected it into the HTML5 code, and the browser added it to the Document Object Model (DOM) tree just as normal. It's also important to note that, in this demonstration, I embedded standard HTML5 tags into the output from the echo statement. Everything that's inside the string value is sent to the client browser, including any HTML5 elements that you specify.

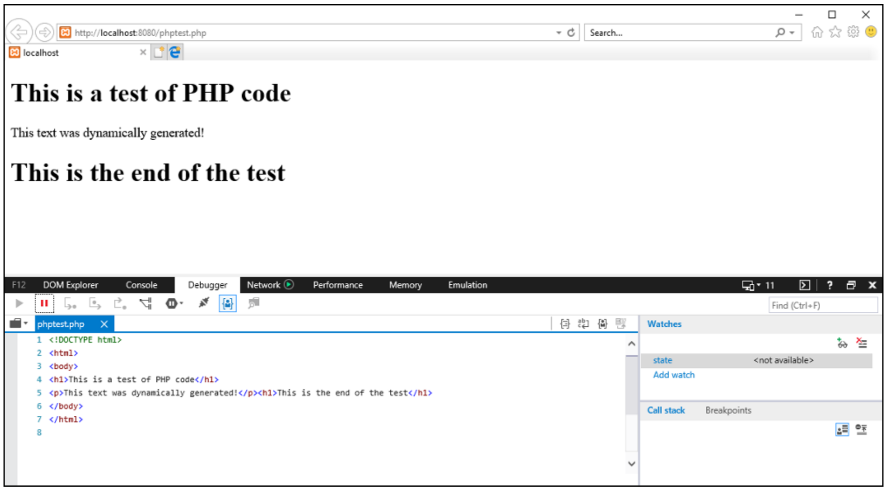
TIP

If you see the PHP code appear in your web page, that means the PHP server didn't process the PHP code. Check to make sure you don't have a typo in the opening <?php tag (note that there are no spaces in the tag) and that the file uses the .php file extension.

Handling new-line characters

There is one oddity that you may have noticed when running the phptest.php demo program. If you use the Developer Tools for your browser (see Book 3, Chapter 5) and look at the HTML5 code generated, it may look a little odd, as shown in Figure 1-3.

Instead of the p element being on a separate line in the code, it got pushed onto the same line as the second h1 element.



This is a test of PHP code

This text was dynamically generated!

This is the end of the test

```
<!DOCTYPE html>
<html>
<body>
<h1>This is a test of PHP code</h1>
<p>This text was dynamically generated!</p><h1>This is the end of the test</h1>
</body>
</html>
```

FIGURE 1-3: Viewing the HTML5 code generated by the `phptest.php` program.

The `echo` statement in PHP doesn't add a new-line character at the end of the output. Because there aren't any new-line characters, any content that you display using the `echo` statement in PHP appears on the same line in the HTML5 code.

REMEMBER

The HTML5 standard ignores any white space between elements in the document, so the fact that the p element is on the same line as the h1 element doesn't effect the output that appears on the web page at all. However, having two elements on the same line can make troubleshooting HTML5 code generated by PHP somewhat complicated. That's especially true as you use PHP to create entire web pages!

TIP

To solve this problem, many PHP developers like to add their own new-line characters to the ends of all `echo` statements in the code, like this:

```
echo "<p>This text was dynamically generated!</p>\n";
```

The `\n` new-line character doesn't change the appearance of anything on the web page as it appears in the browser, but it does separate the p element from the following h1 element when you look at the HTML5 code using the browser Developer Tools features. It adds some extra typing to your development work, but it can save you lots of time trying to troubleshoot HTML5 code issues in your applications!

Working with PHP Variables

The key to dynamic web applications is working with data. Just like any other programming language, PHP allows you to use variables to store data in your programs. *Variables* are placeholders that you assign values to throughout the duration of the program. When the program references the variable, it represents the actual value that the program last assigned to it.

This section walks through what you'll need to know to use variables in PHP.

Declaring variables

In PHP, you identify variables with a leading dollar sign ($) in front of the variable name. You must start a variable name with either a letter or an underscore character (_), and it can contain only letters, numbers, and underscores (the variable name can't contain any spaces or other special characters). Here are some examples of valid PHP variable names:

```
$test
$Test1
$_another_test
```

Just as in JavaScript, PHP variable names are case-sensitive, so be careful when you reference variables in your code. The variable name `$Test` is different from `$test`. Case-sensitivity causes all sorts of headaches when trying to troubleshoot PHP code.

Unlike with JavaScript, with PHP, you don't declare variables with a `var` statement — you just use them. However, the first time you use a variable must be within an `assign` statement, assigning a value to the variable:

```
$test = "This is a test string";
```

The assignment statement assigns the value specified on the right side of the equal sign to the variable specified on the left side. As with all other PHP statements, don't forget the semicolon at the end of assignment statements!

After you assign a value to a variable, you can use it in your application:

```
$value1 = 10;
$value2 = 20;
$result = $value1 + $value2;
```

If you try using the third statement before assigning values to the `$value1` or `$value2` variables, you'll get a warning message from PHP about using a value with no assigned value. However, by default, PHP will assume the unassigned variables contain a value of `0`.

As you can tell from these examples, PHP allows you to store different data types in variables. The next section takes a closer look at that.

Seeing which data types PHP supports

Just as with JavaScript, PHP supports the following data types:

- **Integer:** Stores whole-number values
- **Float:** Also called floating-point or double; stores real numbers
- **Boolean:** Stores a `true` or `false` value
- **String:** Stores a *series* (string) of characters
- **Array:** Stores multiple values referenced by the same variable name
- **Object:** Stores instances of classes
- **Reference:** Stores a pointer to a complex data type

WARNING

In PHP, just as in JavaScript, a single variable can hold any type of data at any time (PHP doesn't enforce strict data typing). Changing the data type stored in a variable can get confusing, and I strongly recommend sticking with one data type per variable name in your programs. Trust me, it'll make your life a lot easier!

Follow these steps to test out using different data types in PHP code:

1. **Open your editor and type the following code:**

```
<!DOCTYPE>
<html>
<head>
<title>Testing PHP Data Types</title>
</head>
<body>
<h1>PHP Data Type Test</h1>
<?php
$name = "Rich";
$age = 100;
$salary = 575.25;
echo "<h2>Information for $name</h2>\n";
echo "Age: $age<br>\n";
echo "Salary: $$salary\n";
?>
```

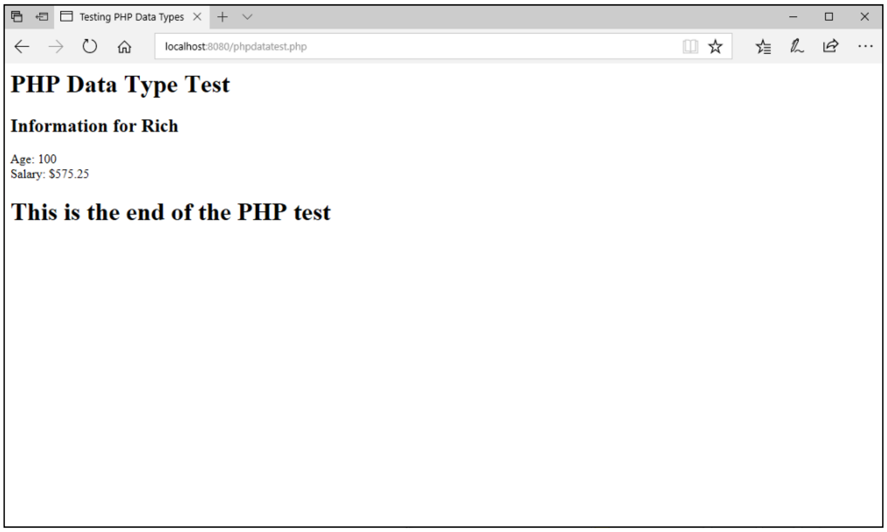
```
<h1>This is the end of the PHP test</h1>
</body>
</html>
```

2. **Save the file as** `phpdatatest.php` **in the** `DocumentRoot` **folder for your web server.**
3. **Ensure that the web server is running, open your browser, and enter the following URL:**

```
http://localhost:8080/phpdatatest.php
```

4. **Close the browser window when you're done.**

When you run the `phpdatatest.php` program, you should see the output as shown in Figure 1-4.



PHP Data Type Test

Information for Rich

Age: 100
Salary: $575.25

This is the end of the PHP test

FIGURE 1-4: Output from the `phpdatatest.php` program.

Let's look at exactly what's going on in this PHP program. First, the code assigns values for three variables:

```
$name = "Rich";
$age = 100;
$salary = 575.25;
```

PHP AND QUOTES

You can use either single or double quotes to define a string value in PHP. They're interchangeable, but there are times when you'll want to use one over the other. Things can get somewhat confusing when you have to use quotes inside the string value itself. When you know you have to use one type of quote in the data, just use the other type to define the string value:

```
$test1 = "This'll work just fine in PHP";
$test2 = 'Rich says "this works, too" in PHP';
```

Where things get tricky is when you need to use both types of quotes inside the data value. To do that, you must escape the quote type that you use to define the string value. Use the backslash to identify the quotes in the data:

```
$test3 = "Rich says \"This'll work, too\" in PHP";
```

Be careful when working with quotes in data — it's easy to miss them and cause errors in your PHP code.

The first statement assigns a string value to the `$name` variable. To assign a string value, you must enclose the data in either single or double quotes. These mark the beginning and end of the string value.

After assigning the three variable values, the code then uses three `echo` statements to display the variable values:

```
echo "<h2>Information for $name</h2>\n";
echo "Age: $age<br>\n";
echo "Salary: $$salary\n";
```

Unlike many other programming languages, PHP allows you to just use a variable directly within a string value in the `echo` statement. However, how the `echo` statement handles the variable depends on the type of quotes you use to define the string (again with the quotes).

If you use double quotes to define the output string, PHP will display the variable value in the output. If you use single quotes to define the output string, PHP will display the variable name in the output:

```
echo "The variable value is $age";
echo 'The variable name is $age';
```

That's extremely versatile, but it can be somewhat confusing, and it takes some time getting used to as you code your PHP programs.

The last `echo` statement in the example code also does something rather odd: It uses two dollar signs in front of the `$salary` variable. That doesn't change anything for the variable — it just displays a dollar sign in front of the value contained in the `$salary` variable. This shows that PHP doesn't get confused when embedding variables inside the output string. You don't need to place spaces before the variable names. Again, though, this can get confusing, and you should take care when embedding variables in your output.

WARNING

There is one oddity with using single quotes for string values in PHP. For some reason, PHP doesn't recognize the `\n` newline character when you use single quotes. For that reason, I tend to stick with using double quotes for my string values.

Grouping data values with array variables

Array variables allow you to group related data values together in a list using a single variable name. You can then either reference the values as a whole by referencing the variable name or handle each data value individually within the array by referencing its place in the list.

PHP supports two types of arrays: numeric and associative. The following sections cover these array types.

Numeric

The standard type of array variable is the *numeric array*. With the numeric array, PHP indexes each value you store in the array with an integer value, starting at 0 for the first item in the array list.

The way to define an array is to use the PHP `array()` function in an assignment statement:

```
$myscores = array(100, 120, 115);
```

Just because the array is a numeric array, that doesn't mean you're restricted to storing only numeric values:

```
$myfamily = array("Rich", "Barbara", "Katie", "Jessica");
```

Starting in PHP version 5.4, you can also define an array using square brackets instead of the `array()` function:

```
$myscores = [110, 120, 115];
```

PHP references each value in the array using a positional number within square brackets after the variable name. The first element in the array is at position 0, the second at position 1, and so on.

For example, to retrieve the first value stored in the array, you'd use `$myfamily[0]`, which would return the value `Rich`.

Associative

The *associative array* variable is similar to what other programming languages call a "dictionary." Instead of using numeric indexes, it assigns a string key value to individual values in the list. You use the special `=>` assignment operator to do that when you define the array:

```
$favs = array("fruit" => "banana", "veggie" => "carrot");
```

This array definition assigns the key value of `fruit` to the data value `banana`, and the key value `veggie` to the data value `carrot`. With associative arrays, to reference a data value you must specify the key value in the square brackets:

```
$favs["fruit"]
```

There is one thing to watch out for, though, when using associative array variables in your PHP code. For some reason, the `echo` statement has a hard time detecting associative array variables, so it needs some help from you.

When you use an associative array variable in an `echo` statement, it's a good idea to enclose it in braces, like this:

```
echo "My favorite fruit is {$favs['fruit']}\n";
```

This separates out the associative array variable from the string, so the `echo` statement can properly process it. Also, notice that the problem with quotes pops up when using associative array variables inside the `echo` statement. Because you want the output to show the value of the associative array variable, you need to use double quotes for the echo statement string. That means you must use single quotes around the associative array variable key.

Using PHP Operators

Now that you know how to store data in variables and display those values on a web page, it's time to take a look at how to dynamically alter the values. The core of any programming language is the ability to let the computer system crunch your data and then display the results for you. To do that, you need data operators. This section covers the operators you'll run into when using arithmetic and string operations in your PHP code.

Arithmetic operators

Arithmetic operators provide the basic mathematical functions that you're used to seeing on your calculator, directly within your PHP programs. You can perform all the standard calculations shown in Table 1-1 in your PHP programs.

TABLE 1-1 **PHP Arithmetic Operators**

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus

Arithmetic operators are normally used in an assignment statement to perform the calculations:

```
$value1 = 10;
$value2 = 20;
$result = $value1 + $value2;
echo "The result is $result\n";
```

The first two lines assign values to the two values used in the arithmetic operation. If you try to use a variable that hasn't been assigned a value in an arithmetic operation, you'll get a warning from PHP.

The third line is where you use the arithmetic operation on the two values. If you've never done programming before, this statement may look a little odd. Don't think of it as a mathematical equation. The equal sign is still acting to perform the assignment in PHP. The PHP server first evaluates the arithmetic operation on the right side of the equal sign and then assigns the result to the `$result` variable specified on the left side.

You can use both integer and float data type values in your arithmetic operations. However, float data type values need a little explaining here.

You can define a float value in one of three ways:

```
$float1 = 3.14159;
$float2 = 2.3e10;
$float3 = 5E-10;
```

The `e` and `E` symbols represent an exponential value applied to the value specified. You can use very large and very small float numbers, but be careful because the precision that PHP uses is somewhat limited, based on the server system. Don't be surprised if you store the value 3.0 in a variable and then later on retrieve it and it shows as 3.00001. Extra care is needed when working with float values.

Arithmetic shortcuts

There are a few different shortcuts you can use when implementing arithmetic operators in your PHP code. A common function in programming code is to perform a mathematical operation on a value stored in a variable and then store the result back in the same variable, like this:

```
$counter = $counter + 1;
```

This code adds 1 to the value currently stored in the `$counter` variable and then saves the result back in the `$counter` variable. PHP provides a handy shortcut method for doing this:

```
$counter += 1;
```

This code accomplishes the exact same thing, but in a shorter form. You can use the same shortcut with any type of arithmetic operator:

```
$total *= 1.10;
```

This example multiplies the value stored in the `$total` variable by 1.10 and stores the result back in the `$total` variable. You can also use variables on the right side of the assignment operation:

```
$total *= $taxrate;
```

This is the same as typing the following:

```
$total = $total * $taxrate;
```

That can really save some typing for you!

Two other types of arithmetic shortcuts are the *incrementor* and *decrementor* operators. The incrementor operator adds 1 to a variable's value:

```
$counter++;
```

The decrementor operator subtracts 1 from the variable's value:

```
$counter--;
```

Now that's *really* saving some typing!

WARNING

The arithmetic shortcut operators assume there's already a value stored in the variable before the operation. If there isn't, PHP will generate a warning message, telling you that it assumes the initial value is 0. It's always a good idea to initialize a variable to a known value before trying to use it in any arithmetic operations.

Boolean operators

Besides the standard arithmetic operators, PHP supports Boolean operators for logical operations with data. Boolean math allows you to work with TRUE and FALSE conditions in your programs. The Boolean operators test whether two values are both TRUE, both FALSE, or one is TRUE and the other is FALSE. Table 1-2 shows the Boolean operators supported by PHP.

TABLE 1-2 PHP Boolean Operators

Operator	Description
and	logical AND
&&	logical AND
or	logical OR
||	logical OR
xor	logical XOR
!	logical NOT

Notice that PHP supports two forms for the AND and OR logical operations — both the symbols and the names. There's no preference as to which method to use, so feel free to use the method you're most comfortable with.

These operators come in handy when you need to evaluate two separate conditions at the same time:

```
if (($age > 50) and ($gender == "F"))
```

This can help to simplify the code in your programs!

String operators

When you think of text string values, you don't necessarily think of arithmetic operations, but PHP does include a string operator that comes in handy when working with string values.

The *concatenation operator* allows you to "add" two string values together to create a single string value. Basically, the concatenation operator appends the second string to the end of the first string.

The concatenation operator in PHP is the period:

```
$string1 = "This is ";
$string2 = "a test";
$result = $string1 . $string2;
```

The result stored in the `$result` variable will be the string `This is a test`. Note that the concatenation operator doesn't add any spaces either before or after the text it concatenates, so it's up to you to do that if you need the space!

Including Files

One feature of PHP that many web developers love is the ability to create and use *include files.* Include files (sometimes referred to as *server-side includes*) allow you to store HTML5 and PHP code in one file and then reference that file in another web page file. There are a couple of ways to do that in PHP.

The include() function

The `include()` function allows you to include the contents of one web page within another web page simply by referencing a filename on the server. The PHP processor includes all the code contained within the included file, both HTML5 and PHP, directly into the PHP code of the main file, exactly where you place the `include()` function. It's just as if you had typed in all the lines of code from the include file yourself into the main file!

Developers often use the `include()` function to create standard header or footer sections on all the web pages in an application. Instead of having to add the same header or footer code to every web page in the application, you just save the header code in one file and the footer code in another file, and then use the `include()` function to include the header and footer files into each web page code.

The format of the `include()` function is simple:

```
include(filename);
```

You just replace *`filename`* with the actual name of the file you need to include in the program code.

Follow these steps to test out using an include file in a web page:

1. **Open your editor and type the following code:**

```
<h1>This is a test header</h1>
<?php
    echo "<p>This is the header text</p>\n";
?>
```

2. **Save the file as `myinclude.inc.php` in the `DocumentRoot` folder for your web server.**
3. **Open your editor to a new document and type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing PHP includes</title>
</head>
<body>
<header>
<?php include("myinclude.inc.php"); ?>
</header>
```

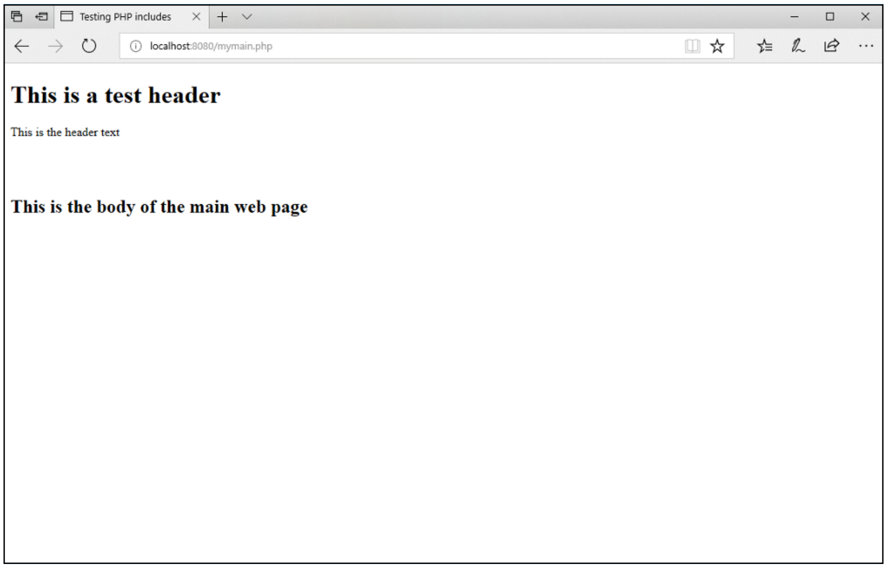
```
<section>
<br><br>
<h2>This is the body of the main web page</h2>
</section>
</body>
</html>
```

4. **Save the file as `mymain.php` in the `DocumentRoot` folder for your web server.**
5. **Ensure that the web server is running, and then open your browser and enter the following URL:**

   ```
   http://localhost:8080/mymain.php
   ```

6. **Close the browser when you're done.**

You should see the output as shown in Figure 1-5.



This is a test header

This is the header text

This is the body of the main web page

FIGURE 1-5: The results of the `mymain.php` program.

If you view the HTML5 code in your browser's Developer Tools, the output appears as a single HTML5 document. The browser is unaware that the code came from two separate files on the server, and your site visitors will have no idea that you cheated when creating all your web pages!

TIP

The filename you specify in the include() function can use any file extension — it's not required to use .php, because the include() function includes it into the main file for processing. However, it's become common to use the .inc.php file extension to identify include files and to separate them out from main PHP files. You can also use either an absolute or relative path name to reference the filename. Because the PHP server is accessing the file as a file and not as a web document, you can't use the http:// URL here — only a file path. Also, your web server must have access to read the file on the server.

The require() function

There is one limitation to the include() function that may cause problems for you. If PHP is unable to find the file you reference, it'll produce a warning, but the PHP server will continue to process the rest of the program code. That may have detrimental effects on your program!

There may be times where you don't want the PHP server to continue on processing code if a crucial include file is missing from the server. Instead, you may want the program to stop immediately and produce an error message instead of just a warning. This is where the require() function comes in.

The require() function works exactly like the include() function, except for one difference: It forces the PHP server to stop processing code if the include file fails to load.

To test this out, follow these steps:

1. **Open the mymain.php code from the previous example into your editor.**
2. **Change the include() function line to this:**

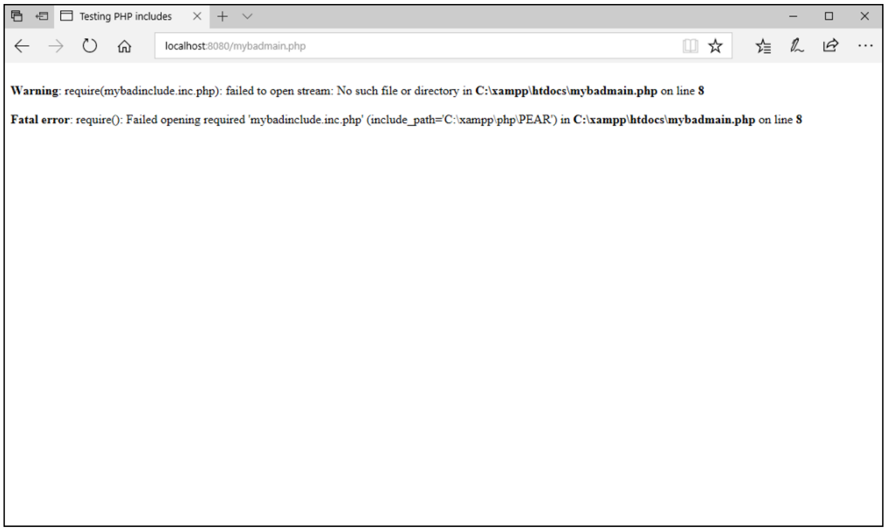
   ```
   require("mybadinclude.inc.php");
   ```

3. **Save the file as mybadmain.php in the DocumentRoot folder for your web server.**
4. **Ensure that the web server is still running and then open your browser and go to the following URL:**

   ```
   http://localhost:8080/mybadmain.php
   ```

5. **Close the browser and shut down the web server when you're done.**

When you run the mybadmain.php program, you may or may not see an error message on your web page, as shown in Figure 1-6.



