

# Javascript

# Introduction

- JavaScript is a computer language specially designed to work with Internet browsers. It lets you create small programs called scripts and embed them inside Hypertext Markup Language (HTML) pages in order to provide interactive content on your Web pages.
- JScript is Microsoft's implementation of JavaScript. In addition to running within Internet Explorer, Microsoft also provides a version of JScript that can be used as a desktop scripting language with the Windows Script Host.

- JavaScript is interpreted languages. This means that scripts written in these languages are not compiled before they are executed .
- Every script statement must first be converted into binary code in order to execute.
- JavaScript statements are processed at execution time.
- This means that they run a little slower than compiled programs.

- JavaScript is object-based scripting languages. This means that they view everything as objects.
- For JavaScripts, the browser is an object, a window is an object, and a button in a window is an object.
- JScript has access to a different set of objects. For example, JScript has the capability to access objects such as files, drives, and printers.

- Every object has properties, and you can use JavaScript to manipulate these properties.
- For example, with JavaScript you can change the background color of a browser window or the size of a graphic image.
- In addition to properties, objects have methods. Methods are the actions that objects can perform. For example, JavaScript can be used to open and close browser windows. By manipulating their properties and executing methods, you can control objects and make things happen.

- JavaScripts support event-driven programming.
- An event is an action that occurs when the user does something such as click on a button or move the pointer over a graphic image.
- JavaScript enables you to write scripts that are triggered by events.
- Did you ever wonder how buttons dynamically change colors on some Web sites when you move the mouse over them? It's simply a JavaScript technique known as a rollover. The event is the mouse moving over the button (object). This triggers the execution of an event handler, which is a collection of JavaScript statements that replaces the button with another one that uses a different color.

- JavaScripts that run within Web browsers have access to objects located on Web pages.

- JavaScript provides your Web pages with the capability to do many exciting things. For ex.
- Display pop-up messages that display and collect information from visitors
- Create rotating banners
- Open new windows
- Redirect people using older browsers to non-JavaScript HTML pages
- Detect the browsers and plug-ins being used by people visiting your Web site
- Validate forms and package their contents in an e-mail message
- Perform simple animations such as rollovers Exercise greater control over HTML frames and forms
- Take control of the status bar and create scrolling messages



- JavaScript can do a lot of different and exciting things.
- However, there is one thing that it cannot do-  
JavaScripts cannot run outside of the browser.  
This "limitation" helps make JavaScript more secure because users do not have to worry about somebody writing a JavaScript that might erase their hard drive or read their address book and extract private information.

- JavaScript, is successfully used for client-side scripting.
- By client-side scripting means scripts that execute within the browsers of people that visit your Web pages.
- A serverside version of JavaScript also exists. This version of JavaScript is designed to run on Web servers and is used by professional Web site developers to create scripts that provide dynamic content based on information received from visitors, as well as from information stored in a server-side database.

# History

- Years ago, the programmers at Netscape Communications Corporation recognized that HTML alone was not robust enough to support interactive Web programming. In 1995 they developed a scripting language called LiveScript, which gave Web page developers greater control over the browser. After successful introduction of Java as a object-oriented lang. LiveScript is renamed as JavaScript.

- JavaScripts are collections of programming statements that you embed in HTML documents by placing them within the tags. These tags can be placed within either the head or body section of an HTML page.
- Version of JavaScript:  
LANGUAGE="JavaScript" LANGUAGE="JavaScript1.1"  
LANGUAGE="JavaScript1.2"  
LANGUAGE="JavaScript1.3"  
LANGUAGE="JavaScript1.5"  
Now we have "JavaScript2.0".

```
<HTML>
  <HEAD>
    <TITLE>Script 1.2 - Sample HTML Page</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      document.write("Hello World");
    </SCRIPT>
  </BODY>
</HTML>
```

These three statements will make up your first JavaScript. You should recognize the first and last lines as script tags that tell your browser to execute the enclosed JavaScript statements. This script has just one statement. This statement tells the browser to write the message "Hello World" on the current document, which is the window in which the HTML page opened.

- JavaScript is a case sensitive programming language (unlike HTML, which enables you to use different capitalization when defining HTML tags).

