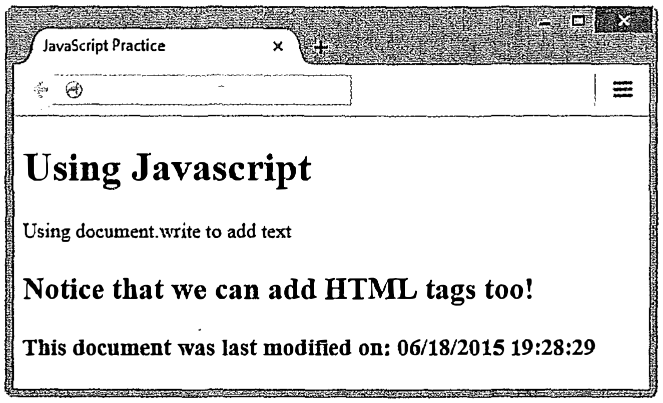
Hands-On Practice 14.2

In this Hands-On Practice, you will practice using the `write()` method of the document and the `lastModified` property of the document. You will use `document.write()` to add text and some HTML tags to an HTML document. You will also use `document.write()` to write the date the file was last saved to the document.

Open the alert.html file (found at chapter14/14.1/alert.html in the student files) and edit the code as indicated:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>JavaScript Practice</title>
  <meta charset="utf-8">
</head>
<body>
<h1>Using JavaScript</h1>
<script>
document.write("<p>Using document.write to add text</p>");
document.write("<h2>Notice that we can add HTML tags too!</h2>");
</script>
<h3>This document was last modified on:
<script>
document.write(document.lastModified);
</script>
</h3>
</body>
</html>
```

Save this file as write.html and view it in the browser. The text should display (see Figure 14.8). If the text does not display, open the Web Console and correct any errors that appear.



Using Javascript

Using document.write to add text

Notice that we can add HTML tags too!

This document was last modified on: 06/18/2015 19:28:29

Figure 14.8 The Firefox browser displays write.html

JavaScript can be seen in the source code. To confirm this, select the top right menu icon > Developer > Page Source to see the source code. Close the source code window when you have finished viewing the code. A suggested solution is located at chapter14/14.2/write.html in the student files.

FAQ **Why would I use `document.write` when I can just type the HTML code by itself?**

In practice, you typically wouldn't use `document.write` to generate your web page if you could just type the HTML code by itself. You would use `document.write` in conjunction with other techniques. For instance, you might use JavaScript to detect the time of day and, if it is before noon, use `document.write` to write "Good morning" to the document. If it is afternoon, write "Good afternoon" to the document, and if it is after 6:00 p.m., write "Good evening" to the document.

14.6 Events and Event Handlers

As the user is viewing a web page, the browser detects mouse movement and events. An **event** is an action taken by the web page visitor, such as clicking the mouse, loading pages, or submitting forms. For instance, when you move your mouse pointer over a hypertext link, the browser detects a mouseover event. Table 14.1 lists a few of the events and their descriptions.

Table 14.1 Events and their descriptions

Event	Description
`click`	The user clicks on an item. This could be an image, hypertext link, or button.
`load`	The browser displays a web page.
`mouseover`	The mouse pointer hovers over an item. The mouse pointer does not have to rest on the object. This could be a hypertext link, image, paragraph, or another object.
`mouseout`	The mouse pointer is moved away from an item that it had previously hovered over.
`submit`	The user clicks the submit button on a form.
`unload`	The web page unloads in the browser. This event occurs just before a new web page loads.

When an event occurs, this can trigger some JavaScript code to execute. One widely used technique is to detect the mouseover and mouseout events and swap images or display a menu.

We need to indicate which events will be acted upon and what will be done when an event occurs. We can use an **event handler** to indicate which event to target. An event handler is embedded in an HTML tag as an attribute and indicates some JavaScript code to execute when the event occurs. Event handlers use the event name with the prefix "on". Table 14.2 shows the event handlers that correspond to the events described in Table 14.1. For example, the **onload** event is triggered when browser renders (loads) a web page. When you move your mouse pointer over a text hyperlink, a **mouseover** event occurs and is detected by the browser. If that hyperlink contains an **onmouseover** event handler, the JavaScript code indicated by the event handler will execute. This code might pop up an alert message, display an image, or display a menu. Other event handlers such as **onclick** and **onmouseout** can cause JavaScript code to execute when their corresponding event occurs.

Table 14.2 Events and event handlers

Event	Event Handler
`click`	`onclick`
`load`	`onload`
`mouseover`	`onmouseover`
`mouseout`	`onmouseout`
`submit`	`onsubmit`
`unload`	`onunload`

Hands-On Practice 14.3

Let's practice using the `onmouseover` and `onmouseout` event handlers and alert messages to indicate when the event handler has been triggered. We will use simple hypertext links and embed the event handlers in the anchor tags. We will not need the `<script>` block because event handlers are placed as attributes within the HTML tags. We'll place the hypertext links in an unordered list so that there's a lot of room in the browser window to move the mouse pointer and test our script.

Open a text editor and enter the code shown below. Note the use of the double and single quotes in the `onmouseover` and `onmouseout` event handlers. We need quotes around the message in the `alert()` method and we need quotes that encapsulate the JavaScript for the event handler. HTML and JavaScript will allow us to use either double quotes or single quotes. The rule is that they must match in pairs. So when you have a situation where you need two sets of quotes, you can use both double and single quotes. Use double quotes for the outer set and single quotes for the inner set. In the anchor tag, the `#` symbol is used for the href value because we don't need the functionality of loading another web page. We need the hypertext link to sense mouseover and mouseout events.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>JavaScript Practice</title>
  <meta charset="utf-8">
</head>
<body>
<h1>Using JavaScript</h1>
<ul>
  <li><a href="#" onmouseover="alert('You moused over');">Mouseover
  test</a></li>
  <li><a href="#" onmouseout="alert('You moused out');">Mouseout
  test</a></li>
</ul>
</body>
</html>
```