```
Warning: require(mybadinclude.inc.php): failed to open stream: No such file or directory in C:\xampp\htdocs\mybadmain.php on line 8

Fatal error: require(): Failed opening required 'mybadinclude.inc.php' (include_path='C:\xampp\php\PEAR') in C:\xampp\htdocs\mybadmain.php on line 8
```

FIGURE 1-6: The output from the `mybadmain.php` program.

If you have the `display_errors` setting enabled in your PHP server configuration file, you'll see the error message. None of the HTML5 code from the main program code appears on the web page, because the PHP server stopped processing code after the `require()` function failed.

IN THIS CHAPTER

» Adding conditional tests

» Looping through code

» Building functions

» Working with event-driven programming

Chapter 2 PHP Flow Control

In the preceding chapter, I cover the basics of creating and running PHP programs. I show you how to use variables to hold data, but you don't really do much with them to test the data and perform operations. In this chapter, I walk through how to use the PHP conditional tests to control how your program behaves, as well as show how to loop through code to perform multiple iterations. In case you have code that you find yourself using frequently, I show how you can convert them into functions to share among your programs. Finally, I cover how to use PHP code in your event-driven web applications to add to your dynamic web applications.

Using Logic Control

Only having variables and `echo` statements in your PHP program would be pretty boring. You need to give your programs some intelligence so that they can make decisions based on what's happening in the application and display different sets of content based on those decisions.

Every programming language has methods for controlling the order the program handles statements, called the *program flow*, and PHP is no different. This section walks through the basics of controlling program flow in your PHP programs.

The if statement

The `if` statement controls which statements PHP should run in the program based on conditions. You use `if` statements in your everyday life (for example, if it's raining, then you'll bring an umbrella). You apply the same logic to your PHP programs.

The basic format for the `if` statement is:

```
if (condition)
    PHP statement to run
```

PHP evaluates the condition defined inside the parentheses to determine whether it should run the specified PHP statement that appears immediately after the `if` statement. The condition uses a special PHP expression called the *comparison operator*, which it uses to compare two values. If the comparison evaluates to a Boolean `TRUE` value, PHP runs the statement listed after the `if` statement. If the comparison evaluates to a Boolean `FALSE` value, PHP skips the statement.

This may sound confusing, but it's not all that hard when you get used to the format. Here's an example of a simple `if` statement:

```
if ($age > 21)
    echo "Sorry, you are too old to play";
```

The condition inside the parentheses checks if the value stored in the variable named `$age` is greater than 21. If it is, the condition evaluates to a `TRUE` value and PHP runs the `echo` statement. If it isn't, the condition evaluates to a `FALSE` value and PHP skips the `echo` statement and moves on.

There are quite a few comparison operators that you have available to use in PHP. Table 2-1 shows the comparison operators available.

WARNING

Notice that the comparison operator used to check if two values are equal is the double equal sign, not a single equal sign. Forgetting that small detail causes all sorts of annoying errors in your PHP code because the equal sign performs an assignment operation, which always returns a `TRUE` value (been there, done that).

The triple equal sign not only compares the value of the variables, but also checks to make sure the variables contain the same data types. For example, a Boolean data type of `TRUE` will match against an integer data type of `1` using the double equal, but not the triple equal.

TABLE 2-1 **PHP Comparison Operators**

Operator	Description
`==`	Equal to the same value
`===`	Equal to the same value, and they're the same data type
`!=`	Not equal to the same value
`<>`	Not equal to the same value
`!==`	Not equal to the same value, or they aren't the same data type
`<`	Less than
`<=`	Less than or equal to
`>`	Greater than
`>=`	Greater than or equal to

If you need to control more than just a single statement using the `if` condition, group the statements using braces:

```
if (condition) {
    statement1
    statement2
    statement3
}
```

You can have as many PHP statements contained within the group block as necessary — they'll all be controlled by the single condition in the `if` statement line. Here's an example:

```
if ($price > 50) {
    $tax = $price * .07;
    $shipping = 10;
    $total = $price + $tax + $shipping;
}
```

In this example, the entire group of statements will only be run by PHP if the `$price` variable value is greater than 50.

The else statement

The `if` statement has a cousin, called the `else` statement. The `else` statement allows you to provide an alternative group of statements to run if the condition in the `if` statement evaluates to a `FALSE` value:

```
if (condition) {
    PHP statements to run if TRUE
} else {
    PHP statements to run if FALSE
}
```

This gives you total control over what PHP statements are run in any condition!

The elseif statement

You can string `if` and `else` statements together, but that uses a new statement in place of the `else` statement, called the `elseif` statement (yes, that's `else` and `if` as one word). An `elseif` statement looks like this:

```
if (condition1){
    PHP statements to run if condition1 is TRUE
} elseif (condition2) {
    PHP statements to run if condition2 is TRUE
}
```

You can string as many `elseif` statements into the code block as necessary to check for alternative conditions. Each `elseif` statement requires its own condition check.

Follow these steps to try out using `if`, `else`, and `elseif` statements:

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
2. **Type the following code into the editor:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing PHP Program Control</title>
</head>
<body>
```

```
<h1>Random number test</h1>
<?php
    $number = rand(1, 100);
    if ($number > 50) {
        echo "<h2>The value $number is big!</h2>\n";
    } elseif ($number > 25) {
        echo "<h2>The value $number is medium</h2>\n";
    } else {
        echo "<h2>The value $number is small</h2>\n";
    }
?>
</body>
</html>
```

3. **Save the file as `phpconditiontest.php` in the `DocumentRoot` folder for your web server.**

   For XAMPP on Windows, use `c:\xampp\htdocs`; for XAMPP on macOS, use `/Applications/XAMPP/htdocs`.

4. **Open the XAMPP Control Panel and start the Apache web server.**
5. **Open your browser and enter the following URL:**

   ```
   http://localhost:8080/phpconditiontest.php
   ```

   You may need to change the TCP port used in the URL to match your web server.

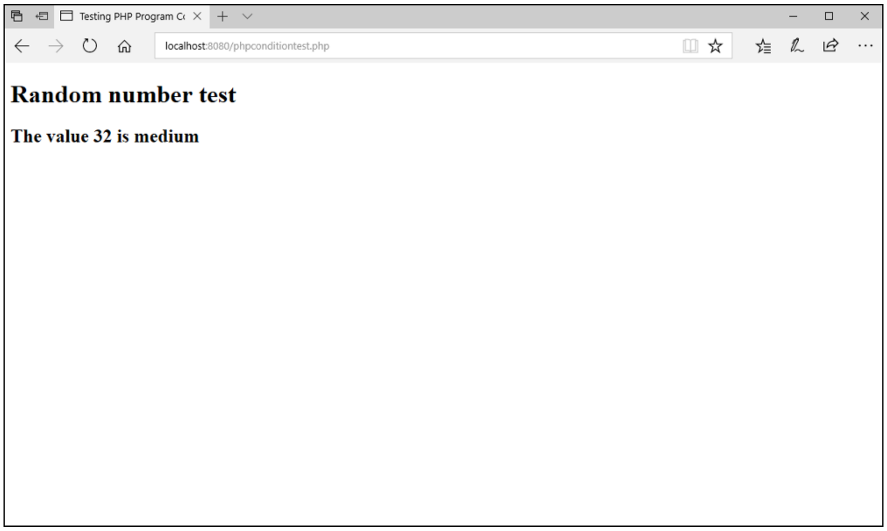
6. **Click the Refresh button on your browser to reload the web page.**

   That should run the PHP program again, selecting a new random number.

7. **Close the browser when you're done.**

The program uses the PHP `rand()` function to select a random number from 1 to 100. The value is compared in two separate condition checks in the `if` and `elseif` statements. If both fail, the code falls through to the final `else` statement. PHP runs the appropriate `echo` statement based on which condition succeeds. Figure 2-1 shows an example of the output you should see in your web page.

Each time you click your browser's Refresh button, the browser makes a new request to the server to reload the web page. That triggers the server to reload the web page in the PHP server, which in turn reruns the program.



Random number test

The value 32 is medium

FIGURE 2-1: The output from the `phpconditiontest.php` program.

The switch statement

Writing long `if`, `elseif`, and `else` statements to check for a long list of conditions can get tedious. To help out with that, PHP provides the `switch` statement. The `switch` statement allows you to perform one check, and then provide multiple values to compare the check against:

```
switch (condition) {
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    default:
        statement3;
}
```

The `switch` statement evaluates the condition you specify against the different values presented in each `case` statement. If one of the values matches the result of the condition, PHP jumps to that section of the code to run the statements contained in that section.

It's important to note, though, that the `case` statements are labels and not code blocks. After PHP runs the statements in the `case` section it jumped to, it continues to run the statements in all the `case` sections after it! To prevent that from happening, use the `break` statement at the end of the `case` code section. That causes PHP to break out of the `switch` statement and skip any remaining `case` sections.

Also, you can place a `default` statement section at the end of the `switch` statement code block. If none of the `case` values matches the condition value, PHP jumps to the `default` section.

Looping

Sometimes you'll find yourself needing to repeat the same operation multiple times, such as when you're displaying all the values in an array variable or database table. You could just write out all the PHP statements yourself, but that could get cumbersome:

```
$family = array("Rich", "Barbara", "Katie", "Jessica");
echo "One member of my family is $family[0]<br>\n";
echo "One member of my family is $family[1]<br>\n";
echo "One member of my family is $family[2]<br>\n";
echo "One member of my family is $family[3]<br>\n";
```

This code would certainly display all the elements contained within the array, but what if there were 100 elements in the array? That would require a lot of coding!

Notice that most of the code in the `echo` statements is the same — the only thing that differs is the index used in the array to reference the specific data element in the array. All that you need to do is iterate through the index numbers and use the same code. Well, that's exactly what you can do using the PHP looping functions.

PHP provides a family of looping functions available for you to use in your code. The following sections walk through the different ways to loop through code in PHP.

The while family

The `while` statement allows you to create a simple loop of code based on a condition that you specify in the statement:

```
while (condition) {
    statements
}
```

In each iteration of the loop, PHP evaluates the condition you specify. If the condition evaluates to a `TRUE` value, PHP runs the statements contained in the `while` code block. As soon as the condition evaluates to a `FALSE` value, PHP breaks out of the loop and continues on with the next statement after the loop.

The `while` statement is tricky in that something inside the loop code must alter the value checked in the condition; otherwise, it will never end (called an *endless loop*). Usually, there's some type of variable that you must change inside the loop and then check in the condition.

Follow these steps to test using the `while` statement to create a loop:

1. **Open your editor and type the following code:**

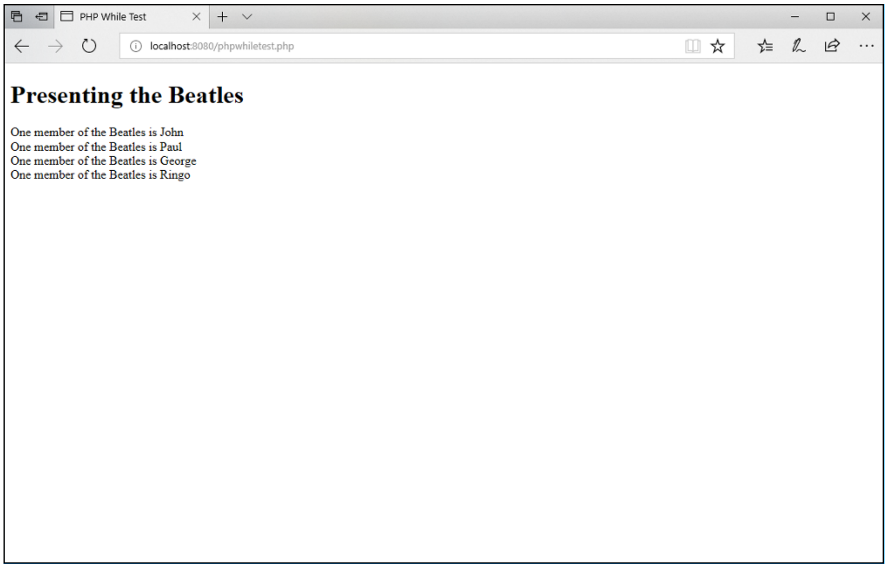
```
<!DOCTYPE html>
<html>
<head>
<title>PHP While Test</title>
</head>
<body>
<h1>Presenting the Beatles</h1>
<?php
$group = array("John", "Paul", "George", "Ringo");
$count = 0;
while ($count < 4) {
  echo "One member of the Beatles is
            $group[$count]<br>\n";
  $count++;
}
?>
</body>
</html>
```

2. **Save the file as** `phpwhiletest.php` **in the** `DocumentRoot` **folder for your web server.**
3. **Ensure that the web server is running and then open your browser and enter the following URL:**

```
http://localhost:8080/phwhiletest.php
```

4. **Close the browser when you're done.**

When you run the program, you should see the output as shown in Figure 2-2.



Presenting the Beatles

```
One member of the Beatles is John
One member of the Beatles is Paul
One member of the Beatles is George
One member of the Beatles is Ringo
```

FIGURE 2-2: The output of the `phpwhiletest.php` program.

Remember that array data indexes always start at 0, so you need to start the `$count` variable at 0 before entering the loop. In the `while` loop condition, you need to check to make sure the `$count` variable value hasn't gotten past the last index in the array. With four data elements in the array, the last index value is 3. So, as long as the `$count` variable value is less than 4, the program can continue iterating through the code in the loop. The code uses the `$count` variable as the `$group` array index to reference each individual data element in the `echo` statement. Finally, there's an incrementor statement to add 1 to the `$count` variable at the end of each loop iteration.

Similar to the `while` statement is the `do...while` statement. The `do...while` statement changes the order of when the condition check is performed:

```
do {
    statements
} while (condition)
```

With the `do...while` loop, PHP doesn't check the condition until after it runs the code inside the loop block. This ensures that the code will be run at least one time, even if the condition evaluates to a `FALSE` value.

The for statement

The `while` loop statement is a great way to iterate through a bunch of data, but it can be a bit cumbersome to use. With the `while` statement, you need to make sure you set a PHP variable that changes value inside the loop code, and make sure you code the condition to stop when that variable reaches a specific value. Sometimes with large blocks of code, that can get complicated to track.

PHP provides an all-on-one type of looping statement called the `for` statement. The `for` statement can keep track of loop iterations for you.

Here's the basic format of the `for` statement:

```
for(statement1; condition; statement2) {
    PHP statements
}
```

The first parameter, *`statement1`*, is a PHP statement that the PHP server runs before the loop starts. Normally, this statement sets the initial value of the counter used in the loop.

The middle parameter, *`condition`*, is the standard PHP condition check that's evaluated after each loop iteration. The last parameter, *`statement2`*, is a PHP statement that's run at the end of each loop iteration. This is normally set to change the value of the counter used in the loop.

Here's the same code used to demonstrate the `while` loop, but using the `for` statement:

```
<?php
$group = array("John", "Paul", "George", "Ringo");
for ($count = 0; $count < 4; $count++ ) {
  echo "One member of the Beatles is
            $group[$count]<br>\n";
}
?>
```

Because the `for` loop does everything for you, you don't need to worry about incrementing the counter value inside the code block. At the end of each iteration, PHP runs the incrementor specified in the for statement for you.

The foreach statement

One problem that you may often run into with PHP is having to iterate through all the data elements contained within an associative array variable.

REMEMBER

An associative array uses text keys, not numbers, to track data values. There's no way you can increment through the keys in an associative array variable using the `for` statement.

Fortunately, the PHP developers have come to your rescue with the `foreach` statement. The `foreach` statement loops through each of the keys created in an associative array and allows you to retrieve both the key and its associated value.

Here's the format of the `foreach` statement:

```
foreach (array as $key => $value) {
    PHP statements
}
```

In each iteration, the `foreach` statement assigns the associative key to the *`$key`* variable, and its associated value to the *`$value`* variable. You can then use those variables in your PHP code inside the code block.

Follow these steps to try out the `foreach` statement with an associative array variable:

1. **Open your editor and type the following code:**

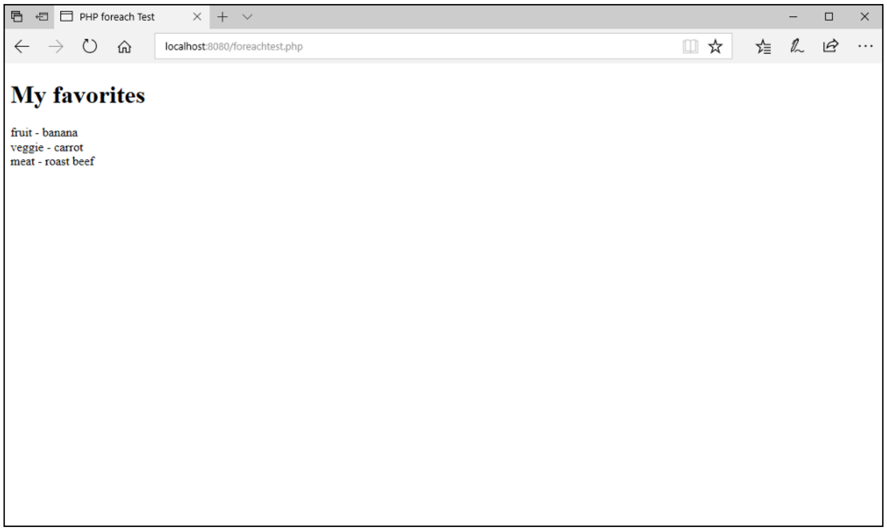
```
<!DOCTYPE html>
<html>
<head>
<title>PHP foreach Test</title>
</head>
<body>
<h1>My favorites</h1>
<?php
$favs = array("fruit"=>"banana","veggie"=>"carrot","meat"
     =>"roast beef");
foreach($favs as $food => $type) {
    echo "$food - $type<br>\n";
}
?>
</body>
</html>
```

2. **Save the file as `foreachtest.php` in the `DocumentRoot` folder of the web server.**
3. **Ensure that the web server is running, and then open your browser and enter the following URL:**

```
http://localhost:8080/foreachtest.php
```

4. **Close the browser when you're done.**

When you run the program, you should get the results shown in Figure 2-3.



My favorites

```
fruit - banana
veggie - carrot
meat - roast beef
```

FIGURE 2-3: The output from the `foreachtest.php` program.

The `foreach` statement iterates through each key contained in the `$favs` associative array variable, assigning the key to the `$food` variable and its value to the `$type` variable. The code then uses the `echo` statement to display the values on the web page.

Building Your Own Functions

While you're coding in PHP, you'll often find yourself using the built-in functions available (such as the `rand()` function you used earlier in the example programs). *Functions* are nothing more than PHP code someone else wrote to accomplish a useful feature that you can use in any program. Instead of having to copy all the code into your application, you just use the function name.

PHP allows you to create your own functions to use in your programs and share with others. After you define a function, you can use it throughout your program. This saves typing if you use a common routine or block of code in lots of places in your application. All you need to do is write the code once in the function definition and then call the function everywhere else you need it.

The basic format for a function definition looks like this:

```
function name(parameters) {
    function code
    return value;
}
```

The *name* must uniquely identify the function. It can’t be one of the existing PHP function names, and it can’t start with a number (although numbers can appear anywhere else in the function name).

The *parameters* identify one or more variables that the calling program can pass to the function (or you can have a function that requires 0 parameters). If there is more than one variable in the parameter list, you must separate them with commas. You can then use the variables anywhere within the function code, but they only apply to inside the function code block. You can’t access the passed parameter variables anywhere else in the program code.

Any variables you define inside the function code apply only to the function code. You can’t use function variables in the PHP code outside the function definition.

The `return` statement allows you to pass a single value back to the calling program. It’s the last statement in the function definition code, and it returns control of the program back to the main code section in your program.

Try out the following steps to experiment with creating a function and using it in your PHP program:

1. **Open your editor and type the following code:**

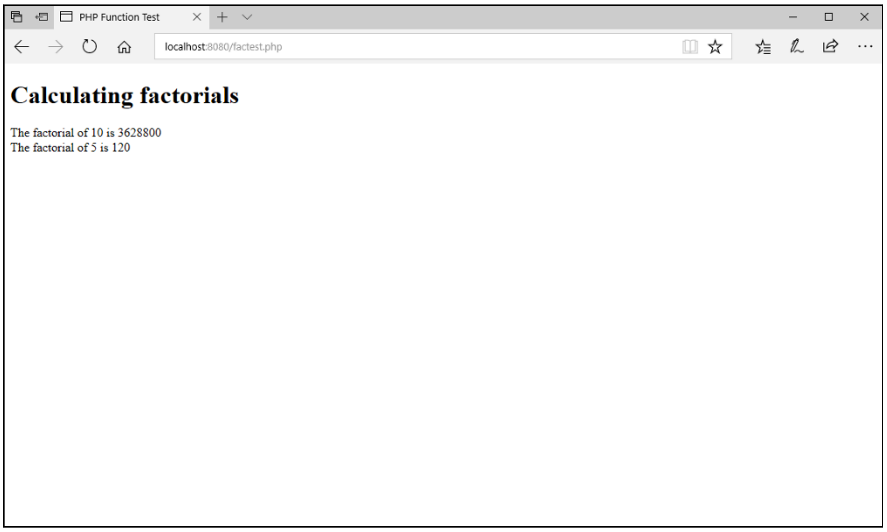
```
<!DOCTYPE html>
<html>
<head>
<title>PHP Function Test</title>
</head>
<body>
<?php
function factorial($value1) {
    $factorial = 1;
    $count = 1;
    while($count <= $value1) {
        $factorial *= $count;
    $count++;
    }
    return $factorial;
}
?>
<h1>Calculating factorials</h1>
<?php
echo "The factorial of 10 is " . factorial(10) . "<br>\n";
echo "The factorial of 5 is " . factorial(5) . "<br>\n";
?>
</body>
</html>
```

2. **Save the file as** `factest.php` **in the** `DocumentRoot` **folder for your web server.**
3. **Open your browser and enter the following URL:**

```
http://localhost:8080/factest.php
```

4. **Close your browser when you're done.**

When you run the `factest.php` program, the output should look like what's shown in Figure 2-4.



