To create the arrays containing the schedule information:

1. In your text editor, open the **calendar.htm** file, located in your Chapter folder for Chapter 3.
2. In the comment section at the top of the document, type your name and today's date where indicated, and then save your work.
3. Scroll through the document to familiarize yourself with its content. The `article` element contains a table consisting of five rows and seven columns. The cells contain no data. Each of the 31 `td` elements that will eventually contain the schedule information has an `id` value. Later in this chapter, you'll use these `id` values to place each array element in the correct cell.
4. Open **calendar.htm** in a browser. As Figure 3-7 shows, the table is displayed as an empty grid.

Figure 3-7: Empty table in calendar.htm

5. Return to your text editor, open a new document, and then enter the following comment, entering your name and today's date where indicated:

```
/*    JavaScript 6th Edition
 *    Chapter 3
 *    Chapter case
 *    Tipton Turbines
 *    Variables and functions
 *    Author:
 *    Date:
 *    Filename: tt.js
 */
```

6. Below the comment section, enter the following code to declare the `daysOfWeek` variable and set its value using an array literal:

```
// global variables
var daysOfWeek = [ "Sunday", "Monday", "Tuesday",
   "Wednesday", "Thursday", "Friday", "Saturday"];
```

This code uses an array literal to declare a variable named `daysOfWeek` whose value is an array that contains the name of each day of the week.

7. Below the `daysOfWeek` variable declaration, enter the following code to declare the `opponents` variable and set its value:

```
1   var opponents =
2      ["Lightning", "Combines", "Combines", "Combines",
3       "Lightning", "Lightning", "Lightning", "Lightning",
4       "Barn Raisers", "Barn Raisers", "Barn Raisers",
5       "Sodbusters", "Sodbusters", "Sodbusters",
6       "Sodbusters", "(off)", "River Riders",
7       "River Riders", "River Riders", "Big Dippers",
8       "Big Dippers", "Big Dippers", "(off)",
9       "Sodbusters", "Sodbusters", "Sodbusters",
10      "Combines", "Combines", "Combines", "(off)",
11      "(off)"];
```

Note

Be sure to start the list of values with an opening bracket ([ ) and end it with a closing bracket ( ]). Also make sure the entire statement ends with a semicolon (;).

This array lists, in order, the Turbines' opponents for each of the 31 days in August 2016. For instance, the first day of the month, August 1, the team is playing the Lightning, on the second day they're playing the Combines, and on the last day, August 31, no game is scheduled.

8. Below the `opponents` variable declaration, enter the following code to declare the `gameLocation` variable and set its value:

```
1   var gameLocation =
2      ["away", "away", "away", "away", "home", "home",
```

```
3       "home", "home", "home", "home", "home", "away",
4       "away", "away", "away", "", "away", "away", "away",
5       "away", "away", "away", "", "home", "home", "home",
6       "home", "home", "home", "", ""];
```

This array lists whether the team is playing away or at home for each of the 31 days in August 2016. Note that this array combines the strings "away" and "home" with empty values, indicated by "", which denote days when no game is scheduled. For instance, the first four days of the month, August 1–4, the games are away, on the next 7 days, August 5–11, the games are at home, and on the last day, August 31, no game is scheduled.

9. Save the file as a text file with the name **tt.js** to your Chapter folder for Chapter 3, return to **calendar.htm** in your text editor, and then add the following line of code at the bottom of the document, just above the closing `</body>` tag:

```
<script src="tt.js"></script>
```

10. Save your changes to **calendar.htm**, and then refresh or reload **calendar.htm** in your browser. Although you've declared three array variables, you haven't yet referenced their values in any JavaScript code, so the web page still matches Figure 3-6. You'll create functions to insert the array values in the table later in this chapter.

Accessing Element Information

You access an element's value just as you access the value of any other variable, except that you include the element index in brackets. For example, the following code assigns the values contained in the first three elements of the `newsSections` array as the content of three web page elements:

```
var sec1Head = document.getElementById("section1");
var sec2Head = document.getElementById("section2");
var sec3Head = document.getElementById("section3");
sec1Head.innerHTML = newsSections[0]; // "world"
sec2Head.innerHTML = newsSections[1]; // "local"
sec3Head.innerHTML = newsSections[2]; // "opinion"
```

In Chapter 2, you used the browser console to check for errors generated by JavaScript code. The browser consoles included in the current versions of all major browsers also enable you to enter JavaScript statements and see the results of the statements displayed. Especially after entering large arrays like the `opponents` and `gameLocation` variables,

it can be helpful to check the array values using the console to verify that you've declared the arrays accurately. You'll use your browser's console to check the arrays in the tt.js file now.

To check the array values using a browser console:

1. Return to **calendar.htm** in your browser. Table 3-1 lists the keyboard shortcuts and menu commands to open the console in the major browsers.

BROWSER	KEYBOARD SHORTCUT	MENU STEPS
Internet Explorer 9+	**F12**, then **Ctrl** + **2**	Click the **Tools** icon, click **F12 developer tools** on the menu, and then in the window that opens, click the **Console** button
Firefox	**Ctrl** + **Shift** + **K** (Win) **option** + **command** + **K** (Mac)	Click the **Open menu** button, click **Developer**, and then click **Web Console**
Chrome	**Ctrl** + **Shift** + **J** (Win) **option** + **command** + **J** (Mac)	Click the **Customize and control Google Chrome** button, point to **Tools**, and then click **JavaScript Console**

Table 3-1: Steps to open the browser console in major browsers

2. Open the console in your browser using the appropriate command from Table 3-1.

Note

If your browser isn't listed, or if one of the listed methods doesn't work, check your browser's documentation for the correct steps, or try a different browser. Remember that different browsers may use different terms for the browser console, including "JavaScript console," "web console," or "error console."