Save this file as mouse.html and load it in the browser. Move your mouse on top of the Mouseover test link. As soon as your mouse touches the link, the mouseover event occurs and the `onmouseover` event handler is triggered. This displays the alert box (see Figure 14.9).

Click the OK button and position your mouse pointer over the Mouseout test link. Notice that nothing happens. This is because the mouseout event has not occurred yet.

Move the mouse pointer away from the link. As soon as the mouse pointer is no longer on the link, the mouseout event occurs and the `onmouseout` event handler is triggered. This displays the alert box (see Figure 14.10). A suggested solution can be found in the student files (chapter14/14.3/mouse.html).

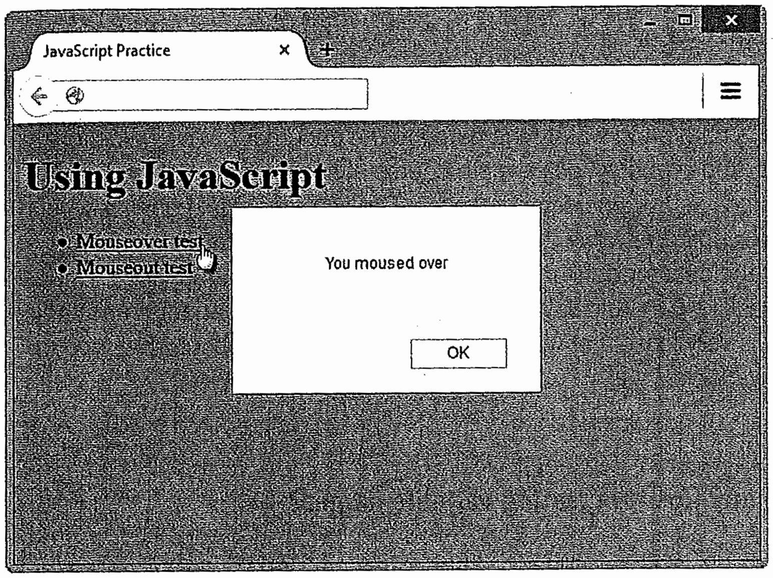


Figure 14.9 Demonstration of onmouseover

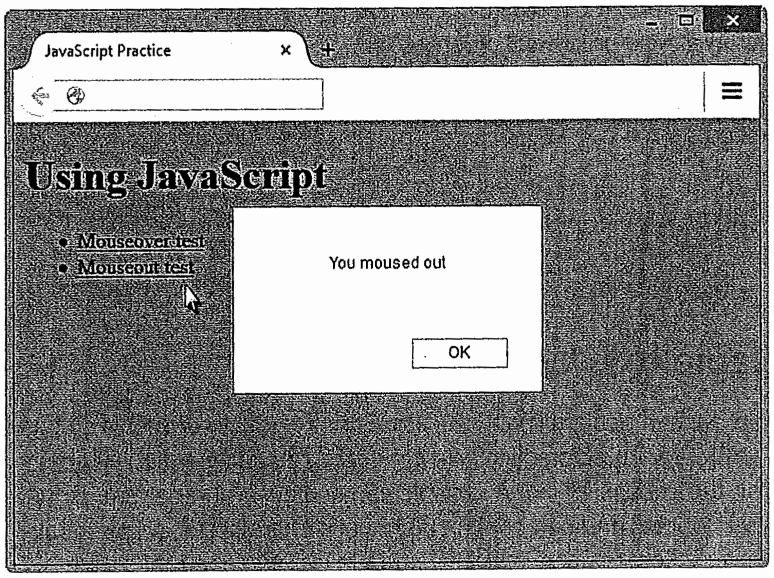


Figure 14.10 Demonstration of onmouseout

Checkpoint 14.2

1. With respect to objects, describe the difference between a property and a method. Feel free to use words like thing, action, description, attribute, and so on.
2. What is the difference between an event and an event handler?
3. Where are event handlers placed in the HTML document?

14.7 Variables

Sometimes we need to be able to collect information from the user. A simple example is prompting the user for a name and writing the name to the document. We would store the name in a **variable**. You probably took a math course at some point and used *x* and *y* as variables in equations as placeholders for values. The same principle applies when using variables in JavaScript. (We won't do any tricky math here, so relax!) JavaScript variables are also placeholders for data and the value of the variable can change. Robust programming languages like C++ and Java have all kinds of rules for variables and their data types. JavaScript is very loose that way. We won't have to worry about what type of data is contained in a variable.

FAQ Are there any tips for creating variable names?

Creating the name of a variable is really something of an art form. First of all, you want to create a variable name that describes the data it contains. The underscore, or an uppercase character, can be used for readability to indicate more than one word. Do not use other special characters, though. Stick to letters and numbers. Be careful not to use a JavaScript **reserved word** or **keyword**, such as `var`, `return`, `function`, and so on. Visit http://webdevfoundations.net/8e/chapter14.html for a list of JavaScript keywords. The following are some variable names that could be used for a product code:

- `productCode`
- `prodCode`
- `product_code`

Writing a Variable to a Web Page

