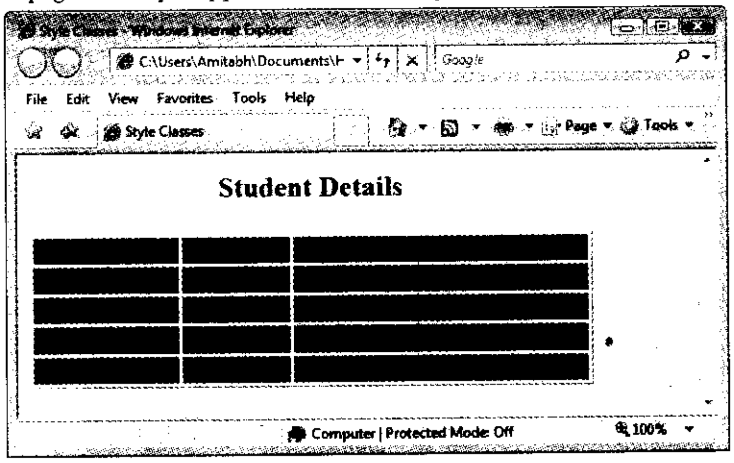
```
<td class="green">15-03-1983</td>
<td class="green">Flat No. 303, Shipra Suncity, Ghaziabad</td>
</tr>
<tr>
<td class="green">Amitabh Kumar</td>
<td class="green">22-02-1984</td>
<td class="green">H.No- 125, Kalkaji, Delhi</td>
</tr>
<tr>
<td class="green">Rohit Jandial</td>
<td class="green">05-07-1983</td>
<td class="green">Flat No- 324, South Ext, Delhi</td>
</tr>
<tr>
<td class="green">Avantika Srivastava</td>
<td class="green">10-12-1984</td>
<td class="green">H.No-541, Vikas Puri, Delhi</td>
</tr>
</table>
</body>
</html>
```

When you open this page, the output appears, as shown in Figure 2.65:


C:\Users\Amitabh\Documents\
Google
File Edit View Favorites Tools Help
Style Classes
Page Tools
Computer | Protected Mode: Off
100%

Student Details

Figure 2.65: Applying Styles Using Style Classes

In Figure 2.65, you can observe that the color of table has changed to the specified color.

Multiple Styles

Multiple styles can be defined by using the different methods to implement CSS. For this reason, the use of several external style sheets results in cascading the styles, which is a combination of styles for various HTML elements. If multiple styles affect the same element, only the last one is used. You can link the external style sheets to the document as follows:

```
<LINK rel=stylesheet type="text/css" href="style1.css">
<LINK rel=stylesheet type="text/css" href="style2.css">
<LINK rel=stylesheet type="text/css" href="style3.css">
```

If multiple conflicting styles are found in the external style sheets, the CSS recommendations allow users to select among several alternative style sheets using the `rel` attribute of the `<STYLE>` tag, which is combined with the `TITLE` attribute to select them by name:

```
<LINK rel="alternate stylesheet" type="text/css" href="style1.css" title="style1">
<LINK rel="alternate stylesheet" type="text/css" href="style2.css" title="style2">
```

```
<LINK rel=stylesheet type="text/css" href="style2.css">
```

Multiple styles are included in a page by using the various possible inclusion methods. However, the style closest to the content is applied when some conflict appears among styles.

Let's create a Web page, named `MultipleStyles.html` to understand how multiple styles work in an HTML document. You can find the `MultipleStyles.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.60 shows the code of the `MultipleStyles.html` page:

Listing 2.60: Linking Multiple Styles

```
<html>
<head>
<title>External Style Sheets</title>
<link rel="stylesheet" type="text/css" href="Style.css">
<link rel="stylesheet" type="text/css" href="style1.css">
<link rel="stylesheet" type="text/css" href="style2.css">
</head>
<body>
<h1>External Style Sheet Example</h1>
<a href= Page1.html  target="_blank">
<h2>Page 1</h2>
<a href= Page2.html target="_blank">
<h2>Page 2</h2>
</body>
</html>
```

To apply the styles that you specified in the code, you need to create those style sheets. In this case, we have created the style1.css and style2.css style sheets, which you can find in the Code\HTML\Chapter 2 folder on the CD. When you open this page, the output appears, as shown in Figure 2.66:

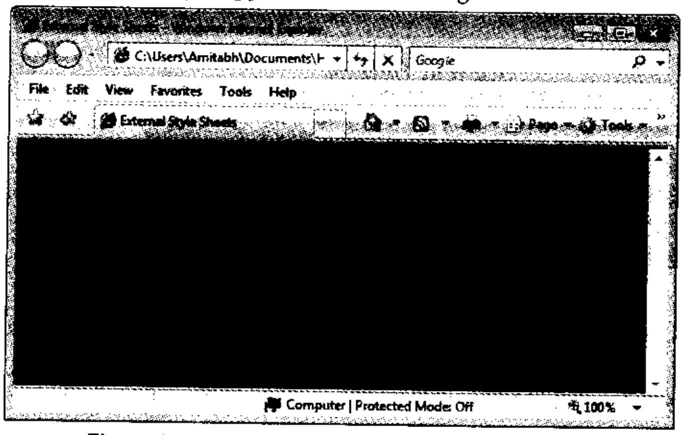

C:\Users\Amitabh\Documents\
Google
File Edit View Favorites Tools Help
External Style Sheets
Page Tools
Computer | Protected Mode: Off
100%

Figure 2.66: Applying Styles Using Multiple Styles

In Figure 2.66, you can note that the `Style2.css` style sheet is applied to the Web page, because it is the last style sheet in the code.

Introducing DHTML

DHTML stands for Dynamic HTML and explains different technologies that are required to make Web pages dynamic and interactive. DHTML is a term used by developers to describe the combination of HTML, CSS, and scripts that helps to create animation on the document. In simple words, you can say that DHTML is a combination of HTML, JavaScript, DOM, and CSS.

JavaScript is a scripting language that is used by DHTML to control, access, and manipulate HTML elements. DOM stands for Document Object Model and defines a standard set of objects for HTML, and a standard way to access and manipulate them. CSS allows Web developers to control the style and layout of Web pages.

Introducing JavaScript

The HTML Web pages that you have worked with in previous sections are plain and static, that is their presentation, style, and content are more or less fixed. Moreover, plain HTML Web pages do not offer a two-way interaction with the users that makes them rather uninteresting and monotonous. You can enhance the Web pages by adding some dynamism and interactivity in them. For instance, you may want to validate the values that the user enter in an HTML form and perform different actions depending on the entered values, such as submitting the form when you click the submit button. In such cases, you need to make the Web page interactive which you can do by using scripts in the HTML document.

A script is a program code that is written using a scripting language. A scripting language is a kind of programming language with less functionality. Some of the commonly used scripting languages are JavaScript, VBScript, ASP, and PHP. Among these, JavaScript is the most popular scripting language used to infuse dynamism and interactivity in Web pages.

JavaScript (originally named `LiveScript` by its creator Netscape and Sun Microsystems) is an interpreted, client-side (the scripts run in the Web browser), and object-based scripting language that offers various functionalities to enliven the static HTML Web pages. Note that JavaScript is also known as `ECMAScript` as it was standardized by the European Computer Manufacturer's Association (ECMA).

The scripts written in JavaScript are processed line by line and that is why JavaScript is an interpreted language. The scripts are interpreted by the JavaScript interpreter that is an in-built component of the Web browser. While working with the JavaScript scripts, you can use the in-built objects and other programmable features, such as control flow statements and events to make your HTML Web pages dynamic.

Client-Side Benefits of using JavaScript over VB Script

Client-side scripting languages, such as JavaScript and VBScript, are popularly used to develop interactive and dynamic Web pages. VBScript and JavaScript are interpreted programming languages most commonly used to manipulate and display HTML and DHTML contents. They both can be used in many other applications. JavaScript, for example, can be embedded in the Shockwave Flash and Adobe PDF documents. VBScript is commonly used as a scripting language for network administrators in the Microsoft Windows environment.

Both of these languages allow a developer to complete complex task in a relatively short period of time. This ability allows developers to add dynamic qualities to static HTML and allows for more robust Web pages. JavaScript has the following benefits over VB Script:

- JavaScript is very fast because any code function can run immediately without waiting for an answer from the server.
- JavaScript is relatively simple to learn and implement.
- JavaScript reduces the demand on the server as it is a client-side scripting language.

Embedding JavaScript in an HTML Page

You can embed a JavaScript script in an HTML document by using the `<script>` and `</script>` tags. The `<script>` and `</script>` are HTML tags that allow you to enclose a script. When an HTML document containing these tags is loaded in the Web browser, it processes content enclosed within the `<script>` and `</script>` tags as script code.

Similar to other HTML tags, the `<script>` tag also has various attributes. Some of the important attributes of the `<script>` tag are:

- **type**—Refers to the Multipurpose Internet Mail Extensions (MIME) type of the script. As per the World Wide Web Consortium (W3C), it is mandatory to specify the `type` attribute. It can have `text/ecmascript`, `text/javascript`, `text/vbscript`, `application/ecmascript`, and `application/javascript` as its value.
- **language**—Refers to the scripting language used to create scripts. It is important to note that this attribute is not required if you have specified the type attribute. However, for backward compatibility with the older

versions of Web browsers that may not support the type attribute, you can specify both the attributes. This attribute can have values, such as `javascript` and `vbscript`.

- **src** – Refers to the URL of another file that has a script. This attribute is used to specify external script files.

You can embed a script in an HTML document by creating a script within it or linking an external script file with it. Let's first learn how to create a script inside an HTML document.

Creating a Script in an HTML Document

You can create an entire script within the HTML document itself. Therefore, the script code is written inside the HTML document making it inline with the HTML content. The script code is written between the `<script>` and `</script>` tags in the HTML document.

You can place the `<script>` and `</script>` tags in the body or head portion of the HTML document as per your requirements. You place the `<script>` and `</script>` tags in the body portion when you want the script to run while the Web page is loading in the Web browser. On the other hand, if you want the script to run only when the user performs an action, such as clicking a link, then you can place the `<script>` and `</script>` tags in the head portion of the Web page.

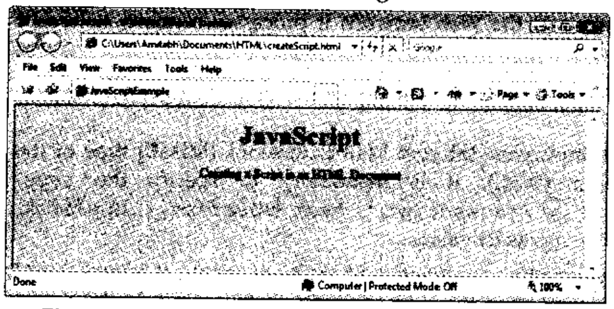
Let's create a Web page, named `createScript.html` to understand how to create a script in an HTML document. You can find the `createScript.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.61 shows the code of the `createScript.html` page:

Listing 2.61: Creating a Script in an HTML Document

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<center>
<h1>JavaScript</h1>
<script type="text/javascript" language="javascript">
document.write("Creating a Script in an HTML Document");
</script>
</body>
</html>
```

In Listing 2.61, the `<script>` and `</script>` tags are placed in the body portion of the Web page, that is, within the `<body>` and `</body>` tags. This implies that the script is loaded in the Web browser at the time when the Web page is loading. In the `<script>` tag, the `type` attribute is set to `text/javascript` and the `language` attribute is set to `javascript`. These two attributes convey to the Web browser that the subsequent content in the HTML document is a script that is written in JavaScript. The `document.write("Creating a Script in an HTML Document")` line written between the `<script>` and `</script>` tags is a JavaScript code. In this line of code, `document` is an object and `write` is a method that allow you to display information on the Web page.

When you open this page, the output appears, as shown in Figure 2.67:


C:\Users\Amitabh\Documents\HTML\createScript.html
File Edit View Favorites Tools Help
JavaScriptExample
Done
Computer | Protected Mode: Off
100%

JavaScript

Creating a Script in an HTML Document

Figure 2.67: Creating a Script in an HTML Document

NOTE

There is no limit to the number of scripts that you can add in an HTML document.

As you can see in Figure 2.67, the Web browser displays the HTML and script contents on the Web page.

Using an External Script File

In some cases, the JavaScript code that you create in an HTML document can extend to tens and hundreds of lines that affect the readability of the HTML document. Moreover, you can use the same script in several Web pages because creating the script in all the Web pages is quite cumbersome. For such cases, you can create an external script file rather than creating the script inside an HTML document. After creating the external script file, you need to link it with the HTML document by using the `<script>` tag.

Let's create an external script file, named `myScript.js`. You can find the `myScript.js` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.62 shows the code of the `myScript.js` script file:

Listing 2.62: Creating an External Script File

```
document.write("Using an External Script File to Add a Script");
```

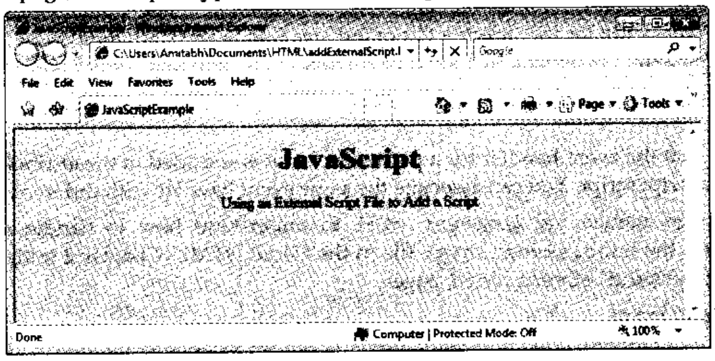
Now, create a Web page, named `addExternalScript.html` to which the script file is added. You can find the `addExternalScript.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.63 shows the code of the `addExternalScript.html` page:

Listing 2.63: Creating the HTML document

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<center>
<h1>JavaScript</h1>
<script src="myScript.js"> </script>
</center>
</body>
</html>
```

In Listing 2.63, the `<script>` and `</script>` tags are enclosed in the body of the HTML document. The `src` attribute of the `<script>` tag is set to the URL of the script file. If the script file is in the same folder as the HTML document, then you just need to specify the name of the script file. However, if the script file and the HTML document are in different folders, then you need to specify the complete URL of the script file. Note that there is no script code written between the `<script>` and `</script>` tags. In case you have written any script code between the two tags, that code is not processed.

When you open this page, the output appears, as shown in Figure 2.68:


C:\Users\Amitabh\Documents\HTML\addExternalScript.l
Google
File Edit View Favorites Tools Help
JavaScriptExample
Page Tools
Done
Computer | Protected Mode: Off
100%

JavaScript

Using an External Script File to Add a Script

Figure 2.68: Adding a Script Using External Script File

As you can observe in Figure 2.68, the Web browser processes the contents of the external script file that was specified in the `src` attribute of the `<script>` tag.

Handling Events

Events are another common and popular way to incorporate interactivity in a Web page. They refer to a particular action performed on the Web page by a user. For instance, an event can occur when the user clicks a hyperlink on the Web page. There are some events that occur without the user's participation, for example, an event occurs when a Web page is fully loaded in the Web browser.

