CHAPTER 6

Using Object-Oriented JavaScript

In this chapter, you will:

- Study object-oriented programming
- Work with the Date, Number, and Math objects
- Define custom JavaScript objects
In this chapter, you will learn how to use object-oriented programming techniques in your JavaScript programs. Essentially, object-oriented programming allows you to use and create self-contained pieces of code and data, called objects, which can be reused in your programs. You already have some experience with object-oriented programming, after working with browser objects (including the Window, Document, and Form objects) in Chapters 4 and 5. The browser objects, however, are part of the Web browser itself. The objects you study in this chapter are part of the JavaScript programming language. Additionally, you will learn how to create your own custom JavaScript objects.

**Introduction to Object-Oriented Programming**

The JavaScript programs you have written so far have mostly been self-contained; most code, such as variables, statements, and functions, exists within a script section. For example, you may create a Web page for an online retailer that uses JavaScript to calculate the total for a sales order that includes state sales tax and shipping. However, the retailer may sell different types of products on different Web pages; one page may sell books, another page may sell sporting goods, and so on. If you want to reuse the JavaScript sales total code on multiple Web pages, you must copy all of the statements or recreate them from scratch for each Web page. Object-oriented programming takes a different approach. It allows you to reuse code without having to copy or recreate it.

**Reusing Software Objects**

Object-oriented programming (OOP) refers to the creation of reusable software objects that can be easily incorporated into multiple programs. The term object specifically refers to programming code and data that can be treated as an individual unit or component. (Objects are also called components.) The term data refers to information contained within variables or other types of storage structures. In Chapter 1, you learned that the procedures associated with an object are called methods, and the variables that are associated with an object are called properties or attributes.

Objects can range from simple controls such as a button, to entire programs such as a database application. In fact, some programs consist entirely of other objects. You’ll often encounter objects that have been designed to perform a specific task. For example, in a retail sales program, you could refer to all of the code that calculates the sales
total as a single object. You could then reuse that object over and over again in the same program just by typing the object name.

Popular object-oriented programming languages include C++, Java, and Visual Basic. Using any of these or other object-oriented languages, programmers can create objects themselves or use objects created by other programmers. For example, if you are creating an accounting program in Visual Basic, you can use an object named Payroll that was created in C++. The Payroll object may contain one method that calculates the amount of federal and state tax to deduct, another method that calculates the FICA amount to deduct, and so on. Properties of the Payroll object may include an employee’s number of tax withholding allowances, federal and state tax percentages, and the cost of insurance premiums. You do not need to know how the Payroll object was created in C++, nor do you need to re-create it in Visual Basic. You only need to know how to access the methods and properties of the Payroll object from the Visual Basic program.

One way of understanding object-oriented programming is to compare it to how personal computers (PCs) are assembled. Many companies manufacturer PCs, but few build all of the components that go into a particular unit. Instead, computer manufacturers usually include components from other vendors. For example, there are many different brands of monitors, keyboards, mice, and so on. Even though different manufacturers build each of these hardware components, if they are designed for a PC, then they all share common ways of attaching to the main computer. Monitors plug into standard monitor ports, keyboards plug into standard keyboard ports, mice plug into mouse ports, and so on. In fact, most of today’s hardware components can plug into a Universal Serial Bus (USB) port, which is a standard interface for connecting computer hardware. Just as all hardware components can plug into the same PC, the software components of an object-oriented program can all “plug into” one application.

An object-oriented accounting program is conceptually illustrated in Figure 6-1. In the figure, the accounting program is composed of four separate components that are plugged into the main accounting program: an Accounts Receivable object, an Accounts Payable object, an Expense Reporting object, and the Payroll object. The important thing to understand is that you do not need to rewrite these four objects for the accounting program; the accounting program only needs to call their methods and provide the correct data to their properties.
What Is Encapsulation?

Objects are encapsulated, which means that all code and required data are contained within the object itself. In most cases, an encapsulated object consists of a single computer file that contains all code and required data. Encapsulation places code inside what programmers like to call a black box; when an object is encapsulated, you cannot see “inside” it—all internal workings are hidden. The code (methods and statements) and data (variables and constants) contained in an encapsulated object are accessed through an interface. The term interface refers to the programmatic elements required for a source program to communicate with an object. For example, interface elements required to access a Payroll object might be a method named `calcNetPay()`, which calculates an employee’s net pay, and properties containing the employee’s name and pay rate.

When you include encapsulated classes in your programs, users can see only the methods and properties of the object that you allow them to see. Essentially, the principle of information hiding states that any methods and properties that other programmers do not need to access or know about should be hidden. By removing the ability to see inside the black box, encapsulation reduces the complexity of the code, allowing programmers who use the code to concentrate on the task of integrating the code into their programs. Encapsulation also prevents other
programmers from accidentally introducing a bug into a program, or from possibly even stealing the code and claiming it as their own.

You can compare a programming object and its interface to a handheld calculator. The calculator represents an object, and you represent a program that wants to use the object. You establish an interface with the calculator object by entering numbers (the data required by the object) and then pressing calculation keys (which represent the methods of the object). You do not need to know about, nor can you see, the inner workings of the calculator object. As a programmer, you are concerned only with an object’s methods and properties. To continue the analogy, you are only concerned with the result you expect the calculator object to return. Figure 6-2 illustrates the idea of the calculator interface.

![Figure 6-2 Calculator interface](image)

In JavaScript, the `Document` object is encapsulated, making it a black box. The `write()` and `writeln()` methods are part of the interface that JavaScript can use to communicate with the `Document` object. Figure 6-3 illustrates the concept of a black box using JavaScript and the `Document` object.

![Figure 6-3 Conceptual example of the Document object black box](image)
Another example of an object and its interface is Microsoft Word. Word itself is actually an object made up of numerous other objects. The program window (or user interface) is one object. The items you see in the interface, such as the buttons on the ribbon are used to execute methods. For example, the Bold button executes a bold method. The text of your document is the data you provide to the program. Microsoft Word is a helpful tool that you can use without knowing how the various methods work. You only need to know what each method does. To get full satisfaction out of Microsoft Word, you only need to provide the data (text) and execute the appropriate methods (such as the bold method), when necessary. In the same way, when using objects in your code, you only need to provide the necessary data (such as an employee’s gross pay) and execute the appropriate method (such as the calcNetPay() method).

Understanding Classes

In object-oriented programming, the code, methods, attributes, and other information that make up an object are organized into classes. Essentially, a class is a template, or blueprint, that serves as the basis for new objects. When you use an object in your program, you actually create an instance of the class of the object. An instance is an object that has been created from an existing class. When you create an object from an existing class, you are said to be instantiating the object.

Later in this chapter, you will learn how to create, or instantiate, an object from built-in JavaScript classes and from custom classes that you write yourself. However, as a conceptual example, consider an object named BankAccount that contains methods and properties that you might use to record transactions associated with a checking or savings account. The BankAccount object is created from a BankAccount class. To use the BankAccount class, you create an instance of the class. A particular instance of an object inherits its methods and properties from a class—that is, it takes on the characteristics of the class on which it is based. The BankAccount object, for instance, would inherit all of the methods and properties of the BankAccount class. To give another example, when you create a new word-processing document, which is a type of object, it usually inherits the properties of a template on which it is based. The template is a type of class. The document inherits characteristics of the template such as font size, line spacing, and boilerplate text. In the same manner, programs that include instances of objects inherit the object’s functionality.

Because objects in the browser object model are actually part of the Web browser, you do not need to instantiate them in order to use
them in your programs. For example, you do not need to instantiate a Document object from the Document class in your JavaScript programs because the Web browser automatically instantiates one for you. However, you do need to instantiate some objects from the built-in JavaScript classes that you will study next.

Using Built-In JavaScript Classes

The JavaScript language includes the eleven built-in classes listed in Table 6-1. Each object contains various methods and properties for performing a particular type of task.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>Creates new array objects</td>
</tr>
<tr>
<td>Boolean</td>
<td>Creates new Boolean objects</td>
</tr>
<tr>
<td>Date</td>
<td>Retrieves and manipulates dates and times</td>
</tr>
<tr>
<td>Error</td>
<td>Returns run-time error information</td>
</tr>
<tr>
<td>Function</td>
<td>Creates new function objects</td>
</tr>
<tr>
<td>Global</td>
<td>Stores global variables and contains various built-in JavaScript functions</td>
</tr>
<tr>
<td>Math</td>
<td>Contains methods and properties for performing mathematical calculations</td>
</tr>
<tr>
<td>Number</td>
<td>Contains methods and properties for manipulating numbers</td>
</tr>
<tr>
<td>Object</td>
<td>Represents the base class for all built-in JavaScript classes; contains several of the built-in JavaScript functions</td>
</tr>
<tr>
<td>RegExp</td>
<td>Contains methods and properties for finding and replacing characters in text strings</td>
</tr>
<tr>
<td>String</td>
<td>Contains methods and properties for manipulating text strings</td>
</tr>
</tbody>
</table>

Table 6-1  Built-in JavaScript classes

Instantiating an Object

You can use some of the built-in JavaScript objects directly in your code, while other objects require you to instantiate a new object. The Math object is one that you can use directly in your programs without instantiating a new object. The following example shows how to use the Math object’s PI (π) property in a script:

```html
<script type="text/javascript">
// The following statement prints 3.141592653589793
document.write("The value of pi is " + Math.PI);
</script>
```
Unlike the Math object, an Array object requires you to instantiate a new object before you can use it. As you learned in Chapter 4, arrays are represented in JavaScript by the Array object, which contains a constructor named Array(). You create new arrays in your code using the new keyword and the Array() constructor. The following statement shows an example of how to instantiate an array named deptHeads:

```javascript
var deptHeads = new Array();
```

You may be wondering why the preceding statement instantiates the new object using the var keyword. As you recall, the var keyword is used for declaring variables. The name you use for an instantiated object is really a variable just like an integer or string variable. In fact, programmers use the terms “variable” and “object” interchangeably. The difference is that the data the variable represents happens to be an object instead of a number or string. Recall from Chapter 1 that variables are the values a program stores in computer memory. Recall, too, that the JavaScript language also supports reference data types, which can contain multiple values or complex types of information, as opposed to the single values stored in primitive data types. In other words, in the same manner that you use a variable name to represent a primitive data type, such as an integer, in computer memory you also use a variable name to represent an object. Because the objects you declare in your JavaScript program are actually a certain type of variable, you can use the var keyword to identify them as variables. You are not required to use the var keyword when declaring any variables or objects in your programs; however, it is good practice always to do so.

**Performing Garbage Collection**

If you have worked with other object-oriented programming languages, then you may be familiar with the term garbage collection, which refers to cleaning up, or reclaiming, memory that is reserved by a program. When you declare a variable or instantiate a new object, you are actually reserving computer memory for the variable or object. With some programming languages, you must write code that deletes a variable or object after you are through with it in order to free the memory for use by other parts of your program or by other programs running on your computer. With JavaScript, you do not need to worry about reclaiming memory that is reserved for your variables or objects; JavaScript knows when your program no longer needs a variable or object and automatically cleans up the memory for you.
Short Quiz 1

1. Why do programmers refer to encapsulation as a black box?

2. What is instantiation as it relates to classes, objects, and object-oriented programming?

3. Explain how to conceptually instantiate an object from a class.

Using the Date, Number, and Math Classes

In this section, you will learn how to work with three of the most commonly used JavaScript classes: Date, Number, and Math. First, you will examine the Date class.