Figure 3-7 shows the browser console in Firefox. The browser console may be displayed differently in different browsers, but all share the same general components indicated in Figure 3-8.

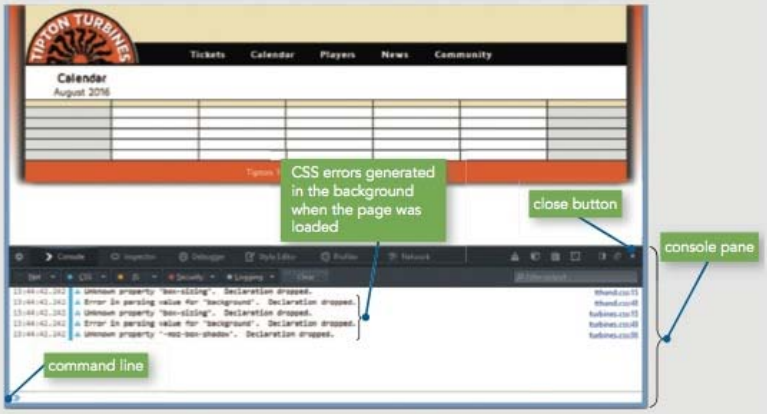

Figure 3-8: Browser console in Firefox
Source: © Mozilla Firefox

3. Click in the command line at the bottom of the console, type **`daysOfWeek[1]`**, and then press **Enter**. As Figure 3-9 shows, the console displays your entry on its own line and then displays the result on the following line. In this case, the result is "Monday".

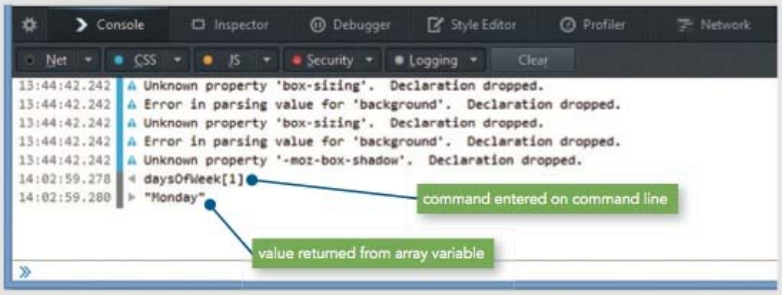

Figure 3-9: Command and result displayed in console

Notice that the console returned the value "Monday" for the command `daysOfWeek[ 1]`, which requests the value of item 1 in the `daysOfWeek` array. The first value in the array is "Sunday." However, remember that array elements are numbered starting with 0, so `daysOfWeek[ 1]` refers to the *second* item in the `daysOfWeek` array, which is "Monday."

Note

Some consoles try to guess what you're typing as you enter it, and display their best guesses on the command line. You can usually press Tab or → to accept the predicted value. When typing an array name, you may be able to save time by typing only the first few letters, accepting the autocompletion, and then typing the number in brackets.

4. On the command line, type **`daysOfWeek[0]`**, and then press **Enter**. The console returns the value "Sunday," which is the first value (number 0) in the array.
5. Repeat Step 4 for each of the values in the first column in Table 3-2, and then verify that the console returns the value indicated in the second column of the table for each.

COMMAND LINE ENTRY	EXPECTED VALUE TO BE RETURNED
`daysOfWeek[ 6]`	`"Saturday"`
`daysOfWeek[ 7]`	`Undefined`
`opponents[ 0]`	`"Lightning"`
`opponents[ 30]`	`"(off)"`
`opponents[ 31]`	`undefined`
`gameLocation[ 0]`	`"away"`
`gameLocation[ 30]`	`""`
`gameLocation[ 31]`	`undefined`

Table 3-2: Command line entries to verify array contents

Note

If your console returns a value different than the one shown in Table 3-2 for any of the listed entries, return to tt.js in your editor, check the definition of the array in question against the previous set of steps, fix any errors, save your work, and then in your browser, refresh or reload calendar.htm and repeat the command line entry.

Modifying Elements

You modify values in existing array elements in the same way you modify values in a standard variable, except that you include the element index in brackets. The following code assigns a new value to the fifth element in the `newsSections` array:

```
newsSections[ 4] = "living";
```

Figure 3-10 illustrates the updated contents of the `newsSections` array after the above statement is executed.

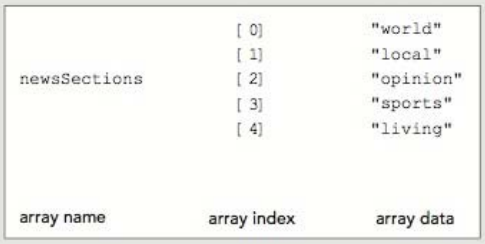

Figure 3-10: `newsSections` array after updating the value of the fifth element

Determining the Number of Elements in an Array

Every array has a `length` property, which returns the number of elements in the array. You append the `length` property to the name of the array whose length you want to retrieve using the syntax *`name`*`.length;`.

Note *Remember that unlike method names, property names are not followed by parentheses.*

Although array indexes are numbered starting from 0, the `length` property counts from 1. This means that, for example, the `length` property for a 10-element array returns a value of 10, even though the final index value in the array is 9.

You'll use the `length` property in your browser's console to verify the number of elements in each of your three arrays.