JavaScript defines several in-built events to assist you in making the Web pages dynamic and interactive. When any of these events occur, its event handler is called. An event handler is a special function that handles a particular event. In JavaScript, you can create an event handler and then associate it with the desired event. The event handlers are associated with the events through certain attributes of various HTML tags. Table 2.12 lists some of the common events and the event handling attributes of the HTML tags:

Table 2.12: Common JavaScript Events and the HTML Event Handling Attributes

Event	Description	Attribute
abort	Occurs when the user stops or cancels the loading of an image	onabort
blur	Occurs when an HTML element loses focus because the user clicks outside the element	onblur
change	Occurs when the user changes the value of a text field in an HTML form	onchange
click	Occurs when the user clicks an HTML element, such as form element, an image, or a link	onclick
dblclick	Occurs when the user double-clicks an HTML element	ondblclick
error	Occurs when an error arises while the Web page or an image is loaded	onerror
focus	Occurs when an HTML element gets input focus	onfocus
keydown	Occurs when the user has pressed a key on the keyboard	onkeydown
keypress	Occurs when the user presses a key on the keyboard for a while	onkeypress
keyup	Occurs when the user has released a key on the keyboard	onkeyup
load	Occurs when a Web page or an image is completely loaded in the Web browser	onload
mousedown	Occurs when the user has just pressed a mouse button	onmousedown
mouseover	Occurs when the user moves the mouse on top of a form element	onmouseover
mouseup	Occurs when the user has just released a mouse button	onmouseup
select	Occurs when the user selects a text or text area field in an HTML form on the Web page	onselect
submit	Occurs when the user submits an HTML form by pressing the submit button	onsubmit

It is important to note that the event handler for a particular event is specified in the attributes of the HTML tags and written in the JavaScript script. You can associate the same event handler with different events.

Let's create a Web page, named `handleEvent.html` to understand how to handle events in an HTML document. You can find the `handleEvent.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.64 shows the code of the `handleEvent.html` page:

Listing 2.64: Handling an Event

```
<html>
<head>
<title>JavaScriptExample</title>
<script type="text/javascript" language="javascript">
```

```
function findPercentage(totalMarks)
{
if(totalMarks<=300)
          document.write("Percentage="+(totalMarks/3)+"%<br/>");
}
</script>
</head>
<body bgcolor="pink">
<center>
<form id="myform">
Enter Total Marks: <input type="text" id="TotalMarks"
value="TotalMarks"/><br/><br/>
<input type="button" id="Percentage" value="Percentage"
onclick="findPercentage(TotalMarks.value)"/>
</form>
</center>
</body>
</html>
```

In Listing 2.64, the HTML form has a text field and button. In the `<input>` tag for the button, the `onclick` attribute is used to specify the `findPercentage()` function as the event handler for the `click` event of the button. This implies that the `findPercentage()` function is called when the user clicks the button. The `findPercentage()` function takes the value entered in the `TotalMarks` text field as its argument and displays the equivalent percentage on a new Web page.

When you open the `handleEvent.html` page, the output appears, as shown in Figure 2.69:

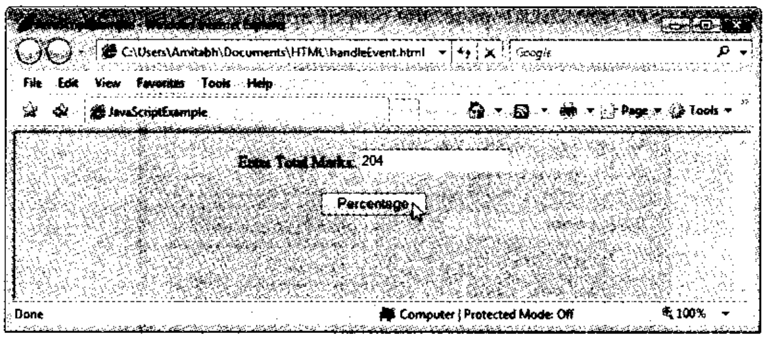
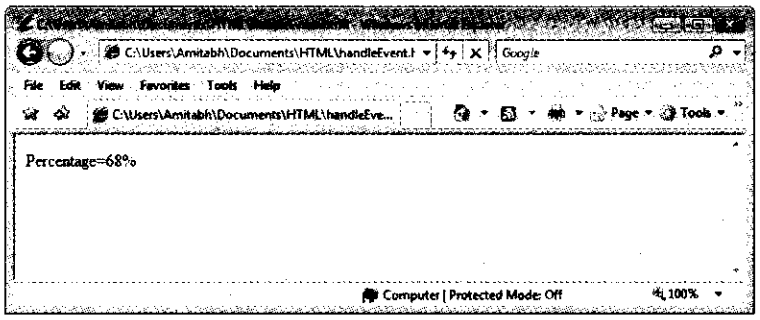

C:\Users\Amitabh\Documents\HTML\handleEvent.html
Google
File Edit View Favorites Tools Help
JavaScriptExample
Page Tools
Enter Total Marks: 204
Percentage
Done
Computer | Protected Mode: Off
100%

Figure 2.69: Handling Event

Enter a value in the `Enter Total Marks` field and click the `Percentage` button, as shown in Figure 2.69. For example, if you enter the value, `204` and then click the `Percentage` button, the equivalent percentage appears on a new Web page, as shown in Figure 2.70:


C:\Users\Amitabh\Documents\HTML\handleEvent.h
Google
File Edit View Favorites Tools Help
C:\Users\Amitabh\Documents\HTML\handleEve...
Page Tools
Computer | Protected Mode: Off
100%

Percentage=68%

Figure 2.70: Output After Calculating the Percentage

In Figure 2.70, you can observe the calculated percentage on the Web page.

Using the onclick Event

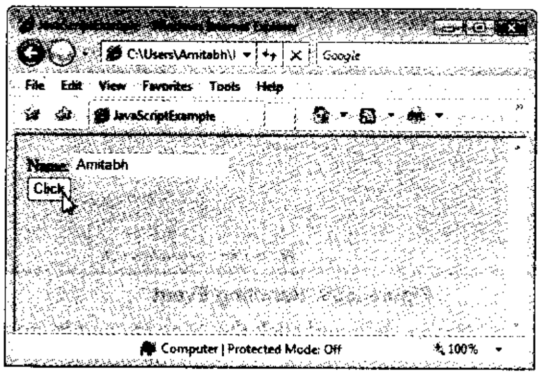
The `onclick` event occurs and runs a specified function when you click an object, such as a button. The `onclick` event can only be added to visible elements on the page, such as form buttons and check boxes. However, it cannot be added within the `<head>` tag.

Let's create a Web page, named `onclickEvent.html` to understand how to handle the `onclick` event in an HTML document. You can find the `onclickEvent.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.65 shows the code of the `onclickEvent.html` page:

Listing 2.65: Using the onclick Event

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">

Name: <input type="text" id="field1" value="Enter Your Name">

<br>

<button onClick= "alert ('Hi ' + document.getElementById('field1').value + ', Welcome to
  Javacript')">Click</button>

</body>
</html>
```

When you open the `onclickEvent.html` page, the output appears after entering your name, as shown in Figure 2.71:

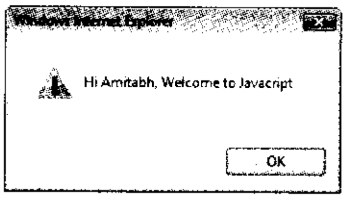

C:\Users\Amitabh\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Click
Computer | Protected Mode: Off
100%

Name: Amitabh

Figure 2.71: Handling the onclick Event

In Figure 2.71, you can notice that when you click the `Click` button on the form, the `onclick` event occurs and the alert message box appears to display a message.

When you enter your name in the `Name` textbox and click the `Click` button, the alert message box appears, as shown in Figure 2.72:



Hi Amitabh, Welcome to Javacript

Figure 2.72: Alert Message Box After Clicking the Button

Let's learn about the usage of onload event in the next section.

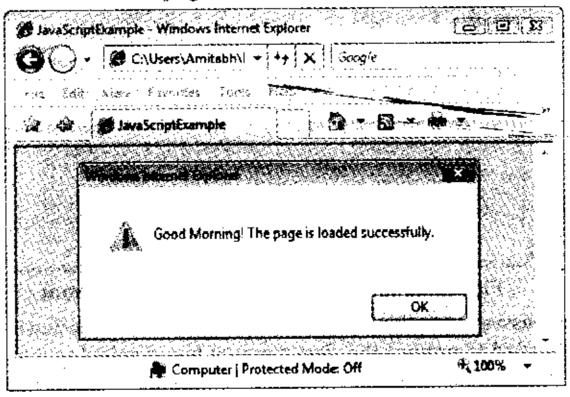
Using the onload Event

The `onload` event is triggered when the user enters the page. This event is generally used to get the information about the browser type and version, and load the Web page according to the information found. You can also use the `onload` event to use the cookies that should be set when a user opens a page. For example, when the user enters the Web page for the first time, you could have a pop-up asking for the user name. Once the user enters the name, it is stored in a cookie. Next time when the visitor enters the same page, you could have another pop-up to wish him with his name. Let's create a Web page, named `onloadEvent.html` to understand how to handle the `onload` event in an HTML document. You can find the `onloadEvent.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.66 shows the code of the `onloadEvent.html` page:

Listing 2.66: Using the onload Event

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<script type="text/javascript">
function hello()
{
alert ("Good Morning! The page is loaded successfully.");
}
</script>
<body bgcolor="pink" onload="hello()">
</body>
</html>
```

When you open the `onloadEvent.html` page, the output appears, as shown in Figure 2.73:


JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
Google
JavaScriptExample
Computer | Protected Mode: Off
100%

Good Morning! The page is loaded successfully.

OK

Figure 2.73: Handling the Onload Event

In Figure 2.73, you can observe that when the body of the form is loaded, the `onload` event is triggered and the alert message box appears to display a message.

Using the onmouseover Event

The `onmouseover` event is triggered when the mouse pointer passes over a specified object. It denotes that something will happen when the mouse pointer moves over the active object.

Let's create a Web page, named `onmouseoverEvent.html` to learn how to handle the `onmouseover` event in an HTML document. You can find the `onmouseoverEvent.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.67 shows the code of the `onmouseoverEvent.html` page:

Listing 2.67: Using the onmouseover Event

```
<html>
<head>
<title>JavaScriptExample</title>
```

```
</head>
<script type="text/javascript">
function mouseover()
{
alert ("Do You want Google as your Home page ?");
}
</script>
<body bgcolor="pink">
<a href="www.google.com" onmouseover="mouseOver()">
<img alt="Google" src="C:\Users\Amitabh\Pictures\google.jpg" id="img1" />
</a>
</body>
</html>
```

NOTE

You need to change the path of the image specified in the src attribute according to the image location on your system.

When you open the `onmouseoverEvent.html` page, the output appears displaying the specified image, as shown in Figure 2.74:

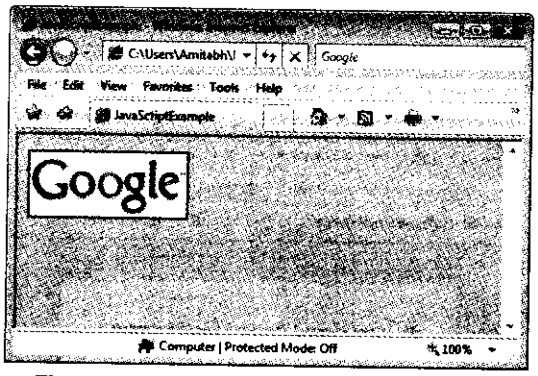
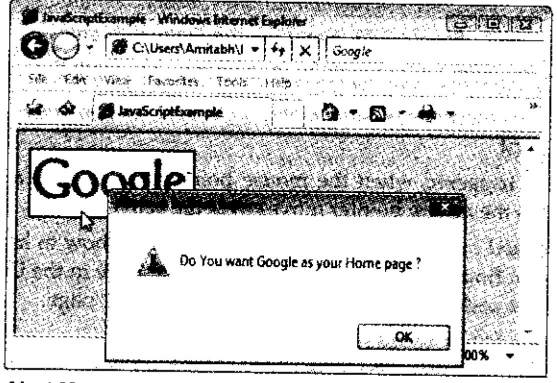

C:\Users\Amitabh\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Google
Computer | Protected Mode: Off
100%

Figure 2.74: Handling the onmouseover Event

When the mouse pointer passes from the image on the form, the `onmouseover` event is triggered and the `mouseOver()` function is called to prompt the alert message box displaying a message, as shown in Figure 2.75:



Do You want Google as your Home page ?

Figure 2.75: Alert Message Box after Moving the Mouse Pointer over the Image

Let's learn to use the onreset event in the next section.

Using the onreset Event

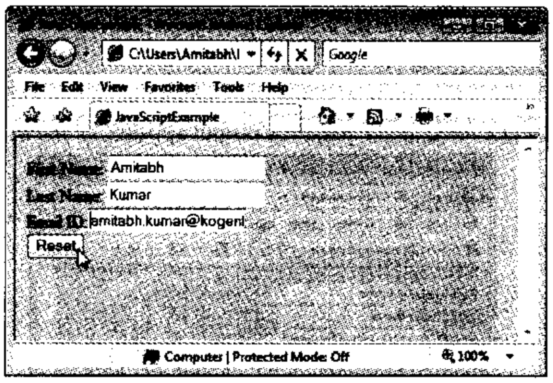
The `onreset` event is triggered when the user clicks the `Reset` button on the form or when JavaScript executes the `form.reset()` method. When the `Reset` button is clicked, the `onreset` event calls a function to be executed.

Let's create a Web page, named `onresetEvent.html` to learn how to handle the `onreset` event in an HTML document. You can find the `onresetEvent.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.68 shows the code of the `onresetEvent.html` page:

Listing 2.68: Using the onreset Event

```
<html>
<head>
<title>JavaScriptExample</title>
</head>

<body bgcolor="pink">

<form onreset="alert('All the data in the form will be reset.')">
First Name: <input type="text" name="fname" value="Enter Your First Name" />

<br>

Last Name: <input type="text" name="lname" value="Enter Your Last Name" />

<br>

Email ID:<input type="text" name="email" value="Enter Your Email ID" />

<br>

<input type="reset" value="Reset">

</form>

</body>
</html>
```

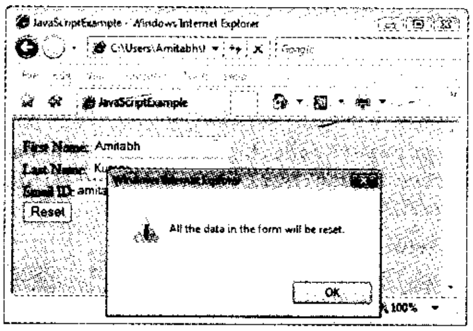
When you open the `onresetEvent.html` page, the output appears after entering the required details in the respective fields, as shown in Figure 2.76:


C:\Users\Amitabh\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

First Name: Amitabh
Last Name: Kumar
Email ID: amitabh.kumar@kogent
Reset

Figure 2.76: Handling the onreset Event

When you click the `Reset` button, the `onreset` event is triggered to display the alert message box, as shown in Figure 2.77:



First Name: Amitabh
Last Name: Ku
Email ID: amita

All the data in the form will be reset.