Before we use a variable, we can declare it with the JavaScript **`var`** keyword. This step isn't necessary, but it is good programming practice. We can assign data to a variable using the assignment operator, the equal sign (=). A variable can contain a number or a string. A **string** is encapsulated in quotes and can contain alphabetic characters, spaces, numbers, and special characters. For instance, a string can be a last name, e-mail address, street address, product code, or paragraph of information. Let's do a practice exercise that assigns data to a variable and writes it to the document.

Hands-On Practice 14.4

In this Hands-On Practice, you will declare a variable, assign string data to it, and write it to the document.

Open a text editor and type the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>JavaScript Practice</title>
  <meta charset="utf-8">
</head>
```

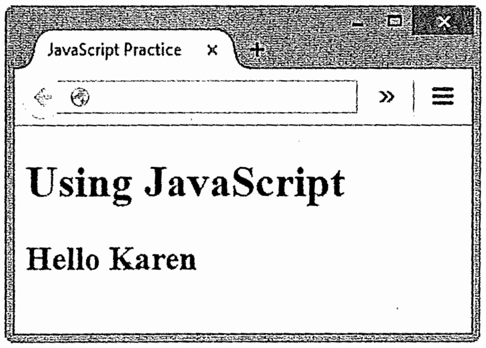
```
<body>
<h1>Using JavaScript</h1>
<h2>Hello
<script>
var userName;
userName = "Karen";
document.write(userName);
</script>
</h2>
</body>
</html>
```

Notice that the `<h2>` tag is placed before the script block and the `</h2>` tag is placed after the script block. This renders the value of `userName` in the `<h2>` heading format. There is also a single space after the "o" in "Hello". If you miss this space, you'll see the `userName` value displayed right after the "o".

Notice that the variable is mixed case. This is a convention used in many programming languages to make the variable readable. Some developers might use an underscore, like user_name. Selecting a variable name is somewhat of an art form, but try to select names that indicate the contents of the variable.

Also notice that the `document.write()` method does not contain quotes. The contents of the variable will be written to the document. If we had used quotes around the variable name, the variable name itself would be written to the document and not the contents of the variable.

Save this document as var.html and load it in the browser. Figure 14.11 shows the var.html file in the browser. Compare your work to chapter14/14.4/var.html in the student files.



Using JavaScript

Hello Karen

Figure 14.11 Browser with var.html displayed

Chopping up the `<h2>` heading so that it is placed before and after the script is a bit cumbersome. We can combine strings using the plus (+) symbol. You'll see later in this chapter that the plus symbol can also be used to add numbers. The practice of combining strings using the plus symbol is called **concatenation**. Let's concatenate the `<h2>` information as a string with the `userName` value and the `</h2>` tag.

Edit the var.html document as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>JavaScript Practice</title>
  <meta charset="utf-8">
</head>
```

```
<body>
<h1>Using JavaScript</h1>
<script>
var userName;
userName = "Karen";
document.write("<h2>Hello " + userName + "</h2>");
</script>
</body>
</html>
```