# Different Ways to Integrate JavaScript into Your HTML Pages

- As you know, there are two places you can put your JavaScripts in an HTML page: in either the head or body section.
- In addition, you can either embed JavaScript directly into the HTML page or reference it in as an external .js file.
- One more way you can integrate JavaScript into an HTML page is as a component in an HTML tag.

```
<BODY>
  <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
    document.write("This JavaScript is located in the body
section");
  </SCRIPT>
</BODY>
```

```
<BODY>
  <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
    document.write("This first JavaScript is located in the
body section");
  </SCRIPT>
  <BR>
  <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
    document.write("This second JavaScript is also located in
the body section");
  </SCRIPT>
</BODY>
```



```
<HTML>
  <HEAD>
    <TITLE>Script 1.3 - Sample HTML Page</TITLE>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      window.alert("This JavaScript is located in the head
section");
    </SCRIPT>
  </HEAD>
  <BODY>
  </BODY>
```

---

```
<HTML>
```

```
<HTML>
  <HEAD>
    <TITLE>Script 1.4 - Sample HTML Page</TITLE>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      function DisplayMsg() {
        window.alert("This JavaScript is located in the head
section");
      }
    </SCRIPT>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

- To store your JavaScripts in external files, you need to save them as plain text files with a .js file extension. You then need to add the SRC attribute to the opening

```
<SCRIPT SRC="Test.js" LANGUAGE="JavaScript"  
TYPE="Text/JavaScript"> </SCRIPT>
```

In this example, an external JavaScript named Test.js has been specified. This external JavaScript can contain any number of JavaScript statements. However, it cannot contain any HTML whatsoever. Otherwise you'll end up with an error. For example, the contents of the Test.js script might be as simple as this:

```
document.write("This is an external JavaScript.");
```

- There are many advantages to putting JavaScripts in externally referenced files.
- For starters, by moving JavaScripts out of your HTML pages you make your HTML pages smaller and easier to work with.
- In addition, you can reuse the JavaScripts stored as external files over and over again by referencing them from any number of HTML pages.
- This way if you create a script that you want to reference from multiple HTML pages, you can do so without having to embed the same script in different HTML pages over and over again.
- As a bonus, should you ever want to modify the functionality of an externally stored script, you may do so without having to visit every HTML page where you would otherwise have embedded it.

# Placing JavaScript in an HTML Tag

```
<BODY onLoad=document.write("Hello World!")> </BODY>
```

- In this example, the JavaScript `onLoad=document.write("Hello World!")` statement has been added to the HTML tag. This particular JavaScript statement tells the browser to write the enclosed text when the browser first loads the HTML page.
- Placing small JavaScript statements inside HTML tags provides an easy way to execute small pieces of JavaScript code. Of course, this option really is beneficial only when executing small JavaScript statements and is impractical for larger JavaScript statements or situations that required multiple lines of code.

# Scripting and Programming

- One possible way to differentiate a script from a program is to define scripts as interpreted programs that are processed one line at a time and to define programs as collections of code that must be compiled before they can execute.

# Values

- JavaScripts store information as values

Value	Description
Boolean	A value that indicates a condition of either <code>true</code> or <code>false</code>
Null	An empty value
Numbers	A numeric value such as <code>99</code> or <code>3.142</code>
Strings	A string of text such as <code>"Welcome"</code> or <code>"Click here to visit ptpmagazine.com"</code>

- Rules for Variable Names Keep the following rules in mind when creating variables:
- Variable names can consist only of uppercase and lowercase letters, the underscore character, and the numbers 0 through 9.
- Variable names cannot begin with a number.
- JavaScript is case sensitive. If you declare a variable with the name `totalCount`, you must refer to it using the exact same case throughout your script.
- Variable names cannot contain spaces.
- You cannot use any reserved words as variable names. Refer to Internet article , "JavaScript Reserved Words," for a list of JavaScript reserved words.

# Defining Variable Scope

- You can create variables that can be accessed by JavaScripts located throughout an HTML page or from any location within a JScript. Alternatively, you can create variables that can only be accessed within the constraints of a small section of code known as a function. Defining the space in which the variable is effective is known as defining the variable's scope. A variable can be either local or global in scope.



# Local Variables

- A local variable is one that is declared explicitly using the 'var' or 'without var' statement inside a function. A function is a collection of script statements that can be called by another script statement to perform a specific action.