Manipulating the Date and Time with the Date Class

You can use dates in your programs to create a calendar, calculate how long it will take to do something, and so on. For instance, a Web page for a dry cleaning business may need to use the current date to calculate when a customer’s dry cleaning order will be ready. The Date class contains methods and properties for manipulating the date and time. The Date class allows you to use the current date and time (or a specific date or time element, such as the current month) in your JavaScript programs. You create a Date object with one of the constructors listed in Table 6-2.

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date()</td>
<td>Creates a Date object that contains the current date and time from the local computer</td>
</tr>
<tr>
<td>Date(milliseconds)</td>
<td>Creates a Date object based on the number of milliseconds that have elapsed since midnight, January 1, 1970</td>
</tr>
<tr>
<td>Date(date_string)</td>
<td>Creates a Date object based on a string containing a date value</td>
</tr>
<tr>
<td>Date(year, month[, date, hours, minutes, seconds, milliseconds])</td>
<td>Creates a Date object with the date and time set according to the passed arguments; the year and month arguments are required</td>
</tr>
</tbody>
</table>

Table 6-2 Date class constructors
The following statement demonstrates how to create a \texttt{Date} object that contains the current date and time from the local computer:

\begin{verbatim}
var today = new Date();
\end{verbatim}

The dates of the month and year in a \texttt{Date} object are stored using numbers that match the actual date and year. However, the days of the week and months of the year are stored in a \texttt{Date} object using numeric representations, starting with zero, similar to an array. The numbers 0 through 6 represent the days Sunday through Saturday, and the numbers 0 through 11 represent the months January through December. The following statement demonstrates how to specify a specific date with a \texttt{Date} constructor function. In this example, the date assigned to the \texttt{independenceDay} variable is July 4, 1776.

\begin{verbatim}
var independenceDay = new Date(1776, 6, 4);
\end{verbatim}

After you create a new \texttt{Date} object, you can then manipulate the date and time in the variable, using the methods of the \texttt{Date} class. Note that the date and time in a \texttt{Date} object are not updated over time like a clock. Instead, a \texttt{Date} object contains the static (unchanging) date and time as of the moment the JavaScript code instantiates the object.

Table 6-3 lists commonly used methods of the \texttt{Date} class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{getDate()}</td>
<td>Returns the date of a \texttt{Date} object</td>
</tr>
<tr>
<td>\texttt{getDay()}</td>
<td>Returns the day of a \texttt{Date} object</td>
</tr>
<tr>
<td>\texttt{getFullYear()}</td>
<td>Returns the year of a \texttt{Date} object in four-digit format</td>
</tr>
<tr>
<td>\texttt{getHours()}</td>
<td>Returns the hour of a \texttt{Date} object</td>
</tr>
<tr>
<td>\texttt{getMilliseconds()}</td>
<td>Returns the milliseconds of a \texttt{Date} object</td>
</tr>
<tr>
<td>\texttt{getMinutes()}</td>
<td>Returns the minutes of a \texttt{Date} object</td>
</tr>
<tr>
<td>\texttt{getMonth()}</td>
<td>Returns the month of a \texttt{Date} object</td>
</tr>
<tr>
<td>\texttt{getSeconds()}</td>
<td>Returns the seconds of a \texttt{Date} object</td>
</tr>
<tr>
<td>\texttt{getTime()}</td>
<td>Returns the time of a \texttt{Date} object</td>
</tr>
<tr>
<td>\texttt{setDate(date)}</td>
<td>Sets the date (1–31) of a \texttt{Date} object</td>
</tr>
<tr>
<td>\texttt{setFullYear(year[, month, day])}</td>
<td>Sets the four-digit year of a \texttt{Date} object; optionally allows you to set the month and the day</td>
</tr>
<tr>
<td>\texttt{setHours(hours[, minutes, seconds, milliseconds])}</td>
<td>Sets the hours (0–23) of a \texttt{Date} object; optionally allows you to set the minutes (0–59), seconds (0–59), and milliseconds (0–999)</td>
</tr>
<tr>
<td>\texttt{setMilliseconds(milliseconds)}</td>
<td>Sets the milliseconds (0–999) of a \texttt{Date} object</td>
</tr>
</tbody>
</table>

\begin{table}[ht]
\centering
\begin{tabular}{|l|l|}
\hline
Method & Description \\
\hline
\texttt{getDate()} & Returns the date of a \texttt{Date} object \\
\texttt{getDay()} & Returns the day of a \texttt{Date} object \\
\texttt{getFullYear()} & Returns the year of a \texttt{Date} object in four-digit format \\
\texttt{getHours()} & Returns the hour of a \texttt{Date} object \\
\texttt{getMilliseconds()} & Returns the milliseconds of a \texttt{Date} object \\
\texttt{getMinutes()} & Returns the minutes of a \texttt{Date} object \\
\texttt{getMonth()} & Returns the month of a \texttt{Date} object \\
\texttt{getSeconds()} & Returns the seconds of a \texttt{Date} object \\
\texttt{getTime()} & Returns the time of a \texttt{Date} object \\
\texttt{setDate(date)} & Sets the date (1–31) of a \texttt{Date} object \\
\texttt{setFullYear(year[, month, day])} & Sets the four-digit year of a \texttt{Date} object; optionally allows you to set the month and the day \\
\texttt{setHours(hours[, minutes, seconds, milliseconds])} & Sets the hours (0–23) of a \texttt{Date} object; optionally allows you to set the minutes (0–59), seconds (0–59), and milliseconds (0–999) \\
\texttt{setMilliseconds(milliseconds)} & Sets the milliseconds (0–999) of a \texttt{Date} object \\
\hline
\end{tabular}
\caption{Commonly used methods of the \texttt{Date} class}
\end{table}
Each portion of a Date object, such as the day, month, year, and so on, can be retrieved and modified using the Date object methods. For example, if you create a new Date object using the statement `var curDate = new Date();`, you can retrieve just the date portion stored in the `curDate` object by using the statement `curDate.getDate();`.

If you want to display the full text for days and months (for example, Wednesday, or January), then you can use a conditional statement to check the value returned by the `getDay()` or `getMonth()` method. For example, the following code uses an `if...else` construct to print the full text for the day of the week returned by the `getDay()` function. Figure 6-4 shows the output when the script is run on a Tuesday.

```javascript
<script type="text/javascript">
var today = new Date();
var curDay = today.getDay();
if (curDay == 0)
    document.write("Today is Sunday.");
else if (curDay == 1)
    document.write("Today is Monday.");
else if (curDay == 2)
    document.write("Today is Tuesday.");
else if (curDay == 3)
    document.write("Today is Wednesday.");
else if (curDay == 4)
    document.write("Today is Thursday.");
else if (curDay == 5)
    document.write("Today is Friday.");
else if (curDay == 6)
    document.write("Today is Saturday.");
</script>
```

You can find a complete listing of Date class methods in the appendix.

The Date class does not contain any properties.
Output of a script with a `getDay()` method

If you need to return the full text of the day or month, you should assign the days of the week or the months of the year to an array. You can then combine the `getDay()` or `getMonth()` method with the array name to return the full text of the day or month. For example, the following code includes an array named `months` with 12 elements that are assigned the full text names of the months of the year:

```javascript
<script type="text/javascript">
var today = new Date();
var months = new Array();
months[0] = "January";
months[1] = "February";
months[2] = "March";
months[3] = "April";
months[4] = "May";
months[5] = "June";
months[6] = "July";
months[7] = "August";
months[8] = "September";
months[9] = "October";
months[10] = "November";
months[11] = "December";
var curMonth = months[today.getMonth()];
document.write("<p>The current month is "
+ curMonth + ".</p>");
</script>
```

In the preceding code, the full text name of the month is assigned to the `curMonth` variable by using the statement `var curMonth = months[today.getMonth()]`. The value of the element is retrieved by placing the `today` object with the `getMonth()` method appended to it between the brackets of the `months` array name. Figure 6-5 shows the output.
Next, you start working on a Web page for Coast City Windsurfing that visitors can use to create group reservations. The files you will need for this project are located in your Chapter folder for Chapter 6. You will use the Date object to generate a monthly calendar that visitors can use to select reservation dates. The script will submit the reservation information to a FormProcessor.html document that is similar to the one you used in Chapter 5. The FormProcessor.html document, located in your Chapter folder for Chapter 6, uses JavaScript code to display the values submitted from a form. The only purpose of the FormProcessor.html document is to display form data and provide a simple simulation of the response you would normally receive from a server-side scripting program.

To create a group reservations page for Coast City Windsurfing that uses the Date object to generate a monthly calendar:

1. Open your text editor, then open the index.html document from your Chapter folder for Chapter 6.
2. Locate <!--[Add code here]--> in the document body, and replace it with the following text and elements in the document body. The form submits the data to the FormProcessor.html document.

```html
<form action="FormProcessor.html" method="get" enctype="application/x-www-form-urlencoded">
</form>
```
3. Add the following elements to the end of the form section:

```html
<table border="0">
  <tr valign="top">
    <td><h3>Windsurfing Date</h3><p><input type="text" name="reservationDate" onclick="displayCalendar()" /></p></td>
    <td><h3>Group Leader</h3></td>
  </tr>
</table>
```

![Figure 6-5](http://example.com/image.png) Output of a script with a getMonth() method
| Last name |<br> | <input type="text" name="leaderLastName" size="40" />
| First name |<br> | <input type="text" name="leaderFirstName" size="40" />
| Telephone |<br> | <input type="text" name="leaderTelephone" size="40" />
| Address |<br> | <input type="text" name="leaderAddress" size="40" />
| City, State, Zip |<br> | <input type="text" name="leaderCity" size="23" />
<input type="text" name="leaderState" size="2" maxlength="2" />
<input type="text" name="leaderZip" size="5" maxlength="5" />
<p><input type="submit" value="Submit Group Reservation"></p>
</td>
</tr>
</table>

4. Add the following script section to the document head:

```html
<script type="text/javascript">
/* <![CDATA[ */
/* ]]> */
</script>
```

5. Add the following statements to the script section, which declare a Date object at the global level, return the number representing the current month, and declare an array to contain the full text for the months of the year. The function creates a variable named dateToday and assigns it as the value of the Windsurfing Date field in the form.

```javascript
var dateObject = new Date();
var month = dateObject.getMonth();
var monthArray = new Array("January","February","March","April","May","June","July","August","September","October","November","December");
function getTodayDate() {
    var dateToday = monthArray[month] + " " + dateObject.getDate() + ", " + dateObject.getFullYear();
    document.forms[0].reservationDate.value = dateToday;
}
```

6. Add an `onload` event handler to the opening `<body>` tag that calls the `getTodayDate()` function, as follows:

```html
<body onload="getTodayDate()">
```
7. Save the document and open it in your Web browser. Figure 6-6 shows how the document appears in a Web browser. The current date should appear in the Windsurfing Date field.

![Figure 6-6 Coast City Windsurfing group reservations page](Coast City Windsurfing - Mozilla Firefox.png)

8. Close your Web browser window.

Next, you modify the Coast City Windsurfing page so that it includes functionality that allows users to select a date from a separate date picker window.