To check the length of your arrays using your browser's console:

1. Return to **calendar.htm** in your browser.
2. On the command line of the console, type **`daysOfWeek.length`**, and then press **Enter**. The console returns 7, which is the number of elements in the `daysOfWeek` array.
3. Repeat Step 2 to check the values of **`opponents.length`** and **`gameLocation.length`**. The console should return 31 for both.

To use your browser's console to check the values of the `li` elements in the navigation bar:

1. Return to **calendar.htm** in your browser.
2. On the command line of the browser console, type **`document.getElementsByTagName("li")[0].innerHTML`**, and then press **Enter**. The browser console returns "Tickets," which is the link code and text for the first item in the navigation bar.
3. On the command line of the browser console, type **`document.getElementsByTagName("li")[1].innerHTML`**, and then press **Enter**. The browser console returns "Calendar," which is the linke code and text for the second item in the navigation bar.
4. Repeat Step 3 to check the third, fourth, and fifth `li` elements. The browser console should return the values "Players," "News," and "Community," respectively.
5. Close the browser console by clicking the console's **Close button**, or by pressing the same key combination you used to open it. Figure 3-8 shows the location of the Close button on the Firefox console.

Programming Concepts *Counting from 0*

The fact that programming languages often count items such as array elements starting with 0 rather than 1 is a common source of confusion for new programmers. While it can take some practice to get used to, counting from 0 rather than from 1 makes some advanced coding tasks much easier. In programming, it's often necessary to iterate through multiple sets of data and compare or combine the values. Using an agreed-on standard for the beginning of numbering makes it easier to perform these operations mathematically.

Short Quiz 1

1. How is an array different from a standard variable?
2. How do you create a new empty array?
3. How do you access an individual element in an array?
4. What property do you use to determine the number of elements in an array?
5. How do you use a browser to check the value of a specific array element?

increment a counter variable, and you'll use the value of the counter variable to identify the array index of the string to work with as well as the index of the `th` element in which to place it.

To create a function using a `while` statement to place the days of the week in the table:

1. Return to the **tt.js** document in your text editor.
2. Below the variable declarations, enter the following code to add a comment and create the structure of a function named **`addColumnHeaders()`**:

```
1  // function to place daysOfWeek values in header row↵
2     cells
3  function addColumnHeaders() {
4
5  }
```

3. Within the command block, enter the following statement:

```
var i = 0;
```

This statement creates a counter variable called `i` and sets its value to 0.

4. Below the variable declaration, enter the following code:

```
while (i < 7) {


}
```

This code creates a `while` statement that sets the condition `i < 7`. This means that every time the loop restarts and the value of the counter variable is less than 7, the loop will go through another iteration.

5. Within the command block for the `while` statement, enter the following statement:

```
document.getElementsByTagName("th")[ i] .innerHTML =↵
   daysOfWeek[ i] ;
```

Starting at the end, the second part of this statement, `daysOfWeek[ i]`, fetches the value from the `daysOfWeek` array that has the index value equal to the counter variable, `i`. The first part of the statement uses the `getElementsByTagName()` method to identify the `th` element that has the index value equal to the counter variable, `i`, and sets the value of its `innerHTML` property to the corresponding value fetched from the `daysOfWeek` array. For instance, in the first iteration of this loop, the value of `i` will be 0. The `getElementsByTagName()` method will identify the first `th` element, representing the column heading for the first column, and set its value to the first value of the `daysOfWeek` array, which is "Sunday".

6. Below the statement you created in the previous step, enter the following statement:

```
i++;
```

This statement increments the counter variable. Following is the completed code for the `addColumnHeaders()` function.

```
1   // function to place daysOfWeek values in header row
2      cells
3      function addColumnHeaders() {
4      var i = 0;
5      while (i < 7) {
6         document.getElementsByTagName("th")[ i] .innerHTML
7            = daysOfWeek[ i] ;
8         i++;
9      }
10  }
```

7. Below the function you just created, enter the following comment and statement:

```
// runs addColumnHeaders() function when page loads
window.addEventListener("load", addColumnHeaders, false);
```

This statement calls the `addColumnHeaders()` function when the page finishes loading in a browser.

8. Save your changes to **tt.js**, and then reload **calendar.htm** in your browser. As Figure 3-14 shows, the table header rows are now populated with the days of the week in the same order you entered them in the `daysOfWeek` array.

Figure 3-14: The calendar.htm document displaying the column header values

In the previous set of steps, you used a `while` statement to add the values from the `daysOfWeek` array to the Tipton Turbines calendar table. The following statement performs the same action using a `do/while` statement instead:

```
var i = 0;
do {
   document.getElementsByTagName("th")[i].innerHTML =↵
      daysOfWeek[i];
   i++;
} while (i < 7);
```

Notice that the code `while (i < 7)` is moved from before the command block to after it, and the word `do` takes its place. Otherwise, the code using `do/while` is the same as the earlier code using just `while`.

Next, you'll add a function to the tt.js document that will add the days of the month to the calendar table using a `do/while` statement.

To add the days of the month to the calendar using a do/while statement:

1. Return to the **tt.js** document in your text editor.
2. Below the `addColumnHeaders()` function, enter the following code to add a comment and create the structure of a new function named `addCalendarDates()`:

```
// function to place day of month value in first p↵
   element
// within each table data cell that has an id
function addCalendarDates() {

}
```

3. Within the command block, enter the following statements:

```
var i = 1;
var paragraphs = "";
```

The first statement creates a counter variable called `i` and sets its value to 1. The second statement creates a variable named `paragraphs` and sets its value to an empty string.