Figure 2.77: Alert Message Box After Clicking the Reset Button

Now, let's learn the usage of onsubmit event in an HTML document.

Using the onsubmit Event

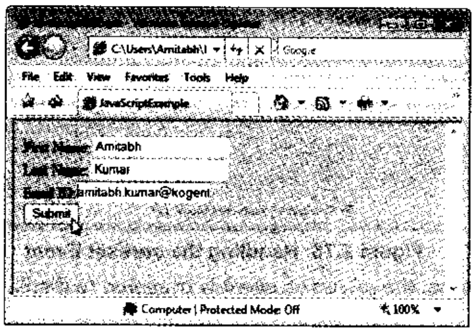
Similar to the `onreset` event, the `onsubmit` event is triggered when the `Submit` button is clicked on the form. The `onsubmit` event is generally used to validate all the values provided in the fields of a form before submitting the form. If the form validation fails and the `onsubmit` event returns false, the data is not sent to the server.

Let's create a Web page, named `onsubmitEvent.html` to learn how to handle the `onsubmit` event in an HTML document. You can find the `onsubmitEvent.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.69 shows the code of the `onsubmitEvent.html` page:

Listing 2.69: Using the onsubmit Event

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<form onsubmit="alert('Are you sure to submit the details ?')">
First Name: <input type="text" name="fname" value="Enter Your First Name" />
<br>
Last Name: <input type="text" name="lname" value="Enter Your Last Name" />
<br>
Email ID:<input type="text" name="email" value="Enter Your Email ID" />
<br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

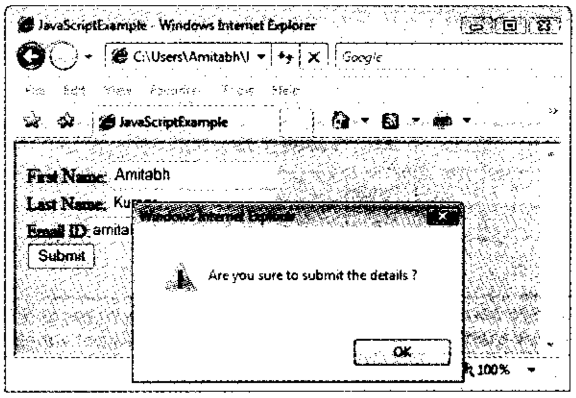
When you open the `onsubmitEvent.html` page, the output appears, as shown in Figure 2.78:


C:\Users\Amitabh\
File Edit View Favorites Tools Help
JavaScriptExample
Submit
Computer | Protected Mode: Off
100%

First Name: Amitabh
Last Name: Kumar
Email ID: amitabh.kumar@kogent

Figure 2.78: Handling the onsubmit Event

when you enter the required information in the form fields and click the `Submit` button, the alert message box appears, as shown in Figure 2.79:



Are you sure to submit the details ?

Figure 2.79: Alert Message Box After Clicking the Submit Button

Let's learn the usage of variables in JavaScript in detail.

Using Variables in JavaScript

In JavaScript, the data can be stored in variables which are the named locations in the memory. Variables are used to temporarily store the data and have a name, value, and memory address. The name of the variable uniquely identifies the variable, the value refers to the data that is stored in the variable, and the memory address refers to the memory location of the variable.

You need to declare a variable before using it to store data. When you declare a variable, it means you are introducing it in the script. While declaring a variable, you need to provide a user friendly and unique name for the variable.

The syntax to declare a variable is:

```
var variable_name;
```

In the preceding syntax, `var` is a keyword and `variable_name` refers to the name of the variable.

NOTE

The name of a variable or any programming entity is knows as an identifier. In a script, some of the identifiers are user-defined and some are predefined. The predefined identifiers are known as keywords and are reserved for specific purposes, such as declaring variables. You cannot use keywords as user-defined identifiers, for example, you cannot use var as a variable name as this may result in an error.

Now, let's learn about the scope of variables in detail in the next section.

Scope of variables

It is important to note that in JavaScript, there is no limit on the length (number of characters) of the name of a variable. However, there are certain rules that define the scope of variables. You need to keep these rules in mind while naming variables in JavaScript:

- The name of the variable must be unique in the script.
- The name of the variable can only contain letters, numbers, and underscores. It cannot contain spaces and certain punctuation characters.
- The name of the variable must begin with an uppercase or lowercase letter or an underscore; it cannot begin with a number.
- The name of the variable cannot be enclosed by single or double quotations.
- The name of the variable cannot be same as any keyword in JavaScript.

In Table 2.13, you can have a look at some examples of invalid and valid names of variables in JavaScript:

Table 2.13: Examples of Invalid and Valid Names of Variables

Invalid Variable Name	Valid Variable Name
89Books	Books89
var	var1
My*Variable//One	My_Variable_One
"count"	count
_Book Titles	_Book_Titles

NOTE

Unlike HTML, JavaScript is a case-sensitive language, that is, it treats uppercase and lowercase letters differently. For instance, the variable names, MyVariable and myvariable represent two different variables and not the same variable.

TIP

While naming a variable, if the name has two or more words, then you can separate the words with an underscore. You can also write the first word in lowercase and the first letter of the subsequent words in uppercase, for example, numberComputerBooks.

Let's take a look at the example, given in Listing 2.70, to declare a variable in a JavaScript script:

Listing 2.70: Declaring a Variable in JavaScript

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var countBooks;
</script>
</body>
</html>
```

In Listing 2.70, you can observe that the Web page has script in which a variable named `countBooks` is declared by using the `var` keyword.

In JavaScript, you can declare multiple variables in the same statement. To declare multiple variables simultaneously, you need to separate the individual variables by a comma (,).

Listing 2.71 shows an example of how to declare multiple variables:

Listing 2.71: Declaring Multiple Variables in JavaScript

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var countBooks, minBookPrice, maxBookPrice, bookName;
</script>
</body>
</html>
```

In Listing 2.71, you can see that the script within the `<body>` and `</body>` tags has four variables named `countBooks`, `minBookPrice`, `maxBookPrice` and `bookName` that are declared in the same statement.

A variable can store only one value at a time; however, it may store different values at different times in the script. In JavaScript, it is not necessary that a variable has to store only a particular type of data. A JavaScript variable can store a value of any data type. For example, the same variable can store both a string and a number at different times in the script. This is why, JavaScript is considered a poorly typed language.

In JavaScript, you can assign values to a variable either at the time of declaration or after that. In addition, you can assign values to a variable as many times as you want as per your requirement. However, the variable stores only the most recent value assigned to it. After assigning a value to a variable, you can use it in the script. You can access the value of a variable by using its name in the script.

The syntax to assign a value to a variable after it is declared is:

```
variable_name = value;
```

In the preceding syntax, `variable_name` refers to the name of the variable, = is the assignment operator, and `value` refers to the value that is to be stored in the variable.

NOTE

You will read more about the assignment operator later in the chapter.

As stated earlier, you can assign values to a variable while declaring it by using the assignment operator.

Let's create a Web page, named `assigningVariables.html` to assign values to a variable. You can find the `assigningVariables.html` file in the Code\HTML\Chapter 2 folder on the CD.

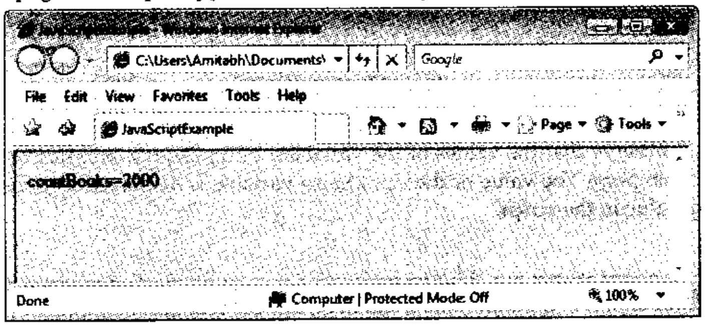
Listing 2.72 shows the code of `assigningVariables.html` page:

Listing 2.72: Assigning Values to Variables

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var countBooks=50;
countBooks=2000;
document.write("countBooks="+countBooks);
</script>
</body>
</html>
```

In Listing 2.72, you can observe that in the first statement of the script, the `countBooks` variable is assigned a value of `50` while declaring it. In the second statement of the script, the `countBooks` variable is assigned a different value, `2000`. Therefore, the `countBooks` variable now stores `2000` and not `50`. The value of the `countBooks` variable is displayed with the `document.write("countBooks="+countBooks)` statement where + is an operator to append two strings.

When you open this page, the output appears, as shown in Figure 2.80:


C:\Users\Amitabh\Documents\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Page Tools
Done
Computer | Protected Mode: Off
100%

```
countBooks=2000
```

Figure 2.80: Assigning Values to Variables

As you can observe in Figure 2.80, the `countBooks` variable stores the most recently assigned value, `2000` and not the initially assigned value, `50`.

If you have declared multiple variables in the same statement, you can assign values to them. You can do this by suffixing the variable name with the assignment operator followed by the value for that variable. You can also assign the value of one variable to another.

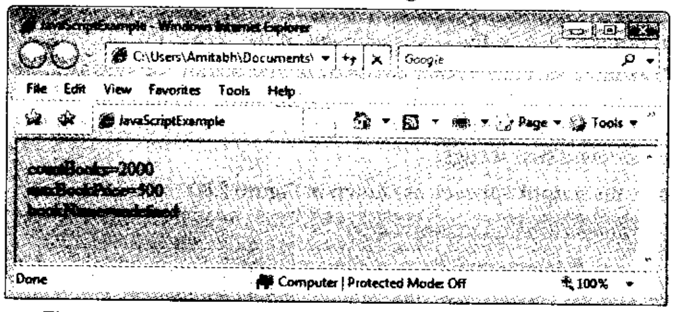
Let's create a Web page, named `assigningMultipleVariables.html` to assign values to multiple variables. You can find the `assigningMultipleVariables.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.73 shows the code of `assigningMultipleVariables.html` page:

Listing 2.73: Assigning Values to Multiple Variables while Declaring

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var countBooks=50, minBookPrice=150, maxBookPrice=3000, bookName;
countBooks=2000;
minBookPrice=500;
maxBookPrice=minBookPrice;
document.write("countBooks="+countBooks+"<br/>");
document.write("maxBookPrice="+maxBookPrice+"<br/>");
document.write("bookName="+bookName);
</script>
</body>
</html>
```

In Listing 2.73, the variables, `countBooks`, `minBookPrice`, `maxBookPrice`, and `bookName` are declared in the same statement. The `countBooks`, `minBookPrice`, and `maxBookPrice` variables are assigned values while they are declared, but the `bookName` variable is not. You can also notice that the value of the `minBookPrice` is assigned to the `maxBookPrice` variable. This implies that the `minBookPrice` and `maxBookPrice` variables have the same value. The value of the `countBooks`, `maxBookPrice`, and `bookName` variables are displayed using the `document` object and its `write()` method. Note that the `<br/>` tag is used in the `write()` method to insert a new line.

When you open this page, the output appears, as shown in Figure 2.81:


JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\Documents\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Page
Tools
Done
Computer | Protected Mode: Off
100%

```
countBooks=2000
maxBookPrice=500
bookName=undefined
```

Figure 2.81: Assigning Values to Variables while Declaring

In Figure 2.81, you can observe that the values of the variables, `countBooks`, `maxBookPrice`, and `bookName`, are displayed on the Web page. The value of the `bookName` variable is displayed as `undefined` because it has not been assigned any value in the script.

NOTE

When you do not assign any value to a variable, it stores an undefined value.

Let's learn to use array in JavaScript in detail.

Using Array in JavaScript

Consider a situation where you need to store and use the price of 100 books. For this, you can declare 100 variables in the script to store the prices. However, declaring so many variables is rather cumbersome and inefficient. In such a case, instead of using the variables, you can use an array to store the book prices. An array is a named collection of different values. The individual values are represented by their respective array elements, each of which has a unique index. The index is a whole number that indicates the position of the array elements with respect to one another.

Similar to variables, you need to introduce the array that you intend to use in the script. In JavaScript, arrays are instances of the `Array` object You can create or define arrays by using the `new` keyword. In JavaScript, the values in an array need not be of the same data type. For instance, an array can have both strings and numbers.

The syntax to define an array is:

```
var array_name=new Array(array_length);
```

In the preceding syntax, `var` and `new` are JavaScript keywords while `Array` refers to the `Array` object. The `array_name` refers to the name of the array and `array_length` refers to the length or the number of array elements. Note that specifying the number of array elements while defining an array is optional.

Let's take a look at an example, given in Listing 2.74, of defining arrays in a script:

Listing 2.74: Defining an Array in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var bookPrices=new Array(3);
var bookNames=new Array();
</script>
</body>
</html>
```

In Listing 2.74, `bookPrices` and `bookNames` have been defined as arrays by using the `new` keyword. The `bookPrices` array has three array elements, while the `bookNames` array has zero array elements.

As you know, you can specify the number of array elements while defining an array. However, the array is not yet populated, that is, the array elements do not store any value and display the `undefined` value. You can assign values to the array elements while defining the array as shown:

```
var bookNames=new Array("Pelican Brief", "Who Moved My Cheese", "The Call of
the Wild");
```

In this code, the `bookNames` array is populated while defining it. The values for the array elements are specified within the parentheses after the `Array` keyword. The individual values are separated by a comma. Note that the `bookNames` array has only string values.

you can access the array elements after defining them. In JavaScript, you can use the array that you have defined in a script by accessing its array elements. As you know, each array element stores a single value and has an index that indicates the relative position of an array element. The starting index of arrays in JavaScript is zero. That is, the first array element has an index of zero, the second array element has an index of one, the third array element has an index of two, and so on.

NOTE

The highest index in an array is one less than the total number of array elements. For instance, if an array has 50 array elements, then the highest index is 49.

The syntax to access an array element is:

```
array_name[array_element_index];
```

In the preceding syntax, `array_name` refers to the name of the array, which is followed by a pair of square brackets. The `array_element_index` refers to the index of a particular array element.

Let's create a Web page, named `accessArray.html` to access the array elements. You can find the `accessArray.html` file in the Code\HTML\Chapter 2 folder on the CD.

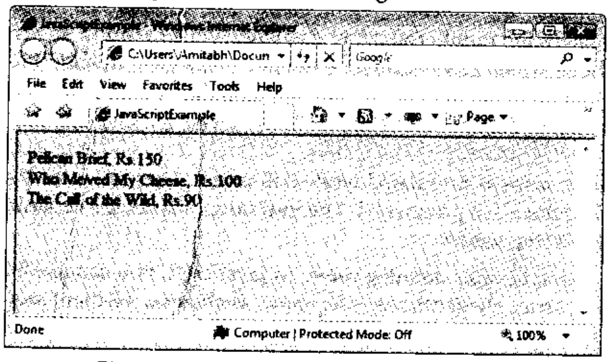
Listing 2.75 shows the code of `accessArray.html` page:

Listing 2.75: Accessing an Array Element in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var bookPrices=new Array(3);
var bookNames=new Array("Pelican Brief", "Who Moved My Cheese", "The Call of
the Wild");
bookPrices[0]=150;
bookPrices[1]=100;
bookPrices[2]=90;
for(var i=0;i<3;i++)
{
document.write(bookNames[i]+", Rs."+bookPrices[i]+"<br/>");
}
</script>
</body>
</html>
```

In Listing 2.75, the `bookPrices` array is populated by accessing and assigning values to the individual array elements. Values of the array elements of the `bookPrices` and `bookNames` arrays are accessed in the `for` loop. The variable `i` declared in the `for` loop is used to specify the index of the array elements. When `i` is 0, it refers to the first element of the `bookNames` and `bookPrices` arrays; when `i` is `1`, it refers to the second elements of both the arrays, and so on.