To add date picking functionality to the Coast City Windsurfing page:

1. Return to the `index.html` document in your text editor.
2. Add the following anchor element before the closing `<p>` tag in the paragraph element that contains the `reservationDate` field. The anchor element contains an `onclick` event handler that calls a function named `displayCalendar()`, which you create next.
3. Next, start building the following `displayCalendar()` function. The statements in the function use the `window.open()` method to create a new window that will display the calendar. Notice that the statements use the `calendarWin` variable (which represents the new window) with `document.write()` statements to create the new window. In order for the contents of the window to be well formed, the code includes `document.write()` statements that create the `<!DOCTYPE>` declaration, `<html>` element, and header information. The last statement begins creating the table that will display the calendar. Add the following function to the end of the script section:

```javascript
function displayCalendar() {
    calendarWin = window.open('', 'CalWindow', "status=no,resizable=yes,width=400,height=220,
left=200,top=200");
    calendarWin.focus();
    calendarWin.document.write("<!DOCTYPE html PUBLIC -//W3C//DTD XHTML 1.0 Strict//EN' "
'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'><html xmlns='http://www.w3.org/1999/xhtml'>
<head><title>Coast City Windsurfing</title>
<meta http-equiv='content-type' content='text/html;charset=iso-8859-1'/>
<link rel='stylesheet' href='js_styles.css' type='text/css' />
</head><body>");
    calendarWin.document.write("<colgroup span='7' width='50' />");
    calendarWin.document.write("<tr><td colspan='7' align='center'><strong>" + monthArray[month] + " " + dateObject.getFullYear() + "</strong></td></tr>");
    calendarWin.document.write("<tr align='center'>
    <td>Sun</td><td>Mon</td><td>Tue</td>
    <td>Wed</td><td>Thu</td><td>Fri</td>
    <td>Sat</td></tr>");
    calendarWin.document.write("<tr align='center'>
    <td>Sun</td><td>Tue</td><td>Wed</td><td>Thu</td><td>Fri</td><td>Sat</td></tr>");
    calendarWin.document.write("<tr align='center'>
    <td>Sun</td><td>Tue</td><td>Wed</td><td>Thu</td><td>Fri</td><td>Sat</td></tr>");
}
```

4. Start building the table by adding the following statements to the end of the `displayCalendar()` function. Notice that the second statement uses the `monthArray` and the `month` variables to print the name of the current month and the `getFullYear()` method of the `dateObject` variable to print the year.

```javascript
calendarWin.document.write("<tbody>");
calendarWin.document.write("<tr><td align='center'><strong>" + monthArray[month] + " " + dateObject.getFullYear() + "</strong></td></tr>");
calendarWin.document.write("<tr align='center'>
    <td>Sun</td><td>Tue</td><td>Wed</td><td>Thu</td><td>Fri</td><td>Sat</td></tr>");
calendarWin.document.write("<tr align='center'>
    <td>Sun</td><td>Tue</td><td>Wed</td><td>Thu</td><td>Fri</td><td>Sat</td></tr>");
calendarWin.document.write("<tr align='center'>
    <td>Sun</td><td>Tue</td><td>Wed</td><td>Thu</td><td>Fri</td><td>Sat</td></tr>");
```
5. Add the following statements to the end of the `displayCalendar()` function. The first statement uses the `setDate()` function to set the date of the `Date` object to the first day of the month. The second statement uses the `getDay()` function to determine which day of the week it is. For instance, if the `getDay()` function returns a value of 3, then the first day of the month starts on Wednesday. Any table cells for days in the first week that are part of the previous month are assigned a nonbreaking space character (&nbsp;) by the `for` statement.

```javascript
dateObject.setDate(1);
var dayOfWeek = dateObject.getDay();
for (var i=0; i<dayOfWeek; ++ i) {
calendarWin.document.write("<td>&nbsp;</td>";
}
```

6. Add the following statements to the end of the `displayCalendar()` function. The first statement calculates the number of days in the first week that require date values. The second statement declares a variable named `dateCounter` that is used to keep track of the next date to write to the calendar. The `for` statement then finishes creating the first row in the table, which represents the first week of the month. Notice that the `for` statement creates anchor elements for each of the dates. When a user clicks on a date, an `onclick` event uses the `opener` property of the `self` object to assign the data value to the Windsurfing Date field in the form on the main Coast City Windsurfing Web page and then close the calendar window. (Recall that the `opener` property refers to the window that opened the current window.)

```javascript
var daysWithDates = 7 - dayOfWeek;
var dateCounter = 1;
for(var i=0; i<daysWithDates; ++i) {
    var curDate = monthArray[moment] + " "
    + dateCounter + ", "
    + dateObject.getFullYear();
calendarWin.document.write("<td><a href='' onclick='self.opener.document.forms[0].reservationDate.value="" + curDate + "\";self.close()'></a></td>";
    ++dateCounter;
}
```
7. Next, add the following variable declaration and if...else statement to determine the number of days in the month variable, which represents the Date object. You need the number of days in the current month in order to determine the number of days to display in the calendar. Add the statements to the end of the displayCalendar() function.

```javascript
var numDays = 0;
// January, March, May, July, August, October, December
if (month == 0 || month == 2 || month == 4 
|| month == 6 || month == 7 || month == 9 
|| month == 11)
    numDays = 31;
// February
else if (month == 1)
    numDays = 28;
// April, June, September, November
else if (month == 3 || month == 5 || month == 8 
|| month == 10)
    numDays = 30;
```

8. Next, add the following for statement to the end of the displayCalendar() function. The calendar needs to consist of six body rows in order to display all of the dates for each month. You already added the first row in Steps 6 and 7. The following for statement adds the remaining five rows, starting and ending each row with <tr align='center'> and </tr>:

```javascript
for (var rowCounter = 0; rowCounter < 5; ++rowCounter) {
    var weekDayCounter = 0;
    calendarWin.document.write("<tr align='center'>");
    calendarWin.document.write("</tr>");
}
```

9. Add the following code between the two write() statements in the for loop. This code adds the dates for each week. The while statement loops through the seven days in a week. While the dateCounter variable is less than or equal to the numDays variable (which represents the total number of days in the month), a table cell and the value of the dateCounter are created for each day. If the dateCounter variable is greater than the numDays variable, nonbreaking characters are added to each table cell for the remaining days in the week that are not part of the current month.
while (weekDayCounter < 7) {
    var curDate = monthArray[month] + " "
        + dateCounter + ", "
        + dateObject.getFullYear();
    if (dateCounter <= numDays)
        calendarWin.document.write("<td><a href='
            onclick='self.opener.document.forms[0].reservationDate.value=" + curDate
            + "';self.close();'>" + dateCounter
            + "</a></td>");
    else
        calendarWin.document.write("<td>&nbsp;</td>");
    ++weekDayCounter;
    ++dateCounter;
}

10. Finally, add the following statements to the end of the
displayCalendar() function. The first statement closes the
<table>, <body>, and <html> elements. The second state-
ment, which calls the close() method of the new window’s
Document object, notifies the Web browser window that you
are finished writing to the window and the document should
be displayed. (You learn more about the Document object’s
close() method in Chapter 10.)
    calendarWin.document.write("</table></body></html>");
    calendarWin.document.close();

11. Save the index.html document, open it in your Web browser,
and then click the Select Date link to display the calendar
window. Figure 6-7 shows how the calendar window appears.
Click a date in the calendar window. The date should be
added to the Windsurfing Date field on the Group Reserva-
tions page and the calendar window should close.

Next, you complete the calendar functionality by modifying the script so it displays different months instead of just the current month.

To add functionality to the calendar script so it displays different months:

1. Return to the index.html document in your text editor.

2. Modify the first statement in the displayCalendar() function definition as follows so that it accepts a single parameter named whichMonth:

   ```javascript
   function displayCalendar(whichMonth) {
   ```
3. Locate the following statement in the `displayCalendar()` function:

```javascript
calendarWin.document.write("<tr><td colspan='7' align='center'><strong>" + monthArray[month] + " " + dateObject.getFullYear() + "</strong></td></tr>");
```

4. Replace the preceding statement with the following statements. The `if` statement determines whether the value assigned to the `whichMonth` parameter is either −1 or 1. If the value is −1, then the `setMonth()` function sets the date assigned to the date object to one month earlier by subtracting a value of one from the value returned with the `getMonth()` function. If the value is 1, then the `setMonth()` function sets the date assigned to the date object to one month later by adding a value of one to the value returned with the `getMonth()` function. The variable declaration statement then uses another `getMonth()` function to assign the new month to the `month` variable. The `document.write()` statement builds the same header row at the beginning of the calendar that you saw in the last exercise, but this time it also creates Previous and Next links that rerun the `displayCalendar()` function when they are clicked.

```javascript
if (whichMonth == -1)  
dateObject.setMonth(dateObject.getMonth() - 1);
else if (whichMonth == 1)  
dateObject.setMonth(dateObject.getMonth() + 1);
var month = dateObject.getMonth();
calendarWin(document.write("<tr><td colspan='7' align='center'><strong>" + monthArray[month] + " " + dateObject.getFullYear() + "</strong></td></tr>");
```

5. Save the `index.html` document, open it in your Web browser, and then click the Select Date link to display the calendar window. The calendar window now includes Previous and Next links, as shown in Figure 6-8.

Be sure to type the text string that is printed with the `calendarWin.document.write()` statement on a single line.
6. Test the **Previous** and **Next** links. The calendar window should update and display the correct dates for the selected month. Click a date in the calendar window to ensure that the date is still added to the Windsurfing Date field on the Group Reservations page and that the calendar window closes.

7. Close your Web browser window.

### Manipulating Numbers with the **Number** Class

The **Number class** contains methods for manipulating numbers and properties that contain static values representing some of the numeric limitations in the JavaScript language (such as the largest positive number that can be used in JavaScript). While you can create a **Number** object using a statement similar to `var myNum = new Number();`, you are not required to. Instead, you can simply append the name of any **Number** class method or property to the name of an existing variable that contains a numeric value.