```
function ShowNotice()  
{  
    var textMessage = "Are you sure you want to quit?";  
    window.alert(textMessage);  
}
```

# Global Variables

- A global variable is any variable defined outside of a function. Such a variable is global because any script statement located in the Web page or script file can refer to it.
- You can create a global variable in several ways, including:
  - Creating the variable by referencing it inside a function without first declaring it using the var keyword.
  - Creating the variable anywhere else in a JavaScript with or without the var keyword.

```
<HTML>
  <HEAD>
    <TITLE>Script 2.3 - A demonstration of variable
scope</TITLE>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements
        glVarMsg1 = "Global variables created in the HEAD
section can be " +
          "referenced anywhere in this page.";
        function CreateVariables()
        {
          var lcVarMsg1 = "Local variables created inside a
function cannot " +
            "be referenced anywhere else.";
          glVarMsg2 = "Global variables created inside functions
in the HEAD " +
            "section can be referenced anywhere in this page.";
        }
      // End hiding JavaScript statements -->
    </SCRIPT>
  </HEAD>
```

```
<BODY>
  <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
    <!-- Start hiding JavaScript statements
      CreateVariables();
      glVarMsg3 = "Global variables created in the BODY
section can be " +

        "referenced anywhere in this page.";
      document.write(glVarMsg1 + "<BR>");
      document.write(glVarMsg2 + "<BR>");
      document.write(glVarMsg3 + "<BR>");
      document.write(lcVarMsg1 + "<BR>");
    // End hiding JavaScript statements -->
  </SCRIPT>
</BODY>
</HTML>
```

The script ran  
but experienced  
an error



Depending on how you have configured your browser, the error message may or may not be displayed. For example, if you are using Internet Explorer to load the HTML page that contains the script and you have not configured it to display all error messages, the only indication of an error will be a small yellow icon and an error message displayed on the browser's status bar.

# Manipulating Variables

- JavaScript also provides a number of other operators you can use to modify the value of variables.

Operator	Description	Example
<code>x + y</code>	Adds the value of x to the value of y	<code>total = 2 + 1</code>
<code>x - y</code>	Subtracts the value of y from x	<code>total = 2 - 1</code>
<code>x * y</code>	Multiplies the value of x and y	<code>total = x * 2</code>
<code>x / y</code>	Divides the value of x by the value of y	<code>total = x / 2</code>
<code>-x</code>	Reverses the sign of x	<code>count = -count</code>
<code>x++</code>	Post-increment (returns x, then increments x by one)	<code>x = y++</code>
<code>++x</code>	Pre-increment (increments x by one, then returns x)	<code>x = ++y</code>

# Assigning Values

Operator	Description	Examples	Result
=	Sets a variable value equal to some value	<code>x = y + 1</code>	6
+=	Shorthand for writing <code>x = x + y</code>	<code>x += y</code>	11
-=	Shorthand for writing <code>x = x - y</code>	<code>x -= y</code>	6
*=	Shorthand for writing <code>x = x * y</code>	<code>x *= y</code>	30
/=	Shorthand for writing <code>x = x / y</code>	<code>x /= y</code>	6
%=	Shorthand for writing <code>x = x % y</code>	<code>x %= y</code>	1

```
<HTML>
  <HEAD>
    <TITLE>Script 2.4 - A demonstration of JavaScript opera
tors</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements
        var y = 5;
        var x= y + 1;
        document.write("x = y + 1 ...Result = " + x);
        x += y;
        document.write("<BR>x += y ...Result = " + x);
        x -= y;
        document.write("<BR>x -= y ...Result = " + x);
        x *= y;
        document.write("<BR>x *= y ...Result = " + x);
        x /= y;
        document.write("<BR>x /= y ...Result = " + x);
      // End hiding JavaScript statements -->
    </SCRIPT>
  </BODY>
</HTML>
```





# Comparing Values

- One of the things you will find yourself doing in your scripts is comparing the values of variables.

Operator	Description	Example	Result
<code>==</code>	Equal to	<code>x == y</code>	true if both x and y are the same
<code>!=</code>	Not equal to	<code>x != y</code>	true if x and y are not the same
<code>&gt;</code>	Greater than	<code>x &gt; y</code>	true if x is greater than y
<code>&lt;</code>	Less than	<code>x &lt; y</code>	true if x is less than y
<code>&gt;=</code>	Greater than or equal to	<code>x &gt;= y</code>	true if x is greater than or equal to y
<code>&lt;=</code>	Less than or equal to	<code>x &lt;= y</code>	true if x is less than or equal to y
<code>!x</code>	Not	<code>!x</code>	true if x is false
<code>&amp;&amp;</code>	Both true	<code>x &amp;&amp; y</code>	true if x and y are both true
<code>  </code>	Either true	<code>x    y</code>	true if either x or y is true