When you open this page, the output appears, as shown in Figure 2.82:



```
Pelican Brief, Rs.150
Who Moved My Cheese, Rs.100
The Call of the Wild, Rs.90
```

Figure 2.82: Accessing an Array Element

In Figure 2.82, you can observe that the name and prices of the different books appears on the Web page.

Using Array Methods

When an array is created, it also gets properties and methods to manipulate the elements contained in the array. Some of these methods are listed as follows:

- ❑ length
- ❑ reverse

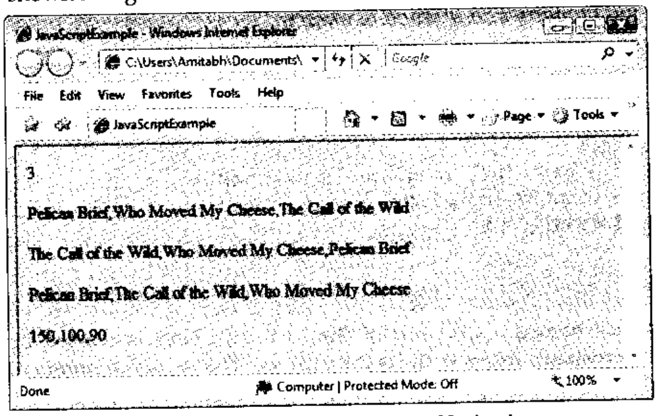
- sort
- concat
- join

Let's create a Web page, named `ArrayMethods.html` to learn the use of these methods. You can find the `ArrayMethods.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.76 shows the code of the `ArrayMethods.html` page:

Listing 2.76: Using Array Methods in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var bookPrices=new Array(3);
var bookNames=new Array("Pelican Brief", "Who Moved My Cheese", "The Call of the Wild");
bookPrices[0]=150;
bookPrices[1]=100;
bookPrices[2]=90;

document.write(bookPrices.length+ "<br>");

document.write("<br>" + bookNames.concat()+ "<br>");

document.write("<br>" + bookNames.reverse()+ "<br>");

document.write("<br>" + bookNames.sort() + "<br>");

document.write("<br>" + bookPrices.join() + "<br>");

</script>
</body>
</html>
```

In Listing 2.76, you can observe that the length property returns the length of the `bookPrices` array. The `concat` method concatenates all the elements of the `bookNames` array; the `reverse` method reverses elements of the `bookNames` array; the `sort` method sorts the elements of the `bookNames` array in ascending order; and the `join` method combines all elements of the `bookNames` array as a single element. When you open this page, the output appears, as shown in Figure 2.83:


JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\Documents\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Page Tools
Done
Computer | Protected Mode: Off
100%

```
3

Pelican Brief,Who Moved My Cheese,The Call of the Wild

The Call of the Wild,Who Moved My Cheese,Pelican Brief

Pelican Brief,The Call of the Wild,Who Moved My Cheese

150,100,90
```

Figure 2.83: Accessing Array Methods

In Figure 2.83, you can observe that the names and prices of the books appear on the Web page by using the different array methods.

Creating Objects in JavaScript

As stated earlier, JavaScript is an object-based scripting language. An object is a programmable entity that you can use in the script. JavaScript provides several in-built objects that represent different aspects of a Web page. Every object has certain properties and methods that help to work with the specific aspects of a Web page. A property of an object refers to an attribute of the object, while a method refers to a particular action or task that can be performed on the object. The properties and methods differ from one object to the other.

In JavaScript, the in-built objects pertain to the type of content displayed on the Web page. For instance, some of them allow you to work with text, while some allow you to work with date and time. Table 2.14 lists the commonly used JavaScript objects and their important properties and methods:

Table 2.14: Common JavaScript Objects and their Important Properties and Methods

Object	Description	Properties	Methods
String	Allows you to create and manipulate a string of characters that are enclosed in single or double quotes	length	bold(), charAt(), concat(), indexOf(), match(), replace(), search(), substring(), toLowerCase(), toUpperCase()
Array	Allows you to create and manipulate a series of values that are represented by a single name	length	concat(), join(), reverse(), sort(), valueOf()
Date	Allows you to create and manipulate dates and times		Date(), getDate(), getDay(), getMonth(), getYear()

You can access and use the properties and methods of the in-built JavaScript objects. For the in-built JavaScript objects, you need to first create an instance of that object by using the new keyword. With the newly created instance, you can then access the properties and methods of the objects by using a dot (.) between the instance name and the property or method name. However, for the DOM objects, you need not create an instance of the object.

NOTE

Keywords are special words that are reserved for some specific purposes in the language. JavaScript has many keywords, for example the new keyword allows you to create instances of objects.

You can use the object name to access its property or method, as shown in Listing 2.77:

Listing 2.77: Accessing the Property and Method of Objects

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var myString=new String("Welcome");
document.write(myString.length);
</script>
</body>
</html>
```

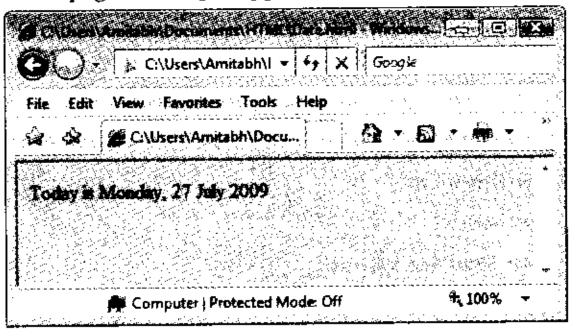
In Listing 2.77, an instance named `myString` of the `String` object is created by using the `new` and `var` keywords. The `myString` instance refers to the text `Welcome`. The length or number of characters in `myString` is accessed by using the `length` property in the `write()` method of the `document` object. Note that the `length` property of `String` object and the `write()` method of the `document` object are accessed by using a dot (.) between them and the object name.

Let's take another example to access the methods of the `Date` object. The date and time that is used in the JavaScript is taken from the system clock and calendar that is running the browser application in which the script is loaded. You can access the system clock information by creating a `Date` object. Let's create a Web page, named `Date.html` to access the system date and time information. You can find the `Date.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.78 shows the code of the `Date.html` page:

Listing 2.78: Accessing the Property and Method of Date Object

```
<html>
<head>
<script language="javascript" type="text/javascript">

var dt=new Date()
var day=dt.getDay()
var month=dt.getMonth()
var year=dt.getYear()
var date=dt.getDate()

var days=new Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")
var months=new
  Array("January","February","March","April","May","June","July","August","September",
  "October","November","December")
</script>
</head>
<body bgcolor="pink">
<script language="JavaScript" type="text/javascript">
document.write("Today is " + days[day] + ", " + date + " " + months[month] + " " + year)
</script>
</body>
</html>
```

In Listing 2.78, we have created two arrays, named `days` and `months` to store the different days and months respectively. By default, the `getDay()` and `getMonth()` methods return the days and months in integer value. When you open the `Date.html` page, the output appears, as shown in Figure 2.84:


C:\Users\Amitabh\
Google
File Edit View Favorites Tools Help
C:\Users\Amitabh\Docu...
Computer | Protected Mode: Off
100%

Today is Monday, 27 July 2009

Figure 2.84: Accessing Date Methods

In Figure 2.84, you can observe that the date appears on the Web page by using the different Date methods.

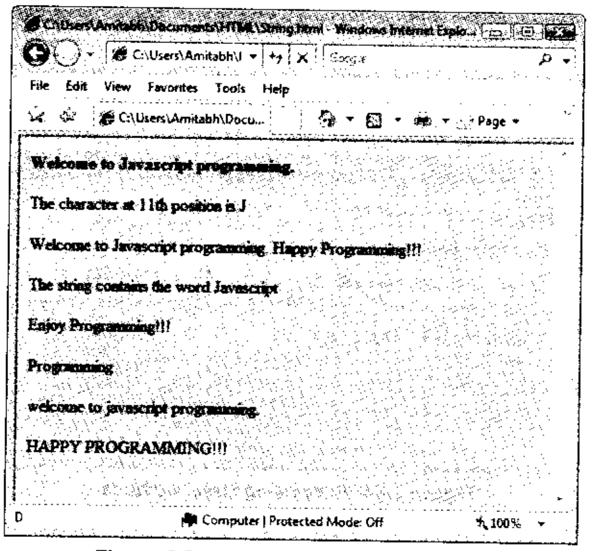
Similarly, let's create another Web page, named `String.html` to access the different methods of the `String` object. You can find the `String.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.79 shows the code of the `String.html` page:

Listing 2.79: Accessing the Property and Method of String Object

```
<html>
<head>
</head>
<body bgcolor="pink">
<script type="text/javascript">
```

```
var st= "Welcome to Javascript programming. "
var str="Happy Programming!!!"
document.write(st.bold() + "<BR><BR>")
document.write("The character at 11th position is " + st.charAt(11) + "<BR><BR> ")
document.write(st.concat(str) + "<BR><BR> ")

if (st.match(/(Javascript)/))
{
document.write("The string contains the word Javascript" + "<BR><BR>")
}

if (str.search(/(Happy)/) != -1)
{
document.write(str.replace(/(Happy)/, "Enjoy") + "<BR><BR>")
}

document.write(str.substring(6, 17) + "<BR><BR>")
document.write(st.toLowerCase() + "<BR><BR>")
document.write(str.toUpperCase() + "<BR><BR>")
</script>
</body>
</html>
```

In Listing 2.79, we have created two variables, named `st` and `str` to store two different string values. Then we have used different methods of the `String` object to perform the desired operations, such as changing the case of characters and searching a particular text in the string. When you open the `String.html` page, the output appears, as shown in Figure 2.85:


C:\Users\Amitabh\Documents\HTML\String.html - Windows Internet Explo...
C:\Users\Amitabh\
File Edit View Favorites Tools Help
C:\Users\Amitabh\Docu...
Page
Computer | Protected Mode: Off
100%

```
Welcome to Javascript programming.

The character at 11th position is J

Welcome to Javascript programming. Happy Programming!!!

The string contains the word Javascript

Enjoy Programming!!!

Programming

welcome to javascript programming.

HAPPY PROGRAMMING!!!
```

Figure 2.85: Accessing String Methods

In Figure 2.85, you can observe that the string appears on the Web page in different formats by using the different `String` methods.

JavaScript also supports various HTML Document Object Model (DOM) objects. The DOM refers to a hierarchical collection of objects that allow you to work with HTML documents. The DOM objects, shown in Figure 2.86, represent different aspects of the Web browser and content in an HTML document in the form of a hierarchy.

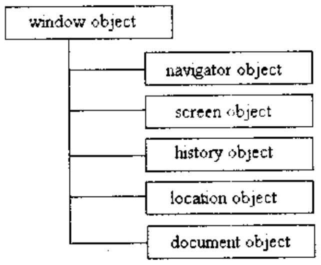


Figure 2.86: DOM Objects

As you can see in Figure 2.86, the `window` object is the container for rest of the `navigator`, `screen`, `history`, `location`, and `document` objects. Note that all these objects are DOM objects.

Similar to the JavaScript objects, the DOM objects also have properties and methods. Table 2.15 lists the DOM objects along with their important properties and methods:

Table 2.15: DOM Objects and their Important Properties and Methods

DOM Object	Description	Properties	Methods
window	Represents the Web browser window. Note that if the Web page is divided into frames, then each frame corresponds to a `window` object.	defaultStatus, name, status, self, closed, history, location, document	alert(), close(), confirm(), focus(), open(), print(), prompt()
navigator	Allows you to access various information about the user's Web browser.	appCodeName, appName, appVersion, cookieEnabled, platform,	javaEnabled()
screen	Allows you to access different information about the user's screen.	availHeight, availWidth, colorDepth, height, width	
history	Represents a list of URLs that the user has already accessed. In case, the Web page has frames, then each frame has its own list of URLs.	length	back(), forward(), go()
location	Allows you to access information about the URL of the Web page that is currently opened in a window or frame.	host, hostname, href, pathname, port, protocol, search	assign(), reload(), replace()
document	Represents the HTML document or Web page that is currently opened in the Web browser.	lastModified, title, URL,	close(), getElementById(), getElementsByName(), getElementsByTagName(), open(), write(), writeln()

NOTE

The history, location, and document objects are also properties of the window object.

Let's learn about the object hierarchy in JavaScript.

Object Hierarchy in JavaScript

We know that JavaScript includes a number of objects that are contained within each other. JavaScript objects have a container to contain object relationship rather than a class and subclass relationship. However, JavaScript properties are not inherited from one type of object to another. There are two main types of JavaScript objects:

- **Language Objects**—Provides by the language and not dependent on other objects.
- **Navigator**—Provides by the client browser. These objects are the subobjects of the navigator object.

Figure 2.87 displays the object hierarchy in JavaScript:

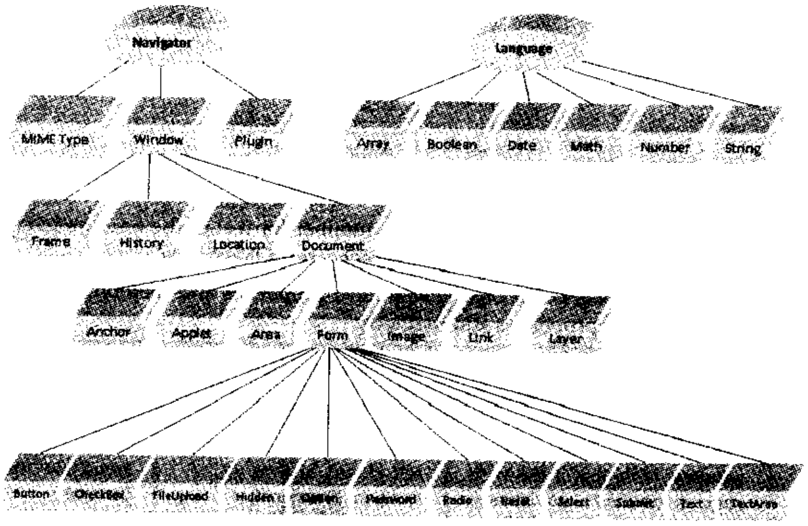


Figure 2.87: JavaScript Object Hierarchy

Now, let's learn to use operators in detail.