### Using **Number** Class Methods

Table 6-4 lists the methods of the **Number** class.
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number()</td>
<td>Number object constructor</td>
</tr>
<tr>
<td>toExponential((\text{decimals}))</td>
<td>Converts a number to a string in exponential notation using a specified number of decimal places</td>
</tr>
<tr>
<td>toFixed((\text{decimals}))</td>
<td>Converts a number to a string with a specified number of decimal places</td>
</tr>
<tr>
<td>toLocaleString()</td>
<td>Converts a number to a string that is formatted with local numeric formatting conventions</td>
</tr>
<tr>
<td>toPrecision((\text{decimals}))</td>
<td>Converts a number to a string with a specific number of decimal places, in either exponential notation or in fixed notation</td>
</tr>
<tr>
<td>toString((\text{radix}))</td>
<td>Converts a number to a string using a specified radix</td>
</tr>
<tr>
<td>valueOf()</td>
<td>Returns the numeric value of a Number object</td>
</tr>
</tbody>
</table>

**Table 6-4** Number class methods

The primary reason for using any of the “to” methods listed in Table 6-4 is to convert a number to a string value with a specific number of decimal places that will be displayed to a user. If you don’t need to display the number for a user, there is no need to use any of the methods. The most useful Number class method is the `toFixed()` method, which you can use to display a numeric value with a specified number of decimal places. For example, you may have a number in your program that represents a dollar value. However, depending on the result of a calculation or a value entered by a user, the number may contain more than the two decimal places that are acceptable in a currency value. The following code shows a simple example of a numeric variable named `salesTotal` that is assigned a value of 49.95. If you apply a discount of 10% to the variable, the new number is equal to 44.955. Before displaying the value, the `write()` statement uses the `toFixed()` method to convert the value of the `salesTotal` variable to a string containing two decimal places.

```javascript
var salesTotal = 49.95;
var discount = salesTotal * .1;
salesTotal -= discount; // new value is 44.955
document.write("\$\n + salesTotal.toFixed(2)); // displays $44.96
```

Another useful Number class method is the `toLocaleString()` method, which you can use to convert a number to a string that is formatted with local numeric formatting conventions. For example, with American numeric formatting conventions, you separate thousands with a comma. The following statements demonstrate how to convert the number 1210349 to the string $1,210,349:
Firefox displays the number in the code to the right without decimal places ($1,210,349), whereas Internet Explorer displays it with decimal places ($1,210,349.00).

```javascript
var salesTotal = 1210349;
salesTotal = salesTotal.toLocaleString();
document.write("\$" + salesTotal); // displays $1,210,349
```

By default, Internet Explorer displays two decimal places for numbers that are converted with the `toLocaleString()` method, whereas Firefox displays the number as a whole number. To convert a numeric value to a specified number of decimal places and to a local string is not intuitive in Firefox. First, you call the `toFixed()` method, which converts the number to a string with the specified number of decimals. Then, because it's a string, you need to convert it back to floating-point number with the `parseFloat()` function. Finally, you call the `toLocaleString()` method to convert the number to a string that is formatted with the local numeric formatting conventions. The following code converts the number 1210349.4567 to the string $1,210,349.46:

```javascript
var salesTotal = 1210349.4567;
salesTotal = salesTotal.toFixed(2);
salesTotal = parseFloat(salesTotal);
salesTotal = salesTotal.toLocaleString();
document.write("\$" + salesTotal); // displays $1,210,349.46
```

Although Internet Explorer will successfully display decimal places for numbers that are converted with the `toLocaleString()` method, you should use the `toFixed()` and `parseFloat()` methods to ensure that code that uses the `toLocaleString()` method is compatible with both Internet Explorer and Firefox.

### Accessing Number Class Properties

Table 6-5 lists the properties of the `Number` class. Note that there is little reason for you to use these properties. However, they are listed here for the sake of completeness.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_VALUE</td>
<td>The largest positive number that can be used in JavaScript</td>
</tr>
<tr>
<td>MIN_VALUE</td>
<td>The smallest positive number that can be used in JavaScript</td>
</tr>
<tr>
<td>NaN</td>
<td>The value NaN, which stands for “not a number”</td>
</tr>
<tr>
<td>NEGATIVE_INFINITY</td>
<td>The value of negative infinity</td>
</tr>
<tr>
<td>POSITIVE_INFINITY</td>
<td>The value of positive infinity</td>
</tr>
</tbody>
</table>

| Table 6-5 Number class properties |
Next, you add code to the Group Reservations page that calculates group discounts.

To add code to the Group Reservations page that calculates group discounts:

1. Return to the index.html document in your text editor.

2. Add the following text and elements above the paragraph containing the Submit button. The text and elements display group discounts and the first <input> element allows users to enter the number of windsurfers in their groups. The onchange event handler in the first <input> element then calls a function named calcGroupDiscount(), which will calculate the group discount according to the size of the group. Notice that an argument of this.value is passed to the function.

   <h3>Group Discounts</h3>
   <ul>
      <li>Daily Rate: $49</li>
      <li>5–10 windsurfers: 10% discount</li>
      <li>11–24 windsurfers: 20% discount</li>
      <li>25+ windsurfers: 25% discount</li>
   </ul>
   How many windsurfers are in your group?
   <input type="text" size="5" value="0"
   onchange="calcGroupDiscount(this.value)" />
   Your group rate is $<input type="text"
   name="discount" size="60" value="0"
   readonly="readonly"
   class="total" />

3. Add the following calcGroupDiscount() function to the end of the script section. The if...else statements in the function calculate the group discount according to the value assigned to the groupSize parameter. The last statement in the function then assigns the discount rate to the discount field in the form.

   function calcGroupDiscount(groupSize) {
      var dailyRate = 49;
      if (groupSize >= 5 && groupSize <= 10)
         dailyRate = 49 / 1.1;
      else if (groupSize > 10 && groupSize < 25)
         dailyRate = 49 / 1.2;
      else if (groupSize > 24)
         dailyRate = 49 / 1.25;
      var groupRate = groupSize * dailyRate;
      document.forms[0].discount.value = groupRate;
   }
4. Save the **index.html** document and then open it in your Web browser. Enter a number in the group size field, then press the **Tab** key or click off the field so that the `calcGroupDiscount()` function is called by the `onchange` event handler. Notice that the group rate field displays all the digits in the calculated number. For example, Figure 6-9 shows a group rate of $356.3636363636363 after entering a value of 8 in the group size field.

![Figure 6-9 Group Reservations page after adding the Group Discounts section](image)

5. Return to the **index.html** document in your text editor

6. Modify the last statement in the `calcGroupDiscount()` function as follows so that the `groupRate` includes the `toFixed()` method to convert the value of the `groupRate` variable to two decimal places.

   ```javascript
   document.forms[0].discount.value = groupRate.toFixed(2);
   ```

7. Save the **index.html** document and then reopen it in your Web browser. Enter a number in the group size field, then press the **Tab** key or click off the field so that the `calcGroupDiscount()` function is called by the `onchange` event handler. The group rate field should display the calculated number with two decimal places.

8. Close your Web browser window.
Performing Math Functions with the Math Class

The Math class contains methods and properties for performing mathematical calculations in your programs.

Using Math Class Methods

Table 6-6 lists the methods of the Math class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(x)</td>
<td>Returns the absolute value of x</td>
</tr>
<tr>
<td>acos(x)</td>
<td>Returns the arc cosine of x</td>
</tr>
<tr>
<td>asin(x)</td>
<td>Returns the arc sine of x</td>
</tr>
<tr>
<td>atan(x)</td>
<td>Returns the arc tangent of x</td>
</tr>
<tr>
<td>atan2(x,y)</td>
<td>Returns the angle from the x-axis</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>Returns the value of x rounded to the next highest integer</td>
</tr>
<tr>
<td>cos(x)</td>
<td>Returns the cosine of x</td>
</tr>
<tr>
<td>exp(x)</td>
<td>Returns the exponent of x</td>
</tr>
<tr>
<td>floor(x)</td>
<td>Returns the value of x rounded to the next lowest integer</td>
</tr>
<tr>
<td>log(x)</td>
<td>Returns the natural logarithm of x</td>
</tr>
<tr>
<td>max(x,y)</td>
<td>Returns the larger of two numbers</td>
</tr>
<tr>
<td>min(x,y)</td>
<td>Returns the smaller of two numbers</td>
</tr>
<tr>
<td>pow(x,y)</td>
<td>Returns the value of x raised to the y power</td>
</tr>
<tr>
<td>random()</td>
<td>Returns a random number</td>
</tr>
<tr>
<td>round(x)</td>
<td>Returns the value of x rounded to the nearest integer</td>
</tr>
<tr>
<td>sin(x)</td>
<td>Returns the sine of x</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>Returns the square root of x</td>
</tr>
<tr>
<td>tan(x)</td>
<td>Returns the tangent of x</td>
</tr>
</tbody>
</table>

Table 6-6: Math class methods

Unlike the Array, Date, and Number classes, the Math class does not contain a constructor. This means that you cannot instantiate a Math object using a statement such as var mathCalc = new Math(). Instead, you use the Math object and one of its methods or properties directly in your code. For example, the sqrt() method returns the square root of a number. The following code shows how to use the sqrt() method to determine the square root of 144:

```javascript
var curNumber = 144;
squareRoot = Math.sqrt(curNumber); // returns '12'
document.write("The square root of " + curNumber + " is " + squareRoot);
```
### Accessing Math Class Properties

Table 6-7 lists the properties of the Math class.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Euler's constant e, which is the base of a natural logarithm; this value is approximately 2.7182818284590452354</td>
</tr>
<tr>
<td>LN10</td>
<td>The natural logarithm of 10, which is approximately 2.302585092994046</td>
</tr>
<tr>
<td>LN2</td>
<td>The natural logarithm of 2, which is approximately 0.6931471805599453</td>
</tr>
<tr>
<td>LOG10E</td>
<td>The base-10 logarithm of e, the base of the natural logarithms; this value is approximately 0.4342944819032518</td>
</tr>
<tr>
<td>LOG2E</td>
<td>The base-2 logarithm of e, the base of the natural logarithms; this value is approximately 1.4426950408889634</td>
</tr>
<tr>
<td>PI</td>
<td>A constant representing the ratio of the circumference of a circle to its diameter, which is approximately 3.1415926535897932</td>
</tr>
<tr>
<td>SQRT1_2</td>
<td>The square root of 1/2, which is approximately 0.7071067811865476</td>
</tr>
<tr>
<td>SQRT2</td>
<td>The square root of 2, which is approximately 1.4142135623730951</td>
</tr>
</tbody>
</table>

Table 6-7: Math class properties

As an example of how to use the properties of the Math object, the following code shows how to use the PI property to calculate the area of a circle based on its radius. The code also uses the `round()` method to round the value returned to the nearest whole number.

```javascript
var radius = 25;
var area = Math.round(Math.PI * radius * radius); // return 1963
document.write("A circle with a radius of " + radius + " has an area of " + area);
```

Next, you modify the `calcGroupDiscount()` function so that it uses the `round()` function of the Math object to round the group discount to the nearest integer instead of displaying decimal places. If you entered a large number in the last exercise when you tested the script, you may have noticed that although the group discount displayed only two decimal places, the number was not formatted with commas or whatever the formatting convention is for your locale. For example, if you entered a value of 38, the group rate is displayed as $1489.60. In American numeric formatting, the convention is to include commas to separate thousands. This means that the value $1489.60 should display as $1,489.60. To ensure that numbers are correctly displayed according to local numeric formatting conventions, you must use
the toLocaleString() function of the Number class, which you will also add to the calcGroupDiscount() function.

**To modify the Group Reservations page so it uses the round() function of the Math object and the toLocaleString() function:**

1. Return to the index.html document in your text editor.
2. Add the following statement immediately after the statement in the calcGroupDiscount() function that declares and initializes the groupRate variable. This statement uses the built-in parseFloat() function to convert the text string to a number and the round() function of the Math object to round the number to the nearest integer.

   ```javascript
   groupRate = Math.round(parseFloat(groupRate));
   ```

3. Modify the last statement in the calcGroupDiscount() function so that it calls the toLocaleString() function instead of the toFixed() function, as follows:

   ```javascript
   document.forms[0].discount.value = groupRate.toLocaleString();
   ```

4. Save the index.html document and then reopen it in your Web browser. Enter a number larger than 25 in the group size field, then press the Tab key or click off the field so that the calcGroupDiscount() function is called by the onchange event handler. The value in the group rate field should be displayed with a comma separator and no decimal places. Figure 6-10 shows how the page appears after entering a value of 35 in the group size field.
Figure 6-10  Group Reservations page after adding Math.round() and toLocaleString() functions

5. Close your Web browser window.

Short Quiz 2

1. How do you refer to the days of the week, dates of the month, and months of the year when working with the Date class?