4. Below the variable declarations, enter the following code:

```
do {


}
```

This code starts a `do/while` statement with the `do` keyword.

5. Within the code block for the `do` statement, enter the following statement:

```
var tableCell = document.getElementById("08-" + i);
```

This statement creates a variable called `tableCell` and uses the `getElementById()` method to set its value to the element with an `id` value that starts with 08- followed by the value of `i`. This means that in the first iteration of the loop, the table cell variable references the element with the `id` of `08-1`, which is the second `td` element in the second row of the table.

6. Below the statement you created in the previous step, enter the following statement:

```
paragraphs = tableCell.getElementsByTagName("p");
```

This statement uses the `getElementsByTagName()` method to look up all `p` elements within the current cell (designated by the `tableCell` variable), and stores the values as an array in the `paragraphs` variable.

7. Below the statement you created in the previous step, enter the following statement:

```
paragraphs[0].innerHTML = i;
```

This statement assigns the value of the counter variable, `i`, as the content of the first paragraph (numbered 0) in the current cell. The first time through the loop, this places the value 1 in the cell for the first day of the month.

8. Below the statement you created in the previous step, enter the following statement:

```
i++;
```

This statement increments the counter variable.

9. After the closing `}` for the `do` command block, but before the closing `}` for the function, add the following statement:

```
while (i <= 31);
```

This conditional statement determines whether the `do` command block is repeated for another iteration. Because the month of August has 31 days, the block should be executed 31 times. After the 31st time, the counter variable will be 32, which is not

less than or equal to 31, so the `do/while` statement will end without iterating again. The following shows the completed code for the `addCalendarDates()` function.

```
// function to place day of month value in first p↵
   element
// within each table data cell that has an id
function addCalendarDates() {
   var i = 1;
   var paragraphs = "";
   do {
      var tableCell = document.getElementById("08-" + i);
      paragraphs = tableCell.getElementsByTagName("p");
      paragraphs[ 0] .innerHTML = i;
      i++;
   } while (i <= 31);
}
```

Because you now have multiple functions to run when the page loads, you'll create a master function to call when the page loads. This function will in turn call each of the functions that needs to run on page load.

10. Below the function you just created, enter the following comment and statement:

```
// function to populate calendar
function setUpPage() {
   addColumnHeaders();
   addCalendarDates();
}
```

This code creates a new function named `setUpPage()` that calls the `addColumnHeaders()` and `addCalendarDates()` functions. You need to change the event listener to call this function when the page finishes loading.

11. In the last two lines of the tt.js file, replace both occurrences of `addColumnHeaders` with **`setUpPage`** so your code matches the following:

```
// runs setUpPage() function when page loads
window.addEventListener("load", setUpPage, false);
```

12. Save your changes to **tt.js**, and then reload **calendar.htm** in your browser. As Figure 3-15 shows, the table cells containing the days of the month have been populated with consecutive numbers from 1-31.

Using a `for` statement is more efficient than a `while` statement because you do not need as many lines of code. Consider the following `while` statement:

```
var count = 1;
while (count < brightestStars.length) {
   document.write(count + "<br />");
   count++;
}
```

You could achieve the same flow control more efficiently by using a `for` statement as follows:

```
for (var count = 1; count < brightestStars.length;↵
      count++) {
   document.write(count + "<br />");
}
```

The following code shows a revised version of the `addColumnHeaders()` function you created earlier in the chapter. You originally created this function using a `while` statement, but the rewritten version below performs the same actions using a `for` statement instead.

```
function addColumnHeaders() {
   for (var i = 0; i < 7; i++) {
      document.getElementsByTagName("th")[i].innerHTML↵
         = daysOfWeek[i];
   }
}
```

Notice that the declaration of the `count` variable, the conditional expression, and the statement that increments the `count` variable are now all contained in the parentheses after the `for` keyword. Using a `for` statement instead of a `do/while` statement simplifies the script somewhat, because you do not need as many lines of code.

Next, you add another function to the tt.js file. This function will use a `for` statement to add the names of the opposing teams from the `opponents` array to the second paragraph in each table cell that represents a date in August.

To add a function to populate the calendar dates with the opposing team names using a `for` statement:

1. Return to the **tt.js** document in your text editor.

2. Below the `addCalendarDates()` function, enter the following code to add a comment and create the structure of a new function named `addGameInfo()`:

```
// function to place opponents values in
// second p element within each table data cell that has an id
function addGameInfo() {

}
```

3. Within the command block, enter the following code:

```
var paragraphs = "";
```

This code creates a variable named `paragraphs` and sets its value to an empty string.

4. Below the variable declaration, enter the following code:

```
for (var i = 0; i < 31; i++) {

}
```

This code starts a `for` statement. It creates a counter variable named `i` and sets its value to 0. It specifies the condition `i < 31`; as long as this condition is satisfied, the `for` command block will repeat. Finally, the code `i++` increments the `i` counter variable with each iteration.

5. Within the code block for the `for` statement, enter the following statement:

```
var date = i + 1;
```

This statement creates a variable called `date` and assigns a value to it equal to the value of the counter variable, `i`, plus 1.

6. Below the statement you created in the previous step, enter the following statement:

```
var tableCell = document.getElementById("08-" + date);
```

This statement creates a variable called `tableCell` and uses the `getElementById()` method to set its value to the element with an `id` value that starts with 08- followed by the value of `i`.