Calculating factorials

The factorial of 10 is 3628800
The factorial of 5 is 120

FIGURE 2-4: The output from the `factest.php` program.

All the code required to calculate the function is contained within the `factorial()` function definition code block. When PHP uses the `factorial()` function, it passes a single value that the function assigns to the `$value1` variable. When the calculation is complete, the function code returns the results back to the main program.

The main program uses the `factorial()` function twice in the code, both embedded in `echo` statements:

```
echo "The factorial of 10 is " . factorial(10) . "<br>\n";
echo "The factorial of 5 is " . factorial(5) . "<br>\n";
```

You can embed variables inside the string values in `echo` statements, but you can't embed functions. To insert the output from the function into the `echo` statement output, the code uses the string concatenation operator (the period) to "glue" the output from the strings and the `factorial()` functions into a single string to display.

TIP

If you have lots of functions that you use in many of your programs, you can define them in a separate file. Then to use the functions in your programs just use the `include()` function to include the function file, and you can then use the functions inside your programs without having to retype them!

Working with Event-Driven PHP

Because PHP is a server-side programming language, you can't associate it directly with events that occur within the browser. However, that said, you can link your PHP web pages to specific events in the web page so that the browser can request a specific web page based on an event.

There are basically two methods for doing that:

- Creating a link to a PHP web page
- Creating a form to pass data to a web page

The following sections describe how to use each of these event-driven methods to launch your PHP web pages.

Working with links

In HTML5, you create hypertext links on the web page using the anchor element:

```
<a href="mypage.html">Click here</a>
```

The text Click Here appears on the web page, and when the site visitor clicks that link, the browser requests the `mypage.html` file from the web server.

You can use this method for passing small amounts of data to the PHP web pages in your web application. As part of the URL, you can embed variable/value pairs after the URL location that get passed to the web server:

```
<a href="mystore.php?content=store">Click to shop</a>
```

The browser sends the data combination of `content` and `store` to the web server as part of the `GET` request for the new web page. If you need to send more data, separate them with the ampersand sign:

```
href="mystore.php?content=buy&prodid=10"
```

This link sends two variable/value pairs to the web server using the `GET` method:

```
content=buy
prodid=10
```

To retrieve the data values passed using the `GET` method in your PHP code, use the special array variable `$_GET[]`. The PHP server populates the `$_GET[]` array variable with all the variable/value pairs passed in the `GET` method from the client browser. You can then access those array variables in your PHP program code.

Follow this example to test out using the `GET` method to pass data from a link click event to a PHP program:

1. **Open your editor and enter the following code:**

   ```
   <!DOCTYPE html>
   <html>
   <head>
   <title>Testing Link Events in PHP</title>
   </head>
   <body>
   <h1>Please select one of the following links:</h1>
   <a href="linktest2.php?content=buy">Buy products</a><br>
   <a href="linktest2.php?content=browse">Browse for products</a><br>
   <a href="linktest2.php?content=help">I need assistance</a><br>
   </body>
   </html>
   ```

2. **Save the file as `linktest.html` in the `DocumentRoot` folder for your web server.**

3. **Open a new window in your editor and enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing Link Events in PHP</title>
</head>
<body>
<h1>Thanks for visiting us!<h1>
<?php
$content = $_GET['content'];
echo "<h2>You are in the $content section</h2>\n";
?>
</body>
</html>
```

4. **Save the file as `linktest2.php` in the `DocumentRoot` folder for your web server.**
5. **Ensure that the web server is running, and then open your browser and enter the following URL:**

```
http://localhost:8080/linktest.html
```

6. **Click one of the links on the web page, and observe what appears in the resulting web page.**
7. **Close the browser when you're done.**

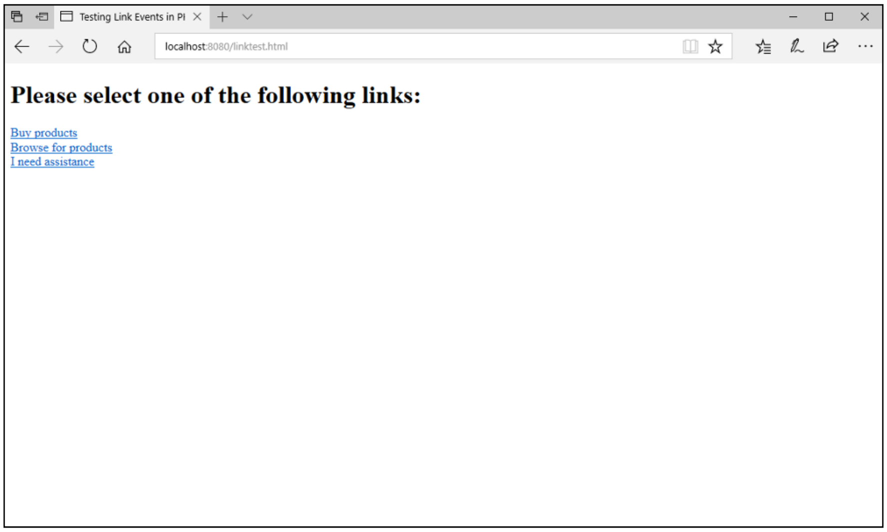
When you open the `linktest.html` web page, you'll see a series of links that simulate a navigation menu bar in a web page. Each link consists of a hyptertext link that points to the same web page (`linktest2.php`) but sets a different value for the `content` variable passed in the `GET` method. When you open the page, you should see the results, as shown in Figure 2-5.

When you click a link on the web page, the browser sends a `GET` request to the web server for the specified web page file, and passes the `content` variable setting assigned in the anchor element tag.

When the Apache web server receives the `GET` request from the client browser, it retrieves the `linktest2.php` file, and because it uses the `.php` file extension, it passes it to the PHP server to process the embedded PHP code. The PHP server detects the `GET` variable/value pair passed and assigns it to the `$_GET[]`

array variable. The code in the `linktest2.php` code retrieves that value from the `$_GET[]` array variable and assigns it to another variable:

```
$content = $_GET['content'];
```



Testing Link Events in Pl

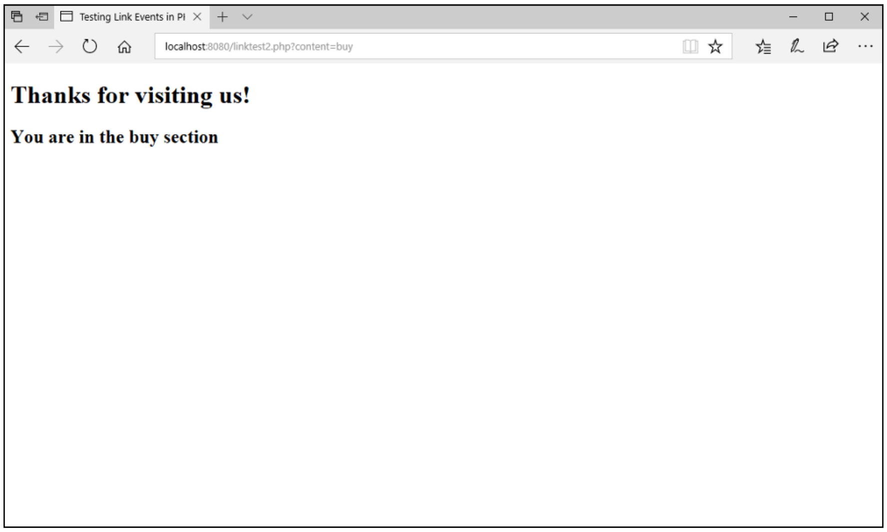
localhost:8080/linktest.html

Please select one of the following links:

Buy products
Browse for products
I need assistance

FIGURE 2-5: The output from the `linktest.html` file.

The code then uses that variable in the `echo` statement to display on the web page. The result is shown in Figure 2-6.



Testing Link Events in Pl

localhost:8080/linktest2.php?content=buy

Thanks for visiting us!

You are in the buy section

FIGURE 2-6: The result of clicking the Buy a Product link on the `linktest.html` web page.

WARNING

Notice the URL that appears in the address bar in the `link2test.php` web page. It contains the content variable, along with the value that was set by the anchor element `href` attribute. Because the values set using the `GET` method appear in the URL, they aren't a secure method of sending data. You should limit using the `GET` method to passing data about web pages and not personal information.

Processing form data

Book 2, Chapter 3, discusses how to build data entry forms using HTML5 code. To refresh your memory, the core of the HTML5 form is the `<form>` tag. This tag defines the beginning and end of the data fields that make up the form. The `<form>` tag uses three main attributes:

- `name`: Specifies a unique name for the form
- `method`: Specifies the HTTP method used to pass data
- `action`: Specifies the web page to pass the form data to

Within the form element, you include HTML elements for text boxes, text areas, radio buttons, check boxes, and other HTML5 form data fields. Each element uses a unique name to identify it in the form data that the browser sends to the action web page.

Because PHP runs on the server, it has no way of knowing when the site visitor is done filling out the form data fields in the browser window. With PHP, it's imperative to have a Submit button in the form to indicate to the browser when to send the form data to the web page specified in the `action` attribute, using the method specified in the `method` attribute.

A simple HTML5 form to use with PHP would look like this:

```
<form name="myform" action="mypage.php" method="POST">
<label>First name</label>
<input type="text" name="fname" size="40"><br>
<label>Last name</label>
<input type="text" name="lname" size="40"><br>
<input type="submit">
</form>
```

After the site visitor fills in the form data, she needs to click the Submit button to send the data to the `mypage.php` file specified in the form `action` attribute. The browser sends the form data embedded behind the scenes in the HTTP communication with the web server.

In the receiving web server, it passes the data received by the POST method to the PHP server, which uses the special $_POST[] array variable to retrieve the form data. You can then access that data in your PHP code using the $_POST[] array variable, along with the form field names:

```
$firstname = $_POST['fname'];
$lastname = $_POST['lname'];
```

The same method works for retrieving data from a <textarea> form field.

To retrieve the value from a select element, the name attribute of the select element defines the field name, and the option element value attribute for the option selected in the field is the value passed in the POST data. Consider the following form field:

```
<select name="age">
<option value="young">18-35</option>
<option value="middleage'>36-55</option>
<option value="old">56+</option>
</select>
```

When the site visitor selects the option labeled 18-35 in the drop-down list, the form sends the value young in the POST data. The PHP code can then access the $_POST['age'] array variable to retrieve the selected value.

To retrieve the value from a radio button element, the name attribute for all the buttons in the same group is the same. The value attribute defines what data is sent to the server as part of the POST data:

```
<input type="radio" name="age" value="young">18-35
<input type="radio" name="age" value="middleage">36-55
<input type="radio" name="age" value="old">56+
```

The PHP code checks the $_POST['age'] variable for the data value passed by the selected radio button.

Working with check box data fields can be a little tricky. The check box doesn't pass any data — it just indicates whether the box is checked. If the box is checked, it sends the value specified by the value attribute assigned to the data field specified name attribute:

```
<input type="checkbox" name="age" value="old">
```

If the site visitor checks the box in the form, the form sends the data field age with a value of old, and your PHP code can retrieve the selection using the $_POST['age'] array variable.

The problem comes in if the site visitor doesn't select the check box. If the check box is not selected, the form doesn't send any data for the form field. In that case, if you try using the `$_POST['age']` array variable, you get an error from PHP that it doesn't exist.

To determine if a check box form field has been selected, you use the `isset()` PHP function. The `isset()` function returns a `TRUE` value if the PHP variable exists and has a value assigned to it or a `FALSE` value if not. You can then write something like this:

```
if (isset($_POST['age'])) {
    $age = $_POST['age'];
} else
    $age = "not selected";
}
```

Now you're able to determine whether the site visitor selected the check box.

Working with forms and PHP can be a bit tricky, but the more you practice, the better you'll get at it. Try out this example to get a feel for how to work with forms and PHP:

1. **Open your editor and type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>PHP Form Test</title>
<style>
    input, textarea {
        margin: 5px;
    }
</style>
</head>
<body>
<h1>Please fill in the form</h1>
<form action="formtest.php" method="post">
<fieldset>
<legend>My test form</legend>
<label>First name</label>
<input type="text" name="fname" size="40"><br>
<label>Last name</label>
<input type="text" name="lname" size="40"><br>
<fieldset>
<legend>Select your favorite sport</legend>
```

```
<input type="radio" name="sport" value="baseball">Baseball<br>
<input type="radio" name="sport" value="football">Football<br>
<input type="radio" name="sport" value="hockey">Hockey<br>
<input type="radio" name="sport" value="soccer">Soccer<br>
</fieldset>
<label>Please type your essay</label>
<textarea name="essay" cols="50" rows="10"></textarea><br>
<input type="submit" value="Submit your form">
</fieldset>
</body>
</html>
```

2. **Save the file as** `formtest.html` **in the** `DocumentRoot` **folder for your web server.**

3. **Open a new window in your editor and type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>PHP Form Test</title>
</head>
<body>
<h1>Form results:</h1>
<?php
$fname = $_POST['fname'];
$lname = $_POST['lname'];
if (isset($_POST['sport'])) {
    $sport = $_POST['sport'];
} else {
    $sport = "not specified";
}
$essay = $_POST['essay'];

echo "<h2>First name: $fname</h2>\n";
echo "<h2>Last name: $lname</h2>\n";
echo "<h2>Favorite sport: $sport</h2>\n";
echo "<h2>Essay response:</h2>\n";
echo "<p>$essay</p>\n";
?>
</body>
</html>
```

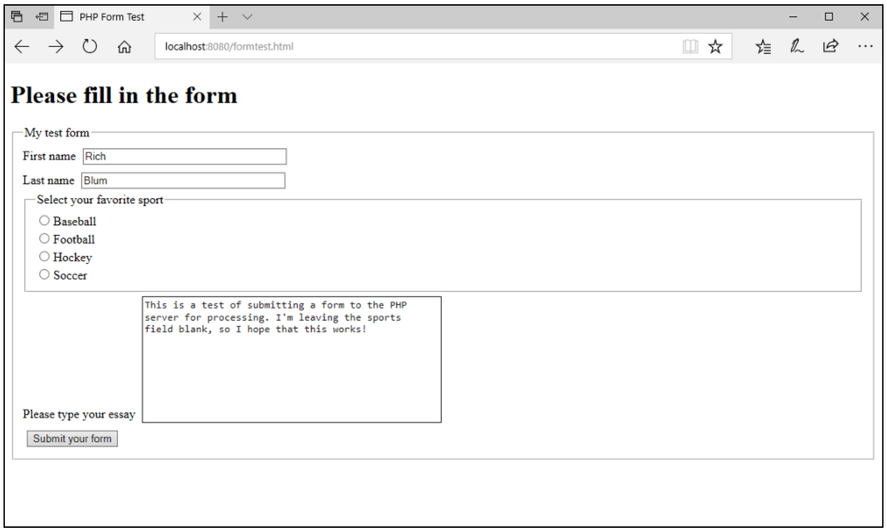
4. **Save the file as** `formtest.php` **in the** `DocumentRoot` **folder for your web server.**

5. **Ensure the web server is running and then open your browser and enter the following URL:**

```
http://localhost:8080/formtest.html
```

6. **Fill in the from data fields, selecting a radio button but leaving the check boxes all unchecked.**
7. **Click the Submit button when you're done filling in the form.**
8. **Close the browser and shut down the web server.**

The `formtest.html` file displays a standard HTML5 form on the web page, as shown in Figure 2-7.



Please fill in the form

My test form

First name Rich

Last name Blum

Select your favorite sport

○ Baseball
○ Football
○ Hockey
○ Soccer

```
This is a test of submitting a form to the PHP
server for processing. I'm leaving the sports
field blank, so I hope that this works!
```

Please type your essay

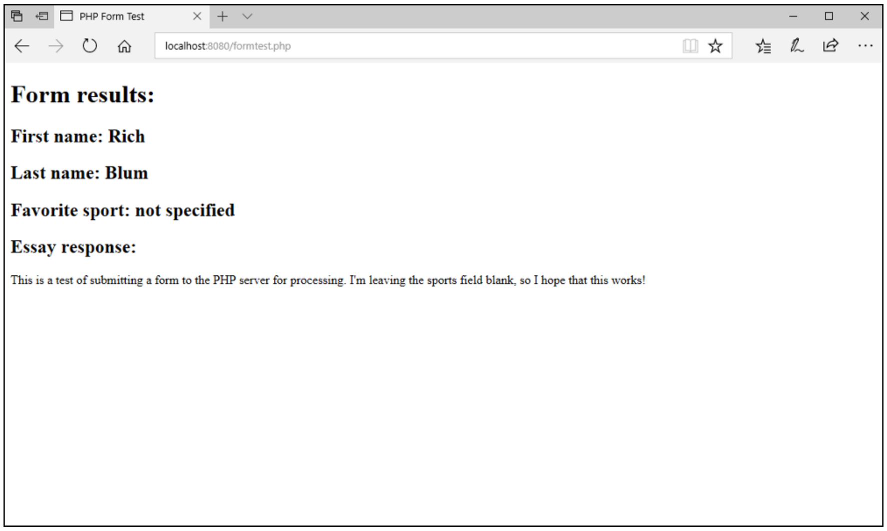
Submit your form

FIGURE 2-7: The web form produced by the `formtest.html` file.

Enter your data in the form, but don't make a selection for your favorite sport. When you click the Submit button, the browser sends the form data as part of a `POST` method to the web server, which passes the form data to the `formtest.php` file as specified in the form `action` attribute.

The `formtest.php` code retrieves the form data and detects that none of the radio buttons was selected. By using the `isset()` function. It displays the data passed from the form, as shown in Figure 2-8.

Now you're ready to process any HTML5 form using your PHP server-side programming skills!



Form results:

First name: Rich

Last name: Blum

Favorite sport: not specified

Essay response:

This is a test of submitting a form to the PHP server for processing. I'm leaving the sports field blank, so I hope that this works!

FIGURE 2-8: The form results as shown from the `formtest.php` file.

IN THIS CHAPTER

» Getting familiar with PHP libraries

» Working with text functions

» Handling numbers

» Using dates

» Playing with images

Chapter 3
PHP Libraries

As you start creating your dynamic web applications, you'll often find yourself wanting to perform certain functions that require quite a bit of coding, such as manipulating data or performing complex mathematical calculations. The true test of a robust programming language is in how much work it can save you by providing prebuilt code libraries that do most of the hard coding work for you. Fortunately, PHP has an extensive set of built-in libraries that can save you lots of development time as you build your web applications! This chapter dives into the basics of using the built-in libraries in PHP.

How PHP Uses Libraries

All programming languages provide libraries of functions that help you with your coding. How many there are and how they do that differs somewhat between programming languages.

Some interpreted programming languages compile all the function libraries into a single monolithic executable program that loads into memory each time the web server runs a program that requires the interpreter. That can be a huge resource hog on your server!

PHP took a more modular approach to things. Instead of compiling all the function libraries in a single program, PHP provides them as separate loadable library files, called *extensions.* That way, you (or your web-hosting company) can opt to load only the extensions you need to use, saving memory on the server and hopefully improving the performance of the PHP server.

The downside to this approach is that you need to be more aware of just what PHP extensions are available and which ones you should load. This section shows you how PHP splits functions up into different extensions and how you can find the functions you need to do your work.

Exploring PHP extensions

More than 150 extensions are available in the PHP package! There are extensions to cover functions as simple as manipulating string values or as complex as interacting with online search engines. The PHP developers have classified these extensions into 27 categories. Table 3-1 shows the different categories, along with a brief description of what each category contains.

TABLE 3-1 PHP Extension Categories

Category	Description
PHP behavior	Functions that control how the PHP server operates
Audio formats	Functions that handle and manipulate audio files
Authentication	Functions that work with authentication services
Command line	Functions that interact with the server command-line environment
Compression	Functions that compress and archive files and folders
Credit card	Functions that process credit card transactions
Cryptography	Functions that encrypt and decrypt data
Database	Functions that interact with database servers
Date and time	Functions that handle dates and times
File system	Functions that interact with the server file system
GUI	Functions that work with user interface features
Human language	Functions that work with character sets
Image processing	Functions that create and manipulate images
Mail	Functions that interact with mail servers
Mathematical	Functions that perform complex mathematical operations

Category	Description
Non-text MIME	Functions that handle binary data in MIME messages
Process control	Functions that interact with processes on the server
Other	Miscellaneous functions that manipulate data
Other services	Functions that interact with network services
Search engine	Functions that interact with online search engines
Session	Functions that handle browser sessions
Text	Functions that manipulate and process text
Variable	Functions that work with complex objects and data structures
Web services	Functions that interact with web service servers and clients
Windows	Functions that access Microsoft Windows features on Windows servers
XML	Functions that handle and manipulate data in XML format

Each category contains multiple extensions that are available for you to load and use in your PHP programs. There are far too many PHP extensions to list them all individually here. For a full and current list of the PHP extensions, go to the PHP online documentation at `www.php.net/manual/en/funcref.php`.

Examining the PHP extensions

You can view which extensions are actively installed in your specific PHP server environment by using the special `phpinfo()` function. Just include that as a single line in a PHP program. When you run the program, the `phpinfo()` function displays a table showing detailed information about the PHP server, including which PHP extensions are currently installed.

Follow these steps to determine which PHP extensions are installed in your PHP server environment.

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
2. **Type the following code:**

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