2. What is the primary reason for using any of the “to” methods in the Number class?

3. How do you use the Math object and one of its methods or properties directly in your code?

Defining Custom JavaScript Objects

JavaScript is not a true object-oriented programming language. You can base objects in your programs on built-in JavaScript classes such as the Array and Date objects. However, you cannot create your own classes in JavaScript. For this reason, JavaScript is said to be an object-based programming language instead of an object-oriented programming language.
Nevertheless JavaScript does allow you to define your own custom objects. Unlike objects that are based on classes, custom objects in JavaScript are not encapsulated, which means that other programmers who use your custom object can see inside of the black box. Even though custom JavaScript objects cannot be encapsulated, you may find them useful, especially if you need to replicate the same functionality an unknown number of times in a script. For example, you may have a Web site that allows customers to place online orders for concert tickets. For each order, you may want to create a new object that uses properties to store information such as the customer's name, concert name, number of tickets, concert date, and so on. The object may also contain methods that calculate sales tax and sales total. Although you could use standard functions and variables to create the same functionality, the ability to treat each order as a self-contained object would make your job as a programmer a little easier.

Declaring Basic Custom Objects

Although JavaScript is not a true object-oriented programming language, you can create basic objects and properties by using the Object object. To declare a custom object with the Object object, you use the following statement:

```javascript
var objectName = new Object();
```

You can also create a custom object by assigning a pair of empty braces to a variable name, as follows:

```javascript
var objectName = {};
```

After you create a custom object, you can assign properties to the object by appending the property name to the object name with a period. For example, the following code creates a new object named ConcertTickets and assigns four properties to it: customerName, concertName, ticketQuantity, and concertDate. You can then access the values assigned to each property the same as you would for other types of objects, as demonstrated with the `document.write()` statements. Notice that the concertDate property is created as a Date object that stores the date and time of the concert. Figure 6-11 shows the output.

```javascript
var ConcertTickets = new Object();
ConcertTickets.customerName = "Don Gosselin";
ConcertTickets.concertName = "Jimmy Buffett";
ConcertTickets.ticketQuantity = 2;
ConcertTickets.concertDate = new Date(2010, 6, 18, 20);
```
Chapter 6 Using Object-Oriented JavaScript

```javascript
document.write("<h1>Ticket Order</h1>");
document.write("Customer: " + ConcertTickets.customerName);
document.write("Concert: " + ConcertTickets.concertName);
document.write("Quantity: " + ConcertTickets.ticketQuantity);
document.write("Date: " + ConcertTickets.concertDate.toLocaleString());
```

![Ticket Order Image](image)

**Figure 6-11** Output of custom object properties

Custom objects created as described in this section are limited to containing only properties. Although you may find it useful to create objects in this manner to organize complex data, in most cases you can use standard variables just as effectively. Objects are most useful when they contain both properties and methods. To create custom objects that contain methods, you must use constructor functions, which are described later in this chapter.

Next, you start adding a Group Members section to the Coast City Windsurfing page. This section allows you to enter information about each windsurfer in the group, including name, address, and phone number.

**To add a Group Members form section to the Coast City Windsurfing page:**

1. Return to the `index.html` document and delete the following text and elements:

   ```html
   <p>How many windsurfers are in your group?
   <input type="text" size="5" value="0"
   onchange="calcGroupDiscount(this.value)"></p>
   ```
2. Add the following code above the paragraph that displays the group rate:

```html
</td>
<td>
<h3>Group Members</h3>
<p><input type="button" value="Add Windsurfer" />
<input type="button" value="Delete Windsurfer" />
<input type="button" value="Update Info" /></p>
<table border="0">
<tr>
<td><select name="contacts" size="13"
    style="width: 150px">
    <option value="contacts">Group Members</option>
</select></td>
<td>Last name<br />
<input type="text" name="lastname" size="50" />
First name<br />
<input type="text" name="firstname" size="50" />
Telephone<br />
<input type="text" name="telephone" size="50" />
Address<br />
<input type="text" name="address" size="50" />
City, State, Zip<br />
<input type="text" name="city" size="34" />
<input type="text" name="state" size="2"
    maxlength="2" />
<input type="text" name="zip" size="5"
    maxlength="5" /></td>
</tr>
</table>
```

3. Add the following array declaration to the end of the script section. The array will contain other objects that represent each windsurfer added to the Group Members list. Using an array to store objects makes it easier to manage each group member as it is added to and deleted from the program.

```javascript
var contactList = new Array();
```

4. Save the `index.html` document and open it in your Web browser. You still need to add functionality to the section before you can enter any windsurfers information. Figure 6-12 shows how the Web page appears after adding the Group Members section.
5. Close your Web browser window.

Defining Constructor Functions

You can define your own custom objects by using a constructor function, which is a function that is used as the basis for a custom object. (Another term for constructor function is object definition.) As with traditional class-based objects, JavaScript objects inherit all the variables and statements of the constructor function on which they are based. Any JavaScript function can serve as a constructor. The following code defines a function named ConcertTickets() with four parameters that can serve as a constructor function:

```javascript
function ConcertTickets(customer, concert, tickets, eventDate) {
    ...
}
```

Use a statement similar to the following to instantiate an instance of a ConcertTickets object:

```javascript
var newOrder = new ConcertTickets();
```

Next, you add a constructor function to the Group Reservations page.

To add a constructor function to the Group Reservations page:

1. Return to the index.html document in your text editor.
2. Type the following constructor function above the
   contactList declaration statement in the script section:
   
   ```javascript
   function Contact() {
   }
   ```

3. Save the index.html document.

Working with Object Properties

In this section, you will learn how to use properties with your custom
JavaScript objects. First, you will learn how to add new properties.

Adding Properties

To add a property to a constructor function, you must add a state-
ment to the function body that uses the `this` keyword with the
following syntax: `this.property_name = value`. In the case of a
custom JavaScript object, the `this` keyword refers to the object that
calls the constructor function. For example, the following construc-
tor function includes four properties: `customerName`, `concertName`,
ticketQuantity, and `concertDate`.

```javascript
function ConcertTickets(customer, event, quantity, eventDate) {
    this.customerName = customer; // customer name
    this.concertName = event; // event name
    this.ticketQuantity = quantity; // number of tickets
    this.concertDate = eventDate; // concert date
}
```

The statements in the preceding constructor function use the `this`
keyword to assign the values of the four arguments (`customer`, `event`,
`quantity`, and `eventDate`) to the `customerName`, `concertName`,
ticketQuantity, and `concertDate` properties of whichever object
called the function. The use of the `this` reference is one of the pri-
mary differences between standard functions and constructor func-
tions. Standard functions do not include a `this` reference, because
they are not used as the basis of objects.

The following code declares a ticketOrder object based on the
ConcertTickets() constructor function, assigns values to its four
properties, and then prints the properties. The output is the same as
that shown in Figure 6-11.

```javascript
var ticketOrder = new ConcertTickets();
ticketOrder.customerName = "Don Gosselin";
ticketOrder.concertName = "Jimmy Buffett";
ticketOrder.ticketQuantity = 2;
ticketOrder.concertDate = new Date(2010, 6, 18, 20);
```
document.write("<h1>Ticket Order</h1>");
document.write("<p>Customer: " + ticketOrder.customerName);
document.write("Concert: " + ticketOrder.concertName);
document.write("<br />Quantity: " + ticketOrder.ticketQuantity);
document.write("<br />Date: " + ticketOrder.concertDate.toLocaleString());
</p>);

You can also assign values to the properties of an object when
you first instantiate the object by using statements similar to the
following:

```javascript
var ticketOrder = new ConcertTickets("Don Gosselin",
      "Jimmy Buffett", 2, new Date(2010, 6, 18, 20));
document.write("<h1>Ticket Order</h1>");
document.write("<p>Customer: " + ticketOrder.customerName);
document.write("Concert: " + ticketOrder.concertName);
document.write("<br />Quantity: " + ticketOrder.ticketQuantity);
document.write("<br />Date: " + ticketOrder.concertDate.toLocaleString());
</p>);
```

Next, you will add properties to the Contact constructor function.

To add properties to the Contact constructor function, along
with a function that copies the values from the Contacts form to
the properties:

1. Return to the `index.html` document in your text editor.
2. Add the following properties to the Contact constructor
   function. Each property is initially assigned an empty string.

   ```javascript
   this.lastName = "";
   this.firstName = "";
   this.telephone = "";
   this.address = "";
   this.city = "";
   this.state = "";
   this.zip = "";
   ```
3. Save the `index.html` document.

**Enumerating Custom Object Properties**

Some custom objects can contain dozens of properties. For example,
a script may create new custom object properties that store sales
prices for each item a customer wants to purchase. Suppose that
you want to discount the individual sales prices by 10% of any items
that cost more than $100. Because there is no way to determine in
advance which items a customer will purchase, you have no way of
knowing which properties have been added to the object for each
individual customer. To execute the same statement or command
block for all the properties within a custom object, you can use the
**for...in statement**, which is a looping statement similar to the `for` statement. The syntax of the `for...in` statement is as follows:

```javascript
for (variable in object) {
    statement(s);
}
```

The variable name in the `for...in` statement constructor holds an individual object property. The object name in the constructor represents the name of an object that has been instantiated in a program. Unlike the other loop statements, the `for...in` statement does not require a counter or any other type of code to control how the loop functions. Instead, the `for...in` statement automatically assigns each property in an object to the variable name, performs the necessary statements on the property, then moves to the next property and starts over. The `for...in` statement ends automatically once it reaches the last property in an object. A typical use of the `for...in` statement is to retrieve the names of properties within an object, as shown in the following code:

```javascript
var ConcertTickets = new Object();
ConcertTickets.customerName = "Don Gosselin";
ConcertTickets.concertName = "Jimmy Buffett";
ConcertTickets.ticketQuantity = 2;
ConcertTickets.concertDate = new Date(2010, 6, 18, 20);
for (prop in ConcertTickets) {
    document.write(prop + "<br />");
}
```

In the preceding code, the variable name `prop` holds the names of each property in the `ConcertTickets` object. The `document.write()` statement then writes the name of each property to the Web browser window, as shown in Figure 6-13.

![Figure 6-13 Property names printed with a for...in statement](image)

The preceding example demonstrates how to use the `for...in` statement with objects instantiated from the `Object` object. Using the
for...in statement with objects instantiated from constructor functions is very similar:

```javascript
function ConcertTickets(customer, concert, tickets, eventDate) {
    this.customerName = customer;   // customer name
    this.concertName = concert; // concert name
    this.ticketQuantity = tickets; // number of tickets
    this.concertDate = eventDate; // concert date
}
var ticketOrder = new ConcertTickets("Don Gosselin", "Jimmy Buffett", 2, new Date(2010, 6, 18, 20));
for (prop in ticketOrder) {
    document.write(prop + "<br />");
}
```

One of the benefits of the for...in statement is that it enumerates, or assigns an index to, each property in an object, which is similar to the way elements in an array are indexed. You can use an enumerated object property to access the values contained within object properties. For example, in the following code, the `document.write()` statement within the body of the for...in statement refers to the prop variable as an index of the ticketOrder object:

```javascript
for (prop in ticketOrder) {
    document.write(ticketOrder[prop] + "<br />");
}
```

Each iteration of the for...in statement in the preceding code now prints the contents of each property rather than just the property names. The code passes the ticketOrder object to the `document.write()` method, along with the prop variable enclosed in brackets (ticketOrder[prop]). You would use this same technique to print the contents of an array. Unlike the elements in an array, however, you cannot refer to the enumerated properties of an object outside of a for...in loop; doing so generates an error. The statement `document.writeln(ticketOrder[prop]);` causes an error outside of a for...in loop.