7. Below the statement you created in the previous step, enter the following statement:

```
paragraphs = tableCell.getElementsByTagName("p");
```

This statement uses the `getElementsByTagName()` method to look up all `p` elements within the current cell (designated by the `tableCell` variable) and stores the values as an array in the `paragraphs` variable.

8. Below the statement you created in the previous step, enter the following statement:

```
paragraphs[ 1] .innerHTML = opponents[ i] ;
```

This statement fetches the value in the `opponents` array whose index matches the current value of the counter variable, `i`, and assigns its value as the content of the second paragraph (numbered 1) in the current cell. The following shows the completed code for the `addGameInfo()` function.

```
1  // function to place opponents in second
2  // p element within each table data cell that has an id
3  function addGameInfo() {
4     var paragraphs = "";
5     for (var i = 0; i < 31; i++) {
6        var date = i + 1;
7        var tableCell = document.getElementById("08-" +↵
8           date);
9        paragraphs = tableCell.getElementsByTagName("p");
10       paragraphs[ 1] .innerHTML += opponents[ i] ;
11    }
12 }
```

9. Within the `setUpPage()` function, before the closing `}`, add the statement **`addGameInfo();`** so your `setUpPage()` function matches the following:

```
1  function setUpPage() {
2     addColumnHeaders();
3     addCalendarDates();
4     addGameInfo();
5  }
```

Note *This book uses a yellow highlight to call attention to changes in existing code.*

Calling the `addGameInfo()` function you just created within the `setUpPage()` function ensures that the calendar is populated with opponent information when the page is loaded.

10. Save your changes to **tt.js**, and then reload **calendar.htm** in your browser. As Figure 3-17 shows, the table cells containing the days of the month now also display the name of each day's opponent, or "(off)" if no opponent is scheduled.

value, then the statements within the command block are executed. After the command block executes, the program continues on to execute any code after the command block. If the condition instead evaluates to a falsy value, the `if` statement command block is skipped, and the program continues on to execute any statements after the command block.

So far, you've created functions to add column headers, dates, and opponent names to the calendar page for the Tipton Turbines. To complete the calendar, you need to add one additional piece of information: whether each game is being played at home or away. You've stored this information in the `gameLocation` array. You'll add `if` statements to the `addGameInfo()` function to insert the appropriate content in the calendar based on whether the game is home or away, or the team is off for the day. If the team is playing at home, you'll insert the value "vs" followed by a space. If the team is playing away, you'll insert the character "@" followed by a space. If the team is off for the day, you won't insert any text.

To add the home/away information to the calendar using `if` statements:

1. Return to the **tt.js** document in your text editor.
2. Within the `addGameInfo()` function, within the data block for the `for` loop, insert a new line before the final statement, enter the code **`if (gameLocation[i] === "away") {`**, press **Enter** twice, and then enter a closing **`}`** as shown below:

```
        paragraphs = tableCell.getElementsByTagName("p");
        if (gameLocation[ i] === "away") {

        }
      }
  }
```

This code starts with the `if` keyword to create an `if` statement, followed by parentheses containing the `if` condition. The statement tests whether the element of the `gameLocation` array with the index value equal to the counter variable, `i`, equals the string "away".

Note

Some developers add comments to their closing braces to indicate what statement each brace ends. For instance, after the first closing brace in Step 2, you might add the comment `// end if` to indicate that the brace closes the preceding `if` statement. Especially when your code includes structures nested several levels deep, such comments can make it easier to add or remove levels of nesting.

3. Within the command block for the `if` statement, enter the statement **`paragraphs[1].innerHTML = "@ ";`** as shown below:

```
var paragraphs = "";
      if (gameLocation[ i] === "away") {
          paragraphs[ 1].innerHTML = "@ ";
       }
```

This statement sets the content of the first paragraph in the current cell equal to "@ ".

4. Below the command block you just created, in the final line of code for the `for` loop, change = to += so the statement matches the following:

```
paragraphs[ 1].innerHTML += opponents[ i];
```

Because you've just added content to the second paragraph using your `if` statement, you want to ensure that the name of the opposing team is concatenated to the existing content (+=) rather than replacing it (=).

5. Save your changes to **tt.js**, and then reload **calendar.htm** in your browser. As Figure 3-19 shows, "@" followed by a space is now displayed before the opponent names in some of the table cells. These dates correspond to the elements in the `gameLocation` array with a value of "away."

Figure 3-19: The calendar.htm document displaying @ before the names of away opponents

To add the text "vs" followed by a space before the names of opponents for home games, you need to create another `if` statement.

6. Return to **tt.js** in your text editor, and then, immediately after the closing `}` for the `if` statement you created in the previous steps, enter the following code to create a second `if` statement:

```
if (gameLocation[ i] === "home") {
   paragraphs[ 1] .innerHTML = "vs ";
}
```

This statement creates a second `if` statement. It checks if the element in the `gameLocation` array with the index equal to the counter variable is equal to "home" and if so, sets the content of the second paragraph of the current cell to "vs ".

7. In the first comment line before the start of the `addGameInfo()` function, after the word "opponents," insert **and gameLocation values**. The following code shows the completed `addGameInfo()` function:

```
1   // function to place opponents and gameLocation values in
2   // second p element within each table data cell that has an id
3   function addGameInfo() {
4      var paragraphs = "";
5      for (var i = 0; i < 31; i++) {
6         var date = i + 1;
7         var tableCell = document.getElementById("08-" +↵
8            date);
9         paragraphs = tableCell.getElementsByTagName("p");
10        if (gameLocation[ i] === "away") {
11           paragraphs[ 1] .innerHTML = "@ ";
12        }
13        if (gameLocation[ i] === "home") {
14           paragraphs[ 1] .innerHTML = "vs ";
15        }
16        paragraphs[ 1] .innerHTML += opponents[ i] ;
17     }
18  }
```

8. Save your changes to **tt.js**, and then reload **calendar.htm** in your browser. As Figure 3-20 shows, "vs" followed by a space is now displayed before the opponent names in some of the table cells. These dates correspond to the elements in the `gameLocation` array with a value of "home."