Be sure to remove the `<h2>` and `</h2>` information above and below the script block. Save the file as var2.html and display it in the browser window. You should not see any difference in the document in the browser. Compare your work to chapter14/14.4/var2.html in the student files.

Collecting Variable Values Using a Prompt

To demonstrate the interactive aspect of JavaScript and variables, we can use the **`prompt()`** method to request data from the user and write this data to the web page. For example, we will build on Hands-On Practice 14.4 and prompt the user for a name rather than hard code this data in the userName variable.

The `prompt()` method is a method of the window object. We could use `window.prompt()`, but the window object is assumed, so we can write this simply as `prompt()`. The `prompt()` method can provide a message to the user. This method is generally used in conjunction with a variable so that the incoming data is stored in a variable. The structure is

```
someVariable = prompt("prompt message");
```

When this command executes, a prompt box pops up that displays the message and an input box for data entry. The user types in the prompt box, clicks the OK button, and the data is assigned to the variable. Let's add this feature to the var2.html file.

Hands-On Practice 14.5

In this Hands-On Practice, you will use the `prompt()` method to gather data from the user and write it to the document.

Edit your file from Hands-On Practice 14.4 (also found in the student files at chapter14/14.4/var2.html) as follows:

```
<script>
var userName;
userName = prompt("Please enter your name");
document.write("<h2>Hello " + userName + "</h2>");
</script>
```

Only the `userName` variable assignment command has changed. The data typed by the user will be assigned to the variable `userName`.

Save the file as var3.html and display it in the browser. The prompt box will appear and you can type a name in the input box and click the OK button (see Figure 14.12). The name should appear in the browser window. Compare your work to chapter14/14.5/var3.html in the student files.

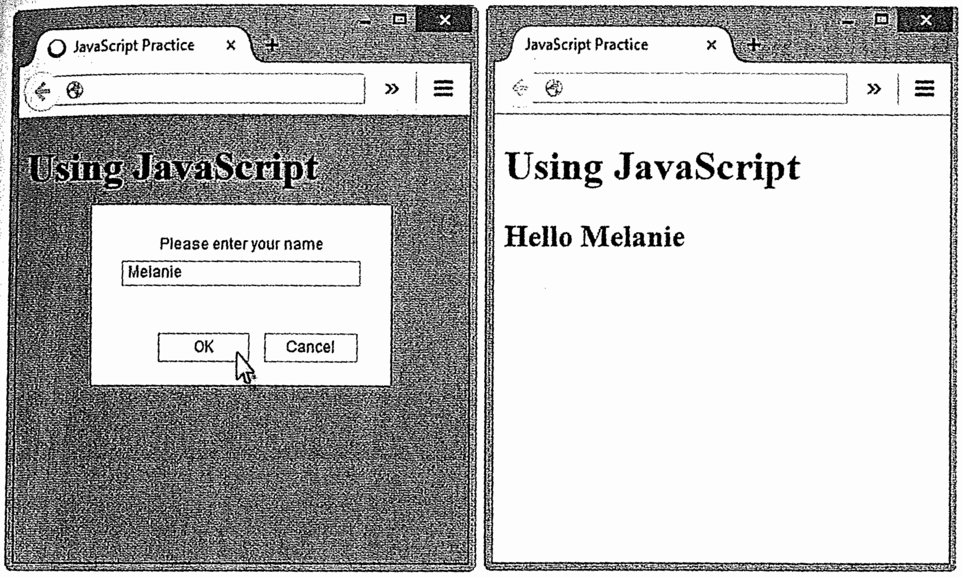


Figure 14.12 The prompt box is displayed in the browser when the web page is loaded. The visitor types their name and clicks OK. Next, the browser accepts the input and writes a welcome message on the page.