```
<HTML>
  <HEAD>
    <TITLE>Script 2.5 - Example of a Value Comparison</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
<!-- Start hiding JavaScript statements
var x = 12;
var y = 5;
if (x > 10);
{
  document.write("x is greater than 10!");
}

  if (y > 10)
  {
    document.write("y is greater than 10!");
  }

  // End hiding JavaScript statements -->
</SCRIPT>
</BODY>
</HTML>
```

# Writing Text with Strings

- One object in particular, the String object, you will find yourself using over and over again in your scripts.
- The String object is made up of a collection of text characters. You create an instance of a String object using the new keyword, as shown in the following example:

```
MyString = new String("This is an example of a text string.");
```

- This statement creates a String object called MyString and assigns text to it. The new keyword is used to create objects. However, you are not required to use it. You can also create a String object by simply referring to it as in the following example:

```
MyString = "This is an example of a text string.";
```

You can change the value assigned to a String object by assigning it a new value. The following example changes the value of the MyString string:

```
MyString = "This is a second example of a text string.";
```

# String Properties

- Because a String object is a built-in JavaScript object, it has properties. In the case of the String object, there is just one property, its length.
- You can refer to an object's property by typing the name of the object, followed by a period and the name of the property.
- In the case of the MyString object's length property, you'd use MyString.length.

```
document.write("The length of MyString is " +  
               MyString.length);
```

# String Methods

- Although the String object has only one property, it has many methods. A method is an action that can be taken against the object. For example, you can use the String object's `bold()` method to display the string in bold, as shown in this example:

```
document.write(MyString.bold());
```

When referencing an object's methods, the correct syntax to use is the name of the object, a period, the name of the method, and the left and right parentheses. In this example, the parentheses did not contain anything. Some methods enable you to pass arguments to the method by placing them inside the parentheses.

- you can display strings in all uppercase characters using the `toUpperCase()` method
  - `document.write(MyString.toUpperCase());`
- Similarly, there is a `toLowerCase()` method. Other methods enable you to set the string's font color and font size or to display it in italic, blinking, or strikethrough text. A particularly useful method is `substring()`. This method enables you to extract a portion of a string's text.
- using the `String` object's `substring()` method, you can extract just the name portion of any string, as shown in the following example:



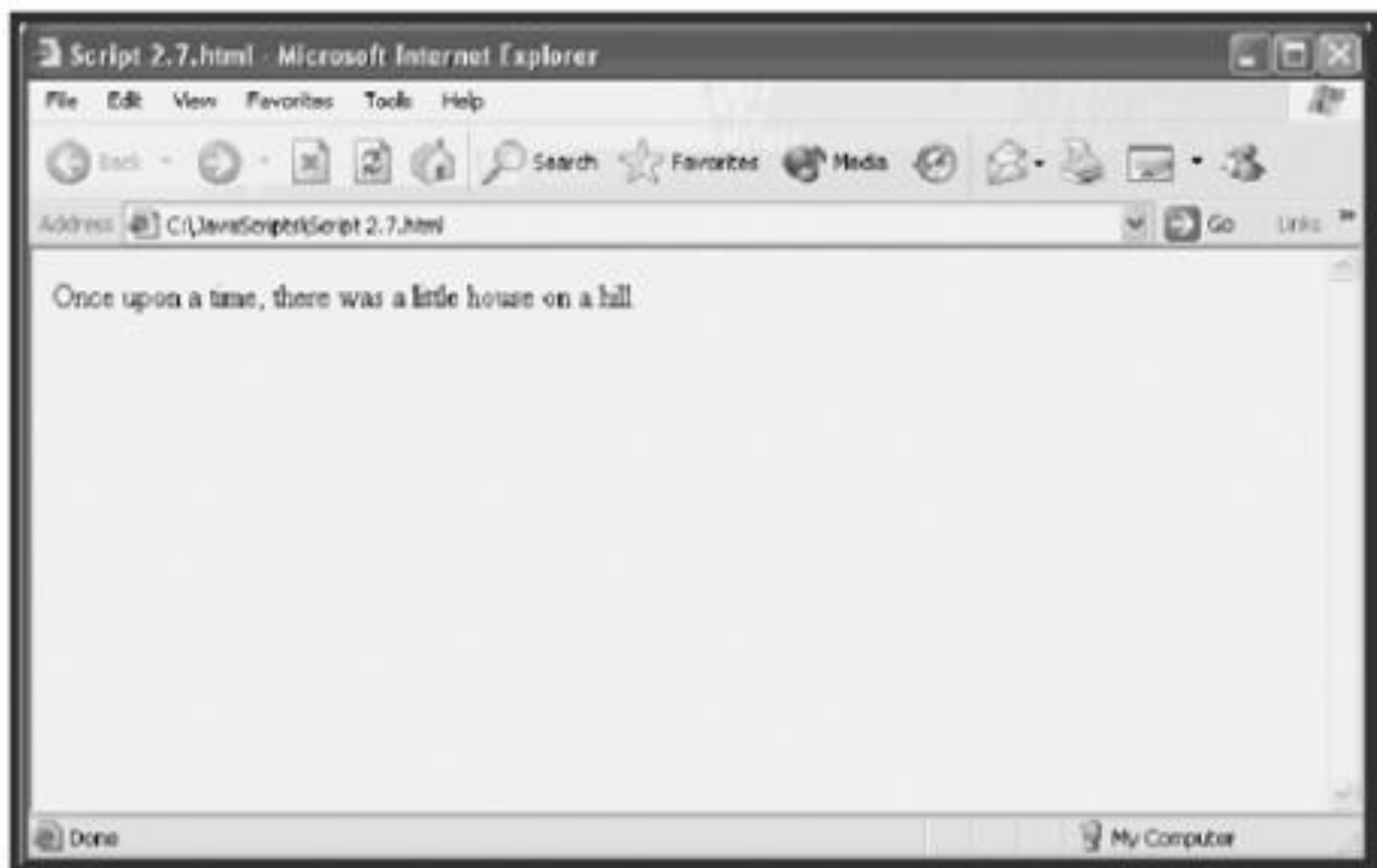
```
<HEAD>
  <TITLE>Script 2.6 - Substring Method Example</TITLE>
</HEAD>
<BODY>
  <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
    <!-- Start hiding JavaScript statements
    var cust1Info = "Bobby B Jones   9995 Park Place Ct
Richmond VA 23232";
    var cust2Info = "Sue K Miller    1112 Rockford Lane
Richmond VA 23232";
    var cust3Info = "Bobby B Jones   9995 Richland Drive
Richmond VA 23223";
    document.write(cust1Info.substring(0,14),"<BR>");
    document.write(cust2Info.substring(0,14),"<BR>");
    document.write(cust3Info.substring(0,14),"<BR>");
    // End hiding JavaScript statements -->
  </SCRIPT>
</BODY>
</HTML>
```



# String Concatenation

- you can concatenate two or more of them together using the + operator, as shown in the following example:

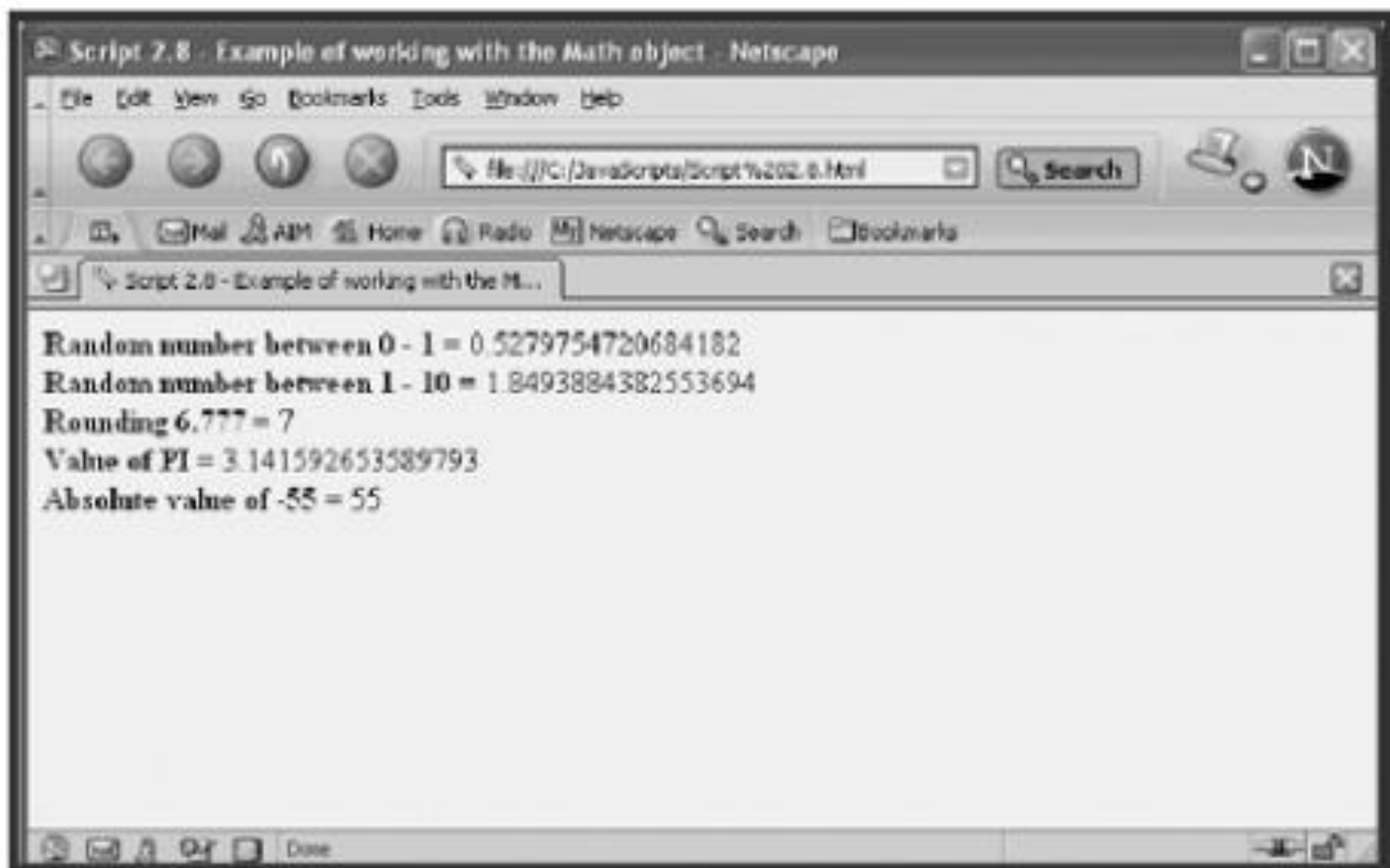
```
<HTML>
  <HEAD>
    <TITLE>Script 2.7 - Example of String