Figure 3-20: The calendar.htm document displaying "vs" before the names of home opponents

Note

You could add a third `if` statement to the `addGameInfo()` function to cover the case when an element of the `gameLocation` array has a value of `""`. However, because no text needs to be inserted in such a case, there's no need to add this extra code to your function.

`if/else` Statements

So far you've learned how to use an `if` statement to execute one or more statements if a condition evaluates to a truthy value. In some situations, however, you may want to execute one set of statements when a condition evaluates to a truthy value and another set of statements when the condition evaluates to a falsy value. You can accomplish this by adding an `else` clause to your `if` statement. For instance, suppose you create a web page form that asks users to indicate by clicking an option button whether they invest in the stock market. An `if` statement in the script might contain a conditional expression that evaluates the user's input. If the condition evaluates to `true` (that is, if the user clicked the "yes" option), then the `if` statement would display a web page on recommended stocks. If the condition

Note

In situations where multiple `if` statements can perform the same actions as a single `if/else` statement, the `if/else` construction is preferable because it makes it clear to anyone reading the code that the statements are connected. This is another example of a choice you can make while programming that results in self-documenting code.

The JavaScript code for the calendar.htm document you created earlier uses multiple `if` statements to evaluate the contents of the `gameLocation` array and decide what value to insert into the current table cell. Although the multiple `if` statements function properly, they can be combined into an `if/else` statement. Next, you will replace the multiple if statements in the `addGameInfo()` function with a single `if/else` statement.

To add an `if/else` statement to the `addGameInfo()` function:

1. Return to the **tt.js** document in your text editor.
2. Within the `addGameInfo()` function, within the data block for the `for` loop, insert `/*` before the start of the first `if` statement, and add `*/` after the closing `}` for the second `if` statement, as shown below:

```
1   for (var i = 0; i < 31; i++) {
2      var date = i + 1;
3      var tableCell = document.getElementById("08-" + date);
4      paragraphs = tableCell.getElementsByTagName("p");
5   /*   if (gameLocation[ i] === "away") {
6           paragraphs[ 1] .innerHTML = "@ ";
7      }
8      if (gameLocation[ i] === "home") {
9           paragraphs[ 1] .innerHTML = "vs ";
10     }*/
11     paragraphs[ 1] .innerHTML += opponents[ i] ;
12  }
```

The `if` statements are now treated as comments, and will not be processed by JavaScript interpreters. This allows you to keep the code available for reference without it being part of your program.

3. Within the command block for the `for` statement, add a new line below the closing `*/` you entered in the previous step, and then enter the following code:

```
if (gameLocation[i] === "away") {
   paragraphs[1].innerHTML = "@ ";
}
```

This is the same `if` statement you entered in the previous set of steps. Instead of standing alone, however, this time it will serve as the start of an `if/else` construction.

4. Below the command block you just created, enter the following code:

```
else {
   paragraphs[1].innerHTML = "vs ";
}
```

In place of the `if` statement from the previous set of steps, which had its own condition, this `else` statement simply provides a single line of code to be executed if the original condition evaluates to `false`. The revised `for` statement in the `addGameInfo()` function should match the following code:

```
1   for (var i = 0; i < 31; i++) {
2      var date = i + 1;
3      var tableCell = document.getElementById("08-" + date);
4      paragraphs = tableCell.getElementsByTagName("p");
5   /*    if (gameLocation[i] === "away") {
6            paragraphs[1].innerHTML = "@ ";
7      }
8      if (gameLocation[i] === "home") {
9            paragraphs[1].innerHTML = "vs ";
10     }*/
11     if (gameLocation[i] === "away") {
12        paragraphs[1].innerHTML = "@ ";
13     }
14     else {
15        paragraphs[1].innerHTML = "vs ";
16     }
17     paragraphs[1].innerHTML += opponents[i];
18  }
```

5. Save your changes to **tt.js**, and then reload **calendar.htm** in your browser. As Figure 3-21 shows, the text "@" and "vs" is still inserted into the cells containing August dates. However, note that "vs" is also inserted before the text "(off)" for the four dates when no game will be played. Because you removed the condition specified by the second `if` statement and replaced it with a default value for all instances where the first condition didn't evaluate to `true`, the text "vs" is displayed in cells whose corresponding `gameLocation` values are "home" as well as in those whose value is "". To fix this issue, you need to use a nested `if` statement.