```
<?php
phpinfo();
?>
</body>
</html>
```

3. **Save the file as `extensions.php` in the `DocumentRoot` folder for your web server.**

For XAMPP on Windows, that's `c:\xampp\htdocs`; for XAMPP on macOS, it's `/Applications/XAMPP/htdocs`.

4. **Open the XAMPP Control Panel, and then start the Apache web server.**

5. **Open your browser, and enter the following URL:**

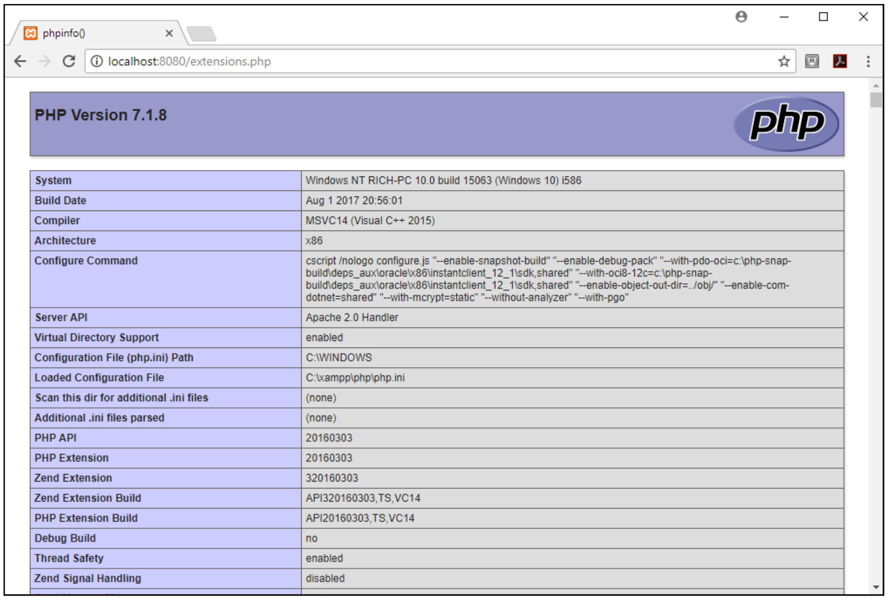
```
http://localhost:8080/extensions.php
```

You may need to change the TCP port in the URL to match your web server.

6. **Examine the output generated by the `phpinfo()` function, looking for which extensions are installed on your system.**

7. **Close the browser when you're done.**

Figure 3-1 shows the results from the XAMPP package running on a Windows workstation.



PHP Version 7.1.8

System	Windows NT RICH-PC 10.0 build 15063 (Windows 10) i586
Build Date	Aug 1 2017 20:56:01
Compiler	MSVC14 (Visual C++ 2015)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x86\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x86\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20160303
PHP Extension	20160303
Zend Extension	320160303
Zend Extension Build	API320160303,TS,VC14
PHP Extension Build	API20160303,TS,VC14
Debug Build	no
Thread Safety	enabled
Zend Signal Handling	disabled

FIGURE 3-1: The output from the `phpinfo()` function.

As you scroll through the listing generated by the `phpinfo()` function, you'll see separate sections devoted to the different extensions and the configuration settings that control how they operate. Most likely, your PHP server has quite a few (if not all) of the extensions already activated. If any are missing, you can usually activate them yourself. That's covered in the next section.

Including extensions

Most PHP server environments include all the extension library files in the PHP server build, but they may not activate all of them to help save memory as the PHP server runs. If you find yourself needing to activate a specific PHP extension, you can easily do that from the PHP configuration file.

The first step is to find the `php.ini` configuration file for your PHP server environment. The easiest way to do that is from the output of the `phpinfo()` function.

If you followed the steps in the previous exercise, you can view the output of the `phpinfo()` function in your browser. In that output, look for the line in the top section for Loaded Configuration File. That shows the path to the configuration file the PHP server is using.

Using your system's file manager program (File Explorer for Windows, Finder for Mac), navigate to the folder where the `php.ini` file is stored, and then double-click the file to open it with a text editor.

Look for the section labeled Dynamic Extensions within the `php.ini` configuration file. This is where the configuration file defines the extensions to install. Each extension is referenced by a single line. For Windows systems, it looks like this:

```
extension=name.dll
```

For Mac and Unix/Linux systems, it looks like this:

```
extension=name.so
```

The extension names are in the format `php_`*`name`* where *name* is the unique name assigned to the extension. For example, the extension for interacting with MySQL servers is called `php_mysqli` (the *i* is added because it's an improved version from the original MySQL extension).

WARNING

Not all PHP server environments use extensions, so you may not see any entries for them in the php.ini configuration file. For example, the XAMPP for the macOS environment compiles all the extensions directly into the main PHP server executable.

Adding additional extensions

As you can probably guess, you can create your own PHP extensions for your own custom functions. This has become quite popular in the PHP developer world, and a clearinghouse has been created for sharing custom-made extensions with other PHP developers.

The PHP Extension Community Library (PECL) hosts a library of custom extensions shared by developers from around the world. You can access PECL at `https://pecl.php.net`. There, you'll find extensions that add additional functionality to the standard PHP libraries, as well as add entirely new features, such as the `html_parse` extension, which provides functions to access a remote web page and parse the DOM tree elements to extract data!

Now that you know about PHP extensions, the following sections take a look at some of the more popular ones and the functions they contain that can help save you some time in your PHP coding.

Text Functions

Just about every web application needs to work with text data. There's a wealth of text processing and manipulation functions available at your fingertips within the PHP extension library. This section walks through some of the more useful ones that may come in handy as you process data in your applications. There are so many text functions provided by PHP that trying to find just what you're looking for in the PHP online manual can be a bit overwhelming. This section breaks up the functions into categories to help simplify things a bit.

Altering string values

PHP provides a handful of functions that manipulate either the text or the text format in string values. Table 3-2 shows the string functions that can be useful when you need to manipulate string values.

TABLE 3-2 PHP String Manipulation Functions

Function	Description
addslashes	Adds an escape character (backslash) in front of single quote, double quote, backslash, and NULL characters.
chop	Removes all whitespace characters from the end of a string.
htmlentities	Converts HTML codes into HTML tags.
htmlspecialchars	Converts any HTML tags embedded in a string into HTML codes.
lcfirst	Changes the first character of the string to lowercase.
ltrim	Removes any whitespace characters from the start of a string.
money_format	Formats a monetary string value into a currency format.
nl2br	Converts newline characters to the `<br>` HTML tag.
number_format	Allows you to specify the format to display a number value.
rtrim	Removes all whitespace characters from the end of a string.
str_replace	Replaces the occurrences of a string with another string.
strip_tags	Removes all HTML and PHP tags from a string.
strtolower	Converts the string to lowercase.
strtoupper	Converts the string to uppercase.
trim	Removes all whitespace characters from the start and end of a string.
ucfirst	Converts the first character of the string to uppercase.

The string manipulation functions don't change the value of the original string — they just return a new string value. If you want to use the result in your program, you have to assign it to another variable:

```
$newvalue = trim($data);
```

The `htmlspecialchars()` and `strip_tags()` functions are extremely helpful if you're creating a web application that accepts data from unknown site visitors. Unfortunately, it's all too common these days for an unseemly website to run robot scripts that scan the Internet looking for websites that allow site visitors to post comments without requiring a login. These robots then post advertisements as comments in the website, and these advertisements more often than not contain a hypertext link to a rogue website.

The `htmlspecialchars()` and `strip_tags()` functions can help block that silliness. They detect any HTML code embedded within a string value and either remove them completely (the `strip_tags()` function) or convert the greater-than and less-than symbols in the tag into the HTML `&gt;` and `&lt;` codes (the `htmlspecialchars()` function). This helps prevent your site visitors from accidentally clicking rogue hypertext links embedded within posts!

The `nl2br()` function comes in handy if your web application processes text files to display on the web page. If the text file contains new-line characters, those won't display on the web page, which may alter the layout of the text. If you pass the data through the `nl2br()` function, it converts any new-line characters in the text to HTML5 `<br>` tags, preserving the text layout on the web page.

Yet another useful string manipulation function you don't often see in other programming languages is the `addslashes()` function. This function is useful when you need to push data submitted by site visitors into a SQL database. It escapes any single or double quotes embedded within the string value, so that they don't conflict with any quotes needed to embed the string into a SQL statement to submit to the database. This little function can save you lots of trouble with handling data for your database!

Splitting strings

Another common function in string manipulation is the ability to split strings into separate substrings. This comes in handy when you're trying to parse string values to look for words. Table 3-3 shows the PHP string splitting functions that are available.

TABLE 3-3 PHP String Splitting Functions

Function	Description
`chunk_split`	Splits a string value into smaller parts of a specified length.
`explode`	Splits a string value into an array based on one or more delimiter characters.
`implode`	Joins array elements into a single string value.
`str_getcsv`	Parses a comma-delimited string into an array.
`str_split`	Splits a string into an array based on a specified length.

The `str_getcsv()` function is extremely useful when you need to parse comma-separated data entered by site visitors, such as search terms. Follow these steps to see a demonstration of how this works:

1. **Open your editor and type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>String Parsing Test</title>
<style>
   input {
      margin: 5px;
   }
</style>
</head>
<body>
<h2>String parse test</h2>
<form action="parseoutput.php" method="post">
<p>Enter a list of search words, separated with commas</p>
<input type="text" name="search" size="40"><br>
<input type="submit" value="Search">
</form>
</body>
</html>
```

2. **Save the file as `parseinput.html` in the `DocumentRoot` folder for your web server.**
3. **Open a new tab or window in your editor and type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>String Parse Test Results</title>
</head>
</body>
<h1>Search word results</h1>
<?php
$search = $_POST['search'];
$words = str_getcsv($search);
```

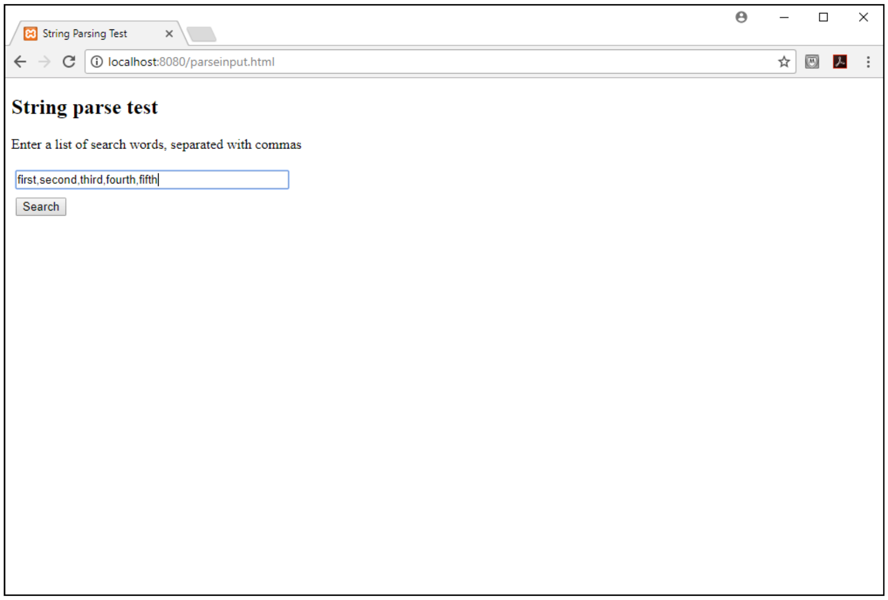
```
foreach ($words as $word) {
   $term = trim($word);
   echo "<p>Search term: '$term'</p>
\n";
}
?>
</body>
</html>
```

4. **Save the file as `parseoutput.php` in the `DocumentRoot` folder for your web server.**
5. **Ensure that your web server is still running, and then open your browser and enter the following URL:**

```
http://localhost:8080/parseinput.html
```

6. **In the text box, type a comma-separated list of words, and then click the Submit button.**
7. **Observe the results in the `parseoutput.php` page.**
8. **Close your browser window when you're done.**

The `parseinput.html` file creates a simple HTML form that contains a single text box for you to enter search words, as shown in Figure 3-2.



String Parsing Test

localhost:8080/parseinput.html

String parse test

Enter a list of search words, separated with commas

first,second,third,fourth,fifth

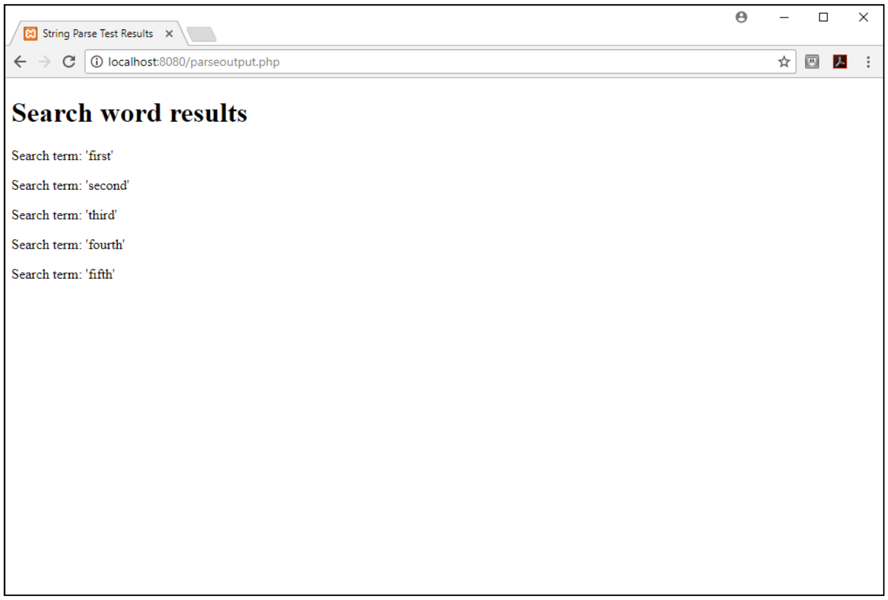
Search

FIGURE 3-2: The web page generated by the `parseinput.php` code.

Type a comma-separated list of words in the text box, and then click the Search button to send them to the `parseoutput.php` file. The `parseoutput.php` code retrieves the list of words using the standard `$_POST[]` array variable:

```
$search = $_POST['search'];
```

Then it uses the `str_getcsv()` function to parse the string and split the words into an array variable. It then uses the `foreach` statement to display the individual words in the web page, as shown in Figure 3-3.



Search word results

Search term: 'first'

Search term: 'second'

Search term: 'third'

Search term: 'fourth'

Search term: 'fifth'

FIGURE 3-3: The web page result from the `parseoutput.php` code.

The `trim()` function is used to remove any extra spaces or tab characters that may have been added between the search terms in the form. These are handy little functions to have in your toolbox as you code your web applications!

Testing string values

A vital function in string manipulation is the ability to test string values for specific conditions. PHP provides several string-testing functions that help with that, as shown in Table 3-4.

TABLE 3-4 PHP String-Testing Functions

Function	Description
`is_bool`	Returns a TRUE value if the string is a valid Boolean value.
`is_float`	Returns a TRUE value if the string is a valid float value.
`is_int`	Returns a TRUE value if the string is a valid integer value.
`is_null`	Returns a TRUE value if the string is a NULL value.
`is_numeric`	Returns a TRUE value if the string is a valid number or numeric string.
`str_word_count`	Returns the number of words in a string or an array of words.
`strcasecmp`	Performs a case-insensitive string comparison.
`strcmp`	Compares the binary values of two string values.
`strlen`	Returns the number of characters in a string.
`strncmp`	Compares the first *n* characters of two string values.

The string-testing functions provide quite a bit of information about the data you receive from your site visitors, as well as performing simple string comparisons to check data. The `strcmp()` function is crucial in evaluating data entered into forms in response to questions in your web applications.

TIP

The `is_numeric()` function is handy to use when testing data submitted in HTML5 forms from unknown site visitors to ensure a numeric value was submitted.

Searching strings

Yet another common string function is searching for a specific value within a string. If you just need to know if a substring value is contained within a string value, use the `strpos()` function. Here's the format of the `strpos()` function:

```
strpos(largestring, substring);
```

PHP will look for the string *`substring`* within the *`largestring`* string value. It returns the position where the substring is found inside the *`largestring`* (with position 0 being the first character of the string). If the substring is not found, it returns a `FALSE` value. Be careful though, because position 0 returns a numeric 0, which is different from a `FALSE` value! To properly test for the difference you must use the === comparison operator, which compares both the value and the data type.

REGULAR EXPRESSIONS

Besides a simple string search, PHP supports more complex *regular expression* string searches. Regular expressions allow you to define a template to compare against the string value. If the string matches the template, it passes the regular expression test.

In the past, PHP supported two different types of regular expressions formats:

- Perl Compatible Regular Expressions (PCRE)
- POSIX Extended Regular Expressions

However, since PHP version 7, support for POSIX regular expressions has been dropped; only the PCRE regular expression format is supported today.

Using regular expressions to search for data is a powerful tool, but also a very complex tool. Entire books and websites have been devoted to explaining all the complexities of regular expression searching. In a nutshell, the key to regular expressions is defining a template that can filter out just the data you want. The template defines what character(s) to look for in a string, and you can even define in what positions the characters should appear within the string if needed. PHP matches the string value against the template, and if it matches, it returns a `TRUE` value. Check out the PHP online manual section on the PCRE regular expressions (`www.php.net/manual/en/book.pcre.php`) for more information on using regular expressions in your PHP code.

Math Functions

Chapter 1 in this minibook shows the basic arithmetic operators that PHP supports. However, there are lots more advanced mathematical features that are available in the PHP extensions! This section discusses the different math functions you can add to your web applications to help save you from having to create complex code for your calculations.

Number theory

Number theory functions provide handy mathematical features, such as finding the absolute value, square root, or factorial of a number. PHP has lots of different number theory functions built in and ready for you to use in your calculations. Table 3-5 lists some of the more common ones you'll use.

TABLE 3-5 PHP Number Functions

Function	Description
abs	Returns the absolute value of a number.
ceil	Rounds a value up to the next largest integer.
floor	Rounds a value down to the next lowest integer.
fmod	Returns the floating point remainder of the division.
intdiv	Performs an integer division.
is_finite	Returns TRUE if the value is a finite number.
is_infinite	Returns TRUE if the value is infinite.
is_nan	Returns TRUE if the value is not a proper float value.
max	Returns the largest value in an array.
min	Returns the smallest value in an array.
pi	Returns a float approximation of pi.
rand	Returns a random number.
sqrt	Returns the square root of a value.

The `rand()` function is handy when you need to generate random numbers for applications (such as guessing games). Without any parameters, the `rand()` function returns a random integer value between 0 and the maximum integer value supported by the server (you can determine that using the `getrandmax()` function). If you need a value from a smaller range, you can specify the min and max range as parameters. The range values are inclusive, so if you specify the following, the `rand()` function will return a random number from 1 to 10:

```
$number = rand(1, 10);
```

WARNING

Despite what you might think from its name, the `is_nan()` function does not work to test input provided by site visitors to determine if the value is a number. The `is_nan()` function only works for float values to determine if the float is in the correct floating point notation. Use the `is_numeric()` string function instead.

Calculating logs and exponents

PHP supports several logarithmic functions that can help with some of your more complex mathematical operations. Table 3-6 shows what tools you have available for that.

TABLE 3-6 **PHP Logarithmic Functions**

Function	Description
`exp`	Calculates the exponent of *e*.
`expm1`	Calculates the exponent of *e* minus 1.
`log`	Performs a standard natural logarithm.
`log10`	Performs a base-10 logarithm.
`log1p`	Calculates a log(1 + number).
`pow`	Calculate the base raised to a power.

Since version 5.6, PHP has included the `**` operator to perform exponentiation as well as the `pow()` function. You can use either one in your mathematical calculations to get the same result.

Working the angles

If trigonometry is your thing, you'll be glad to know that PHP includes all the standard trig functions in the math extension. These are shown in Table 3-7.

TABLE 3-7 **PHP Trigonometric Functions**

Function	Description
`acos`	Calculates the arc cosine.
`asin`	Calculates the arc sine.
`atan`	Calculates the arc tangent.
`cos`	Calculates the cosine.
`deg2rad`	Returns the radian value of a degree.
`hypot`	Calculates the length of the hypotenuse of a right triangle.
`rad2deg`	Returns the degree value of a radian.
`sin`	Calculates the sine.
`tan`	Calculates the tangent.

All the PHP trig functions require that you specify the angle values in radians instead of degrees. If your application is working with degree units, you'll need to use the `deg2rad()` function to convert the values to radians before using them in your calculations.

Hyperbolic functions

Somewhat related to trigonometric functions are the hyperbolic functions. Whereas trigonometric functions are derived from circular calculations, hyperbolic functions are derived from a hyperbola calculation. Table 3-8 shows the hyperbolic functions that PHP supports.

TABLE 3-8 PHP Hyperbolic Functions

Function	Description
`acosh`	Returns the inverse hyperbolic cosine.
`asinh`	Returns the inverse hyperbolic sine.
`atanh`	Returns the inverse hyperbolic tangent.
`cosh`	Returns the hyperbolic cosine.
`sinh`	Returns the hyperbolic sine.
`tanh`	Returns the hyperbolic tangent.

Just as with the trigonometric functions, you must specify the hyperbolic function values in radian units instead of degrees.

Tracking statistics

The PHP statistics extension contains functions commonly used for statistical calculations. It uses the open-source library of C routines for Cumulative Distributions Functions, Inverses, and Other parameters (DCDFLIB) created by Barry Brown and James Lavato.

The library contains about 70 functions for calculating statistical values from beta, chi-square, *f*, gamma, Laplace, logistic, normal, Poisson, *t*, and Weinbull distributions. If you understand any of those things, this is the extension for you! Check out the available statistical functions in the PHP online manual at `www.php.net/manual/en/ref.stats.php`.

Date and Time Functions

Working with times and dates in web applications can be a tricky thing. If your application needs to perform date arithmetic (such as calculating when 60 days is from now), PHP has some useful functions for you! This section first walks through just how PHP handles time and dates, then shows you some functions that can help with your date calculations.

Generating dates

PHP provides the `date()` function for generating human-readable dates and times. The `date()` function takes either one or two parameters:

```
date(format [, timestamp])
```

The *format* parameter is required. It specifies how you want PHP to display the date and/or time values. The *timestamp* parameter is optional. It represents the date and time you want to display as an integer timestamp value. The timestamp value represents the date and time as the number of seconds since midnight, January 1, 1970 (it's an old Unix standard). If you omit the timestamp value, PHP assumes the current date and time.

The *format* is a string value that uses a complicated code to indicate how you want the time and date to appear in the output. Table 3-9 shows the format codes that are available.

TABLE 3-9 The PHP date() Function Format Codes

Code	Description	Example
a	Morning or evening as am or pm	am
A	Morning or evening as AM or PM	AM
B	The Swatch international time format	952 (for 9:52 pm)
c	The date in ISO 8601 format	2018-05-15T22:51:52+01:00
d	The day of the month as a two-digit value with leading zero if necessary	15
D	The day of the week as a three-letter abbreviation	Mon
e	Time zone identifier	America/New_York
F	The month of the year in full text	January

(continued)

PHP Libraries

TABLE 3-9 ***(continued)***

Code	Description	Example
`g`	The hour of the day in 12-hour format	4
`G`	The hour of the day in 24-hour format	16
`h`	The hour of the day in 12-hour format with leading zero	04
`H`	The hour of the day in 24-hour format with leading zero	16
`i`	Minutes past the hour with leading zero	05
`I`	Whether the time zone is using daylight saving time	0 (for not using daylight savings time)
`j`	The day of the month as a number without leading zeroes	5
`l`	The day of the week in full text	Monday
`L`	Whether the year is a leap year	0 (for non-leap years)
`m`	The month of the year as a two-digit number with leading zero	01
`M`	The month of the year as a three-letter abbreviation	Jan
`n`	The month of the year as a number without leading zero	1
`o`	The year in ISO 8601 format	2018
`O`	The difference between the current time zone and GMT	-0500
`r`	The date and time in RFC822 format	Mon, 15 Jan 2018 22:56:35 +0100
`s`	Seconds past the minute in two-digit format with leading zero	05
`S`	Ordinal suffix of the date in two-letter format	th (for 15)
`t`	The total number of days in the date's month	31
`T`	The time zone setting of the server	EST
`U`	The date and time in Unix timestamp format	1516053508
`w`	The day of the week as a single digit	1
`W`	The week number in the year	03
`y`	The year in two-digit format with leading zero	18
`Y`	The year in four-digit format	2018
`z`	The day of the year as a number	78
`Z`	Offset for the current time zone in seconds	-18000

As you can see from the list of codes in Table 3-9, the `date()` function output is very flexible! For example, if you use the following format:

```
$today = date("l, F jS, Y");
```

The `$today` variable would display the current date as:

```
Thursday, January 4th, 2018
```

Or if you prefer, you can just use:

```
$today = date("m/d/Y");
```

To display the date as:

```
01/04/2018
```

With the `date()` function codes, you can display the date and time in any format you need!

WARNING

Displaying dates in a website that can be seen internationally can be somewhat tricky. Keep in mind the differences in how the United States represents dates and how the rest of the world represents dates. To accommodate that difference, the Organization of International Standards (ISO) has created the ISO 8601 standard for displaying dates. It follows none of the common date formats, but instead, uses its own style: 2018-01-04, which represents January 4, 2018. Later in the book, when you work with dates in the MySQL server, you need to use this format to store dates in the database.

Using timestamps

The second parameter of the `date()` function allows you to specify a different date/time to display using a timestamp value. The problem, though, is that you most likely don't know what the timestamp value for a date is! No worries — you have the handy `strtotime()` function to help you out.

The `strtotime()` function converts a date/time string value in just about any format into a timestamp value. For example, if you want to find out what day of the week the Fourth of July is in the year 2020, just use the following code:

```
$timestamp = strtotime("07/04/2020");
$holiday = date("l", $timestamp);
```

The `strtotime()` function returns the value 1593820800, which is the timestamp representation for midnight on that day. You then use that as the second parameter in the `date()` function, and use the `l` (lowercase letter *L*) code format for the output. The output will be the day of the week, Saturday.

WARNING

There is a looming problem with using timestamp values in your PHP code. Because the timestamp format is an old format, to remain backward-compatible, systems store the value as a 32-bit integer data type. As you can guess, at some point in the future, that value will overflow the storage capability of the integer data type. That date happens to be January 14, 2038. If your application needs to work with dates past then, you have to use some other way to handle dates.

Calculating dates

You have a couple of different ways to handle date calculations at your disposal in PHP. One method is to work with timestamp values. If you know the timestamp for the current date/time, you can add the number of seconds needed to represent another date/time.

For example, to calculate the time ten minutes from now, you'd use the following code:

```
$start = strtotime("07/04/2020 10:00:00");
$end = $start + (60 * 10);
$duedate = $date("H:i:s", $end);
```

The first line returns the timestamp value for the start date. The second line adds the number of seconds for 10 minutes (60 seconds × 10 minutes) to the date timestamp. Finally, the third line returns the resulting time.

With timestamp values, you can perform all types of calculations, adding and subtracting values from any start point. Just remember that you're working with seconds, so you need to convert the values into the appropriate timestamp values, and add or subtract the appropriate number of seconds.

The other method for performing date calculations is to use the `strtotime()` function itself. The `strtotime()` function is extremely versatile and can recognize all sorts of common date representations. For example, if you want to find out yesterday's date, you use the following:

```
$yesterday = strtotime("yesterday");
```

And the `strtotime()` function will return the timestamp value for yesterday! You can also use some basic calendar math:

```
$duedate = strtotime("today + 120 days");
```

PHP will calculate that for you automatically! That saves you from having to do the calculations yourself using timestamp values.

Image-Handling Functions

These days, it's a common requirement to work with images in your web pages. Whether vacation pictures on a blog or an online catalog of products, images have become a crucial part of most web applications.

PHP doesn't disappoint here. The php_gd2 extension is a complete graphical manipulation library for processing images directly in your PHP applications. Instead of having to rely on an external image manipulation program such as Photoshop or GIMP, you can edit images directly in your application as you or your site visitors upload them!

Not only can you manipulate uploaded images, but the php_gd2 extension also has functions that allow you to create new images on the fly in your PHP code! To create a new image, use the `imagecreatetruecolor()` function. This function takes two parameters: the width and height of the new image, specified in pixels. It returns a resource variable value that you use to reference the new image as you add components to the image.

For example, to create a new image that is 80 pixels wide by 60 pixels high, use this code:

```
$myimage = imagecreatetruecolor(80, 60);
```

After creating the new image, you'll probably want to draw something in it. First, you must allocate colors to use for the background and foreground objects:

```
$bg = imagecolorallocate($myimage, 255, 255, 255);
$fg = imagecolorallocate($myimage, 0, 0, 0);
```

The `imagecolorallocate()` function takes four parameters. The first parameter is the image resource value returned when you create the image. The next three parameters are the color, defined by the RGB value, just as you do with CSS style colors. The value 255, 255, 255 represents white, while the 0, 0, 0 value represents black.

After allocating the colors you need, you're ready to start drawing on your canvas. Table 3-10 covers the functions you have available for drawing lines, shapes, and even text.

TABLE 3-10 **The GD2 Library Drawing Functions**

Function	Description
`imageline`	Draws a line between two specified points, using a defined color.
`imagechar`	Draws an alphanumeric character using a specified font, color, and location.
`imagerectangle`	Draws a rectangle outline between four points using a defined color.
`imagefilledrectangle`	Draws a solid rectangle between four points using a defined color.
`imagestring`	Draws a string of characters using a specified font, color, and location.

So, to create a new image file with the words *Test Image*, you'd use this code:

```
$image = imagecreatetruecolor(80, 60);
$bc = imagecolorallocate($image, 255, 255, 255);
$fc = imagecolorallocate($image, 0, 0, 0);
imagefilledrectangle($image, 0, 0, 80, 60, $bc);
imagestring($image, 5, 20, 5, "Test", $fc);
imagestring($image, 5, 10, 20, "Image", $fc);
imagejpeg($image, "myimage.jpg");
imagedestroy($image);
```

You should recognize most of these image functions. The `imagestring()` function defines a font size followed by the X and Y coordinates of where to start the string, followed by the string, followed by the color.

The `imagejpeg()` function converts the referenced image object in memory to either an image on the web page or saves it to a file. I specified a filename to save the image to. The `imagedestroy()` function removes the image from memory to free up space. This is especially necessary when working with large images.

One of the biggest problems I often run into when using images in web applications is that they're too big to fit nicely in the spaces I allocate on the web page. If you run a web application that allows site visitors to upload their own images for posting, you never know quite what to expect. Some visitors upload tiny picture files, while others upload mega-sized images. The trick to a good web page is to standardize all the images to make them fit nicely on the web page.

Sure, you can do that by manually downloading all the images, opening them in Photoshop, resizing them, and then uploading the new images back to the web server. That works, but it's extremely time consuming and awkward. Fortunately, the php_gd2 library has just the tool for you!

The `imagecopyresampled()` function allows you to resample an existing image to a new image. Resampling rebuilds the image pixel by pixel, at a different resolution, using special algorithms to maintain the picture clarity.

By resampling the image, you can make it larger or smaller. The php_gd2 extension library takes care of all the mathematical routines required to do that. Follow these steps to try that out:

1. **Open your editor and enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Image Manipulation Test</title>
<style>
    input {
        margin: 5px;
    }
</style>
</head>
<body>
<h2>Please select an image to upload</h2>
<form action="imageconvert.php" method="post"
      enctype="multipart/form-data">
<input type="file" name="picture"><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

2. **Save the file as `imageupload.html` in the `DocumentRoot` folder for your web server.**
3. **Open a new tab or window in your editor and enter the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Image Manipulation Test</title>
</head>
```

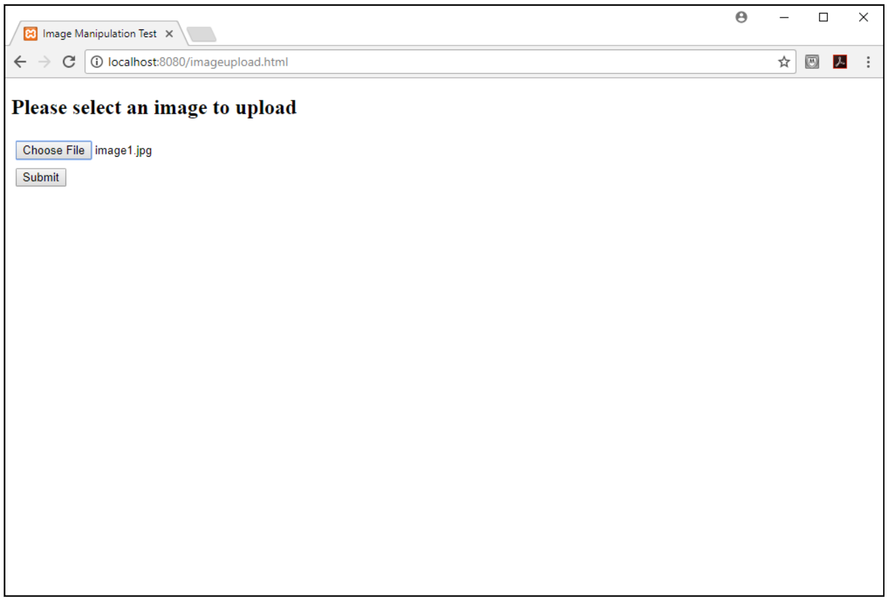
```
<body>
<h1>The uploaded image:</h1>
<?php
$file = $_FILES['picture']['tmp_name'];
$picture = file_get_contents($file);
$sourceImage = imagecreatefromstring($picture);

$width = imageSX($sourceImage);
$height = imageSY($sourceImage);

$newheight = 400;
$newwidth = $newheight * ($width / $height);

$newImage = imagecreatetruecolor($newwidth, $newheight);
$result = imagecopyresampled($newImage, $sourceImage,
          0, 0, 0, 0,
          $newwidth, $newheight, $width, $height);
imagejpeg($newImage, "newimage.jpg");
?>
<img src="newimage.jpg">
</body>
</html>
```

4. **Save the file as `imageconvert.php` in the `DocumentRoot` folder for your web server.**
5. **Have an image file handy that you want to copy and convert to a different sized image.**
6. **Ensure the web server is running, and then open your browser to the URL:**

```
http://localhost:8080/imageupload.html
```

7. **Click the file chooser button for the file upload text box as it appears in your browser, navigate to your image file, select it, and then click the Open button to select the name.**
8. **Click the Submit button to upload the image file for converting.**
9. **You should see the resized image appear on the resulting web page.**
10. **Close your browser and shut down the web server when you're done.**

The `imageupload.html` file creates a simple HMTL5 form using the `file` data input type. The browser will provide a method for you to select a local file to enter into the file input field, as shown in Figure 3-4 for the Chrome browser.



Please select an image to upload

FIGURE 3-4: The output from the `imageupload.html` program.

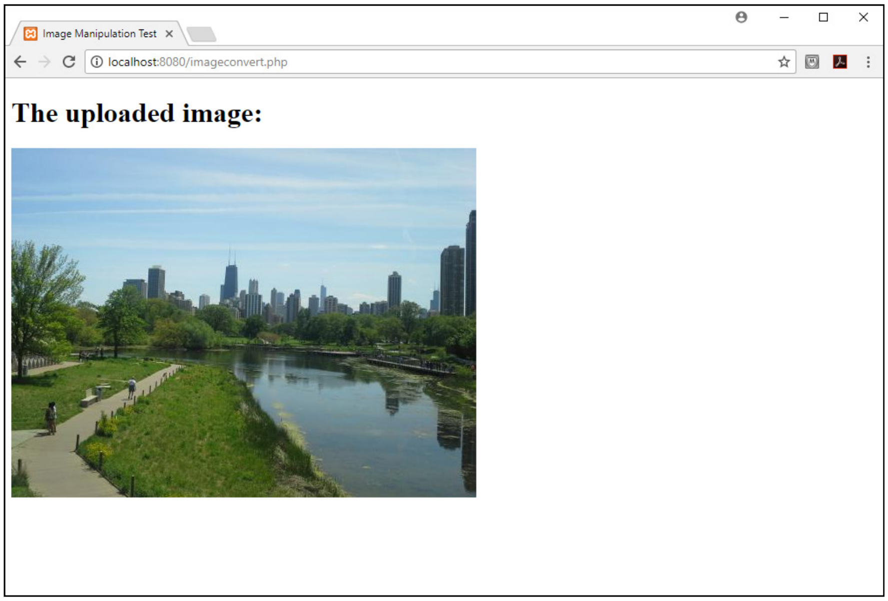
The `imageconvert.php` code retrieves the uploaded image from the PHP server using the special `$_FILES[]` array variable. The `$_FILES[]` array provides information about files uploaded to the server within an HTML5 form. The `tmp_name` array element contains the name of the temporary file the server creates to store the uploaded file.

After retrieving the uploaded file, the code converts it to an editable php_gd2 library object using the `imagecreatefromstring()` function.

Using the uploaded image object, the code calculates the width and height of the original image using the `imageSX()` and `imageSY()` functions. Then with a little bit of algebra, the code sets the new image height to a set height, and calculates the new width required to keep the original aspect ratio of the image. This ensures that all images that appear on the web page use the same height.

With the new width and height values calculated, the code then uses the `imagecopyresampled()` function to copy and resample the original image to the resized image object. The `imagejpeg()` function saves the new image as the file `newimage.jpg` in the `DocumentRoot` folder of the web server. Finally, the code displays the new image on the web page using a standard `<img>` HTML5 tag, as shown in Figure 3-5.

Now you can resize uploaded image files on the fly, without any intervention required on your part!



The uploaded image:

FIGURE 3-5: Displaying the resampled and resized image.

IN THIS CHAPTER

» **Identifying PHP attacks**

» **Stopping cross-site scripts**

» **Hiding your files**

» **Watching for data spoofing**

» **Handling data safely**

Chapter 4 Considering PHP Security

Web application security is a hot topic these days, and for good reason! It seems that almost every day there's a news story about some company being attacked and having important data stolen. These breaches are costly — both for the company and for the thousands of customers who have personal information stolen.

As a web application developer, your job is to put security first in all your design and coding work. You're the front line in the battle of data security! This chapter helps with that job, by giving you an idea of the types of attacks you need to watch out for and then walking you through how to avoid those attacks with your PHP code.

Exploring PHP Vulnerabilities

To avoid attacks, you first need to know where they'll come from. It doesn't do any good to barricade the front door, if you leave the windows wide open. The majority of attacks against your web applications are avoidable by following some basic PHP coding rules.

There are thousands of different ways for an attacker to break into your PHP program, but most of them boil down into four general categories:

- Cross-site scripting
- Data spoofing
- Invalid data
- Unauthorized file access

Each of these attacks has different causes and results, as well as different methods for you to use to block them. The following sections examine each of these attacks in depth.

Cross-site scripting

Cross-site scripting (known as XSS) is quite possibly the most dangerous type of attack made on dynamic web applications. The main idea of an XSS attack is to embed malicious JavaScript code in data that the attacker submits to the web application as part of the normal data input process. When the web application tries to display the data in a client browser, the JavaScript is pushed to the client browser that's viewing the website and runs.

Follow these steps to watch an XSS exploit in action:

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
2. **Type the following code into the editor window:**

```
<!DOCTYPE html>
<html>
<head>
<title>XSS Test</title>
<style>
   input {
        margin: 5px;
    }
</style>
</head>
<body>
<h2>Please enter your first name:</h2>
<form action="xsstest.php" method="post">
<input type="text" name="fname"><br>
```

```
<input type="submit" value="Submit name">
</form>
</body>
</html>
```

3. **Save the file as `xssform.html` in the `DocumentRoot` folder for your web server.**

   For XAMPP on Windows, that's `c:\xampp\htdocs`; for XAMPP on macOS, that's `/Applications/XAMPP/htdocs`.

4. **Open a new tab or window in your browser, and type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>XSS Test</title>
</head>
<body>
<h1>XSS Test</h1>
<?php
    $fname = $_POST['fname'];
   echo "<p>Welcome, $fname</p>\n";
?>
<h2>This is the end of the test</h2>
</body>
</html>
```

5. **Save the file as `xsstest.php` in the `DocumentRoot` folder for your web server.**
6. **Open the XAMPP Control Panel, and then start the Apache web server.**
7. **Open your browser, and enter the following URL:**

```
http://localhost:8080/xssform.html
```

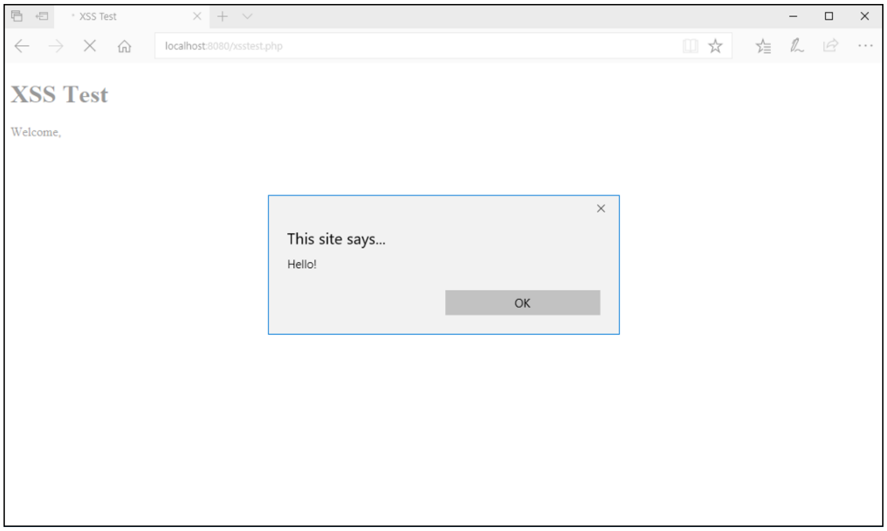
   You may need to change the TCP port used to match your web server.

8. **In the form, type the following code in the Name text box:**

```
<script>alert("Hello!");</script>
```

9. **Click the Submit button to continue.**
10. **Close the browser window when you're done with the test.**

When you submit the form with the embedded JavaScript code, you should get the output as shown in Figure 4-1.



XSS Test

Welcome,

This site says...

Hello!

FIGURE 4-1: The output from entering embedded JavaScript in a form.

The PHP code sent the JavaScript to the browser as part of the `echo` statement output, and the browser dutifully ran the JavaScript code. This example is harmless, because it just displays a simple alert message, but a real attacker would embed much more malicious code.

TIP

Some browsers, such as Safari and Chrome, have built-in XSS attack detection, which may trigger on this test and block the JavaScript code from running. If you don't see the `alert()` pop-up box, open the developer tools for your browser and see if there's a notice that the browser blocked a potential XSS attempt.

The "cross-site" part of the XSS name comes from where the `<script>` tag sends the browser to retrieve the JavaScript code file. In the previous example, I just submitted embedded JavaScript code directly within the script element. ***Remember:*** The `<script>` HTML5 tag can also reference an external JavaScript file, which the browser will load and run. An attacker can specify the `src` attribute in the `<script>` tag to redirect the browser to run JavaScript located on a rogue server anywhere in the world.

There are two different methods of carrying out an XSS attack:

- **Reflected attack:** The attacker places the rogue script as a link in the submitted data. Victims must actively click the link to launch the XSS attack.

» **Persistent attack:** The attacker places the rogue script as data that the browser displays when the web page loads (as in the previous example).

Persistent attacks are very dangerous. The malicious script runs as soon as an unsuspecting website visitor opens the web page that contains the script as part of the content, without any actions required by the victim. For example, if an attacker posts a blog comment that contains malicious JavaScript code, every time the web application displays that blog comment on a client browser, the malicious script is run.

Data spoofing

Our dynamic web applications use all types of data to produce content. All too often, though, we assume the values stored in a particular variable are placed there by our program and are correct. However, that may not always be the case.

Another popular form of attack is *data spoofing* (externally inserting fraudulent data into a PHP program code). The biggest culprit of this attack is the `register_globals` setting in the `php.ini` configuration file for the PHP server (see Book 1, Chapter 2).

The `register_globals` setting was originally intended to make life easier for PHP developers. When that setting is enabled, PHP automatically converts any data passed via the `GET` or `POST` methods into a PHP variable.

For example, let's say you build a form that contains the following input element:

```
<input type="text" name="fname">
```

When the PHP server receives the form data, it automatically creates a PHP variable named `$fname`, and assigns it the value passed from the form data field with that name. This feature certainly makes your coding life easier, but it adds a new problem.

Suppose your application uses an authentication method to validate the administrators of your website. When an administrator logs in, you set a variable indicating that the session is an administrative session and then check that variable whenever the user attempts to do some admin work. The code for that would look something like this:

```
if ($admin == 1) {
   do some admin functions
} else {
   echo "Sorry, you do not have permission";
}
```

The application assumes the `$admin` variable is set to a value of `1` when the user is an authenticated administrator.

Now, consider what would happen if an attacker figured this out and the `register_globals` setting in PHP were enabled. All the attacker would need to do is spoof the `$admin` variable with a phony value. And all that attack requires is to use this URL:

```
http://yourhost.com/index.php?admin=1
```

The `register_globals` setting allows the PHP server to retrieve the value set in the `GET` method, create the variable `$admin`, and set it to a value of `1`. This will then allow the attacker to perform the admin function in the application without having to log in!

WARNING

Newer versions of PHP disable the `register_globals` setting by default, but that setting is still present. It's never a good idea to enable the `register_globals` setting. Just retrieve any data you need using the standard `$_GET[]` and `$_POST[]` array variables. It's worth the effort!

Invalid data

Invalid data comes in all shapes and sizes. Often invalid data is just the result of a site visitor not paying close enough attention to the form fields and entering the wrong data into the wrong field, such as typing a zip code into a city name data field. Other times there may be some malicious intent to the invalid data, such as entering an invalid email address into a contact form on purpose to remain anonymous. It's your job as the application developer to anticipate invalid data and try to prevent it before it becomes a problem in the application.

There are two schools of thought on data validation:

- Client-side data validation
- Server-side data validation

The following sections dig a little deeper into just how these two methods differ.

Client-side data validation

As you can probably guess, *client-side data validation* requires adding some Java-Script code to your web-page form to ensure site visitors enter the proper data into the proper data fields. Book 3, Chapter 4, details how to watch for form events and trigger JavaScript code to check as the site visitor types the data. If any invalid data is entered, the JavaScript can block sending the form data to the server.

WARNING

Don’t rely on JavaScript data validation alone, though. Your website visitors can disable JavaScript in their browsers to get around it!

You can also use the HTML5 data-filtering elements and attributes that limit the range of possible values for a form field — for example, by using the new phone or email input element types instead of just a standard text input element. The browser won’t accept data that doesn’t match the format defined by the filters.

A combination of JavaScript, HTML5, and CSS produces a three-pronged approach to client-side data validation. That combination allows you to monitor the data your site visitors type into the data fields and then change the styles applied to the data field accordingly. A common feature is to use the background color style to indicate invalid data in a data field. When the site visitor enters invalid data, JavaScript changes the data field background color to red.

Server-side data validation

Because the focus of this chapter is PHP, I talk more about *server-side data validation.* Server-side data validation is a little trickier in that you must wait for the site visitor to submit the form before you can validate the data in your PHP code. You can’t detect invalid data in real time, but you do have a few more tools available for validating the data in your PHP code.

When the client browser sends the form data to the server, your PHP code retrieves it from the `$_GET[]` or `$_POST[]` array variables and then can work on determining which data is valid and which is invalid. Usually, there’s a set process that you can undertake to validate data, such as making sure numeric values are really numbers or that text values don’t contain any extraneous characters that shouldn’t be there (such as the semicolon character discussed in the SQL injection sidebar).

One common method used in PHP development is to create an array to contain the “clean” data values retrieved from the table. As the code validates each data field value, that value is placed into the array with the corresponding variable name used as the key. The application doesn’t use any of the data retrieved directly from the form; instead, it only accesses data values from the array of cleaned data values. That ensures that you won’t make any mistakes by accidentally using a data value that hasn’t been validated.

There are a few PHP functions to help out with the data validation process, which I discuss later in this chapter.

SQL INJECTION

Possibly the most dangerous attack involving invalid data is the SQL injection attack. With SQL injection, an attacker embeds a SQL statement inside form field data, hoping that the application will forward the data to a database server without validating it and that the database server will run the SQL statement. The embedded SQL statement usually performs some type of malicious action, such as deleting a table, or at least all the data within the table.

If your application uses a database, it's important to block SQL injection attempts within form data. SQL injections usually involve embedding a semicolon to separate out the SQL statement. Always validate input data looking for embedded semicolons to block these types of attacks.

Unauthorized file access

The PHP code that you write for your web applications may contain lots of privileged information, whether it's database user accounts for accessing a database or admin passwords that it checks to validate admin login attempts. Being able to properly protect your PHP files from unauthorized viewing is a must.

By default, any `.php` files accessed via the web server are passed to the PHP server and processed, so if attackers try to access a `.php` file directly, they only see the output from the file, not the actual code. However, if an attacker manages to break into the `DocumentRoot` folder using some attack, your PHP code will be wide open. Your job as a PHP developer is to try to hide your code from these types of attacks.

One method of doing that is to utilize the `include()` function. Chapter 1 of this minibook covers how to use the `include()` function to access PHP and HTML5 code located in a separate file from within a program file. The `include()` function isn't bound by the web server `DocumentRoot` setting folder location; it can retrieve data from anywhere on the server that it has read access to.

You can leverage that feature by storing all your application PHP code as include files outside the `DocumentRoot` boundaries. Then you only need to place the main `index.php` template file into the `DocumentRoot` folder for site visitors to access.

The main template file defines the different sections of the web page, and calls the appropriate include files for each one:

```
<body>
<header>
<?php include("/secretlocation/header.inc.php"); ?>
</header>
<nav>
<?php include("/secretlocation/navigation.inc.php");?>
</nav>
<main>
<?php
$content = $_GET['content'];
switch ($content) {
    case "initial":
        include("/secretlocation/initial.inc.php");
        break;
    case "registration":
        include("/secretlocation/registration.inc.php");
        break;
    case "query":
        include("/secretlocation/query.inc.php");
        break;
    case "newdata":
        include("/secretlocation/newdata.inc.php");
        break;
    default:
        echo "<p>Sorry, invalid page location</p>\n";
}
?>
</main>
<aside>
<?php include("/secretlocation/aside.inc.php"); ?>
</aside>
<footer>
<?php include("/secretlocation/footer.inc.php"); ?>
</footer>
</body>
```

Each section of the web page uses a separate include file to load the content for the section. A `GET` variable controls what content displays in the main section of each web page. The `content` HTML variable contains the name of the include file to use for each feature of the application. If an attacker tries to set the `content` HTML variable to some other value, an error message displays.

Now all the actual PHP code is safely stored away in include files located outside the `DocumentRoot` folder area of the web server. This method isn't foolproof, but it does provide an extra layer of security for your data.

PHP Vulnerability Solutions

Fortunately, the PHP programming language provides several features that you can utilize to help you avoid all these types of attacks. This section walks through the different tools that you have at your disposal, showing you how best to use them to protect your website data and code.

Sanitizing data

Just like sanitizing your kitchen is a good idea to help protect you from nasty bugs and viruses, sanitizing your PHP data helps render any harmful code injected into the data harmless. The idea is to detect any embedded HTML code and make it harmless by removing the HTML5 tags that trigger actions in the browser. This stops any type of XSS attack dead in its tracks.

The best defense against XSS attacks is to block any types of HTML code from the data your site visitors enter, both as they try to input it and as your application tries to output it. Two functions are good for this:

- `htmlspecialchars()`
- `filter_var()`

The following sections takes a closer look at how to use these functions to help make your web application safer.

Using htmlspecialchars()

The `htmlspecialchars()` function detects HTML5 tags embedded in a data string and converts the greater-than and less-than symbols in the tags to the HTML5 entity codes `&gt;` and `&lt;`. This doesn't remove the tags from the data; instead, it turns them to ordinary text that displays as normal content.

Here's the format for the `htmlspecialchars()` function:

```
htmlspecialchars(string [, flags [,encoding [,double]]])
```

By default, the `htmlspecialchars()` function encodes the following characters that it finds in the data string:

- Ampersand (&)
- Double quote (")

- Single quote (')
- Less than (<)
- Greater than (>)

You can pick and choose which of these items the `htmlspecialchars()` function converts and which ones it allows through by specifying one or more flags. Table 4-1 shows the flags that are available to choose from.

TABLE 4-1 htmlspecialchars Flags

Flag	Description
`ENT_COMPAT`	Converts only double quotes.
`ENT_QUOTES`	Converts both single and double quotes.
`ENT_NOQUOTES`	Doesn't convert either single or double quotes.
`ENT_IGNORE`	Doesn't convert anything.
`ENT_SUBSTITUTE`	Replaces invalid code with Unicode replacement characters instead of returning an empty string.
`ENT_DISALLOWED`	Replaces invalid code with Unicode replacement characters instead of leaving them as is.
`ENT_HTML401`	Handles the code as HTML version 4.01.
`ENT_XML1`	Handles the code as XML version 1.
`ENT_XHTML`	Handles the code as XHTML.
`ENT_HTML5`	Handles the code as HTML5.

The *`encoding`* parameter allows you to define what character set encoding the data uses, and the *`double`* parameter allows PHP to double-encode the data, also looking for HTML5 entity codes embedded in the data and converting them as well.

The best way to get a handle on what `htmlspecialchars()` does is to watch it in action. Follow these steps to test this out:

1. **Open the `xsstest.php` file in your editor, program editor, or IDE package.**
2. **Change the line of code that retrieves the `$_POST['fname']` array variable to make it look like the following:**

```
$fname = htmlspecialchars($_POST['fname']);
```

3. **Save the file as `xsstest.php` in the `DocumentRoot` folder of your web server.**

4. **Ensure that the web server is running and then open your browser and enter the following URL:**

   ```
   http://localhost:8080/xssform.html
   ```

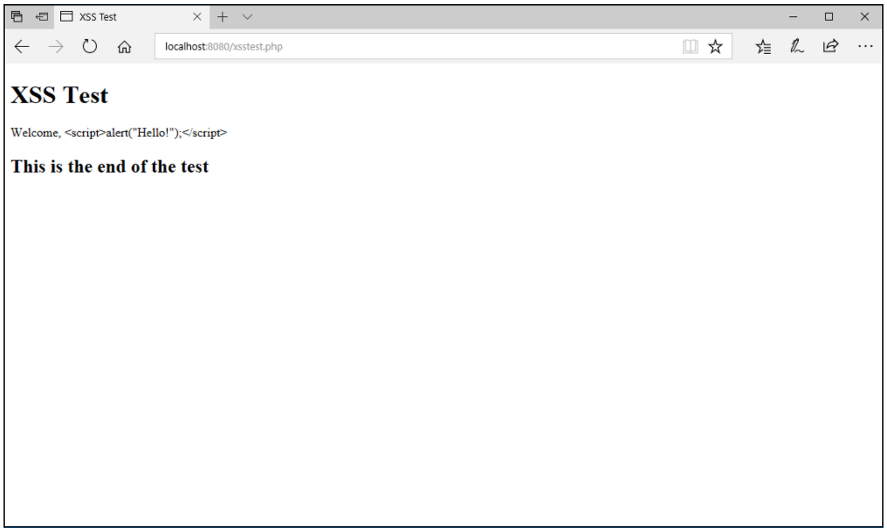
5. **Enter the following text in the text box:**

   ```
   <script>alert("Hello!");</script>
   ```

6. **Click the Submit button to submit the text.**

7. **Observe the output in the `xsstest.php` web page and then close the browser window.**

With the simple addition of the `htmlspecialchars()` function, you should now see the output shown in Figure 4-2.



XSS Test

Welcome, <script>alert("Hello!");</script>

This is the end of the test

FIGURE 4-2: The output from adding the `htmlspecialchars()` function.

The `htmlspecialchars()` function converted the script element tags into plain text and displayed the JavaScript code as regular text in the output. That's not ideal, but it did block the XSS attack from hitting the browser.

Using filter_var()

The `filter_var()` function is the Swiss Army knife of functions for protecting data in your PHP applications. It provides a host of customized filters for finding and sanitizing different types of data that could potentially cause harm in your PHP application.

You control the behavior of the `filter_var()` function by specifying both options and flags as parameters:

```
filter_var(string [, filter] [, flags])
```

The *`filter`* and *`flags`* parameters are optional, but almost always you'll at least specify the filter to use. The *`filter`* defines what class of characters the `filter_var()` function should look for, and the *`flags`* parameter fine-tunes subsets of characters within the `filter` class.

What makes the `filter_var()` function so versatile is that it can both sanitize (remove) and validate (test) string data. Table 4-2 shows the data-sanitizing options that you can use.

TABLE 4-2 The filter_var Data-Sanitizing Options

Option	Description
`FILTER_SANITIZE_EMAIL`	Removes invalid characters from an email address.
`FILTER_SANITIZE_ENCODED`	Encodes a string to make a valid URL.
`FILTER_SANITIZE_MAGIC_QUOTES`	Escapes embedded quotes.
`FILTER_SANITIZE_NUMBER_FLOAT`	Removes all characters except digits and float symbols.
`FILTER_SANITIZE_NUMBER_INT`	Removes all characters except digits and integer symbols.
`FILTER_SANITIZE_SPECIAL_CHARS`	Removes quotes, as well as greater-than, less-than, and ampersand characters.
`FILTER_SANITIZE_FULL_SPECIAL_CHARS`	Converts the greater-than and less-than symbols in HTML5 tags to entity codes (the same as `htmlspecialchars()`).
`FILTER_SANITIZE_STRING`	Removes all HTML5 tags.
`FILTER_SANITIZE_STRIPPED`	Removes all HTML5 tags.
`FILTER_SANITIZE_URL`	Removes all invalid URL characters.
`FILTER_UNSAFE_RAW`	Does nothing, the default action.

The filter_var() function allows you to customize just what data gets sanitized from the input data and what data is allowed to pass through. Follow these steps to test this out:

1. **Open the xsstest.php file in your editor.**
2. **Change the line that assigns the $fname variable to this:**

```
$fname = filter_var($_POST['fname'], FILTER_SANITIZE_STRING);
```

3. **Save the file as xsstest.php in the DocumentRoot folder for your web server.**
4. **Ensure that your web server is running, and then open your browser and type the following URL:**

```
http://localhost:8080/xsstest.php
```

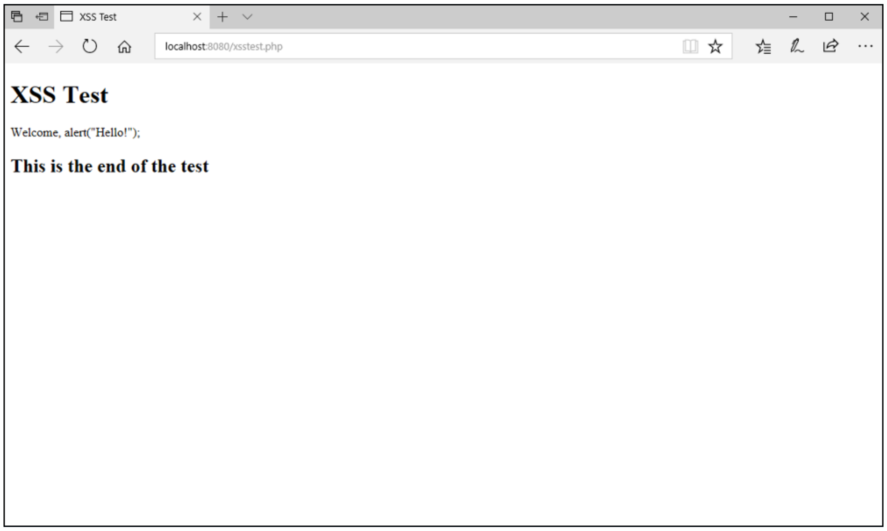
5. **Enter the following text into the text box:**

```
http://localhost:8080/xssform.html
```

6. **Click the Submit button.**
7. **Observe the output from the xsstest.php program and then close the browser window.**

The filter_var() function not only disables the script element in the text, but also completely removes the opening and closing tags, as shown in Figure 4-3.

The embedded JavaScript code is still visible, but at least the <script> tags are completely removed from the data, rendering the attack useless.



XSS Test

Welcome, alert("Hello!");

This is the end of the test

FIGURE 4-3: The output from adding the filter_var() function.

TIP

The `filter_var()` function is also a great way to extract numeric data from a string, using the `FILTER_SANITIZE_NUMBER_INT` option.

Validating data

Detecting all types of invalid data can be impossible, but PHP provides a few ways for you to at least detect some types of invalid data to help make things at least a little bit easier. This section describes the PHP functions available for helping detect when a site visitor has attempted to input invalid data into a form data field.

Validating data types

One primary goal for catching invalid data is to at least determine that the input data is the correct data type. PHP provides a series of functions to do that (see Table 4-3).

TABLE 4-3 PHP Data Validation Functions

Function	Description
`is_bool()`	Returns `TRUE` if the value is a Boolean data type.
`is_float()`	Returns `TRUE` if the value is in valid float format.
`is_int()`	Returns `TRUE` if the value is an integer value.
`is_null()`	Returns `TRUE` if the value is `NULL`.
`is_numeric()`	Returns `TRUE` if the value is in a valid numeric format.
`is_string()`	Returns `TRUE` if the value is a string as opposed to a number.

Of these, the `is_numeric()` function is the most useful. It comes in handy to validate simple numeric data that your site visitors enter into forms, such as ages or quantities.

To test this out, follow these steps:

1. **Open your editor and type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Data Type Test</title>
<style>
    input {
        margin: 5px;
    }
</style>
</head>
<body>
<h1>Please enter data into the form fields</h1>
<form action="typetest.php" method="post">
<label>Last Name</label>
<input type="text" name="name"><br>
<label>Email address</label>
<input type="text" name="email"><br>
<label>Age</label>
<input type="text" name="age"><br>
<input type="submit" value="Submit form">
</form>
</body>
</html>
```

2. **Save the file as `typetest.html` in the `DocumentRoot` folder for your web server.**

3. **Open a new tab or window in your editor, and type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Data Type Test</title>
<style>
    .warning {
        color:red;
    }
</style>
</head>
<body>
<h1>Form results:</h1>
<?php
$name = htmlspecialchars($_POST['name']);
$email = htmlspecialchars($_POST['email']);
$age = htmlspecialchars($_POST['age']);

echo "<p>Name: $name</p>\n";
echo "<p>Email: $email</p>\n";
if (is_numeric($age)) {
   echo "<p>Age: $age</p>\n";
} else {
   echo "<p class='warning'>Please enter a valid age</p>\n";
}
?>
<br>
<a href="typetest.html">Return to form</a>
</body>
</html>
```

4. **Save the file as** `typetest.php` **in the** `DocumentRoot` **folder for your web server.**
5. **Ensure that the web server is running and then open your browser and enter the following URL:**

```
http://localhost:8080/typetest.html
```

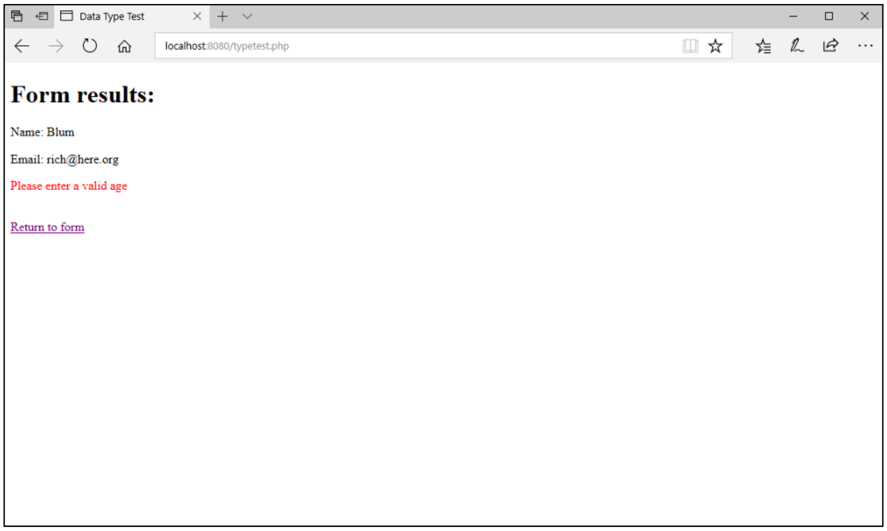
6. **Enter your name and numeric age into the form and click the Submit button.**
7. **Observe the results on the** `typetest.php` **page output and then click the Return to Form link.**
8. **This time, enter your name and a text value for the age and then click the Submit button.**
9. **Observe the results in the** `typetest.php` **page output.**
10. **Close your browser window when you're done.**

In this example, the `is_numeric()` function detects when the site visitor enters an invalid value for the age and displays a warning message, as shown in Figure 4-4.

The `is_numeric()` function can't stop site visitors from lying about their ages, but at least it can prevent someone from entering text into the age data field.

Validating data format

Testing for valid data types is fine when you're working with numeric values, but it doesn't help all that much for text values such as names, home addresses, and email addresses. The `is_string()` function can tell you that the value is a valid string value, but not the format of the data contained within the string.



Form results:

Name: Blum

Email: rich@here.org

Please enter a valid age

Return to form

FIGURE 4-4: The result from entering an invalid age value into the `typetest.html` form.

This is another time where the `filter_var()` function can come in handy. Not only can the `filter_var()` function sanitize data, but it can also validate data formats for us! Table 4-4 shows the data validation options that are available for the `filter_var()` function.

TABLE 4-4 The filter_var() Data Validation Options

Option	Description
`FILTER_VALIDATE_BOOLEAN`	Returns TRUE if the value is a valid Boolean value.
`FILTER_VALIDATE_EMAIL`	Returns TRUE if the value is in a valid email address format.
`FILTER_VALIDATE_FLOAT`	Returns TRUE if the value is in a valid float format.
`FILTER_VALIDATE_INT`	Returns TRUE if the value is in a valid integer format.
`FILTER_VALIDATE_IP`	Returns TRUE if the value is in a valid IP address format.
`FILTER_VALIDATE_MAC`	Returns TRUE if the value is in a valid MAC address format.
`FILTER_VALIDATE_REGEXP`	Returns TRUE if the value matches the specified regular expression.
`FILTER_VALIDATE_URL`	Returns TRUE if the value is in a valid URL format.

The email address check in `filter_vars()` comes in handy when you need to validate email addresses entered into contact forms. Follow these steps to test that out:

1. **Open the `typetest.php` file in your editor.**
2. **Modify the code so that it looks like the following:**

```
<!DOCTYPE html>
<html>
<head>
<title>Data Type Test</title>
<style>
    .warning {
        color:red;
    }
</style>
</head>
<body>
<h1>Form results:</h1>
<?php
$name = htmlspecialchars($_POST['name']);
$emal = htmlspecialchars($_POST['email']);
$age = htmlspecialchars($_POST['age']);

echo "<p>Name: $name</p>\n";
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "<p>Email: $email</p>\n";
} else {
    echo "<p class='warning'>Please enter a valid
            email address</p>\n";
}
if (is_numeric($age)) {
   echo "<p>Age: $age</p>\n";
} else {
   echo "<p class='warning'>Please enter a valid
           age</p>\n";
}
?>
<br>
<a href="typetest.html">Return to form</a>
</body>
</html>
```

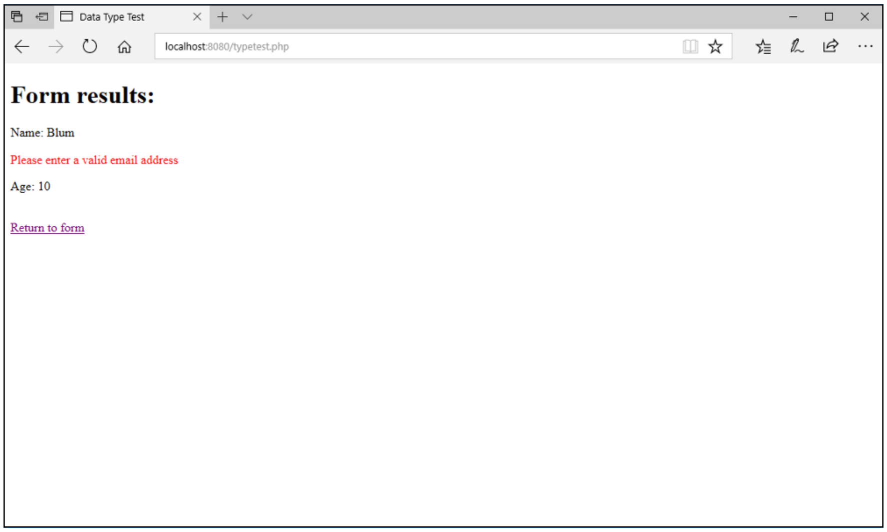
3. **Save the file as `typetest.php` in the `DocumentRoot` folder of your web server.**

4. **Ensure that the web server is running and then open your browser and enter the following URL:**