Using Operators

As you know, variables are used to temporarily store the data that you want to use in a script. You can now modify or change the data by manipulating the respective variables. For instance, you have declared two variables named `marksMaths` and `marksScience` and want to add the value of these two variables. In such a case, you need to manipulate these variables to get the desired result. One of the most common ways of manipulating variables is by using operators. An operator is a symbol or a word that is reserved for a special task or action. Every operator works on one or more operands, that is, an operator takes the values of its operands, performs an action, and returns the result of that action.

In JavaScript, there is a whole gamut of operators that you can use as per your requirements. Some of the operators work on a single operand, while some work on two operands. Moreover, some operators work on numbers, while others work on strings and Boolean values. Table 2.16 lists some of the commonly used operators in JavaScript:

Table 2.16: Commonly used Operators in JavaScript

Operator	Description	Example
Arithmetic Operators		
+	Adds two numbers or joins two strings. It also represents a positive number when it is prefixed to a number.	`45+10` returns `55` "My " + "Name" returns "My Name"
-	Subtracts two numbers or represents a negative number.	`-45+10` returns `-35`
*	Multiplies two numbers.	`45*10` returns `450`
/	Divides two numbers evenly and returns the quotient.	`45/10` returns `4.5`
%	Divides two numbers and returns the remainder.	`45%10` returns `5`

Table 2.16: Commonly used Operators in JavaScript

Operator	Description	Example

++	Increments the value of a number by one. It can be prefixed or suffixed to a number. When prefixed, the value is incremented in the current statement, and when suffixed, the value is incremented after the current statement.	`myVar1=45`
		`myVar2=++myVar1` assigns `46` to `myVar2`
		`myVar2=myVar1++` assigns `45` to `myVar2`

--	Decrements the value of a number by one. It can be prefixed or suffixed to a number. When prefixed, the value is decremented in the current statement, and when suffixed, the value is decremented after the current statement.	`myVar1=45`
		`myVar2=--myVar1` assigns `44` to `myVar2`
		`myVar2=myVar1--` assigns `45` to `myVar2`

Assignment Operators		
=	Assigns the value on the right hand side to the variable on the left hand side	`myVar=90`

+=	Adds the right hand side operand to the left hand side operand and assigns the result to the left hand side operand.	`myVar1=45, myVar2=10`
		`myVar1+=myVar2` assigns `55` to `myVar1`

-=	Subtracts the right hand side operand from the left hand side operand and assigns the result to the left hand side operand.	`myVar1=45, myVar2=10`
		`myVar1-=myVar2` assigns `35` to `myVar1`

*=	Multiplies the right hand side operand and the left hand side operand and assigns the result to the left hand side operand.	`myVar1=45, myVar2=10`
		`myVar1*=myVar2` assigns `450` to `myVar1`

/=	Divides the left hand side operand by the right hand side operand and assigns the quotient to the left hand side operand.	`myVar1=45, myVar2=10`
		`myVar1/=myVar2` assigns `4.5` to `myVar1`

%=	Divides the left hand side operand by the right hand side operand and assigns the remainder to the left hand side operand.	`myVar1=45, myVar2=10`
		`myVar1%=myVar2` assigns `5` to `myVar1`

Comparison Operators		
==	Returns `true` if both the operands are equal, otherwise returns `false`.	`45==10` returns `false`
!=	Returns `true` if both the operands are not equal, otherwise returns `false`.	`45!=10` returns `true`
>	Returns `true` if the left hand side operand is greater than the right hand side operand, otherwise returns `false`.	`45>10` returns `true`
>=	Returns `true` if the left hand side operand is greater than or equal to the right hand side operand, otherwise returns `false`.	`45>=10` returns `true`
<	Returns `true` if the left hand side operand is less than the right hand side operand, otherwise returns `false`.	`45<10` returns `false`
<=	Returns `true` if the left hand side operand is less than or equal to the right hand side operand, otherwise returns `false`.	`45<=10` returns `false`

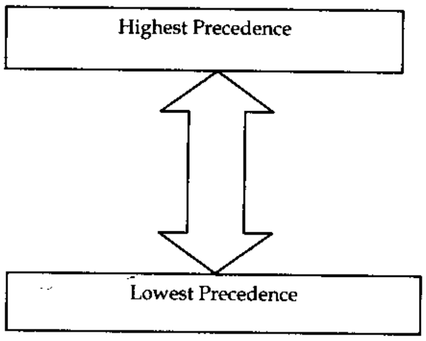
Table 2.16: Commonly used Operators in JavaScript

Operator	Description	Example
Logical Operators		
&&	Returns `true` only if both the operands are `true`, otherwise returns `false`.	`true&&false` returns `false`
||	Returns `true` only if either of the operands are `true`, It returns `false` when both the operands are `false`.	`true||false` returns `true`
!	Negates the operand, that is, returns `true` if the operand is `false` and returns `false` if the operand is `true`.	`!true` returns `false`
Conditional Operator		
?:	Returns the second operand if the first operand is `true`, otherwise, returns the third operand.	`myVar1=45, myVar2=10` `myResult=(myVar1<myVar2)?myVar1: myVar2` returns `10`

You can use operators to perform various functions, such as mathematical calculations, modifying strings, and making decisions. The combination of operators and their operands form expressions. An expression either has a specific value or evaluates to a value, for example, `45` and `var1+var2`. In JavaScript and other programming languages, expressions are evaluated in a particular order. The order of evaluation is determined by the precedence of the operators with respect to one another. Operators are evaluated in the order of higher precedence to lower precedence. Table 2.17 lists the operators in the decreasing order of their precedence:

Table 2.17: Precedence of Some Commonly Used Operators from Highest to Lowest

Operator
!, - (negative sign), ++, --, typeof
*, /, %
+, - (subtraction)
<, <=, >,>=
==, !=
&&, ||, ?:
=, +=, -=, *=, /=,%=,



In case there are two or more operators of the same precedence in an expression, the operators are evaluated as they appear in the expression from left to right. For instance, the expression, `45+5-2*10` evaluates to `30`. The * operator is evaluated first, followed by the + operator and the - operator.

It is very essential that you are aware of the precedence of the operators; otherwise, you may get unexpected results. However, you can also override the precedence of an operator by using parentheses. The parentheses have the highest precedence and any expression that is enclosed within parentheses is evaluated first.

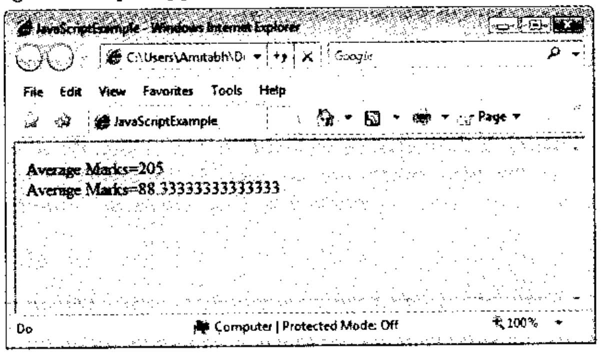
Let's create a Web page, named `OperatorPrecedence.html` to learn the uses of the operators. You can find the `OperatorPrecedence.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.80 shows the code of the `OperatorPrecedence.html` page:

Listing 2.80: Working with Operators

```
<html>
<head>
<title>JavaScriptExample</title>
```

```
</head>
<body bgcolor="pink">

<script type="text/javascript" language="javascript">
var math=95,english=80, science=90,avgMarks;
avgMarks=math+english+science/3;
document.write("Average Marks="+avgMarks);
avgMarks=(math+english+science)/3;

document.write("<br/>Average Marks="+avgMarks);
</script>
</body>
</html>
```

In Listing 2.80, the script within the `<body>` and `</body>` tags has four variables, namely, `math`, `english`, `science`, and `avgMarks`. These variables store the marks in Mathematics, English, and Science subjects and the average marks respectively. The `avgMarks` variable is first assigned the value of the `math+english+science/3` expression. Note that in this case, because of the higher precedence of the / operator, `science/3` is first evaluated, which is then added to the value `math+english`. However, in the fifth script statement, `avgMarks` is assigned the value of `(math+english+science)/3`. In this case, the `math+english+science` expression is evaluated first as it is enclosed within parentheses and then divided by `3`. When you open this page, the output appears, as shown in Figure 2.88:


JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\D
Google
File Edit View Favorites Tools Help
JavaScriptExample
Page
Do
Computer | Protected Mode: Off
100%

```
Average Marks=205
Average Marks=88.33333333333333
```

Figure 2.88: Using the Operators

In Figure 2.88, you can observe that the result of the first expression is 205 because the expression first performs the divide operation and then the addition operation. However, in the second expression, the expression first adds all the variables and then performs the division operation.

Working with Control Flow Statements

The scripts written in JavaScript are sequentially executed in a top-down manner, which means that the first statement in the script is the first to be executed and the last statement in the script is the last to be executed. This is the simplest and most straightforward way to execute the scripts. However, if you want to change the sequence in which the script statements are executed, then you can use the control flow statements.

As the name suggests, control flow statements are special statements that control or alter the flow of execution of script statements. You can make decisions about the execution of statements with the help of control flow statements. The decisions are based on the value of a condition, which is a JavaScript expression that evaluates to a Boolean value (`true` or `false`). When the condition evaluates to `true`, a particular group of statements is executed and when the condition evaluates to `false`, another group of statements is executed.

In JavaScript, the control flow statements can be classified as:

- **Selection statements**—Allows the execution of a group of statements from multiple groups of statements
- **Loops**—Allows a repeated execution of a group of statements

- **Jump Statements**—Allows the execution to skip or jump over certain statements

Let's learn about each of them in detail starting with selection statements.

Working with Selection Statements

Suppose your Web page has an HTML form to collect certain information, such as the name, phone number, and age of the users. When a user fills the form on the Web page, you may want to ensure that the user enters the information correctly. For instance, you want to ascertain that the user does not enter a number in place of name or enter a string or text for age. To incorporate such functionality and logic, you can use the selection statements in the script.

The selection statements use a condition to select or determine the statements that are to be executed. These statements assist you to make decisions and change the flow of execution of the statements. In JavaScript, there are three selection statements-`if`, `if...else`, and `switch`. Let's learn about them in detail one at a time starting with the `if` statement.

Using the if Statement

The `if` statement is one of the most basic and simplest control flow statement. You can use the `if` statement when you want to execute a group of one or more script statements only when a particular condition is met.

The syntax for the `if` statement is:

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

In the preceding syntax, `if` is a JavaScript keyword that signifies an `if` statement and `condition` refers to the condition that is evaluated. Note that the condition for the `if` statement is enclosed within parentheses immediately after the `if` keyword. If the condition evaluates to `true`, then the script statement, represented by `statement1`, enclosed within the curly braces are executed. If the condition evaluates to `false`, then the statement enclosed within the curly braces are skipped and the statement immediately after the closing curly brace (}) is executed.

NOTE

The curly braces in the if statement are used to group multiple script statements, known as a block of statements. The curly braces indicate that the block of statements is to be executed as a single statement.

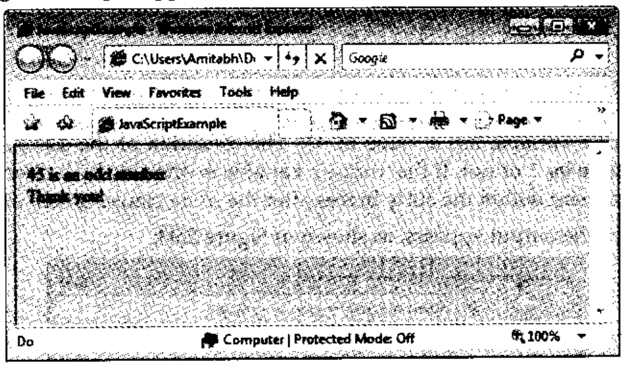
Let's create a Web page, named `ifStatement.html` to learn the working of the `if` statement. You can find the `ifStatement.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.81 shows the code of the `ifStatement.html` page:

Listing 2.81: Using the `if` Statement in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var Number=45;
if((Number%2) != 0)
{
document.write(Number + " is an odd number");
}
document.write("<br/>Thank you!");
</script>
</body>
</html>
```

In Listing 2.81, the script enclosed within `<body>` and `</body>` tags has a variable named `Number` that is assigned a value of `45`. The `if` statement in the script has a condition (the `Number%2==0` expression) that checks if the `Number` variable is divisible by `2` or not. In case, the remainder is not equal to zero, the statement within the curly braces of the `if` statement is executed. The execution then follows the next statement after the `if` statement. However, if the remainder is equal to zero, the statement within the curly braces is ignored and the statement immediately after the `if` statement is executed.

When you open this page, the output appears, as shown in Figure 2.89:



```
45 is an odd number
Thank you!
```

Figure 2.89: Using the if Statement

In Figure 2.89, you can observe that the if statement checks the expression and then returns the value 45 as an odd number.

Using the if...else Statement

As you know that the `if` statement allows you to execute a set of statements only when a particular condition is `true`. However, if you want to execute another set of statements when the condition is `false`, then you can use the `if...else` statement.

The syntax for the `if...else` statement is:

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

In the preceding syntax, `if` and `else` are keywords that signify the `if...else` statement. The condition of the `if...else` statement is also enclosed within parentheses. If the condition is `true`, then the group of statements, represented by `statement1`, enclosed within the first curly braces is executed. If the condition is `false`, then execution of `statement1` is skipped and is passed on to the `else` clause. The group of statements, represented by `statement2`, of the `else` clause is then executed.

Let's create a Web page, named `ifelseStatement.html` to learn the working of the `if...else` statement. You can find the `ifelseStatement.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.82 shows the code of the `ifelseStatement.html` page:

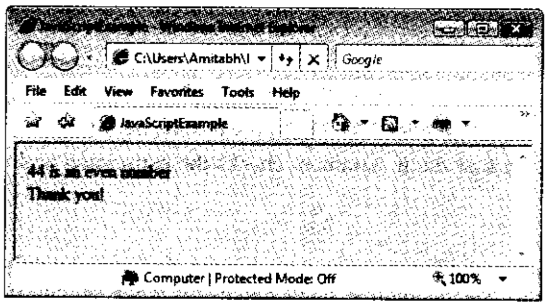
Listing 2.82: Using the `if...else` Statement in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
```

```
var Number=44;
if((Number%2) != 0)
{
document.write(Number + " is an odd number");
}
else
{
document.write(Number + " is an even number");
}
document.write("<br/>Thank you!");
</script>
</body>
</html>
```

In Listing 2.82, the `Number` variable has a value of `44`. The condition of the `if...else` statement checks if the `Number` variable is divisible by `2` or not. If the `Number` variable is divisible by `2`, then the execution passes on to the `else` clause. The statement within the curly braces after the `else` clause is then executed.