TIPTON TURBINES

Tickets Calendar Players News Community

Calendar
August 2016

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1 @ Lightning	2 @ Combines	3 @ Combines	4 @ Combines	5 vs Lightning	6 vs Lightning
7 vs Lightning	8 vs Lightning	9 vs Barn Raisers	10 vs Barn Raisers	11 vs Barn Raisers	12 @ Sodbusters	13 @ Sodbusters
14 @ Sodbusters	15 @ Sodbusters	16 vs (off)	17 @ River Riders	18 @ River Riders	19 @ River Riders	20 @ Big Dippers
21 @ Big Dippers	22 @ Big Dippers	23 vs (off)	24 vs Sodbusters	25 vs Sodbusters	26 vs Sodbusters	27 vs Combines
28 vs Combines	29 vs Combines	30 vs (off)	31 vs (off)			

Tipton Turbines Baseball • Tipton, Iowa

Figure 3-21: The calendar.htm document populated using an `if/else` statement

Nested `if` and `if/else` Statements

As you have seen, you can use a decision-making statement such as an `if` or `if/else` statement to allow a program to make decisions about which statements to execute. In some cases, however, you may want the statements executed by the decision-making statement to make other decisions. For instance, you may have a program that uses an `if` statement to ask users if they like sports. If users answer yes, you may want to run another `if` statement that asks users whether they prefer team sports or individual sports. You can include any code you like within the code block for an `if` statement or an `if/else` statement, and that includes other `if` or `if/else` statements.

Nesting one decision-making statement within another decision-making statement creates a **nested decision-making structure**. An `if` statement contained within an `if` statement or within an `if/else` statement is called a nested `if` statement. Similarly, an `if/else` statement contained within an `if` or `if/else` statement is called a nested `if/else`

statement. You use nested `if` and `if/else` statements to perform conditional evaluations that must be executed after the original conditional evaluation. For example, the following code evaluates two conditional expressions before the `write()` statement is executed:

```
var salesTotal = 75;
if (salesTotal > 50) {
   if (salesTotal < 100) {
      document.write("<p>The sales total is between 50↵
         and 100.</p>");
   }
}
```

The `document.write()` statement in the preceding example is executed only if the conditional expressions in both `if` statements evaluate to `true`.

The `if/else` statement you added to the `addGameInfo()` function doesn't work correctly because it can't differentiate between `gameLocation` values of `"home"` and `""`. To add back the ability to make this distinction, you'll make the second clause a nested `if` statement that tests for the second condition.

To add a nested `if` statement to the `addGameInfo()` function:

1. Return to the **tt.js** document in your text editor.
2. Within the `addGameInfo()` function, add a new line below the code `else {`, and then enter the code **`if (gameLocation[i] === "home") {`**.
3. Add a new line below the statement `paragraphs[1].innerHTML = "vs ";`, and then type a closing **`}`**. Your completed `if/else` statement containing a nested `if` statement should match the code shown below:

```
if (gameLocation[i] === "away") {
   paragraphs[1].innerHTML = "@ ";
}
else {
   if (gameLocation[i] === "home") {
      paragraphs[1].innerHTML = "vs ";
   }
}
```

The nested `if` statement you just created specifies that the text "vs " should be inserted only if the current element in the `gameLocation` array has a value of "home". This means that any `gameLocation` elements with values of `""` will not meet the condition and the text "vs " will not be added to those cells.

4. Save your changes to **tt.js**, and then reload **calendar.htm** in your browser. As Figure 3-22 shows, the text "vs" is still inserted into the cells for home games, but it is no longer added to cells that contain the text "(off)".

Figure 3-22: The calendar.htm document populated using a nested `if` statement

`else if` Statements

JavaScript supports a compact version of nested `if/else` statements known as an **`else if` construction**. In an `else if` construction, you combine an `else` statement with its nested `if` statement. The resulting statement requires fewer characters to create and is easier to read. For example, you could replace the `else` statement and its nested `if` statement from the previous steps with the following `else if` statement:

```
else if (gameLocation[i] === "home") {
   paragraphs[1].innerHTML = "vs ";
}
```

These three lines of code replace five lines in the previous version. In addition, these lines consist of only a single command block, rather than one nested inside another.

The `else if` construction is commonly used to enhance event listeners so they're backward-compatible with older browsers. Although modern browsers have all standardized on the `addEventListener` method for creating event listeners, Internet Explorer version 8 and earlier used the `attachEvent` method instead. You can use an `else if` statement to create code that checks if either method is supported in the current browser, and if so, adds

```
            return "New York";
            break;
        default:
            return "United States";
            break;
    }
}
document.write("<p>" + city_location("Boston") + "</p>");
```

Note that case labels must be discrete values and cannot use operators. This means that for numeric values, you cannot use greater than, less than, or ranges for case labels—only individual values. If you need your code to make decisions based on comparison operators, such as `> 25`, then `if/else` and `else if` statements are a better choice.

Next, you will modify the `addGameInfo()` function program to use a `switch` statement instead of the `if/else if` statements. Each `case` statement in the modified function will check the value that is passed from the `gameLocation` array. The `switch` statement makes better programming sense than the `if/else if` statements, because it eliminates the need to check the `gameLocation` value multiple times.

To add a `switch` statement to the `addGameInfo()` function:

1. Return to the **tt.js** document in your text editor.
2. Within the `addGameInfo()` function, within the data block for the `for` loop, insert **`/*`** before the start of the `if/then` statement, and add **`*/`** after the closing `}` for the second `if/then` statement, as shown below:

```
for (var i = 0; i < 31; i++) {
   var date = i + 1;
   var tableCell = document.getElementById("08-" + date);
   paragraphs = tableCell.getElementsByTagName("p");
/*    if (gameLocation[i] === "away") {
       paragraphs[1].innerHTML = "@ ";
   }
   if (gameLocation[i] === "home") {
      paragraphs[1].innerHTML = "vs ";
   }*/
/*    if (gameLocation[i] === "away") {
       paragraphs[1].innerHTML = "@ ";
   }
```

```
14      else {
15         if (gameLocation[ i]  === "home") {
16            paragraphs[ 1] .innerHTML = "vs ";
17         }
18      } */
19      paragraphs[ 1] .innerHTML += opponents[ i] ;
20   }
```

3. Within the command block for the `for` statement, add a new line below the closing `*/` you entered in the previous step, and then enter the following code:

```
switch (gameLocation[ i] ) {

}
```

This code uses the `switch` keyword to create a `switch` statement, and specifies in parentheses the expression to evaluate. In this code, the expression is the element from the `gameLocation` array that corresponds to the current value of the counter variable.