Let's do a variation on this and allow the user to type a color name. The user's preference will be used as the background color of the document. We will use the `bgColor` property of the document object and set it to the user's color preference. Be sure that an uppercase C is used when typing `bgColor`.

Edit the var3.html document as follows and save it as var4.html:

```
<script>
var userColor;
userColor = prompt("Please type the color name blue or red");
document.bgColor = userColor;
</script>
```

We are prompting the user to type the color name "blue" or "red". You know from your HTML experience that there are more options for color names. Feel free to experiment!

Save the document and display it in the browser. The prompt box will appear and you can type a color name and click the OK button. You should notice the background color change immediately. Compare your work to chapter14/14.5/var4.html in the student files.

14.8 Introduction to Programming Concepts

Until now, we have used the DOM to access properties and methods for the window and document. We have also used some simple event handlers. There is another aspect of JavaScript that is more like programming. In this section, we'll touch on just a small part of this to get a feel for the power of using programming concepts and build on this later to test input on a form.

Arithmetic Operators

When working with variables, it is often useful to be able to do some arithmetic. For instance, you may be creating a web page that calculates the tax on a product. Once the user has selected a product, you can use JavaScript to calculate the tax and write the result to the document. Table 14.3 shows a list of **arithmetic operators**, descriptions, and some examples.

Table 14.3 Commonly used arithmetic operators.

Operator	Description	Example	Value of Quantity
=	Assign	quantity = 10	10
+	Addition	quantity = 10 + 6	16
–	Subtraction	quantity = 10 – 6	4
×	Multiplication	quantity = 10 × 2	20
/	Division	quantity = 10 / 2	5

Programming languages differ greatly in capabilities, but they all have a few things in common. They all allow the use of variables and have commands for decision making, command repetition, and reusable code blocks. Decision making would be used when different outcomes are required, depending on the input or action of the user. In the next Hands-On Practice, we will prompt the user to enter a number and write different text on the web page document based on the value entered. Repetition of commands comes in handy when performing a similar task many times. For instance, it is tedious to create a select list containing the numbers 1 through 31 for the days of the months. We can use JavaScript to do this with a few lines of code. Reusable code blocks are handy when you want to refer to a block of code in an event handler rather than typing many commands in the HTML tag's event handler. Because this chapter is meant to be a very brief introduction, it is beyond our scope to elaborate further. We will touch on decision making and reusable code in the Hands-On Practice.

Decision Making

As we've seen, we can use variables in JavaScript. We may wish to test the value of a variable and perform different tasks based on the value of the variable. For instance, perhaps an order form requires that the user enter a quantity greater than `0`. We could test the quantity input box to verify that the number entered is greater than `0`. If the quantity is not greater than `0`, we could pop up an alert message that instructs the user to enter a quantity greater than `0`. The **`if`** control structure will be

```
if (condition) {
... commands to execute if condition is true
} else {
... commands to execute if condition is false
}
```

Notice that there are two types of grouping symbols used: parentheses and brackets. The parentheses are placed around the condition and the brackets are used to encapsulate a block of commands. The `if` statement includes a block of commands to execute if the condition is true and a block of commands to execute if the condition is false. The brackets are aligned so that you can easily see the opening brackets and closing brackets. It's very easy to miss a bracket when you're typing and then you would have to hunt for the missing bracket. Aligning them makes it much easier to track them visually. As you are typing JavaScript code, remember that parentheses, brackets, and quotes always are used in pairs. If a script isn't working as intended, verify that each of these items has a "partner."

If the condition evaluates as true, the first command block will be executed and the else block will be skipped. If the condition is false, the first command block will be skipped and the else block will execute.

This overview should give you a sense of how conditions and the `if` control structure can be useful. The condition must be something that can be evaluated as either true or false. We can think of this as a mathematical condition. The condition will generally make use of an operator. Table 14.4 lists commonly used **comparison operators**. The examples in Table 14.4 could be used as conditions in an `if` control structure.