When you open this page, the output appears, as shown in Figure 2.90:


C:\Users\Amitabh\I
Google
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
44 is an even number
Thank you!
```

Figure 2.90: Using the if...else Statement

In Figure 2.90, you can observe that the if...else statement checks the expression to find out whether the value 44 is even or odd.

In JavaScript, you can nest one `if...else` statement within another. Nested `if...else` statements are useful when you want additional checking or validation on the Web page. You can nest as many `if...else` statements within an `if...else` statement. In most cases, an `if...else` statement is nested inside the `else` clause of another `if...else` statement.

Let's create another Web page, named `NestedifelseStatement.html` to learn how to nest one `if...else` statement within another. You can find the `NestedifelseStatement.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.83 shows the code of the `NestedifelseStatement.html` page:

Listing 2.83: Nesting `if...else` Statements

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var letter="I";
if (letter=="A")
  document.write("vowel A");
  else
      if(letter=="E")
          document.write("vowel E");
      else
          if(letter=="I")
              document.write("vowel I");
```

```
                else
                    if(letter=="O")
                        document.write("vowel O");
                    else
                        if(letter=="U")
                            document.write("vowel U");
                        else
                            document.write("consonant");
        document.write("<br/>Thank you!");
    </script>
    </body>
    </html>
```

In Listing 2.83, the `letter` variable in the script is assigned a value of `I`. The first `if...else` statement has a condition (`letter=="A"`) that checks whether `letter` is equal to `A`. This condition is `false` resulting in executing the first `else` clause, which has an `if...else` statement nested inside it. The second `if...else` statement has a condition that checks if `letter` is equal to `E` or not. This condition is also `false` resulting in passing the execution to the `else` clause of the second `if...else` statement. The second `else` clause also has an `if...else` statement that checks whether `letter` is equal to `I`. This condition is `true`, as a result of which the `vowel I` message is displayed. When you open this page, the output appears, as shown in Figure 2.91:

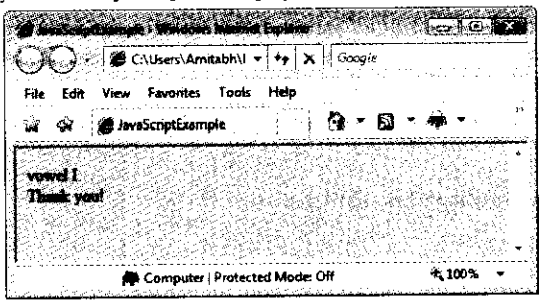

```
vowel I
Thank you!
```

Figure 2.91: Using the Nested if...else Statement

In Figure 2.91, you can observe that when you check the value of the letter variable to ensure whether the value is a consonant or vowel, the value I passes through the different if...else blocks and returns that the value is vowel. With the nested `if...else` statements, you can create a series of yes-no questions to make decisions. Another alternative to the nested `if...else` statements is the `switch` statement, which is discussed next.

Using the switch Statement

The `switch` statement provides a simpler alternative to the nested `if...else` statements. You can use the `switch` statement to select a particular group of statements to be executed among several other groups of statements. The group of statements that is to be executed is selected on the basis of a numeric or string expression.

The syntax of the `switch` statement is:

```
switch(expression)
{
case  value1:
statement1
break;
case value2:
statement2
break;
case value3:
statement3
break;
default:
statement_default
break;
}
```

In the preceding syntax, `switch`, `case`, and `break` are keywords in JavaScript. The `switch` keyword indicates the `switch` statement. The expression that is to be evaluated is specified within parentheses. This expression is checked against each of the case values specified in the case statements (indicated by the `case` keyword). If any of the case value matches with the value of the expression, the group of statements (`statement1`, `statement2`, or `statement3`) specified by the respective case statement is executed. If none of the case values matches with the value of the expression, then the default statement, specified by the `default` keyword, is executed. The default statement is generally placed at the very end of the `switch` block; however, you can place it anywhere within the `switch` block.

NOTE

We have shown the preceding syntax with only three case statements. However, you can have as many case statements as you want but only one default statement.

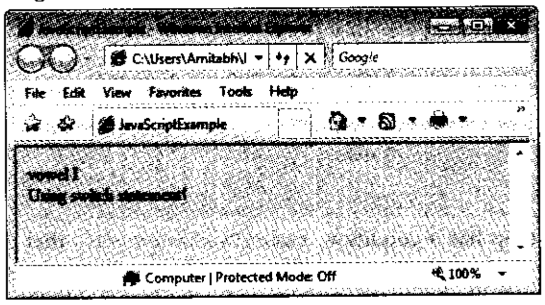
Every group of statements specified by the case statements has a `break` statement that is indicated by the `break` keyword. The `break` statement prevents the execution to pass on to the next case or default statement after a match is found.

Let's create a Web page, named `switchStatement.html` to learn the working of the `switch` statement. You can find the `switchStatement.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.84 shows the code of the `switchStatement.html` page:

Listing 2.84: Using the `switch` Statement in a Script

```
<html>
  <head>
      <title>JavaScriptExample</title>
  </head>
  <body bgcolor="pink">
      <script type="text/javascript" language="javascript">
          var letter="I";
          switch(letter)
          {
          default:
              document.write("consonant");
              break;
          case "A":
              document.write("vowel A");
              break;
          case "E":
              document.write("vowel E");
              break;
          case "I":
              document.write("vowel I");
              break;
          case "O":
              document.write("vowel O");
              break;
          case "U":
              document.write("vowel U");
              break;
          }
          document.write("<br/>Using switch statement!");
      </script>
  </body>
</html>
```

In Listing 2.84, the `letter` variable is the expression for the `switch` statement. The value of the `letter` variable is compared with the case values one by one. When a match is found, which in this case in the third case statement, the set of statements specified by that case statement is executed. When you open this page, the output appears, as shown in Figure 2.92:


C:\Users\Amitabh\I
Google
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
vowel I
Using switch statement
```

Figure 2.92: Using the switch Statement

In Figure 2.92, you can observe that when the value I of the letter variable is passed in the switch statement, the statement checks the value through different cases and finally returns I as vowel.

Working with Loops

Loops or iteration statements are control flow statements that allow you to execute a particular group of statements repeatedly. The number of times the group of statements is executed depends on a particular condition that is a Boolean expression. If the condition is `true`, then the group of statements is executed. The condition is again checked and if it is `true`, the group of statements is again executed. A single execution of a group of statements is known as iteration. In this way, the loop keeps on executing until the condition becomes `false`. When the condition becomes `false`, the execution of the loop halts, the current group of statements is skipped, and the statement immediately following the loop is executed.

In JavaScript, you can use any of the three loops—`while`, do...`while`, and `for` loop. The condition is placed either at the start or at the end of the loop depending on the type of loop.

Let's learn about the three loops one by one starting with the `while` loop.

Using the while Loop

You can use the `while` loop when you want to check the condition at the start of the loop. The group of statements that is to be executed is specified after the condition. This implies that if the condition is `false` in the first iteration itself, the group of statements in the `while` loop is never executed. On the other hand, the group of statements keeps on executing until the condition becomes `false`. Therefore, in a `while` loop, there can be zero or more iterations.

The syntax for the `while` loop is:

```
while(condition)
{
statement1
}
```

In the preceding syntax, `while` is a JavaScript keyword. The `condition` for the `while` loop is specified in parentheses and the group of statements (`statement1`) is specified within the curly braces.

Let's create a Web page, named `whileLoop.html` to understand the working of the `while` loop. You can find the `whileLoop.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.85 shows the code of the `whileLoop.html` page:

Listing 2.85: Using the `while` Loop in a Script

```
<html>
<head>
      <title>JavaScriptExample</title>
  </head>
```

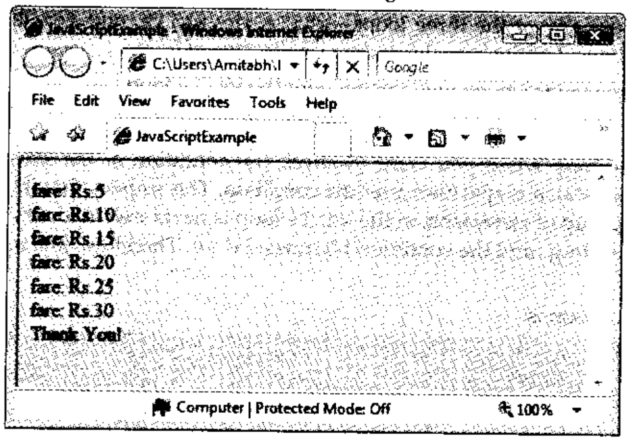
```
    <body bgcolor="pink">
        <script type="text/javascript" language="javascript">
            var totalDistance=0, fare=0;
            while(totalDistance<=10)
            {
                fare=fare+5;
                totalDistance=totalDistance+2;
                document.write("fare: Rs."+fare+"<br/>");
            }
            document.write("Thank You!");
        </script>
    </body>
</html>
```

In Listing 2.85, the `while` loop has a condition, `totalDistance<=10`, that checks whether the value of the variable `totalDistance` is less than or equal to `10`. In the first iteration, the condition is `true` as `totalDistance` is `0`. As a result, the group of statements specified inside the `while` loop is executed, that is, the `fare` variable is incremented by `5` and the value of `totalDistance` is incremented by `2`. The condition of the `while` loop is evaluated again in the second iteration, which is also `true`, as a result of which the values of `fare` and `totalDistance` are incremented again. Note that in this way, the `while` loop executes six times. At the end of the sixth iteration, the `totalDistance` variable is equal to `12` and when the condition is evaluated in the seventh iteration, it becomes `false`. As a result, the `while` loop is exited and the `Thank You!` message is displayed.

NOTE

If the condition of a while loop is always true, then the loop keeps executing indefinitely. Such loops are known as infinite loops as they have an infinite number of iterations.

When you open this page, the output appears, as shown in Figure 2.93:


JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
fare: Rs.5
fare: Rs.10
fare: Rs.15
fare: Rs.20
fare: Rs.25
fare: Rs.30
Thank You!
```

Figure 2.93: Using the while Loop

In Figure 2.93, you can observe that the value of fare is printed for six times as the while loop is also executed for the six times.

Using the do while Loop

You learned that in the `while` loop, if the condition becomes `false` in the very first iteration, the group of statements specified in the `while` loop is never executed. However, if you want the group of statements to execute at least once, then you can use the `do...while` loop. This is possible because the condition of the `do...while` loop is placed after the group of statements at the end of the loop.

The syntax of the `do...while` loop is:

```
do
{
statement1
}
while (condition);
```

In the preceding syntax, do and `while` are JavaScript keywords. The group of statements, represented by `statement1`, is enclosed within curly braces. The `condition` is enclosed within parentheses immediately after the `while` keyword. Note that there is semicolon after the closing parenthesis.

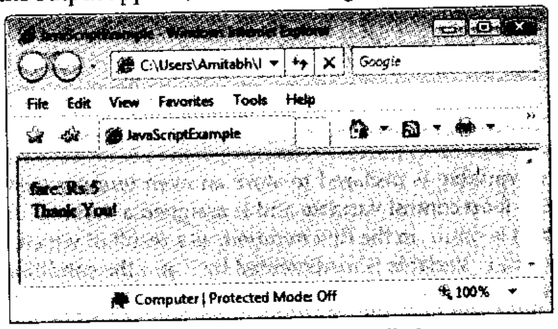
Let's create a Web page, named `dowhileLoop.html` to understand the working of the do...`while` loop. You can find the `dowhileLoop.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.86 shows the code of the `dowhileLoop.html` page:

Listing 2.86: Using the do...`while` Loop in a Script

```
<html>
<head>
      <title>JavaScriptExample</title>
  </head>
  <body bgcolor="pink">
      <script type="text/javascript" language="javascript">
            var totalDistance=10, fare=0;
            do
            {
                  fare=fare+5;
                  totalDistance=totalDistance+2;
                  document.write("fare: Rs."+fare+"<br/>");
            }
            while(totalDistance<=10);
            document.write("Thank You!");
      </script>
  </body>
</html>
```

In Listing 2.86, after the `totalDistance` and `fare` variables are assigned values, the group of statements inside the do...`while` loop is executed. After this, the condition is evaluated. Note that in the first iteration itself, the condition is `false` because after the group of statements is executed, the `totalDistance` variable has a value of `12`. As a result, the do...`while` loop is exited and the `Thank You! message` is displayed.

When you open this page, the output appears, as shown in Figure 2.94:


C:\Users\Amitabh\I
Google
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
fare: Rs.5
Thank You!
```

Figure 2.94: Using the do while Loop

As you can see in Figure 2.94, the do...`while` loop executes the group of statements at least once irrespective of whether the condition is `true` or `false`.

Using the for Loop

The `for` loop is one of the most common loops in JavaScript and other programming languages. The `for` loop allows you to execute a block of statements for a predetermined number of times, that is, the number of iterations of a `for` loop is known beforehand. The condition of the `for` loop is placed at the beginning of the loop.

The syntax of the `for` loop is:

```
for(initialization_statement; condition; updation_statement)
{
  statement1
}
```

In the preceding syntax, `for` is a JavaScript keyword that is immediately followed by a pair of parentheses that encloses the `initialization_statement`, `condition`, and `updation_statement`, each of which is separated by a semicolon. The `initialization_statement` refers to the statement in which the loop control variable is declared or assigned an initial value. A loop control variable, as the name suggests, is a variable that is used in `condition` and `updation_statement` to control the execution of the `for` loop. The `updation_statement` refers to the statement that updates the value of the loop control variable.

Note that in the first iteration of the `for` loop, `initialization_statement` is the first statement to be executed followed by the evaluation of `condition`, which if `true` results in the execution of the group of statements (`statement1`) inside the `for` loop. After the group of statements is executed, the `updation_statement` is executed completing the first iteration. The condition of the `for` loop is again evaluated for the second iteration and so on.

Let's create a Web page, named `forLoop.html` to understand the working of the `for` loop. You can find the `forLoop.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.87 shows the code of the `forLoop.html` page:

Listing 2.87: Using the `for` Loop in a Script

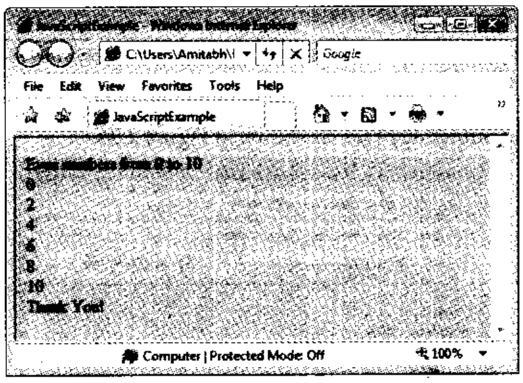
```
<html>

<head>
      <title>JavaScriptExample</title>
  </head>
  <body bgcolor="pink">
      <script type="text/javascript" language="javascript">
         var Number;
         document.write("Even numbers from 0 to 10 <br/>");
         for(Number=0;Number<=10;Number=Number+2)
         {
             document.write(Number+"<br/>");
         }
         document.write("Thank You!");
      </script>
  </body>

</html>
```

In Listing 2.87, the `Number` variable is declared to store an even number from `0` to `10`. In the `for` loop, the `Number` variable is used as a loop control variable and is assigned a value of `0`. The condition, `Number<=10`, is then evaluated. This condition is `true` in the first iteration, as a result of which, value of the `Number` variable is displayed. After this, the `Number` variable is incremented by `2` and the condition is again evaluated. In this way, the `for` loop iterates six times until the value of `Number` becomes `12`.

NOTE

The initialization statement, condition, and updation statement are optional and can be omitted in the for loop. You can omit the condition or updation statement if you want an infinite for loop.

When you open this page, the output appears, as shown in Figure 2.95:


JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
Even numbers from 0 to 10
0
2
4
6
8
10
Thank You!
```

Figure 2.95: Using the for Loop

In Figure 2.95, you can observe that the for loop is executed until the variable Number is less than or equal to 10.