Next, you start adding a function named `addContact()`, which will add windsurfers to the contact list.

**To start adding the `addContact()` function:**

1. Return to the `index.html` document in your text editor.
2. Add the following `addContact()` function definition to the end of the script section:
   ```javascript
   function addContact() {
   }
   ```
3. Add the following statement to the `addContact()` function. This statement uses the length property of the `Array` class to determine the next array index that will store the new contact.

```javascript
var newContact = contactList.length;
```

4. Add the following statements to the end of the `addContact()` function. The `if` statement checks to see if the last name and first name fields have been filled in. If not, then an alert dialog box appears instructing users to enter the contact's first and last name. If both fields contain values, then an else statement instantiates a new `Contact` object in the `contactList` array. The remaining statements assign values to the `Contact` object property, create a new `Option` object, and assign the contents of the new `Contact` property to the new `Option` object.

```javascript
if (document.forms[0].lastname.value == "" || document.forms[0].firstname.value == "")
  window.alert("You must enter the contact's first and last names.");
else {
  contactList[newContact] = new Contact();
  contactList[newContact].lastName = document.forms[0].lastname.value;
  contactList[newContact].firstName = document.forms[0].firstname.value;
  contactList[newContact].telephone = document.forms[0].telephone.value;
  contactList[newContact].address = document.forms[0].address.value;
  contactList[newContact].city = document.forms[0].city.value;
  contactList[newContact].state = document.forms[0].state.value;
  contactList[newContact].zip = document.forms[0].zip.value;
  var createContact = new Option();
  createContact.value = contactList[newContact].lastName + ", " + contactList[newContact].firstName;
  createContact.text = contactList[newContact].lastName + ", " + contactList[newContact].firstName;
  document.forms[0].contacts.options[newContact] = createContact;
  calcGroupDiscount(newContact + 1);
}
```
5. Add the following event handler to the Add Snowboarder element:

   ```html
   <input type="button" value="Add Snowboarder" onclick="addContact()" />
   ```

6. Save the `index.html` document, and open the document in your Web browser. Test the script by adding some contacts to the contact list.

---

**Referring to Object Properties as Associative Arrays**

JavaScript allows you to refer to object properties using associative array syntax. An **associative array** is an array whose elements are referred to with an alphanumeric key instead of an index number. For example, with associative arrays you can create a company’s payroll information that uses each employee’s last name instead of an index number to refer to an element in the array. To refer to an element in an associative array, you place an element’s key in single or double quotation marks inside the array brackets. For example, the following statements create the elements in an array named `hotelReservation` using associative array syntax:

```javascript
var hotelReservation = new Array(4);
hotelReservation["guest"] = "Don Gosselin";
hotelReservation["nights"] = 2;
hotelReservation["price"] = 89.95;
hotelReservation["nonsmoking"] = true;
document.write("<p>Guest: " + hotelReservation["guest"]);
document.write("<br />Nights: " + hotelReservation["nights"]);
document.write("<br />Price: " + hotelReservation["price"]);
document.write("<br />Non-smoking room: " + hotelReservation["nonsmoking"] + "</p>");
```

You can also use associative array syntax to refer to the properties of an object. The following example demonstrates how to refer to the `customerName` property of the `ConcertTickets` object using associative array syntax (`ConcertTickets["customerName"]`) instead of standard property syntax (`ConcertTickets.customerName`):

```javascript
var ConcertTickets = new Object();
ConcertTickets.customerName = "Don Gosselin";
document.write("<p>Customer: " + ConcertTickets["customerName"]);
```
One of the benefits of using associative array syntax with object properties is that you can dynamically build property names at runtime. For example, the following statements use associative array syntax to create a property consisting of the word "employee" and an employee ID in an object named `employeeList`:

```javascript
var employeeList = new Object();
var employeeID = 56725;
employeeList["employee" + employeeID] = "Don Gosselin";
```

You can print the contents of the property created with the preceding statements by using any of the following:

```javascript
document.write(employeeList["employee" + employeeID]);
document.write(employeeList.employee56725);
document.write(employeeList["employee56725"]);
```

### Deleting Properties

To delete a specific property in a custom object, you use the `delete` operator with the syntax `delete object.property`. For example, the following statement deletes the `concertDate` property of the `ConcertTickets` object:

```
delete ConcertTickets.concertDate;
```

Next, you add a `deleteContact()` function to the Group Reservations page that deletes selected windsurfers from the Group Members section.

**To add a `deleteContact()` function:**

1. Return to the `index.html` document in your text editor.
2. Add the following `deleteContact()` function definition to the end of the script section:

   ```javascript
   function deleteContact() {
   }
   ```
3. Add the following statements to the `deleteContact()` function. These statements loop through each option in the selection list to determine which option is selected. Once the selected item is located, its index value is assigned to the `selectedContact` variable.
var contactSelected = false;
var selectedContact = 0;
for (var i=0; i<document.forms[0].contacts.options.length; ++i) {
  if (document.forms[0].contacts.options[i].selected == true) {
    contactSelected = true;
    selectedContact = i;
    break;
  }
}

4. Add the following statements to the end of the deleteContact() function. The if...else statement first checks the value assigned to the contactSelected variable. If the contactSelected variable contains a value of false, an alert dialog box informs the user that they must select a contact in the list. If the contactSelected variable contains a value of true, the if statement uses the remove() method of the Option class to remove the contact from the list and then uses the splice() method of the Array class to delete the array element that stores the object. The statements at the end of the if statement clear the values in the form's text boxes.

if (contactSelected == true) {
  for (prop in contactList) {
    delete contactList[prop]
  }
  for (var i=0; i<document.forms[0].contacts.options.length; ++i) {
    contactList["contact" + i] = new Contact();
  contactList["contact" + i].lastName = document.forms[0].lastname.value;
    contactList["contact" + i].firstName = document.forms[0].firstname.value;
    contactList["contact" + i].telephone = document.forms[0].telephone.value;
    contactList["contact" + i].address = document.forms[0].address.value;
    contactList["contact" + i].city = document.forms[0].city.value;
    contactList["contact" + i].state = document.forms[0].state.value;
  }
}
5. Add the following event handler to the Delete Windsurfer element:

```html
<input type="button" value="Delete Windsurfer"
onclick="deleteContact()" />
```

6. Save the `index.html` document, and open the document in your Web browser. Test the script by adding and deleting some contacts to and from the contact list. Note that you still need to add functionality to repopulate the group member fields when selecting a new contact in the list.

7. Close your Web browser window.

Creating Methods

You can create a function that will be used as an object method by referring to any object properties it contains with the `this` reference. For example, the following code defines a method that prints the `customerName`, `concertName`, `ticketQuantity`, and `concertDate` properties of the `ConcertTickets` constructor function:

```javascript
function displayConcertTickets() {
    document.write("<p>Customer: " + this.customerName);
    document.write("<br />Concert: " + this.concertName);
    document.write("<br />Quantity: " + this.ticketQuantity);
    document.write("<br />Date: " + this.concertDate.toLocaleString() + "</p>");
}
```

After a method is created, it must be added to the constructor function, using the syntax `this.methodname = functionname;`. The `methodname` following the `this` reference is the name that is being assigned to the function within the object. Remember not to include the parentheses following the function name, as you would when calling a function in JavaScript. The statement `this.methodname = functionname();` is incorrect, because it includes parentheses. To add the `displayConcertTickets()` function to the `ConcertTickets` function definition as a method named `showOrder()`, you include the statement `this.showOrder = displayConcertTickets;` within the function definition braces.
The following code shows the `ConcertTickets()` constructor function, the `displayConcertTickets()` function that creates the `showOrder()` method, and statements that instantiate two `ConcertTickets` objects and print the values of their properties. Figure 6-14 shows the output.

```javascript
function ConcertTickets(customer, concert, tickets, eventDate) {
    this.customerName = customer; // customer name
    this.concertName = concert; // concert name
    this.ticketQuantity = tickets; // number of tickets
    this.concertDate = eventDate; // concert date
    this.showOrder = displayConcertTickets;
}

function displayConcertTickets() {
    document.write("<p>Customer: " + this.customerName);
    document.write("Concert: " + this.concertName);
    document.write("Quantity: " + this.ticketQuantity);
    document.write("Date: " + this.concertDate.toLocaleString() + "</p>);
}

var ticketOrder = new ConcertTickets("Don Gosselin", "Jimmy Buffett", 2, new Date(2010, 6, 18, 20));
ticketOrder.showOrder();
var ticketOrder = new ConcertTickets("Don Gosselin", "John Mayer", 2, new Date(2010, 8, 30, 20));
ticketOrder.showOrder();
```

Figure 6-14  Output of a script with two instantiated custom objects
Now you will add methods to the Contact constructor function.

To add methods to the Contact constructor function:

1. Return to the index.html document in your text editor, and add the following function to the end of the script section. The function updates the fields in the contacts form with the properties of the selected Contact object. An onclick event handler you will add to the <select> element later in these steps will call the function.

   ```javascript
   function getContactInfo() {
   document.forms[0].lastname.value = this.lastName;
   document.forms[0].firstname.value = this.firstName;
   document.forms[0].telephone.value = this.telephone;
   document.forms[0].address.value = this.address;
   document.forms[0].city.value = this.city;
   document.forms[0].state.value = this.state;
   document.forms[0].zip.value = this.zip;
   }
   ```

2. Next, add the following function to the end of the script section. This function updates the selected contact's information within the Contact object. You will pass the curIndex parameter from an onclick event handler that you will add to the Update Contact button.

   ```javascript
   function updateSelectedContact(curIndex) {
   this.lastName = document.forms[0].lastname.value;
   this.firstName = document.forms[0].firstname.value;
   this.telephone = document.forms[0].telephone.value;
   this.address = document.forms[0].address.value;
   this.city = document.forms[0].city.value;
   this.state = document.forms[0].state.value;
   this.zip = document.forms[0].zip.value;
   document.forms[0].contacts.options[curIndex].value = this.lastName + "", " + this.firstName;
   document.forms[0].contacts.options[curIndex].text = this.lastName + "", " + this.firstName;
   window.alert("Contact information updated.");
   }
   ```

3. Add the following two statements to the end of the Contact constructor function, which declare getContactInfo() and updateSelectedContact() as methods of the Contact object:

   ```javascript
   this.getContacts = getContactInfo;
   this.updateContact = updateSelectedContact;
   ```

4. Add an onclick event handler to the Update Info button, as follows. The event handler passes to the updateContact() function the selectedIndex property of the option that is selected in the selection list.
5. Finally, add the following event handler to the `<select>` element. The event handler updates the data displayed in the contact information fields after a new option is clicked in the selection list.

```html
onclick="contactList[this.selectedIndex].getContacts();"
```

6. Save the `index.html` document, and then open it in your Web browser. Test the script by adding some contacts to the contact list. Try clicking on a previously entered contact. The contact's information should be displayed in the contact information fields. Also, try updating a previously entered contact. Figure 6-15 shows how the form appears after adding several contacts.