Table 14.4 Commonly used comparison operators

Operator	Description	Example	Sample Values of a Quantity that Would Result in True
==	Double equal signs (equivalent); "is exactly equal to"	`quantity == 10`	10
>	Greater than	`quantity > 10`	11, 12 (but not 10)
>=	Greater than or equal to	`quantity >= 10`	10, 11, 12
<	Less than	`quantity < 10`	9, 8 (but not 10)
<=	Less than or equal to	`quantity <= 10`	10, 9, 8

FAQ What can I do when my JavaScript code doesn't seem to be working?

You can try the following debugging techniques:

- Open the Web Console in Firefox (Tools > Web Developer > Web Console) to see if there are any errors. Common errors include missing a semicolon at the end of a line and typing errors in commands.
- Use `alert()` to print variables to verify the contents. For instance, if you have a variable named quantity, try `alert(quantity);` to see what is contained in the variable.
- Ask a classmate to look at your code. It's difficult to edit your own code because you tend to see what you think you wrote rather than what you actually wrote. It's easier to edit someone else's code.

- Try to explain your code to a classmate. Often, talking through the code will help you uncover errors.
- Verify that you are not using any JavaScript reserved words as variable names or function names. Visit http://webdevfoundations.net/8e/chapter14.html for a list of reserved words.

Hands-On Practice 14.6

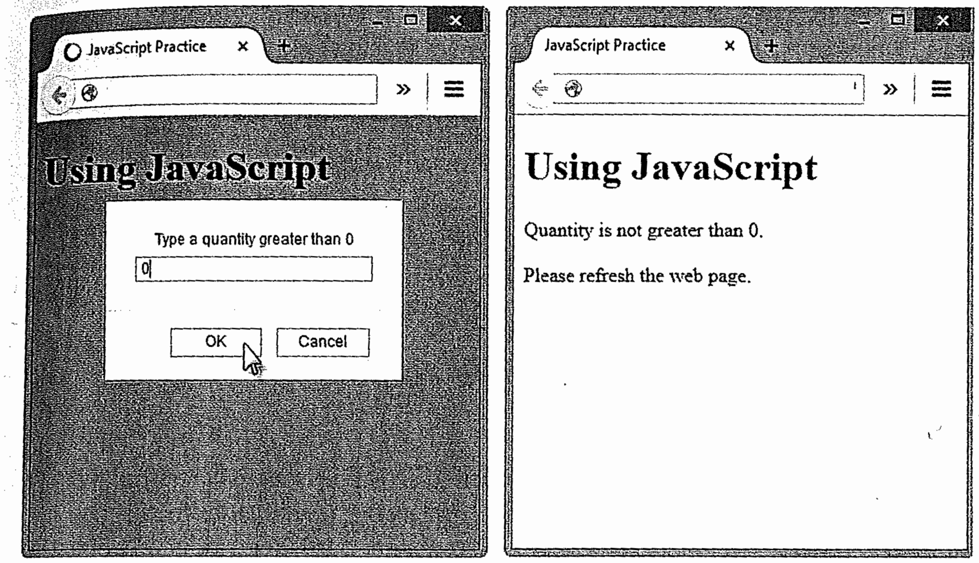
In this Hands-On Practice, you will code the quantity example described earlier. The user will be prompted for a quantity and must enter a quantity greater than 0. We will assume that the user will enter a number. If the user enters a value of 0 or a negative number, there will be an error message displayed. If the user enters a value greater than 0, a message will be displayed thanking the user for the order. We will use a prompt and will write messages to the document.