4. Within the command block you just created, enter the following code:

```
case "away":
   paragraphs[ 1] .innerHTML = "@ ";
   break;
```

This code uses the `case` keyword to create a `case` statement. The `case` label "away" specifies that if the value of the switch expression—`gamelocation[ i]` —equals "away," the JavaScript processor should execute the code that follows. The first statement adds "@ " as the content for the second paragraph in the current cell, and the second statement, `break;`, specifies that the JavaScript processor should end execution of the `switch` statement.

5. Below the last line of code you entered in the previous step, enter the following code:

```
case "home":
   paragraphs[ 1] .innerHTML = "vs ";
   break;
```

This code creates a second `case` statement. The `case` label "home" specifies that if the value of the `switch` expression equals "home," the JavaScript processor should execute the code that follows. The first statement adds "vs " as the content for the second paragraph in the current cell, and the second statement specifies that the JavaScript processor should end execution of the `switch` statement.

6. Save your changes to **tt.js**, and then reload **calendar.htm** in your browser. The calendar content still matches that shown earlier in Figure 3-21, meaning that the `switch` statement has exactly replicated the results created by the `if/else` and `if` statements.

The following code shows the final version of the tt.js file.

```
// global variables
var daysOfWeek = [ "Sunday", "Monday", "Tuesday", "Wednesday",
   "Thursday", "Friday", "Saturday"];
var opponents = [ "Lightning", "Combines", "Combines",
    "Combines", "Lightning", "Lightning", "Lightning",
    "Lightning", "Barn Raisers", "Barn Raisers",
    "Barn Raisers", "Sodbusters", "Sodbusters", "Sodbusters",
    "Sodbusters", "(off)", "River Riders", "River Riders",
    "River Riders", "Big Dippers", "Big Dippers",
    "Big Dippers", "(off)", "Sodbusters", "Sodbusters",
    "Sodbusters", "Combines", "Combines", "Combines",
    "(off)", "(off)"];
var gameLocation =
   ["away", "away", "away", "away", "home", "home", "home",
    "home", "home", "home", "home", "away", "away", "away",
    "away", "", "away", "away", "away", "away", "away",
    "away", "", "home", "home", "home", "home", "home",
    "home", "", ""];
// function to place daysOfWeek values in header row cells
function addColumnHeaders() {
   var i = 0;
   while (i < 7) {
      document.getElementsByTagName("th")[ i] .innerHTML =↵
         daysOfWeek[ i];
      i++;
   }
}
// function to place day of month value in first p element
// within each table data cell that has an id
function addCalendarDates() {
   var i = 1;
```

```
32      var paragraphs = "";
33      do {
34         var tableCell = document.getElementById("08-" + i);
35         paragraphs = tableCell.getElementsByTagName("p");
36         paragraphs[ 0] .innerHTML = i;
37         i++;
38      } while (i <= 31);
39   }
40   // function to place opponents and gameLocation values in
41   // second p element within each table data cell that has an id
42   function addGameInfo() {
43      var paragraphs = "";
44      for (var i = 0; i < 31; i++) {
45         var date = i + 1;
46         var tableCell = document.getElementById("08-" + date);
47         paragraphs = tableCell.getElementsByTagName("p");
48   /*       if (gameLocation[ i] === "away") {
49            paragraphs[ 1] .innerHTML = "@ ";
50         }
51         if (gameLocation[ i] === "home") {
52            paragraphs[ 1] .innerHTML = "vs ";
53         }*/
54   /*       if (gameLocation[ i] === "away") {
55            paragraphs[ 1] .innerHTML = "@ ";
56         }
57         else {
58            if (gameLocation[ i] === "home") {
59               paragraphs[ 1] .innerHTML = "vs ";
60            }
61         } */
62         switch (gameLocation[ i] ) {
63            case "away":
64               paragraphs[ 1] .innerHTML = "@ ";
65               break;
66            case "home":
```

```
67              paragraphs[ 1] .innerHTML = "vs ";
68              break;
69          }
70          paragraphs[ 1] .innerHTML += opponents[ i] ;
71      }
72  }
73  // function to populate calendar
74  function setUpPage() {
75      addColumnHeaders();
76      addCalendarDates();
77      addGameInfo();
78  }
79  // runs setUpPage() function when page loads
80  if (window.addEventListener) {
81      window.addEventListener("load", setUpPage, false);
82  } else if (window.attachEvent) {
83      window.attachEvent("onload", setUpPage);
84  }
```

Note

Normally when you finish writing JavaScript code, you remove any comments that aren't relevant to the final product. On an actual JavaScript project, you would remove the `if` and `if/else` versions of the code you replaced with a `switch` statement. However, while you are learning, it can be useful to be able to compare different versions of code you've created, so you'll leave these comments in your file.

Short Quiz 3

1. What does an `if` statement do when its condition evaluates to a falsy value?
2. What can you do with an `if/else` statement that you can't do with an `if` statement?
3. Why would you nest decision-making statements?
4. How do you specify possible values for the expression in a `switch` statement?
5. What statement should you include at the end of the code for each `case` label in a `switch` statement? Why is it important to include this statement?