![Figure 6-15 Contacts form after adding several contacts](Image)

7. Close your Web browser window.
Your final step is to modify the Group Reservations page so that the group discount is automatically calculated as individuals are added to and deleted from the Group Members list.

To modify the Group Reservations page so that it automatically calculates group discounts:

1. Return to the `index.html` document in your text editor, and add the following statement to the end of `else` clause in the `addContact()` function. This statement calls the `calcGroupDiscount()` function and passes to it the value of the `newContact` variable incremented by one.
   ```javascript
   calcGroupDiscount(newContact + 1);
   ```

2. In the `deleteContact()` function, add the following statement to the end of the `if` statement that determines whether the `contactSelected` variable contains a value of true. This statement also calls the `calcGroupDiscount()` function, but it passes to it the `length` property of the `options[]` array, which indicates the number of items in the group members list.
   ```javascript
   calcGroupDiscount(document.forms[0].contacts.options.length);
   ```

3. Save the `index.html` document, and then validate it with the W3C Markup Validation Service at validator.w3.org/file-upload.html. Once the document is valid, close it in your text editor.

4. Open the `index.html` document in your Web browser, and try adding and deleting some windsurfers. The group discount should be updated automatically.

Using the prototype Property

As explained earlier, objects inherit the properties and methods of the constructor functions from which they are instantiated. When you instantiate a new object named `ticketOrder`, based on the `ConcertTickets` constructor function, the new object includes the `customerName`, `eventName`, `numTickets`, and `concertDate` properties along with the `showOrder()` method. After instantiating a new object, you can assign additional properties to the object, using a period. The following code creates a new object based on the `ConcertTickets` constructor function, then assigns to the object a new property named `orderDate`. The statement uses the `Date()` constructor function without any arguments, which assigns the current date to the `orderDate` property.
var ticketOrder = new ConcertTickets("Don Gosselin", "Jimmy Buffett", 2, new Date(2010, 6, 18, 20));
ticketOrder.orderDate = new Date();

When you add a new property to an object that has been instantiated from a constructor function, the new property is only available to that specific object; the property is not available to the constructor function or to any other objects that were instantiated from the same constructor function. However, if you use the prototype property with the name of the constructor function, any new properties you create will also be available to the constructor function and any objects instantiated from it. The prototype property is a built-in property that specifies the constructor from which an object was instantiated. The following code uses the prototype property to add the orderDate property to the ConcertTickets constructor function. By using a prototype property, you ensure that all objects that extend the ConcertTickets constructor function also have access to the orderDate property.

var ticketOrder = new ConcertTickets("Don Gosselin", "Jimmy Buffett", 2, new Date(2010, 6, 18, 20));
ticketOrder.prototype.orderDate = new Date();

Object definitions can use the prototype property to extend other object definitions. That is to say, you can create a new object based on an existing object. The new object inherits the properties and methods of the original object. You can then add additional properties and methods to the new object that will not be available to the existing object. Consider an object definition named Event that contains generic properties and methods that might be used for planning an event. You may need to create additional object definitions that extend Event and that contain properties and methods specific to certain types of events. To extend one object definition (the derived object definition) from another object definition (the base object definition), append the prototype property to the derived object definition, followed by the new keyword and the name of the base object definition using the following syntax: derived_object.prototype = new base_object();. The following code shows an example of a RetirementEvent object definition that extends the Event object definition. The Event class definition contains some generic properties, eventLocation, eventDate, and eventCost, that apply to all types of events, along with a calcEventCost() method that calculates the cost of an event. The RetirementEvent class includes guestOfHonor and company properties along with a showEventDetails() method. Figure 6-16 shows the output.
<script type="text/javascript">
/* <![CDATA[ */
function Event(location, date) {
  this.eventLocation = location;
  this.eventDate = date;
  this.eventCost = 0;
  this.calcEventCost = calcCost;
}
function calcCost(guests) {
  this.eventCost = guests * 25; // $25 per head
}
function RetirementEvent(retiree, company) {
  this.guestOfHonor = retiree;
  this.companyName = company;
  this.showEventDetails = eventDetails;
}
function eventDetails() {
  document.write("Guest of honor: " + this.guestOfHonor);
  document.write("<br />Company: " + this.companyName);
  document.write("<br />Event date: " + this.eventDate);
  document.write("<br />Event Location: " + this.eventLocation);
  document.write("<br />Event cost: $" +
    this.eventCost.toLocaleString() + "</p>");
}
RetirementEvent.prototype = new Event();
var wertherRetirement = new RetirementEvent("Jacob Werther", "Forestville Funding");
wertherRetirement.eventLocation = "Forestville, CA";
wertherRetirement.eventDate = "January 27, 2010";
wertherRetirement.calcEventCost(175);
wertherRetirement.showEventDetails();
/* ]]> */
</script>

Guest of honor: Jacob Werther
Company: Forestville Funding
Event date: January 27, 2010
Event location: Forestville, CA
Event cost: $4,375

Figure 6-16  Output of a script with an extended object definition

Some object-oriented programming languages allow objects to inherit from more than one object definition. JavaScript, however, only allows objects to inherit from a single object definition.
Short Quiz 3

1. Why would you create a custom object with the `Object` object?

2. Explain how to create custom objects using a constructor function.

3. How do you add properties to a custom object?

4. How and why do you enumerate custom object properties?

5. How do you add methods to a custom object?

Summing Up

- The term object-oriented programming (or OOP) refers to the creation of reusable software objects that can be easily incorporated into another program.

- Reusable software objects are often referred to as components.

- In object-oriented programming, an object is programming code and data that can be treated as an individual unit or component. Data refers to information contained within variables or other types of storage structures.

- Objects are encapsulated, which means that all code and required data are contained within the object itself.

- An interface refers to the programmatic elements required for a source program to communicate with an object.

- The principle of information hiding states that any class members that other programmers do not need to access or know about should be hidden.

- In object-oriented programming, the code, methods, attributes, and other information that make up an object are organized using classes.

- An instance is an object that has been created from an existing class. When you create an object from an existing class, you are said to be instantiating the object.
An object inherits, or takes on, the characteristics of the class on which it is based.

The `Date` class contains methods and properties for manipulating the date and time.

The `Number` class contains methods for manipulating numbers and properties that contain static values representing some of the numeric limitations in the JavaScript language (such as the largest positive number that can be used in JavaScript).

The `Math` class contains methods and properties for performing mathematical calculations in your programs.

You can define your own custom objects by using a constructor function (also known as an object definition), which is a function that is used as the basis for a custom object.

The `this` keyword refers to the current object.

The `prototype` property is a built-in property that specifies the constructor from which an object was extended.

Comprehension Check

1. Which of the following terms refer to programming code and data that can be treated as an individual unit or component? (Choose all that apply.)
   a. methods
   b. components
   c. objects
   d. properties

2. Explain the principle of information hiding. What does a “black box” refer to?

3. A(n) ________________ is an object that has been created from an existing class.
   a. constructor
   b. instance
   c. template
   d. structure
4. Explain why programmers use the terms “variable” and “object” interchangeably.

5. JavaScript automatically performs garbage collection. True or false?

6. Which of the following Date class constructors creates a Date object that contains the current date and time from the local computer?
   a. Date()
   b. Date(milliseconds)
   c. Date(date_string)
   d. Date(year, month[, date, hours, minutes, seconds, milliseconds])

7. Explain how to display the full text for days and months in a Date object.

8. Which of the following methods of the Number class converts a number to a string that is formatted with local numeric formatting conventions?
   a. toString()
   b. toLocaleString()
   c. toFixed()
   d. valueOf()

9. What is the primary reason for using most of the Number class methods?

10. What is the correct syntax for using the sqrt() method of the Math class?
    a. firstNumber = Math.sqrt(secondNumber);
    b. var result = firstNumber.sqrt(secondNumber);
    c. var result = sqrt.Math(firstNumber, secondNumber);
    d. var result = Math.sqrt(firstNumber, secondNumber);
11. A function that is used as the basis for an object is called an object definition or a(n) _____________.
   a. object variable
   b. class
   c. method
   d. constructor function

12. Explain why JavaScript is not a true object-oriented programming language.

13. In the case of a custom JavaScript object, the this keyword refers to _____________.
   a. the currently executing JavaScript statement
   b. the current object that called the constructor function
   c. the XHTML document
   d. the Web browser window

14. What is the correct syntax for creating an object named currentSale from a constructor function named Transaction that requires the two arguments quantity and price?
   a. currentSale = new Transaction (2, 59.95);
   b. currentSale(2, 59.95) = new Transaction;
   c. new Transaction(2, 59.95) = currentSale;
   d. currentSale() = new Transaction 2, 59.95;

15. Explain how to assign properties to a custom object created with the Object object.

16. The Object object can contain both properties and methods. True or false?

17. Which of the following statements is the correct syntax for assigning the value of a parameter named priceQuote to a property named quote in a constructor function?
   a. quote = priceQuote;
   b. this.quote = priceQuote;
   c. this.quote = this.priceQuote;
   d. quote = this.priceQuote;
18. What is the correct syntax for adding a new property named squareFeet to a constructor function named RealEstate?
   a. RealEstate.prototype = squareFeet("");
   b. prototype.RealEstate.squareFeet = "";
   c. squareFeet.RealEstate.prototype = "";
   d. RealEstate.prototype.squareFeet = "";

19. What is the correct syntax for adding an object method named submitBid() to a constructor function named Auction?
   a. Auction = new submitBid();
   b. myMethod = this.submitBid;
   c. this.submitBid = myMethod();
   d. this.submitBid = submitBid;

20. The built-in property that specifies the constructor from which an object was extended is called the _________ property.
   a. source
   b. origination
   c. default
   d. prototype

Reinforcement Exercises

Exercise 6-1

One problem with the calendar in the Group Reservations Web page that you worked on in this chapter is that it does not display 29 days for the month of February during leap years. In Chapter 3, you created a simple form that tested for leap years. As you may recall, you can determine whether a year is a leap year by testing if it is divisible by 4. However, years that are also divisible by 100 are not leap years, unless they are also divisible by 400, in which case they are leap years (1900 was not a leap year; 2000 was). In this exercise, you will add code to the Group Reservations Web page that displays 29 days for the month of February during leap years.
1. Copy the `index.html` document from your Chapter folder for Chapter 6 to your Exercises folder for Chapter 6. Rename the document as `CVSGroupsLeapYear.html`.

2. In the `displayCalendar()` function, locate the following `else...if` statement that determines the number of days for the month of February:

   ```javascript
   else if (month == 1)
       numDays = 28;
   ```

3. Modify the `else...if` statement so that it includes the calculations for determining whether it is a leap year, as follows:

   ```javascript
   else if (month == 1) {
       var thisYear = dateObject.getYear();
       if (thisYear %4 != 0)
           numDays = 28;
       else if (thisYear % 400 == 0)
           numDays = 29;
       else if (thisYear % 100 == 0)
           numDays = 28;
       else
           numDays = 29;
   }
   ```

4. Save the `CVSGroupsLeapYear.html` document.

5. Use the W3C Markup Validation Service to validate the `CVSGroupsLeapYear.html` document, and fix any errors that the document contains. Once the document is valid, close it in your text editor.