Open a text editor and enter the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>JavaScript Practice</title>
  <meta charset="utf-8">
</head>
<body>
<h1>Using JavaScript</h1>
<script>
var quantity;
quantity = prompt("Type a quantity greater than 0");
if (quantity <= 0) {
   document.write("<p>Quantity is not greater than 0.</p>");
   document.write("<p>Please refresh the web page.</p>");
} else {
   document.write("<p>Quantity is greater than 0.</p>");
}
</script>
</body>
</html>
```

Save this document as quantityif.html and display it in a browser. If the prompt box does not appear, remember to check the Web Console for errors. When the prompt box appears, type the number 0 and click the OK button. You should see the error message you have created in the browser window (see Figure 14.13).

Now, refresh the page and enter a value greater than 0 (see Figure 14.14). Compare your work to chapter14/14.6/if.html in the student files.



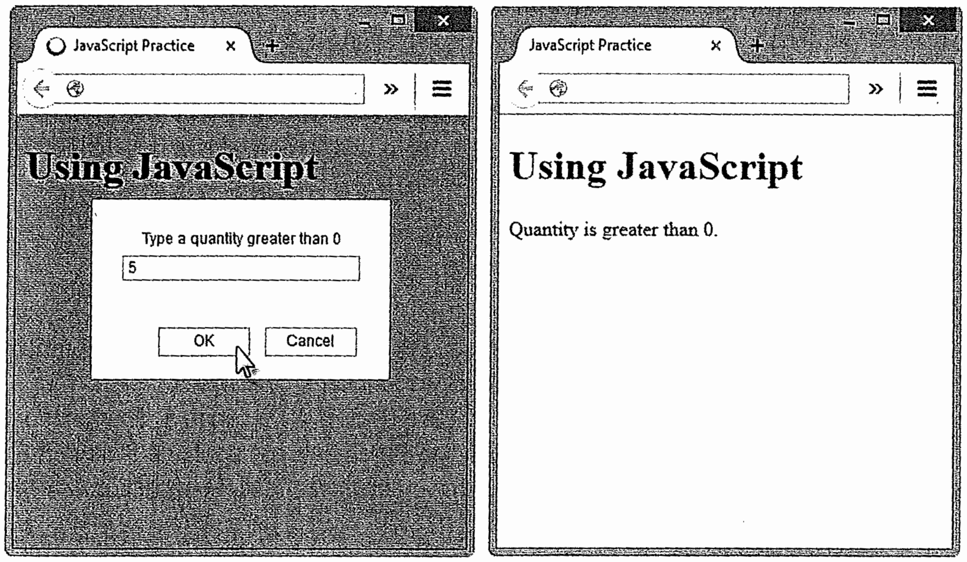
Using JavaScript

Using JavaScript

Quantity is not greater than 0.

Please refresh the web page.

Figure 14.13 The browser on the left shows the prompt box with input of 0 and the browser on the right shows the result



Using JavaScript

Using JavaScript

Quantity is greater than 0.

Figure 14.14 The browser on the left shows the prompt box with an input value that is greater than 0 and the browser on the right shows the result

Functions

In Hands-On Practice 14.6, you coded a prompt box that pops up as soon as the page loads. What if we prefer to allow the user to decide when a particular script should be interpreted or run by the browser? Perhaps we could use an onmouseover event handler and run the script when the user moves the mouse pointer over a link or image. Another method, perhaps more intuitive for the user, is to make use of a button and direct the user to click the button to run the script. The web page visitor doesn't need to be aware that a script will run, but can click a button to initiate some sort of functionality.

Three types of buttons were introduced in Chapter 9:

- A submit button, `<input type="submit">`, is used to submit a form.
- A reset button, `<input type="reset">`, is used to clear values entered on a form.
- The third type of button, `<input type="button">`, does not have any default action related to forms.

In this section, we will make use of the button, `<input type="button">`, and the `onclick` event handler to run a script. The onclick event handler can run a single command or multiple commands. The sample HTML is

```
<input type="button" value="Click to see a message"
  onclick="alert('Welcome!');">
```

In this sample, the button will display the text "Click to see a message". When the user clicks the button, the click event occurs and the `onclick` event handler executes the `alert('Welcome!');` statement. The message box appears. This method is very effective when there is only one JavaScript statement to execute. It quickly becomes unmanageable when there are more statements to execute. When that happens, it makes sense to place all JavaScript statements in a block and somehow point to the block to execute. If the statement block has a name, we can execute the block by pointing to the name. In addition to providing a shortcut name, this code is also easily reused. We can provide a name for a statement block by creating a function.

A **function** is a block of JavaScript statements with a specific purpose that can be run when needed. A function can contain a single statement or a group of statements and is defined as

```
function function_name() {
...  JavaScript statements
}
```

The function definition starts with the keyword `function` followed by the name of the function. The parentheses are required and more advanced functions make use of them. You can choose a name for the function just like you choose a name for a variable. The function name should indicate the purpose of the function. The statements are contained within the brackets. The block of statements will execute when you **invoke**, or call, the function.

Here's an example of a function definition:

```
function showAlerts() {
  alert("Please click OK to continue.");
  alert("Please click OK again.");
  alert("Click OK for the last time to continue.");
}
```

The function can be invoked using

```
showAlerts();
```

Now, we could include the `showAlerts()` function call in a button as