Working with Jump Statements

As the name suggests, jump statements allow you to jump over or skip certain statements in a script and execute some other statement. You can use jump statements to exit or break a loop. In JavaScript, there are two jump statements—`break` and `continue`. Let's learn about the `break` statement first.

Using the break Statement

As you know, the `switch` statement uses the `break` statement to prevent the execution of the subsequent case statements. The `break` statement also allows you to break or exit a loop. When used inside a loop, the `break` statement stops executing the loop and causes the loop to be immediately exited. In case, the loop has statements after the `break` statement, then those remaining statements are skipped.

Let's create a Web page, named `breakStatement.html` to understand the working of the `break` statement. You can find the `breakStatement.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.88 shows the code of the `breakStatement.html` page:

Listing 2.88: Using the `break` Statement in a Script

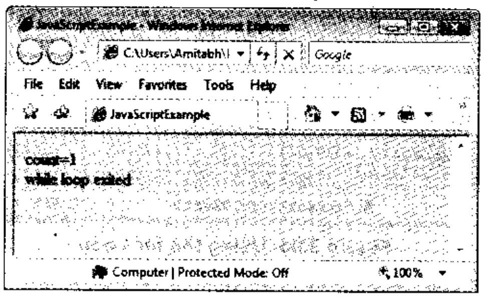
```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var count=0;
while(count<10)
{
++count;
if((count%2==0))
break;
else
document.write("count="+count+"<br/>");
}
document.write("while loop exited");
</script>
</body>
</html>
```

In Listing 2.88, if the value of the `count` variable is less than `10`, the `while` loop is executed. The `if…else` statement specified inside the `while` loop checks if the `count` variable is divisible by `2` or not. If `count` is divisible by `2`, then the `break` statement is executed. As a result, the `while` loop is immediately exited and the statement immediately following the `while` loop is executed.

NOTE

If break statement is used in a nested loop, then only the loop which contains the break statement is exited and not the loop which encloses the nested loop.

When you open this page, the output appears, as shown in Figure 2.96:


JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
count=1
while loop exited
```

Figure 2.96: Using the break Statement

In figure 2.96, you can observe that when the variable count is divided by 2 and the remainder is 0, the while loop is exited.

Using the continue Statement

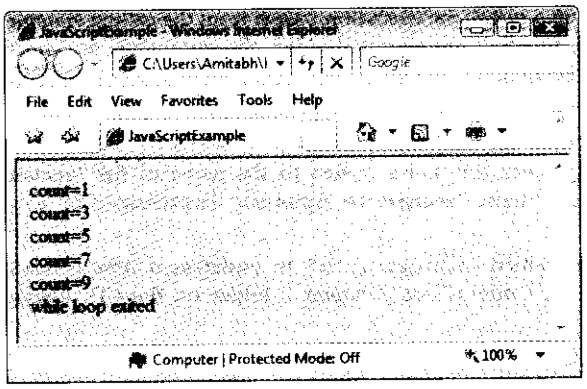
Similar to the `break` statement, the `continue` statement can also be used to stop the execution of a loop. However, the `continue` statement does not exit the loop; it causes the evaluation of the condition for the next iteration of the loop. If the loop has any statements after the `continue` statement, then those statements are not executed.

Let's create a Web page, named `continueStatement.html` to understand the working of the `continue` statement. You can find the `continueStatement.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.89 shows the code of the `continueStatement.html` page:

Listing 2.89: Using the `continue` Statement in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var count=0;
while(count<10)
{
++count;
if(count%2==0)
  continue;
else
  document.write("count="+count+"<br/>");
}
document.write("while loop exited");
</script>
</body>
</html>
```

In Listing 2.89, the `if…else` statement inside the `while` loop has a `continue` statement. The condition of the `if…else` statement checks whether the `count` variable is divisible by `2 or not`. If `count` is divisible by `2`, then the `continue` statement is executed. As a result, execution of the loop is halted, the remaining statements are skipped, and the condition of the `while` loop is evaluated for the next iteration. Note that in this case, the `continue` statement is executed for every even number between `0` and `10`. When you open this page, the output appears, as shown in Figure 2.97:



JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
count=1
count=3
count=5
count=7
count=9
while loop exited
```

Figure 2.97: Using the continue Statement

As you can observe in Figure 2.97, for every even number between `0` and `10`, the value of `count` is not displayed.

Working with Functions

Consider a situation where you need to calculate the average of three given numbers at different places in the script. In this case, if you write the statements or code to find the average at each of those places in the script, the script becomes rather long. This in turn may affect the readability and efficiency of the script. So in all such situations, you can make use of functions. A function refers to a cohesive block of statements that has a name and can be accessed or called from anywhere and any number of times in the script.

In JavaScript, you can quickly create your own function in the script to perform an action or task. You can access or call a function in the script as many times as you want. Every time that you call a function, the specified action or task is performed. The statement in which the function is called is known as the calling statement.

In most cases, the calling statement and the function exchange information, that is, the calling statement sends some information that the function uses or the function returns some information that the calling statement uses. You can exchange information between the calling statement and the function by using parameters, which are variables inside the function. You need to declare the parameters of a function while creating the function.

Let's first learn how to create a function in JavaScript. The syntax to create a function is:

```
function function_name (parameter1, parameter2, . . . parameterN)
{
  statement1
}
```

In the preceding syntax, `function` is a JavaScript keyword that indicates the start of a function in the script. The `function_name` refers to the name of the function, which is immediately followed by a pair of parentheses. The parentheses enclose the parameters (`parameter1`, `parameter2`, and so on) of the function. A function can have zero or more number of parameters. Note that even if you do not specify any parameters of the function, you need to provide the parentheses. When the function is called in the script, the statements (represented by `statement1`) enclosed in the curly braces of the function are executed.

NOTE

While providing a name for a function and the parameters, you should follow the same rules as specified for naming variable. The parameters as well as any variables declared inside a function are local variables, that is, they can be accessed and used only inside that function.

After creating a function, you can access or use it by calling it in the script. You can call a function by simply using its name and providing arguments for the parameters of the function. An argument is the value that is passed in the function call and is assigned to the parameters. It is essential that the number and order of arguments provided in the function call have to be same as the number of parameters in the function.

When you call a function in a script, the execution passes from the calling statement to the function. As a result, the statements inside the function are executed. After the statements are executed, the control of execution passes back to the calling statement.

The syntax of a function call is:

```
function_name(argument1, argument2, . . . , argumentN);
```

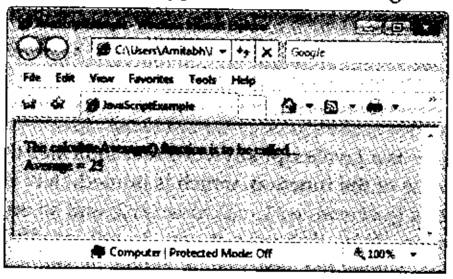
In the preceding syntax, `function_name` refers to the name of the function that you want to call. The parentheses after `function_name` enclose the arguments (represented by `argument1`, `argument2`, and so on).

Let's create a Web page, named `function.html` to understand how a function works. You can find the `function.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.90 shows the code of the `function.html` page:

Listing 2.90: Creating and Calling a Function in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
function calculateAverage()
{
  var a=15, b=25; c=35;
  var average= (a+b+c)/3;
  document.write("Average = "+average);
}
document.write("The calculateAverage() function is to be called...<br/>");
calculateAverage();
</script>
</body>
</html>
```

In Listing 2.90, the `calculateAverage()` function is called by specifying its name and an empty pair of parentheses. Note that you need to provide the parentheses even if there are no arguments to be passed to the function. When you open this page, the output appears, as shown in Figure 2.98:


C:\Users\Amitabh\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
The calculateAverage() function is to be called...
Average = 25
```

Figure 2.98: Creating and Calling a Function

In Figure 2.98, you can observe that the calculateAverage() function is called to return the average of variables a, b, and c.

JavaScript functions can also return some information or value to the calling statement. You can use a variable in the calling statement to store the value returned by the function. To return a value to the calling statement, you need to use the `return` statement.

The `return` statement allows you to return a single value to the calling statement. Though the `return` statement is specified at the end of the function just before the closing curly brace (}), you can specify the `return` statement anywhere in the function. When the `return` statement is executed, any statements after the `return` statement are ignored and the control of execution passes back to the calling statement.

The syntax of the `return` statement is:

```
return value;
```

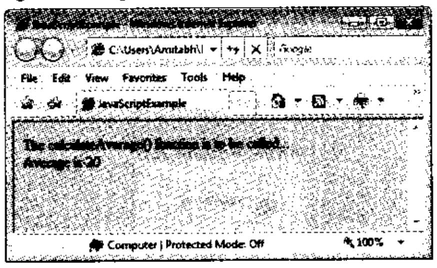
In the preceding syntax, `return` is a JavaScript keyword and `value` refers to the value returned to the calling statement. You can also return the value of a variable by providing the name of a variable in place of `value`.

Let's create a Web page, named `returnFunction.html` to understand how a function returns a value. You can find the `returnFunction.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.91 shows the code of the `returnFunction.html` page:

Listing 2.91: Returning a Value from a Function

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
function calculateAverage(a, b, c)
{
  var average= (a+b+c)/3;
  return average;
}
document.write("The calculateAverage() function is to be called...<br/>");
var avg=calculateAverage(10,20,30);
document.write("Average is "+avg);
</script>
</body>
</html>
```

In Listing 2.91, the `calculateAverage()` function has three parameters that are used inside the function to find their average, which is stored in the `average` variable in the function. When the `calculateAverage()` function is called, three arguments are passed to it. These arguments are assigned to the three parameters in the same order. The value of the `average` variable is then evaluated and returned to the calling statement through the `return` statement. As a result, value of the `average` variable is assigned to the `avg` variable in the calling statement. When you open this page, the output appears, as shown in Figure 2.99:


C:\Users\Amitabh\
Google
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
The calculateAverage() function is to be called...
Average is 20
```

Figure 2.99: Returning a Value from a Function

There are a number of in-built functions included in JavaScript to work with the JavaScript applications. Some of the most frequently used functions are as follows:

- alert()
- prompt()
- confirm()
- eval()
- isFinite()
- isNaN()
- parseInt()
- parseFloat()

- Number()
- String()
- escape()
- unescape()

Let's discuss these functions one by one with their uses.

Using the alert () Function

The `alert()` function is used to display information in a message box. This function is generally used to display the results of your JavaScript processing where you do not want to update the Web page itself. You can use this function to display error messages once you validate a form.

The `alert()` function is also useful when you test JavaScript. You can use it to display the value held in a variable at that stage in the processing.

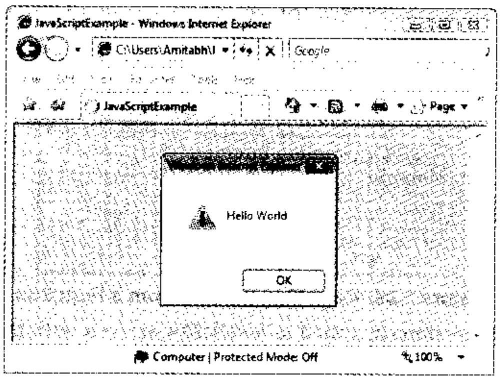
Let's create a Web page, named `alertFunction.html` to understand how to use the `alert()` function in a script. You can find the `alertFunction.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.92 shows the code of the `alertFunction.html` page:

Listing 2.92: Using the alert() Function in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
alert( "Hello World" );
</script>
</body>
</html>
```

In Listing 2.92, you can observe that the `alert()` function displays a message box containing the text `Hello World`.

When you open this page, the output appears, as shown in Figure 2.100:



Hello World

JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
Google
JavaScriptExample
Page
OK
Computer | Protected Mode: Off
100%

Figure 2.100: Using the alert() Function

As you can observe in Figure 2.100, the alert message box contains the `OK` button. You can click this button to continue running the JavaScript code.

Using the confirm() Function

The `confirm()` function is the advanced form of the `alert()` function. It is used to display a message as well as to return a true or false value. The `confirm()` function displays a dialog box with two buttons: `OK` and `Cancel` at the bottom of the message box. When you click the `OK` button, the function returns `true`. When you click the `Cancel` button, the function returns `false`. This allows you to interrupt your JavaScript processing to ask the user a question and then continue processing based on which button is clicked.

Let's create a Web page, named `confirmFunction.html` to understand how to use the `confirm()` function in a script. You can find the `confirmFunction.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.93 shows the code of the `confirmFunction.html` page:

Listing 2.93: Using the confirm() Function in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var qry= confirm ( "Are you sure to send this file to Recycle Bin ? " );
if (qry)
document.write ("The file is deleted");
else
document.write ("Welcome to JavaScript");
</script>
</body>
</html>
```

In Listing 2.93, you can observe that the `var` variable stores the value of the query based on whether you click the `OK` button or the `Cancel` button.

When you open this page, the output appears, as shown in Figure 2.101:

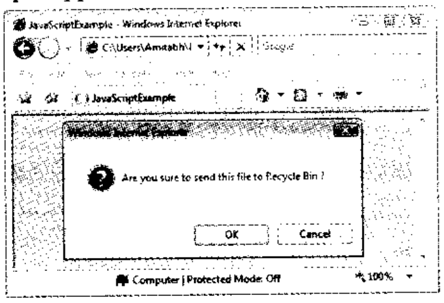

Are you sure to send this file to Recycle Bin ?
JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
JavaScriptExample
Windows Internet Explorer
OK
Cancel
Computer | Protected Mode: Off
100%

Figure 2.101: Using the confirm() Function

In figure 2.101, you can observe that as the page is loaded, the confirm dialog box appears.

When you click the `OK` button, the corresponding message appears, as shown in Figure 2.102:

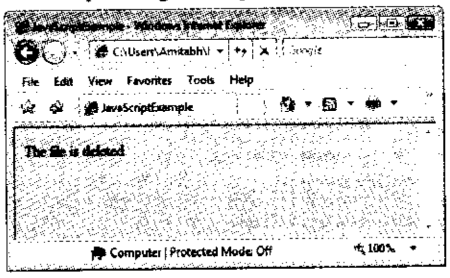

The file is deleted
JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

Figure 2.102: Displaying a Message After Clicking the OK Button

Let's now learn about the usage of the prompt() function, in the next section.

Using the prompt() Function

The `prompt()` function is the most advanced among the `alert()`, `confirm()`, and `prompt()` functions because it accepts two parameters instead of one. The `prompt()` function displays a message box containing a text box with OK and Cancel buttons. It returns a text string when OK is clicked and null when Cancel is clicked.

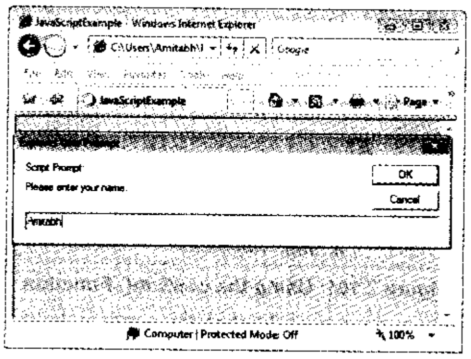
Let's create a Web page, named promptFunction.html to understand how to use the prompt() function in a script. You can find the promptFunction.html file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.94 shows the code of the promptFunction.html page:

Listing 2.94: Using the prompt() Function in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
var name= prompt( "Please enter your name: " );
if (name==null || name=="") name = " visitor "; {
document.write ("Hi " + name + ", Welcome to JavaScript");
}
</script>
</body>
</html>
```

In Listing 2.94, you can observe that the name variable contains the value that you enter in the prompt message box. When you click the OK button, the processing continues and the entered name is then displayed with a message in the window.