6. Open the `CVSGroupsLeapYear.html` document in your Web browser, and click the `Select Date` link to display the calendar window. Click the `Next` or `Previous` link to navigate to the month of February in a leap year, such as 2012. The calendar should display 29 days.

7. Close your Web browser window.

---

**Exercise 6-2**

Another problem with the Group Reservations Web page is that if you submit the form, none of the windsurfers in the Group Members list are submitted unless their names are selected in the selection list. Even if you select the names in the list, only the last and first names are submitted. Furthermore, the values of the group member information fields (last name, first name, telephone, and so on) are submitted; these fields should not be submitted because their values are
stored in the Contact objects that contain the information for each windsurfer. In this project, you will add an onSubmit event handler that selects all the options in the Group Members selection list and submits the properties in each Contact object along with each group member’s name. You will also remove the name attributes from each of the group member information fields to prevent them from being submitted with the form. (Recall that only form fields with name attributes are submitted with a form.) Finally, you will modify the script so that it refers to each group member information field by its position in the options[] array instead of by its name attribute.

1. In your text editor, open the CVSGroupsLeapYear.html document from your Exercises folder for Chapter 6 and immediately save it as CVSGroupsSubmission.html.

2. Add to the opening <form> tag an onSubmit attribute that calls an event handler function named submitReservation(). (You create the event handler function in Step 3.)

   <form action="FormProcessor.html" method="get" enctype="application/x-www-form-urlencoded" onSubmit="return submitReservation();">

3. Add multiple="multiple" to the opening <select> tag of the Group Members selection list.

4. Type the following submitReservation() function at the end of the script section. The for loop iterates through the items in the Group Members selection list. For each item, it appends the properties of the associated object to the value property, using associative array syntax; this ensures that all information for each windsurfer is submitted with the form. The last statement in the for loop uses the selected property to select the item.

   function submitReservation() {
       for (var k=0; k<document.forms[0].contacts.length; ++k) {
           document.forms[0].contacts.options[k].value += "", "
           + contactList["contact" + k].telephone + "",
           + contactList["contact" + k].address + "",
           + contactList["contact" + k].city + "",
           + contactList["contact" + k].state + "",
           + contactList["contact" + k].zip;
           document.forms[0].contacts.options[k].selected = true;
       }
       return true;
   }
5. Remove the name attributes from the lastname, firstname, telephone, address, city, state, and zip elements in the Group Members fields.

6. Finally, replace each of the references to the name properties in the script section of each of the Group Members fields with the element’s associated index in the elements[] array. The indexes for the fields in the elements[] array are: lastname=elements[12], firstname=elements[13], telephone=elements[14], address=elements[15], city=elements[16], state=elements[17], and zip=elements[18]. For example, the statements in the getContactInfo() function should be modified as follows:

```javascript
function getContactInfo() {
    document.forms[0].elements[12].value = this.lastName;
    document.forms[0].elements[13].value = this.firstName;
    document.forms[0].elements[14].value = this.telephone;
    document.forms[0].elements[15].value = this.address;
    document.forms[0].elements[16].value = this.city;
    document.forms[0].elements[17].value = this.state;
    document.forms[0].elements[18].value = this.zip;
}
```

7. Save the CVSGroupsSubmission.html document.

8. Use the W3C Markup Validation Service to validate the CVSGroupsSubmission.html document, and fix any errors that the document contains. Once the document is valid, close it in your text editor and then open it in your Web browser. Add several entries to the Group Members list, and submit the form. The FormProcessor.html file should display the values for each of your entries, but it should not display the group member information fields.


**Exercise 6-3**

In this exercise, you will create a script that displays the current date and time and welcomes the user with “Good morning!”, “Good afternoon!”, or “Good evening!”, depending on the time of day.

1. Create a new document in your text editor.

2. Type the <!DOCTYPE> declaration, <html> element, header information, and <body> element. Use strict DTD and “Welcome” as the content of the <title> element.
3. Add the following text and elements to the document body:

```html
<h1>Welcome to My Web page</h1>
```

4. Add the following script section to the end of the document body:

```javascript
<script type="text/javascript">
/* <![CDATA[ */
/* ]]> */
</script>
```

5. Add the following variable declarations to the script section. The first variable instantiates a `Date` object. The second and third variables will be assigned text strings containing a greeting and the current time. The fourth and fifth variables are assigned the minute and hour values from the `Date` object.

```javascript
var dateObject = new Date();
var greeting = " ";
var curTime = " ";
var minuteValue = dateObject.getMinutes();
var hourValue = dateObject.getHours();
```

6. Add the following code to the end of the script section. The first `if` statement evaluates the `minuteValue` variable and adds a 0 to the beginning of the value if it is less than 10. This forces the minutes to always display as two decimals. The `if...else` structure evaluates the `hourValue` variable and builds the strings that are assigned to the `greeting` and `curTime` variables.

```javascript
if (minuteValue < 10) 
    minuteValue = "0" + minuteValue;
if (hourValue < 12) {
    greeting = "<p>Good morning! ";
    curTime = hourValue + ":" + minuteValue + " AM";
} else if (hourValue == 12) {
    greeting = "<p>Good afternoon! ";
    curTime = hourValue + ":" + minuteValue + " PM";
} else if (hourValue < 17) {
    greeting = "<p>Good afternoon! ";
    curTime = (hourValue-12) + ":" + minuteValue + " PM";
} else {
    greeting = "<p>Good evening! ";
    curTime = (hourValue-12) + ":" + minuteValue + " PM"
}
```
7. Add the following arrays to the end of the script section. These arrays contain the full text for days and months.

```javascript
var dayArray = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday");
```

8. Type the following statements at the end of the script section to retrieve the current day and month values from the `dateObject` variable:

```javascript
var day = dateObject.getDay();
var month = dateObject.getMonth();
```

9. Finally, add the following statement to the end of the script section to display the current date and time and a welcome message:

```javascript
document.write("<p>" + greeting + "It is" + curTime + "on" + dayArray[day] + "," + monthArray[month] + "" + dateObject.getFullYear() + "," + dateObject.getDate() + "</p>"):
```

10. Save the `WelcomeDateTime.html` document in your Exercises folder for Chapter 6.

11. Use the W3C Markup Validation Service to validate the `WelcomeDateTime.html` document, and fix any errors that the document contains. Once the document is valid, close it in your text editor and then open it in your Web browser. You should see the appropriate welcome message along with the time and date.


## Exercise 6-4

In this exercise, you will create a tip calculator.

1. Create a new document in your text editor.

2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and the `<body>` element. Use strict DTD and “Tip Calculator” as the content of the `<title>` element.

3. Add the following heading and form to the document body. The form uses a table to format the display of the fields. The form is fairly straightforward. Several of the fields use an
onchange event handler to call a `figureTip()` method, which you will add in the next step.

```html
<h2>Tip Calculator</h2>
<form action="">
<table border="0">
<tr><td>Bill:</td><td><input type="text" name="bill" value="0"
onchange="figureTip();" /></td></tr>
<tr><td>Tip %:</td><td><input type="text" name="tip_percent" value="0"
onchange="figureTip();" /></td></tr>
<tr><td># of People:</td><td><input type="text" name="num_people" value="1"
onchange="figureTip();" /></td></tr>
<tr><td>Tip amount:</td><td><input type="text" name="tip_amount" value="0"
readonly="readonly" /></td></tr>
<tr><td>Total:</td><td><input type="text" name="total" value="0"
readonly="readonly" /></td></tr>
<tr><td>Total per Person:</td><td><input type="text" name="total_person" value="0"
readonly="readonly" /></td></tr>
</table>
</form>

4. Next, add the following script section to the document head. The function uses the Document object to retrieve the values entered in the form fields. The calculated results use the `Math.round()` method to round the figures to whole numbers.

```javascript
<script type="text/javascript">
/* <![CDATA[ */
function figureTip() {
    var bill = document.forms[0].bill.value;
    var tipPercent = document.forms[0].tip_percent.value;
    var numPeople = document.forms[0].num_people.value;
    var tipAmount = Math.round(bill * ("." + tipPercent));
    var total = Math.round(bill * (1 + "." + tipPercent));
    var totalPerPerson = Math.round(total / numPeople);
    document.forms[0].tip_amount.value = tipAmount.toLocaleString();
    document.forms[0].total.value = total.toLocaleString();
    document.forms[0].total_person.value = totalPerPerson.toLocaleString();
}
/* ]]> */
</script>
5. Save the document as **TipCalculator.html** in the Exercises folder for Chapter 6, and then validate it with the W3C Markup Validation Service at validator.w3.org/file-upload.html. Once the TipCalculator.html document is valid, close it in your text editor, and then open it in your Web browser. Test the calculator’s functionality.


## Discovery Projects

Save your Discovery Projects files in the Projects folder for Chapter 6. Be sure to validate the files you create with the W3C Markup Validation Service.

### Project 6-1

Create a Web page that allows a coach to submit a roster for a baseball team. Use a custom JavaScript object similar to the one you created for the Group Reservations Web page. Include fields for the team’s name and head coach’s contact information. Create a Team Members section, similar to the Group Members section on the Group Reservations page, to allow the coach to enter information for each team member, including his or her contact information and the position played. Include functionality that allows the coach to add, delete, and modify player information. Also include a check box that the coach can select if permission has been received from the player’s guardian. Write functionality that submits all of the team member’s information (including the permission field) to the FormProcessor.html document (a copy is located in your Projects folder for Chapter 6), and create the form so that the team member information fields are not submitted. Save the document as **TeamRoster.html**.

### Project 6-2

Use the Date object to create an age calculator. Create three selection lists that allow users to select the date, month, and year when they were born. For the year selection list, display the years 1950 to 2010. Include a push button that performs the calculation along with text boxes that display the following data: the user’s age in years, months, and days, and how long a person has been living in months, days, hours, and minutes. Finally, include a text box that displays the number of days until the user’s next birthday. Save the document as **AgeCalculator.html**.
Project 6-3

A popular use of the Date object is to create a digital clock. However, recall that the date and time in a Date object are not updated over time like a clock. Instead, a Date object contains the static date and time as of the moment the JavaScript code instantiates the object. You can simulate a digital clock that appears to “click” off each second by using a setInterval() method that continuously executes code to retrieve the current time. Use this technique to create a digital clock that displays a 12-hour clock in a form’s text box. The clock should display hours, minutes, seconds, and either AM or PM. Format the displayed time so that any minutes or seconds that are less than 10 are preceded by a 0, such as 10:08:07 AM. Save the document as DigitalClock.html.

Project 6-4

Create a Web page for a digital photo development company. Include a form that allows clerks at the company to enter orders for each customer. The form should allow you to enter one item per order. Include radio buttons for different types of items that can be created with digital images, including hard-copy prints, posters, coffee mugs, and T-shirts. Use a Date object to calculate the date an item will be ready automatically, based on the current date. For example, hard-copy prints and posters should be ready one day from today, coffee mugs two days from today, T-shirts three days from today, and so on. Submit the form to the FormProcessor.html document (a copy is located in your Projects folder for Chapter 6). Save the document as DigitalPhotos.html.