```
<input type="button" value="Click to see alerts."
onclick="showAlerts();">
```

When the user clicks the button, the `showAlerts()` function will be called and the three alert messages will appear one after the other. Try out the example in the student files (chapter14/function.html) to see this in action.

Function definitions are typically placed in the head section of the HTML document. This loads the function definition code, but it does not execute or run until it is invoked. This ensures that the function definition is loaded and ready to use before the function is called. Another feature of a function is that variables declared within the function are only available within that function; they have limited **scope** and are not available outside of the function. In contrast, variables that are declared outside of functions have global scope and can be available to all JavaScript associated with the page. Limiting scope prevents conflicts with variable names when using JavaScript libraries such as jQuery.

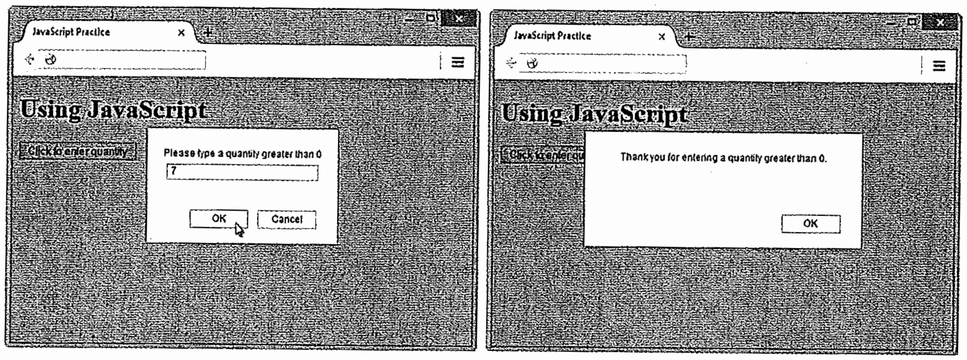
Hands-On Practice 14.7

In this Hands-On Practice, you will edit your file from Hands-On Practice 14.6 (found in the student files at chapter14/14.6/if.html) to move the prompting script into a function and call it with an `onclick` event handler. There are a few things to note. The script has been moved into the head section and is included in a function definition. The `document.write()` methods have been changed to `alert()` methods and the messages have been altered slightly. The `document.write()` methods will not work well after the page has already been written, as is the case in this exercise. Also, there have been some comments added to the end brackets for the `if` statement and the function definition. These comments can help you keep track of the code blocks within the script. The indentation of the code blocks also helps to identify which brackets begin and end various statements. Launch a text editor and edit the if.html file as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>JavaScript Practice</title>
  <meta charset="utf-8">
<script>
function promptQuantity() {
    var quantity;
    quantity = prompt("Please type a quantity greater than 0");
    if (quantity <= 0) {
        alert("Quantity is not greater than 0.");
    } else {
        alert("Thank you for entering a quantity greater than 0.");
    } // end if
} // end function promptQuantity
</script>
</head>
<body>
<h1>Using JavaScript</h1>
<input type="button" value="Click to enter quantity"
onclick="promptQuantity();">
</body>
</html>
```

Save the document as if2.html and display it in a browser. Open the Web Console to check whether there are typing errors when you run the script.

Click the button to test the script. If the prompt box does not appear, check the Web Console and correct any errors. Figure 14.15 shows the browser and the prompt box after the button has been clicked, as well as the resulting alert box. Be sure to test for a value larger than 0 and a value of 0 or less. Compare your work to chapter14/14.7/if2.html in the student files.



Please type a quantity greater than 0

7

Thank you for entering a quantity greater than 0.

Figure 14.15 The browser on the left shows the prompt box and input; the browser on the right shows the alert box displayed after the input

Checkpoint 14.3

1. Describe a method that can be used to gather a piece of data such as the user's age.
2. Write the JavaScript code to display an alert message for users who are under 18 years old and a different alert message for users who are 18 years or older.
3. What is a function definition?

14.9 Form Handling

As you discovered in Chapter 9, the data from a web form can be submitted to a CGI or a server-side script. This data can be added to a database or used for some other purpose; therefore, it is important that the data submitted by a user is as accurate as possible. When the user enters information in a form, there is always a chance that the information will be incorrect or inaccurate. This is particularly true when text input boxes are used because the user can easily mistype data. Often, the form data is checked for invalid data before it is submitted. Form data validation can be done by the server-side script, but it can also be done client-side, using JavaScript. Again, this topic is simplified here, but we can get a sense of how this might be done.