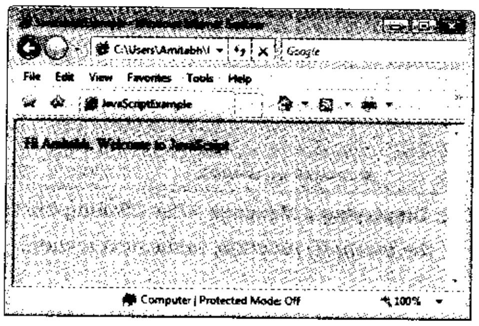
When you open this page, the output appears after entering your name in the text field of the input box, as shown in Figure 2.103:


JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\I
Google
JavaScriptExample
Page
OK
Cancel
Computer | Protected Mode: Off
100%

Script Prompt:
Please enter your name.
Amitabh

Figure 2.103: Using the prompt() Function

In Figure 2.103, you can enter your name in the text box and click the OK button, the message appears, as shown in Figure 2.104:



Hi Amitabh, Welcome to JavaScript

Figure 2.104: Displaying a Message after Clicking the OK Button

Let's now learn about the eval() function in the next section.

Using the eval() Function

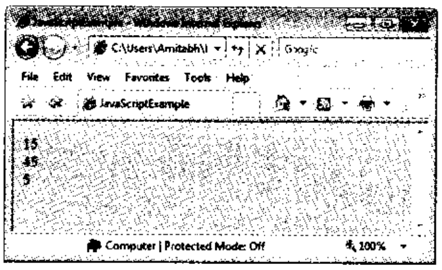
The `eval()` function is generally used to introduce variable values into a string. It accepts a string containing JavaScript code that it evaluates as an argument and then returns the resulting value.

Let's create a Web page, named `evalFunction.html` to understand how to use the `eval()` function in a script. You can find the `evalFunction.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.95 shows the code of the `evalFunction.html` page:

Listing 2.95: Using the eval() Function in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
eval ("x=25; y=10; document.write (x-y)");
document.write ("<br>");
document.write (eval ("9*5"));
document.write ("<br>");
var a = 15;
document.write (eval (a/3));
</script>
</body>
</html>
```

In Listing 2.95, you can observe how the `eval()` function evaluates the strings in different ways. When you open this page, the output appears, as shown in Figure 2.105:



```
15
45
5
```

Figure 2.105: Using the eval() Function

In Figure 2.105, you can observe that the eval() function is evaluated in three different ways and returns the output depending on the value passed to it.

Using the isFinite() Function

The `isFinite()` function checks whether a value is finite or not. The `isFinite()` function returns a Boolean value: true or false indicating whether the argument passed to it is finite or infinite. The `isFinite()` function always returns true, unless it considers an argument to be equal to infinity.

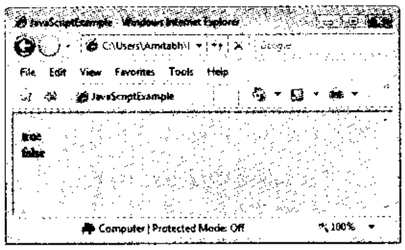
Let's create a Web page, named `isFiniteFunction.html` to understand how to use the `isFinite()` function in a script. You can find the `isFiniteFunction.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.96 shows the code of the `isFiniteFunction.html` page:

Listing 2.96: Using the isFinite() Function in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
```

```
<script type="text/javascript" language="javascript">
document.write (isFinite (998));
document.write ("<br>");
document.write (isFinite("Hello World"));
</script>
</body>
</html>
```

In Listing 2.96, you can observe how the `isFinite()` function checks an integer and a string value to ensure whether the value is finite or infinite. When you open this page, the output appears, as shown in Figure 2.106:


JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
true
false
```

Figure 2.106: Using the isFinite() Function

In Figure 2.106, you can observe that the Boolean value is true for 998 and false for Hello World.

Using the isNaN() Function

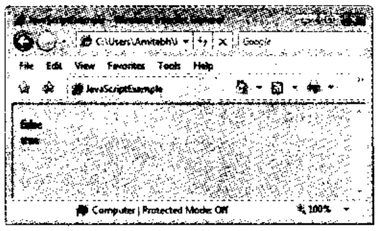
The `isNaN()` function checks whether a value is a number or not. The NaN stands for Not a Number, which only checks for the numerical nature of a value, not whether the value is finite or infinite. Based on the result, the `isFinite()` function returns a Boolean value: true or false indicating whether the argument passed to it is a number or string.

Let's create a Web page, named `isNaNFunction.html` to understand how to use the `isNaN()` function in a script. You can find the `isNaNFunction.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.97 shows the code of the `isNaNFunction.html` page:

Listing 2.97: Using the isNaN() Function in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
document.write (isNaN (998));
document.write ("<br>");
document.write (isNaN("Hello World"));
</script>
</body>
</html>
```

In Listing 2.97, you can observe how the `isNaN()` function checks an integer and a string value to ensure whether the value is a number. When you open this page, the output appears, as shown in Figure 2.107:


C:\Users\Amitabh\
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

```
false
true
```

Figure 2.107: Using the isNaN() Function

In Figure 2.107, you can observe that because the 998 is a number, the isNaN() function returns false and Hello World is a string.

Using the parseInt() and parseFloat() Functions

You can use the parseInt() and parseFloat() functions to extract a number from the beginning of a string. The parseInt() function parses the string and returns the first integer value that is found in the string. Similarly, the parseFloat() function parses the string and returns the first floating-point value found in the string. However, if the parser finds any non-numeric value before returning the value, then a special value NaN (Not a Number) is returned.

When you parse a string, JavaScript uses the radix parameter to specify which numeral system is used. For example, a radix of 8 (octal) shows that the number in the string should be parsed from an octal number to a decimal number. If the radix parameter is excluded, JavaScript assumes the following:

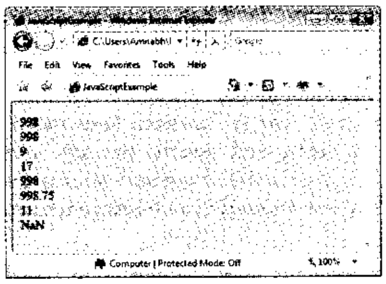
- If 0x precedes the string, the radix is 16 (hexadecimal)
- If 0 precedes the string, the radix is 8 (octal)
- If any other value precedes the string, the radix is 10 (decimal)

Let's create a Web page, named parseInt&parseFloatFunction.html to understand how to use the parseInt() and parseFloat() functions in a script. You can find the parseInt&parseFloatFunction.html file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.98 shows the code of the parseInt&parseFloatFunction.html page:

Listing 2.98: Using the parseInt() and parseFloat() Functions in a Script

```
<html>

<head>

<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
document.write (parseInt ("998"));
document.write ("<br>");
document.write (parseInt ("998.75"));
document.write ("<br>");
document.write (parseInt ("011"));
document.write ("<br>");
document.write (parseInt ("0x11"));
document.write ("<br>");
document.write (parseFloat ("998"));
document.write ("<br>");
document.write (parseFloat ("998.75"));
document.write ("<br>");
document.write (parseFloat ("011"));
document.write ("<br>");
document.write (parseFloat ("He is 25"));
document.write ("<br>");
</script>

</body>

</html>
```

In Listing 2.98, you can observe how the parseInt() and parseFloat() functions check an integer and a string value to ensure whether the value is a number. When you open this page, the output appears, as shown in Figure 2.108:



```
998
998
9
17
998
998.75
11
NaN
```

Figure 2.108: Using the parseInt() and parseFloat() Functions

In Figure 2.108, you can observe that the parseInt() function returns only the integer values whereas the parseFloat() function returns the floating point values with fractional parts.

Using the Number() Function

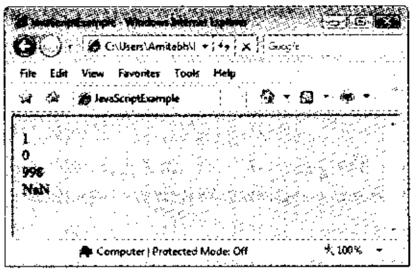
The `Number()` function is used to convert the value of an object into a number. If the value is Boolean, the `Number()` function returns the value `1` for true and the value `0` for false. Similarly, in case of the `Date` object, the `Number()` function returns the number of milliseconds since midnight January 1, 1970 UTC (Universal Time Coordinated). Moreover, if the value of the object cannot be converted to a number, the `Number()` function returns NaN.

Let's create a Web page, named `NumberFunction.html` to understand how to use the `Number()` function in a script. You can find the `NumberFunction.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.99 shows the code of the `NumberFunction.html` page:

Listing 2.99: Using the Number() Function in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
document.write (Number (Boolean (true)));
document.write ("<br>");
document.write (Number (Boolean (false)));
document.write ("<br>");
document.write (Number (String ("998")));
document.write ("<br>");
document.write (Number (String ("998 899")));
document.write ("<br>");
</script>
</body>
</html>
```

In Listing 2.99, you can observe that the `Number()` function converts the different object values to a number. When you open this page, the output appears, as shown in Figure 2.109:



```
1
0
998
NaN
```

Figure 2.109: Using the Number() Function

In Figure 2.109, you can observe that the Number() function changes the value of true to 1, false to 0. However, the value of 998 is not changed because it is already a number and the value of 998 899 is NaN because the value contains a space.

Using the escape() and unescape() Functions

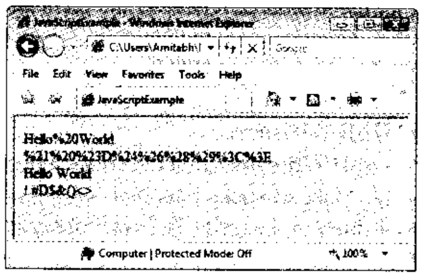
The `escape()` function is used to encode a string so that the string can be read on all computers. The `escape()` function encodes special characters except *, @, -, _, +, ., and /. On the other hand, you can use the `unescape()` function to decode the strings that are encoded with the `escape()` function. You cannot use the `escape()` and `unescape()` functions to encode or decode Uniform Resource Identifiers (URIs).

Let's create a Web page, named `escape&unescapeFunction.html` to understand how to use the `escape()` and `unescape()` functions in a script. You can find the `escape&unescapeFunction.html` file in the Code\HTML\Chapter 2 folder on the CD. Listing 2.100 shows the code of the `escape&unescapeFunction.html` page:

Listing 2.100: Using the escape() and unescape() Functions in a Script

```
<html>
<head>
<title>JavaScriptExample</title>
</head>
<body bgcolor="pink">
<script type="text/javascript" language="javascript">
document.write (escape ("Hello World"));
document.write ("<br>");
document.write (escape ("! #D$&()<>"));
document.write ("<br>");document.write (unescape ("Hello%20World"));
document.write ("<br>");
document.write (unescape ("%21%20%23D%24%26%28%29%3C%3E"));
document.write ("<br>");
</script>
</body>
</html>
```

In Listing 2.100, you can observe that the `escape()` function encodes the different special characters. Alternatively, the `unescape()` function decodes the same encoded string to the original string. When you open this page, the output appears, as shown in Figure 2.110:



```
Hello%20World
%21%20%23D%24%26%28%29%3C%3E
Hello World
! #D$&()<>
```

JavaScriptExample - Windows Internet Explorer
C:\Users\Amitabh\
File Edit View Favorites Tools Help
JavaScriptExample
Computer | Protected Mode: Off
100%

Figure 2.110: Using the escape() and unescape() Functions

In Figure 2.110, you can observe that the escape() and unescape() functions encode and decode the specified strings.

With this we come to the end of the chapter. Let's now summarize the main points of the chapter.

Summary

In this chapter, you learned about the structure of an HTML document, different HTML tags, HTML forms and form controls, and cascading style sheets. You have also learned about embedding JavaScript in an HTML page, variables, arrays, objects, operators, control and looping statements in JavaScript. Finally, you learned about some common events and functions used in a JavaScript application.

In the next chapter, we discuss about Personal Home Pages (PHP).

Quick Revise

Q1. ________tag is used to create a paragraph in HTML.

Ans: <p> tag

Q2. CSS stands for ________________.

Ans: Cascading Style Sheets

Q3. You can create both the vertical and horizontal frames in a Web page at the same time. (True/False)

Ans: True

Q4. The name of a variable can begin with a number. (True/False)

Ans: False

Q5. What is the use of conditional operator?

Ans: The conditional operator returns the second operand if the first operand is `true`, otherwise, returns the third operand

Q6. What is cascading style sheets? Name the different ways to apply a CSS to an HTML document.

Ans: Cascading Style Sheets (CSS) or simply style sheets are text files that contain one or more rules in the form of property/value pairs to determine how elements in a Web page should be displayed. You can create styles in four ways:

- Using inline styles
- Using external style sheets
- Using internal style sheets
- Using style classes

Q7. What do you mean by Image Map?

Ans: Image Map is a technique that divides the image into multiple sections and allows linking of each section to different Web pages. Linked regions of an Image Map are called hot regions and each hot region is associated with an HTML file that is loaded when the hot region is clicked.

Q8. What is the use of the onsubmit event?

Ans: The `onsubmit` event is triggered when the submit button is clicked on the form. The `onsubmit` event is generally used to validate all the values provided in the form fields before submitting the form. If the form validation fails and the `onsubmit` event returns false, the data is not sent.

Q9. What are the ways to embed JavaScript in an HTML page?

Ans: You can embed a script in an HTML document by creating a script within the document or linking an external script file with the HTML document.

Q10. What are functions in JavaScript? Write the syntax to create a function.

Ans: A function refers to a cohesive block of statements that has a name and can be accessed or called from anywhere and any number of times in the script. In JavaScript, you can quickly create your own function in the script to perform an action or task. After you have created a function, you can access or call it in the script as many times as you want. Every time that you call a function, the specified action or task is performed. The syntax to create a function is:

```
function function_name (parameter1, parameter2, . . . parameterN)
{
   statement1
}
```