```
http://localhost:8080/typetest.html
```

5. **Enter a valid name and age, but enter an email address in an invalid format.**
6. **Click the Submit button to submit the form data.**
7. **Observe the output.**
8. **Click the link to return to the form and try out different email address formats to see what gets caught by the data validation and what doesn't.**
9. **Close your browser window when you're done.**

The added `filter_var()` validation check looks for the email address to be in the proper format of `name@hostname`. If it is, the `filter_var()` function returns a `TRUE` value, which triggers the `if...else` statement to display the data. If it isn't, the `else` code block triggers and displays a warning, as shown in Figure 4-5.



Form results:

Name: Blum

Please enter a valid email address

Age: 10

Return to form

FIGURE 4-5: The result from entering an invalid email address.

Again, this check is not foolproof — it can only check the format of an email address. It doesn't test the account to make sure it's a live account. But at least this is a start!

IN THIS CHAPTER

» Defining object-oriented programming

» Creating objects

» Using objects

» Customizing objects

Chapter 5 Object-Oriented PHP Programming

So far, all the PHP scripts presented in this minibook have followed the procedural style of programming. With procedural programming, you create variables and functions within your code to perform certain procedures, such as storing values in variables, and then checking them with conditional statements. The data you use and the functions you create are completely separate entities, with no specific relationship to one another. With object-oriented programming, on the other hand, variables and functions are grouped into common objects that you can use in any program. In this chapter, you learn what object-oriented programming is and how to use it in your web applications.

Understanding the Basics of Object-Oriented Programming

Before you can start working on object-oriented programming (OOP), you need to know how it works. OOP uses a completely different paradigm from coding than what I cover earlier in this minibook. OOP requires that you think differently about how your programs work and how you code them.

With OOP, everything is related to objects. (I guess that's why they call it object-oriented programming!) Objects are the data you use in your applications, grouped together into a single entity.

For example, if you're writing a program that uses cars, you can create a `Car` object that contains information on the car's weight, size, color, engine, and number of doors. If you're writing a program that tracks people, you might create a `person` object that contains information about each person's name, date of birth, height, weight, and gender.

OOP uses classes to define objects. A *class* is the written definition in the program code that contains all the characteristics of the object, using variables and functions. The benefit of OOP is that after you create a class for an object, you can use that same class in any other application. Just plug in the class definition code and put it to use!

An OOP class contains *members.* There are two types of members:

- **Properties:** Class *properties* (also called *attributes*) denote features of the object, such as the car's weight or the person's name. A class can contain many properties, with each property describing a different feature of the object.
- **Methods:** Class *methods* are similar to the standard PHP functions that you've been using. A method performs an operation using the properties in a class. For instance, you could create class methods to retrieve a specific person from a database, or change the address property for an existing person. Each method should be contained within the class and perform operations only in that class. The methods for one class shouldn't deal with properties in other classes.

Defining a class

Defining a class in PHP isn't too different from defining a function. To define a new class, you use the `class` keyword, along with the name of the class, followed by any statements contained in the class.

Here's an example of a simple class definition:

```
class Product {
    public $description;
    public $price;
    public $inventory;
    public $onsale;
    public function buyProduct($amount) {
        $this->inventory -= $amount;
    }
}
```

TIP

The class name you choose must be unique within your program. Class names follow the same rules as PHP variable names. Although it's not required, programmers often start class names with an uppercase letter to help distinguish them in program code.

This example defines four property members and one method member. Each member is defined using one of three *visibility* classifications. The visibility of the member determines where you can use or reference that member. There are three visibility keywords used in PHP:

- `public`: The member can be accessed from outside the class code.
- `private`: The member can only be accessed from inside the class code.
- `protected`: The member can only be accessed from a child class. (I talk about that a little later in the "Extending Classes" section.)

The `Product` class example declares all the members to be public, so you can reference them anywhere in your PHP code.

The `buyProduct()` method uses an odd variable name in the function:

```
$this->inventory
```

The `$this` variable is a special identifier that references the current object of the class. In this example, it points to the `$inventory` property of the object. Notice the removal of the dollar sign from the `inventory` property when referencing it this way. This helps PHP know that you're referencing the `$inventory` property from within the class object and not the class itself.

This code defines the makeup of the class, but it doesn't actually do anything with it. The next section shows you how to actually use your class template to create objects.

Creating an object instance

To use a class, you have to *instantiate* it. When you instantiate a class, you create what's called an *instance* of the class in your program. Each instance represents one occurrence of the object within the program. To instantiate an object in PHP code, you use the following format:

```
$prod1 = new Product();
```

This creates the object called `$prod1` using the `Product` class. When you instantiate an object, you can access the public members of that class directly from your program code:

```
$prod1->description = "carrot";
$prod1->price = 1.50;
$prod1->inventory = 10;
$prod1->onsale = false;
```

This code sets values for each of the properties for the object. Notice the `->` symbol in use again. It tells PHP that you're referencing the properties and methods specifically for the `$prod1` object.

The `$prod1` variable now contains these values set for the object properties, and you can use it anywhere in your PHP code to reference the properties. The same applies when you need to use a public method of an object:

```
$prod1->buyProduct(4);
```

This calls the `buyProduct()` method for the class object, passing the value of `4`. Because the `buyProduct()` method alters the `$inventory` property of the object, the next time you reference the `$prod1->inventory` property in your code, it'll have the value of 6.

You can instantiate as many instances of a class as you need within your program. Just make sure that each instance uses a different variable name:

```
$prod2 = new Product();
$prod2->description = "eggplant";
$prod2->price = 2.00;
$prod2->inventory = 5;
$prod2->onsale = true;
```

PHP will keep the two instances of the `Product` class completely separate, maintaining the property values for each one.

Follow these steps to test out creating and using classes in PHP:

1. **Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
2. **Type the following code into the editor window:**

```
<!DOCTYPE html>
<html>
<head>
<title>PHP OOP Test</title>
</head>
<body>
<h1>Testing PHP OOP code</h1>
<?php
class Product {
    public $description;
    public $price;
    public $inventory;
    public $onsale;

    public function buyProduct($amount) {
        $this->inventory -= $amount;
    }
}

$prod1 = new Product();
$prod1->description = "Carrots";
$prod1->price = 1.50;
$prod1->inventory = 10;
$prod1->onsale = false;
echo "<p>Just added $prod1->description<p>\n";

$prod2 = new Product();
$prod2->description = "Eggplants";
$prod2->price = 2.00;
$prod2->inventory = 5;
$prod2->onsale = true;
echo "<p>Just added $prod2->description<p>\n";

echo "<p>Now buying 4 carrots...<p>\n";
$prod1->buyProduct(4);
```

```
echo "<p>Inventory of $prod1->description is now
        $prod1->inventory</p>\n";
echo "<p>Inventory of $prod2->description is still
        $prod2->inventory</p>\n";
?>
</body>
</html>
```

3. **Save the file as** `ooptest1.php` **in the** `DocumentRoot` **folder for your web server.**

   For XAMPP on Windows, that's `c:\xampp\htdocs`; for XAMPP on macOS, that's `/Applications/XAMPP/htdocs`.

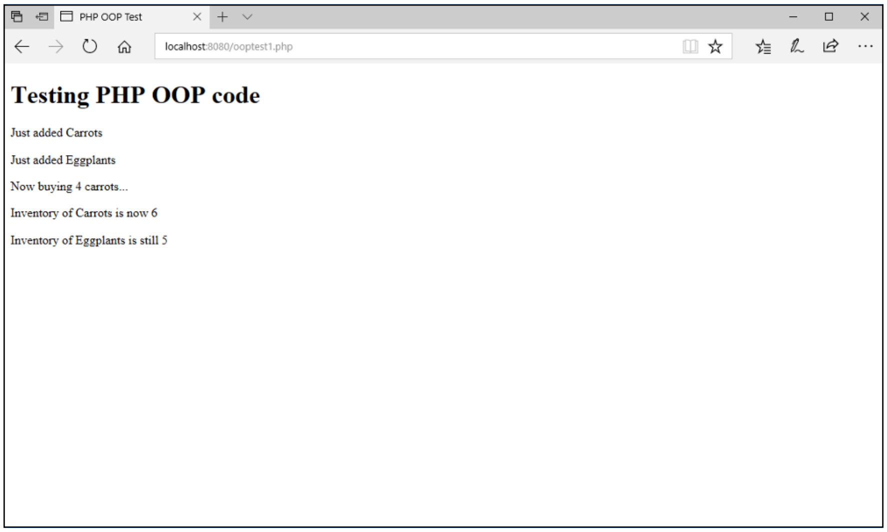
4. **Open the XAMPP Control Panel and then start the Apache web server.**
5. **Open your browser, and enter the following URL:**

   ```
   http://localhost:8080/ooptest1.php
   ```

   You may need to change the TCP port to match your web server.

6. **Close the browser window when you're done.**

When you run the `ooptest1.php` file, you should see the output shown in Figure 5-1.



Testing PHP OOP code

Just added Carrots

Just added Eggplants

Now buying 4 carrots...

Inventory of Carrots is now 6

Inventory of Eggplants is still 5

FIGURE 5-1: The output from the `ooptest1.php` program.

The example code defines the `Product` class, which contains the four properties and one method that has already been discussed. After the `Product` class definition, the code creates two instances of the `Product` class: `$prod1` and `$prod2`. When using classes, you need to define the class first in the code before you create an instance of it.

After creating the two instances, the code uses the `buyProduct()` method for the `$prod1` instance to reduce the inventory by 4. Then it uses two `echo` statements to display the inventory properties for the two instances. Notice that the `buyProduct()` method reduced the inventory of the `$prod1` instance, but not the `$prod2` instance, showing that the two instances are, indeed, separate objects in the program.

Using Magic Class Methods

No, you won't be learning any new tricks involving smoke and mirrors. *Magic class methods* are built-in method names in PHP that apply to all class objects. You can redefine them in your code to provide additional functionality to your PHP classes. This process is called *overloading* or *overriding.* In overloading, you define a method in your class code with the same name as an existing method. PHP uses the newly defined method when you call it from your program code in the class object.

Magic class methods are most often used to help provide common functionality for classes, such as creating a new class object, copying an existing class object, or displaying class objects as text. The PHP developers identify magic class methods by using a double underscore at the start of the method name.

The following sections walk through how to use some of the more common magic class methods in your own classes.

Defining mutator magic methods

Mutator magic methods are methods that change the value of a property that you set with the `private` visibility. These are also commonly called *setters.*

The class example in the previous section used the `public` visibility feature for the class properties, but that's not always a good thing to do. That means that any application can directly access the properties and change them to whatever values it wants. That could be dangerous, and it's somewhat frowned upon in OOP circles.

The preferred way to handle class properties is to make them private so external programs can't change them directly. Instead, to manipulate the data, external programs are forced to use mutator magic class methods that interface with the properties.

The mutator magic method in PHP is `__set()` (note the leading double underscores). You use the mutator magic method to set all the values of the properties in the class with a single method definition:

```
public function __set($name, $value) {
    $this->$name = $value;
}
```

The mutator uses two parameters: the name of the property to set and the value to assign to the property. Where the magic comes into play is with how PHP uses the mutator. In your PHP application code, you don't actually have to call the `__set()` mutator method. You can define the `$description` property just by using a simple assignment statement:

```
$prod1->description = "Carrots";
```

PHP automatically knows to look for the `__set()` mutator method defined for the class and runs it, passing the appropriate property name and value.

Even though the `$description` property is set to the `private` visibility, by defining the mutator magic method you can allow external programs to assign a value to the property. The benefit of using mutators, though, is that you can control how external programs use the properties you define for the class.

With the mutator definition, you can place any code you need to control property features, such as ranges of values allowed or the allowed settings applied to the property. For example, you could so something like this:

```
public function __set($name,$value) {
    if ($name == "price" && $value < 0) {
        $this->price = 0;
    } else {
        $this->$name = $value;
    }
}
```

This example checks if the property being set is the `$price` property. If it is, it checks if the value is less than 0. If the value is less than 0, the price is set to 0 instead of the supplied price value. This gives you a way to control the value that is set for the price from external programs that use the class object.

Defining accessor magic methods

Accessor magic methods are methods you use to access the private property values you define in the class. Creating special methods to retrieve the current property values helps create a standard for how other programs use your class objects. These methods are often called *getters* because they retrieve (get) the value of the property.

You define the accessor using the special `__get()` method:

```
public function __get($name) {
    return $this->$name;
}
```

That's all there is to it! Accessor methods aren't overly complicated; they just return the current value of the property. To use them you just reference the property name as normal:

```
echo "<p>Product: $prod1->description</p>\n";
```

PHP automatically looks for the accessor method to retrieve the property value. Follow these steps to try creating and using a class definition with mutators and accessors:

1. **Open your editor and type the following code:**

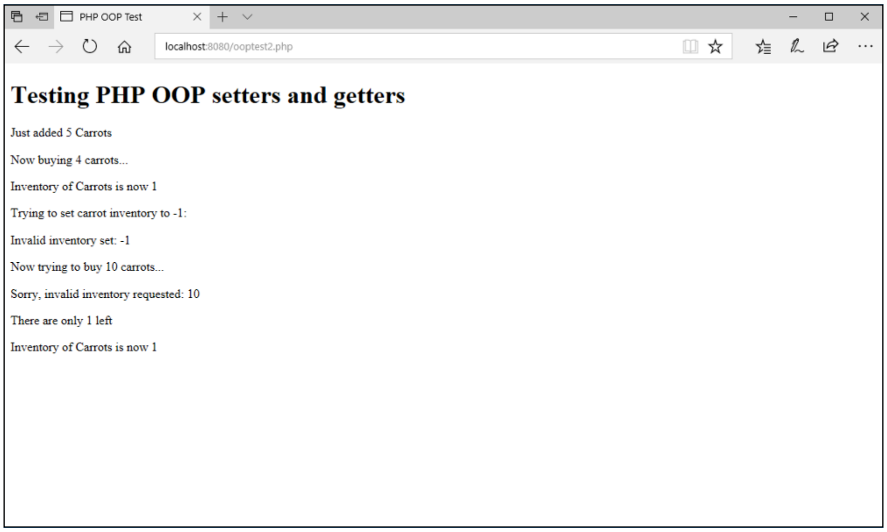
```
<!DOCTYPE html>
<html>
<head>
<title>PHP OOP Test</title>
</head>
<body>
<h1>Testing PHP OOP setters and getters</h1>
<?php
class Product {
    private $description;
    private $price;
    private $inventory;
    private $onsale;

    public function __set($name, $value) {
        if ($name == "price" && $value < 0) {
            echo "<p>Invalid price set<p>\n";
            $this->price = 0;
        } elseif ($name == "inventory" && $value < 0) {
            echo "<p>Invalid inventory set: $value</p>\n";
        } else {
            $this->$name = $value;
        }
    }

    public function __get($name) {
        return $this->$name;
    }

    public function buyProduct($amount) {
        if ($this->inventory >= $amount) {
            $this->inventory -= $amount;
        } else {
            echo "<p>Sorry, invalid inventory requested:
                $amount</p>\n";
            echo "<p>There are only $this->inventory
                left</p>\n";
        }
    }
}

$prod1 = new Product();
$prod1->description = "Carrots";
$prod1->price = 1.50;
$prod1->inventory = 5;
$prod1->onsale = false;

echo "<p>Just added $prod1->inventory $prod1->description</p>\n";

echo "<p>Now buying 4 carrots...<p>\n";
$prod1->buyProduct(4);
echo "<p>Inventory of $prod1->description is now $prod1->inventory</p>\n";

echo "<p>Trying to set carrot inventory to -1:</p>\n";
$prod1->inventory = -1;

echo "<p>Now trying to buy 10 carrots...</p>\n";
$prod1->buyProduct(10);
echo "<p>Inventory of $prod1->description is now $prod1->inventory</p>\n";
?>
</body>
</html>
```

2. **Save the file as `ooptest2.php` in the `DocumentRoot` folder for your web server.**
3. **Ensure that the web server is running, and then open your browser and enter the following URL:**

```
http://localhost:8080/ooptest2.php
```

4. **Close the browser window when you're done.**

Figure 5-2 shows the output that you should see when you run the program in your browser.



```
Testing PHP OOP setters and getters

Just added 5 Carrots

Now buying 4 carrots...

Inventory of Carrots is now 1

Trying to set carrot inventory to -1:

Invalid inventory set: -1

Now trying to buy 10 carrots...

Sorry, invalid inventory requested: 10

There are only 1 left

Inventory of Carrots is now 1
```

FIGURE 5-2: The output from the `ooptest2.php` program.

There's a lot going on in this example, so hang in there with me! First, the PHP code defines the `Product` class, using the four properties, but this time it defines them with private visibility. Following that, the mutator and accessor magic methods are defined. The mutator checks to ensure the price and inventory properties can't be set to a negative value.

After the class definition, the code creates an instance of the `Product` class, and experiments with the inventory values. First, it uses the `buyProduct()` method to purchase four carrots. That works just fine.

Next, it uses the mutator to set the `inventory` property for the carrot object to a negative value. The mutator code intercepts that request and prevents the inventory from being set, instead producing an error message.

Finally, the code tries to use the `buyProduct()` method to purchase more carrots than what's set in inventory. The added code in the `buyProduct()` method prevents that from happening.

Now the class definition is starting to do some useful functions for the application. But wait, there are more magic methods available for you to use!

The constructor

Having to set property values using the mutator methods each time you instantiate a new object can get old, especially if you have lots of properties in the class. The *constructor* magic class method makes that job a lot easier.

The constructor magic method allows you to define values for the properties when you create the new object instance. You can define as many or as few of the properties as you like within the class constructor definition. You do that with the `__construct()` magic method:

```
public function __construct($name, $cost, $quantity) {
    $this->description = $name;
    if ($price > 0) {
        $this->price = $cost;
    } else {
        $this->price = 0;
    }
    $this->inventory = $quantity;
    $this->onsale = false;
}
```

This constructor for the `Product` class uses three parameters to assign values to three of the class properties when you instantiate the class. It also automatically sets the `$onsale` property to a `false` value for each new class instance. To use the constructor, you just provide the three property values as parameters to the class:

```
$prod1 = new Product("Carrot", 1.50, 10);
```

WARNING

When you define a constructor, you have to make sure that the property values are provided in the correct order and data type when you instantiate a new object. If you provide too few arguments to the constructor, PHP will produce an error message. If you provide the right number of arguments but in the wrong order, you may not run into a problem until PHP tries to use the properties in the code.

The destructor

Handling memory management in PHP programs is normally a lot easier than with some other programming languages. By default, PHP recognizes when a class instance is no longer in use and automatically removes it from memory. However, sometimes a program might need to do some type of "cleanup" work for the class object before PHP removes it from memory.

You can specify a magic class method that PHP automatically attempts to run just before it removes the instance from memory. These methods are called *destructors.*

Destructors come in handy with a class that works with files or databases to ensure that the files or database connections are properly closed before the system removes the class instance. This helps prevent any corruption in the data from an improperly closed session being stopped.

You use the `__destruct()` magic class method to define any final statements to process before PHP removes the class instance from memory:

```
public function __destruct() {
    statements
}
```

The `__destruct()` method doesn't allow you to pass any parameters into the method. All the statements you specify in the method need to be self-contained and must not rely on any data from the main program. They can, however, rely on properties within the class, because those should be available when the class object is removed.

You can also manually remove an instance of an object from memory using the `unset()` function:

```
unset($prod1);
```

When you run this command, PHP will process the destructor for the `Product` class for the instance.

WARNING

Although PHP will normally attempt to process a class destructor any time it removes a class instance from memory, it may not always be successful. If the program crashes or stops due to a fatal error, there's no guarantee that PHP will be able to run the destructor method code. If your application relies on closing open files or database connections, it's a good idea to use the `unset()` function to manually remove the object from memory when you're done using it!

Copying objects

You can copy objects within PHP, but not using the standard assignment statement. Instead you need to use the `clone` keyword:

```
$prod1 = new Product("Carrot", 1.50, 100);
$prod2 = clone $prod1;
```

Now the `$prod2` variable contains a second object instance of the `Product` class, with the same property values as the `$prod1` instance.

You may however have a situation where you don't want the copy of the object to have all the same property values as the original. To do that, you can override the `__clone()` magic method in your class code:

```
public function __clone() {
    $this->price = 0;
    $this->inventory = 0;
    $this->onsale = false;
}
```

With this code, when you clone an object only the `description` property will copy over; all the other property values will be reset.

Displaying objects

Most likely, at some point in your application, you'll want to display the properties of your objects in the web page. However, if you try to use the `echo` statement to display the object instance, you'll get a somewhat ugly error message from PHP:

```
Recoverable fatal error: Object of class Product could not be converted to
   string
```

You can solve that problem by defining the `__toString()` magic class method in the class definition.

The `__toString()` magic method defines how you want PHP to handle the properties when you try to use the object as a string value, such as in the `echo` statement. You just build the string value from the properties and store the output in a variable. Then use the `return` statement to return the output variable back to the main program. That code looks like this:

```
public function __toString() {
    $output = "<p>Product: " . $this->description . "<br>\n";
```

```
        $output .= "Price: $" . number_format($this->price,2) . "<br>\n";
        $output .= "Inventory: " . $this->inventory . "<br>\n";
        $output .= "On sale: ";
        if ($this->onsale) {
            $output .= "Yes</p>\n";
        } else {
            $output .= "No</p>\n";
        }
        return $output;
    }
```

With the __toString() magic method defined, you can now use an instance of the Product class in an echo statement just like any variable:

```
echo $prod1;
```

And you'll get the following output in your web page:

```
Product: Carrots
Price: 1.50
Inventory: 10
On sale: No
```

With the __tostring() magic method, displaying your class objects in the web page is as easy as any other type of variable value!

Loading Classes

At the beginning of this chapter, I mention that OOP helps make it easy to reuse program code in multiple applications. After you create the Product class for one application, you can use the same code to use the Product class in any other application that uses products.

However, having to retype the entire Product class code definition in each application that uses it can be somewhat tedious, especially for complicated classes. To solve that problem you can use our friend the include() function.

Just save your class definitions in separate PHP code files; then use the include() function to include the files in any code that uses the class definitions. This enables you to include only the files for the classes the application uses, without having to retype the entire class code definition! That's good, but there may still be a downside to that.

Complex applications may use dozens or possibly even hundreds of separate class objects to manage and manipulate data in the application. Having to list each of the class include files can still be somewhat tedious, as well as be prone to typing mistakes that will cause errors. To solve that problem, the PHP developers created the *autoload feature*, which determines when a class is being instantiated in the program and then tries to load the appropriate include file that defines that class. You implement that using the `spl_autoload_register()` function.

With the `spl_autoload_register()` function, you define the location for all of the class include files based on the class name. With a little bit of programming magic, you can make that task a breeze:

```
spl_autoload_register(function($class) {
    include $class . ".inc.php";
});
```

The anonymous function provided to the `spl_autoload_register()` function defines the include file to load whenever a class is instantiated in the PHP code. The anonymous function attempts to load the include file with the same name as the class name, with an `.inc.php` file extension. Using this method, you must be careful to save the class definition files using the class name as the filename, plus the `.inc.php` file extension.

Follow these steps to try out using the autoload feature in PHP:

1. **Open your editor and type the following code:**

```
<?php

class Product {
    private $description;
    private $price;
    private $inventory;
    private $onsale;

    public function __construct($name, $cost, $quantity, $sale) {
        $this->description = $name;
        $this->onsale = $sale;
        if ($cost < 0) {
            $this->price = 0;
        } else {
            $this->price = $cost;
        }
        if ($quantity < 0) {
            $this->inventory = 0;
        } else {
            $this->inventory = $quantity;
        }
    }

    public function __set($name, $value) {
        if ($name == "price" && $value < 0) {
            echo "<p>Invalid price set<p>\n";
            $this->price = 0;
        } elseif ($name == "inventory" && $value < 0) {
            echo "<p>Invalid inventory set: $value</p>\n";
        } else {
            $this->$name = $value;
        }
    }

    public function __get($name) {
        return $this->$name;
    }

    public function __clone() {
        $this->price = 0;
        $this->inventory = 0;
        $this->onsale = false;
    }

    public function __toString() {
        $output = "<p>Product: " . $this->description . "<br>\n";
        $output .= "Price: $" . number_format($this->price,2) . "<br>\n";
        $output .= "Inventory: " . $this->inventory . "<br>\n";
        $output .= "On sale: ";
        if ($this->onsale) {
            $output .= "Yes</p>\n";
        } else {
            $output .= "No</p>\n";
        }
        return $output;
}

    public function buyProduct($amount) {
        if ($this->inventory >= $amount) {
            $this->inventory -= $amount;
        } else {
            echo "<p>Sorry, invalid inventory requested:
                $amount</p>\n";
            echo "<p>There are only $this->inventory
                left</p>\n";
        }
    }

    public function putonsale() {
        $this->onsale = true;
    }
    public function takeoffsale() {
        $this->onsale = false;
    }
}
?>
```

2. **Save the file as** `Product.inc.php` **(note the capitalization) in the** `DocumentRoot` **folder for your web server.**

3. **Open a new tab or window in your editor and type the following code:**

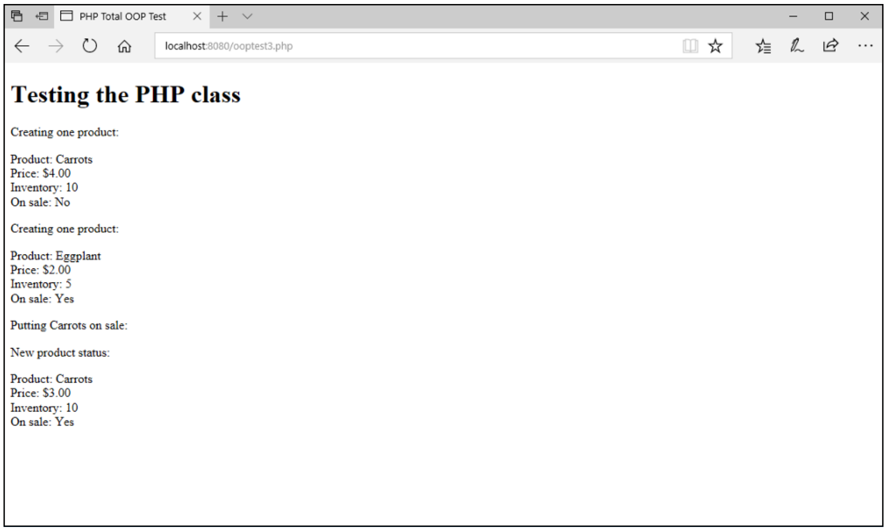
```
<!DOCTYPE html>
<html>
<head>
<title>PHP Total OOP Test</title>
</head>
<body>
<h1>Testing the PHP class</h1>
<?php

spl_autoload_register(function($class) {
    include $class . ".inc.php";
});

$prod1 = new Product("Carrots", 4.00, 10, false);
echo "<p>Creating one product:</p>\n";
echo $prod1;

$prod2 = new Product("Eggplant", 2.00, 5, true);
echo "<p>Creating one product:</p>\n";
echo $prod2;
echo "<p>Putting $prod1->description on sale:</p>\n";
$prod1->price = 3.00;
$prod1->putonsale();
echo "<p>New product status:</p>\n";
echo $prod1;
?>
</body>
</html>
```

4. **Save the file as** `ooptest3.php` **in the** `DocumentRoot` **folder for your web server.**
5. **Ensure that the web server is running and then open your browser and enter the following URL:**

```
http://localhost:8080/ooptest3.php
```

6. **Close the browser window when you're done.**

When you run the `ooptest3.php` file, you should see the output shown in Figure 5-3.



```
Testing the PHP class

Creating one product:

Product: Carrots
Price: $4.00
Inventory: 10
On sale: No

Creating one product:

Product: Eggplant
Price: $2.00
Inventory: 5
On sale: Yes

Putting Carrots on sale:

New product status:

Product: Carrots
Price: $3.00
Inventory: 10
On sale: Yes
```

FIGURE 5-3: The output from the `ooptest3.php` program.

The code saves the `Product` class definition code in the `Product.inc.php` file and then uses the autoloader feature to load the `Product` class include file when needed. It instantiates two `Product` class objects using the constructor and displays them on the web page.

Following that, the code changes the price for the `$prod1` object using the class mutator and uses the `putonsale()` method to place the product on sale. The code finishes with an `echo` statement so you can see the changes made to the class object. Now things are really starting to get fancy!

WARNING

Be very careful when naming class include files. If you're using a server that's case-sensitive with filenames (such as Linux or macOS), then the include filename must match the case of the class name.

Extending Classes

No, I'm not talking about making you stay after school! OOP provides a way to extend an existing class by adding additional members to an existing class. That's the whole beauty of OOP: You can take classes and use them as is, or you can modify just the pieces you need to fit your particular application.

Defining a new class that's an extension of another class is called *inheritance.* The new class (called the *child*) inherits all the public or protected members from the original class (called the *parent*). You can then add new members to the child class and even override members of the parent class. If you use the overridden members, the child members take precedence over the parent members.

WARNING

Members marked with private visibility aren't inherited by child classes. If you want a child class to inherit properties but don't want them visible to external programs, use the `protected` visibility setting.

To create a child class, you use the normal class definition format, along with the `extends` keyword and the name of the class you're extending:

```
class Soda extends Product {
```

For the new class, `Soda`, to inherit the `Product` class properties, you need to change the visibility of the properties to `protected`:

```
class Product {
    protected $description;
    protected $price;
    protected $inventory;
    protected $onsale;
    ...
```

With the properties set to `protected` visibility, the `Soda` child class will automatically inherit the `description`, `price`, `inventory`, and `onsale` properties from the `Product` class, along with all the public class methods.

In the child class definition, you can add additional properties and methods that are unique to the child class:

```
private ounces;

public function restock($amount) {
    $this->inventory += $amount;
}
```

Notice that the `restock()` method uses the `inventory` property that was inherited from the `Product` parent class. When you define a method in a child class, it's only available for objects that are instantiated from the child class. Objects instantiated from the parent class won't see that method.

Because the `Soda` child class contains an additional property, you need to override the `__construct()` method from the parent class to add the new property. That code looks like this:

```
public function __construct($name, $value, $amount, $sale,
                            $size) {
        parent::__construct($name, $value, $amount,
                             $sale);
        $this->ounces = $size;
}
```

The new constructor for the `Soda` child class requires five parameters. Note the first line in the constructor code:

```
parent::__construct($name, $value, $amount, $sale);
```

The `parent::` keyword tells PHP to run the constructor from the parent object. This assigns values to those properties inherited from the parent. The property unique to the child class is assigned a separate value from the parameters.

To instantiate a new child object you'd then just use the following:

```
$rootbeer = new Soda("Root Beer", 1.50, 10, false, 18);
```

Inside the child class definition, you can override any or all of the parent methods. Any methods that don't override the child class objects can use the parent methods.

Follow these steps to test out using inheritance in your OOP PHP code.

1. **Open the `Product.inc.php` file in your editor.**
2. **Change the `private` visibility keyword to `protected`.**

   Look for these four lines:

   ```
   private $description;
   private $price;
   private $inventory;
   private $onsale;
   ```

   And change them to the following:

   ```
   protected $description;
   protected $price;
   protected $inventory;
   protected $onsale;
   ```

3. **Save the file as `Product.inc.php` in the `DocumentRoot` folder for your web server.**
4. **Open a new tab or window in your editor, and type the following code:**

   ```
   <?php

   include("Product.inc.php");

   class Soda extends Product {
       private $ounces;

       public function __construct($name, $value, $amount, $sale, $size) {
           parent::__construct($name, $value, $amount, $sale);
           $this->ounces = $size;
       }

       public function __toString() {
           $output = "<p>Product: " . $this->description . "<br>\n";
           $output .= "Price: $" . number_format($this->price,2) . "<br>\n";
           $output .= "Inventory: " . $this->inventory . "<br>\n";
           $output .= "On sale: ";
           if ($this->onsale) {
               $output .= "Yes<br>\n";
           } else {
               $output .= "No<br>\n";
   ```

```
        }
        $output .= "Ounces: " . $this->ounces . "</p>\n";
        return $output;
    }


    public function restock($amount) {
        $this->inventory += $amount;
    }
}
?>
```

5. **Save the file as `Soda.inc.php` in the `DocumentRoot` folder for your web server.**

6. **Open yet another new tab or window in your editor, and type the following code:**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing PHP Inheritance</title>
</head>
<body>
<h1>Testing inheritance in PHP OOP</h1>
<?php

spl_autoload_register(function($class) {
    include $class . ".inc.php";
});

$prod1 = new Soda("Root Beer", 1.25, 10, false, 18);
echo $prod1;

echo "<p>Buying 6 bottles:</p>\n";
$prod1->buyProduct(6);
echo $prod1;

echo "<p>Restocking 4 bottles:</p>\n";
$prod1->restock(4);
echo $prod1;

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

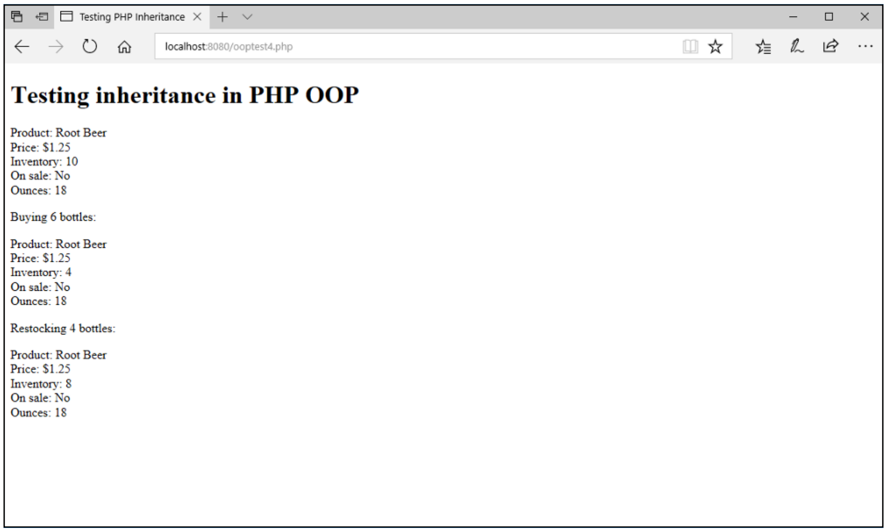
7. **Save this file as `ooptest4.php` in the `DocumentRoot` folder for your web server.**

8. **Ensure that the web server is running, and then open your browser and enter the following URL:**

```
http://localhost:8080/ooptest4.php
```

9. **Close the browser, and stop the web server.**

When you run the `ooptest4.php` code, you should see the output as shown in Figure 5-4.



```
Testing inheritance in PHP OOP

Product: Root Beer
Price: $1.25
Inventory: 10
On sale: No
Ounces: 18

Buying 6 bottles:

Product: Root Beer
Price: $1.25
Inventory: 4
On sale: No
Ounces: 18

Restocking 4 bottles:

Product: Root Beer
Price: $1.25
Inventory: 8
On sale: No
Ounces: 18
```

FIGURE 5-4: The output from the `ooptest4.php` program.

The `Soda` class code overrides both the constructor and the `__toString()` methods of the `Product` parent class to accommodate the additional `$ounces` property. The `ooptest4.php` code creates an instance of the `Soda` class, uses the `buyProduct()` method from the parent class to buy bottles, and then uses the `restock()` method from the child class to restock them. Notice that the child class object has access to the public `buyProduct()` method from the parent class.

IN THIS CHAPTER

» Storing data

» Using cookies

» Working with sessions

» Playing with carts

Chapter 6 Sessions and Carts

In the previous chapters of this minibook, I show you how to use the HTTP `GET` and `POST` methods to send data from one web page to another. Although they work fine for clicking links and submitting forms, they're somewhat impractical to use for sharing data between all the web pages in an application. To do that requires some other form of persistent data, someplace where you can temporarily store it so that your PHP programs can access the data at any time from any page. This is where sessions and carts help out. This chapter explains how they work, why you shouldn't be afraid of them, and how to use them as another piece of your dynamic web applications.

Storing Persistent Data

Most dynamic web applications require some way of temporarily storing data while site visitors work their way through the application web pages. I'm not talking about long-term storage of data (I cover that in the next minibook). I'm talking about short-term storage of data that one web page can store and another web page retrieves, such as passing an authenticated user's info through the website. This helps your application track the site visitors and what they're doing within the application.

This is where HTTP cookies come into play. Cookies have received somewhat of a bum rap in the web world, mainly because of a misunderstanding of how companies use them. A company can't track all of your browsing history using cookies, but it can track which of its advertisements you've visited. This helps the marketing gurus target advertising to your browser based on which of the company's links you've already visited. Cookies do have a valid place in the assembly line of dynamic web application tools, playing a crucial function in being able to keep track of individual site visitors in your application. It's crucial that you know how they work and how to use them.

This section walks through the basics of cookies, why you need them, and how to safely (and responsibly) use them in your dynamic web applications.

The purpose of HTTP cookies

In the mainframe computer world, people who need to access programs running on the system must first log in to the system. This usually requires entering some type of data that uniquely identifies you, such as typing a user ID, placing your finger on a scanner, or inserting a smart ID card that includes a unique encrypted key. When the system authenticates that you are who you say you are, it allows you access to the system and your data. This process starts what's called a *session.*

The mainframe tracks every transaction you perform within the session. A system administrator can look through the log file and identify the user who performed each transaction on the system.

When you've finished entering transactions, you must log off of the system to stop the session. If you forget to log out, another user can come in and enter new transactions that the mainframe credits to your session.

On a mainframe system, keeping track of sessions is easy, because each user logs in from a specific device (either a directly connected terminal or a persistent network connection), performs transactions, and then logs out. Unfortunately, it's not that easy in our dynamic web applications.

The HTTP standard was intended to retrieve data from a remote server in an anonymous, stateless manner. This means not having to deal with the formalities of a session. In essence, a web session consists of a single transaction, and it doesn't even require an ID to identify the user.

Dynamic web applications are somewhat of a hybrid of these two environments. You want to maintain the ease of an HTTP anonymous session, but you need to track users and their transactions like a mainframe session. This is where cookies come to save the day.

Cookies are data that a server can temporarily store in the browser of each site visitor. When the browser stores the cookie data, the server can retrieve that information in later transactions with the site visitor. This allows the server (and, thus, the server-side application) to identify individual site visitors and keep track of what they're doing within the application. This is the beginning of a true web session.

Types of cookies

Before you start thinking chocolate chip and oatmeal raisin, let me start out by saying we're not talking about those types of cookies here. There are several different characteristics of HTTP cookies, each one defining a different way to use the cookie. Table 6-1 lists the different HTTP cookie types you can use.

TABLE 6-1 Types of HTTP Cookies

Type	Description
`HttpOnly`	Can only be accessed via HTTP, not via JavaScript
`Persistent`	Expires at a specific date/time or after a specific length of time
`SameSite`	Can only be sent in requests from the same origin as the target domain
`Secure`	Can only be sent in HTTPS connections
`Session`	Expires when the client browser window closes
`Supercookie`	Uses a top-level domain as the origin, allowing multiple websites access
`Third-party`	Uses a domain that doesn't match the URL domain for the web page

The standard type of cookie is the *persistent cookie.* Persistent cookies are sent by the web server to be stored in the client browser for a specific amount of time. Your application can store data in a persistent cookie and then access that data any time in the future until the cookie expires.

As opposed to persistent cookies, *session cookies* only last for as long as the client browser window stays open. When the site visitor closes the browser window, the session cookies (and the data they contain) go away.

Third-party cookies are what gave cookies a bad name. With persistent and session cookies, a web server can only retrieve and read the cookies that it sets — it doesn't have access to cookies set by other servers. This helps protect the privacy of site visitors by preventing a single server from determining all the websites a site visitor has visited. Third-party cookies use a loophole to get around that.

These days it's very common for a web page to contain embedded advertisements from other websites. Those embedded advertisements run code created by the remote website and can set cookies from the remote website, storing the location of the main website the advertisement is embedded in. This allows a company to purchase advertising space on multiple common websites and then determine which site visitors have visited which website by tracking the cookies that it sets in the advertisements. Now that's sneaky!

TIP

Most modern browsers allow you to block third-party cookies separate from session or persistent cookies, allowing you to use cookies for normal operations but block third-party cookies trying to track your website history.

The anatomy of a cookie

The HTTP standard defines how web servers set and retrieve cookies within the HTTP session with a client browser. When a client browser requests to view a web page on a server, it sends an HTTP `GET` request:

```
GET /index.php
Host: www.myserver.com
```

The request specifies the web page to retrieve and the host from where to retrieve it (usually the same server the request is sent to). The host server returns an HTTP response, which includes the status code for the request, along with any cookies that it wants to set using the `Set-Cookie` statement and then the HTML for the requested web page:

```
HTTP/1.0 GET OK
Content-type: text/html
Set-Cookie: name1=value1; attributes
Set-Cookie: name2=value2; attributes
    Web page HTML content
```

The cookie information appears before the HTML from the requested web page. The server assigns each cookie a unique name and a value, and possibly adds optional attributes that define the cookie type. The client browser stores each cookie as a separate temporary file on the client workstation.

The `Set-Cookie` statement can list one or more optional attributes for the cookie. Table 6-2 lists the cookie attributes that you can set.

TABLE 6-2 **HTTP Cookie Attributes**

Attribute	Description
`Domain=`*`site`*	Specifies the domain the cookie applies to. If omitted the server is the default location.
`Expires=`*`datetime`*	Specifies the expiration date for the cookie as an HTTP timestamp value.
`HttpOnly`	Specifies that the cookie can only be retrieved in an HTTP session.
`Max-Age=`*`number`*	Specifies the expiration time for the cookie in seconds.
`Path=`*`path`*	Indicates the path in the URL that must exist in the requested resource.
`SameSite=`*`setting`*	Specifies if the cookie can only be accessed from the same site that set it. Values are `Strict` or `Lax`.
`Secure`	Specifies that the cookie can only be sent in an HTTPS secure session.

If either the `Expires` or `Max-Age` attributes are set, the cookie is a persistent cookie. It will remain available until the expiration date and/or time. If no attributes are specified, the cookie is a session cookie and will be deleted when the client browser window closes.

The `Expires` attribute specifies an exact date and time the cookie will expire:

```
Set-Cookie: id=25; Expires=Mon 12 May 2025 13:30:00 GMT;
```

The `Max-Age` attribute sets a time duration (in seconds) that the cookie should remain valid:

```
Set-Cookie: id=25; Max-Age=3600
```

After the server sets a cookie, the next time the client browser requests a web page from the same destination, it sends all the cookies set from that destination in the HTTP request using a single `Cookie` statement:

```
GET /index.php
Host: www.myserver.com
Cookie: name1=value1; name2=value2
```

The `Cookie` statement just sends the name/value pair for all the cookies set by that server. It doesn't send any attributes that the server had set for the cookies. The server can then extract the separate cookie names and values and pass them to any server-side programming language (such as your PHP programs).

Cookie rules

Overall, the implementation of cookies in browsers is somewhat nonstandard. No two client browsers may handle cookies the same way. There are however a few minimum requirements that the HTTP standard specifies:

- The browser must support cookies up to 4,096 bytes in size.
- The browser must support at least 50 cookies per website.
- The browser must be able to store at least 3,000 cookies total.

Most browsers exceed these requirements, but it's best not to test the limits in your applications. If you need to store large amounts of data for an application, it's best to use some other type of persistent data storage, such as a database. You can store a key identifying the site visitor as a cookie, and then use that key to reference the larger amounts of data stored in the database associated with that site visitor.

WARNING

Be careful when using session cookies. There is still some controversy in the browser world over how to handle session cookies, especially now that tabbed browsers have become all the rage. Most browsers consider all the web page tabs within the same browser window as a single session. To close the session, your site visitor must close the entire browser window, not just the tab for the web page. Also, many browsers now have a feature that allows for the option of saving sessions by storing session cookie data rather than removing it when the browser window closes. This somewhat circumvents the whole idea of session cookies!

PHP and Cookies

PHP allows you to fully interact with cookies in your web applications. You can set cookies from one web page, retrieve and read them in another web page, and remove them from yet another web page. This section walks through the code you need to use to implement cookies in your PHP applications.

Setting cookies

PHP uses the `setcookie()` function to set new cookies and update existing cookies. Here's the basic format of the `setcookie()` function:

```
setcookie(name [, value] [, expire] [, path] [, domain] [, secure] [, httponly])
```