Concatenation</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements
        var stringOne = "Once upon a time, ";
        var stringTwo = "there was a little house on a hill.";
        var stringThree = stringOne + stringTwo;
        document.write(stringThree);
      // End hiding JavaScript statements -->
    </SCRIPT>
  </BODY>
</HTML>
```



# The Math Object

- This object is created automatically in every JavaScript, so you do not have to create an instance of it as you do with the String object. You can simply refer to the Math object when you need to. Like other objects, the Math object has properties and methods associated with it.
- The Math object's properties contain mathematical constants. For example, the Math property PI stores the value of pi. The Math object's methods contain built-in mathematical functions. There are methods that round numbers up and down, generate random numbers, and return the absolute value of a number.

```
<HTML>
  <HEAD>
    <TITLE>Script 2.8 - Example of working with the Math
object</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements
      //Generate a random number between 0 and 1
      document.write("<B>Random number between 0 - 1 = </B>", +
        Math.random(), "<BR>");
      //Generate a random number between 0 and 10
      document.write("<B>Random number between 0 - 10 = </B>", +
        Math.random() * 10, "<BR>");
      //Rounding a number to the nearest whole number
      document.write("<B>Rounding 6.777 = </B>",
Math.round(6.777), "<BR>");
      //Getting the value of PI
      document.write("<B>Value of PI = </B>", Math.PI, "<BR>");
      //Getting the absolute value of a number
      document.write("<B>Absolute value of -55 = </B>",
Math.abs(-55));
      // End hiding JavaScript statements -->
    </SCRIPT>
  </BODY>
</HTML>
```

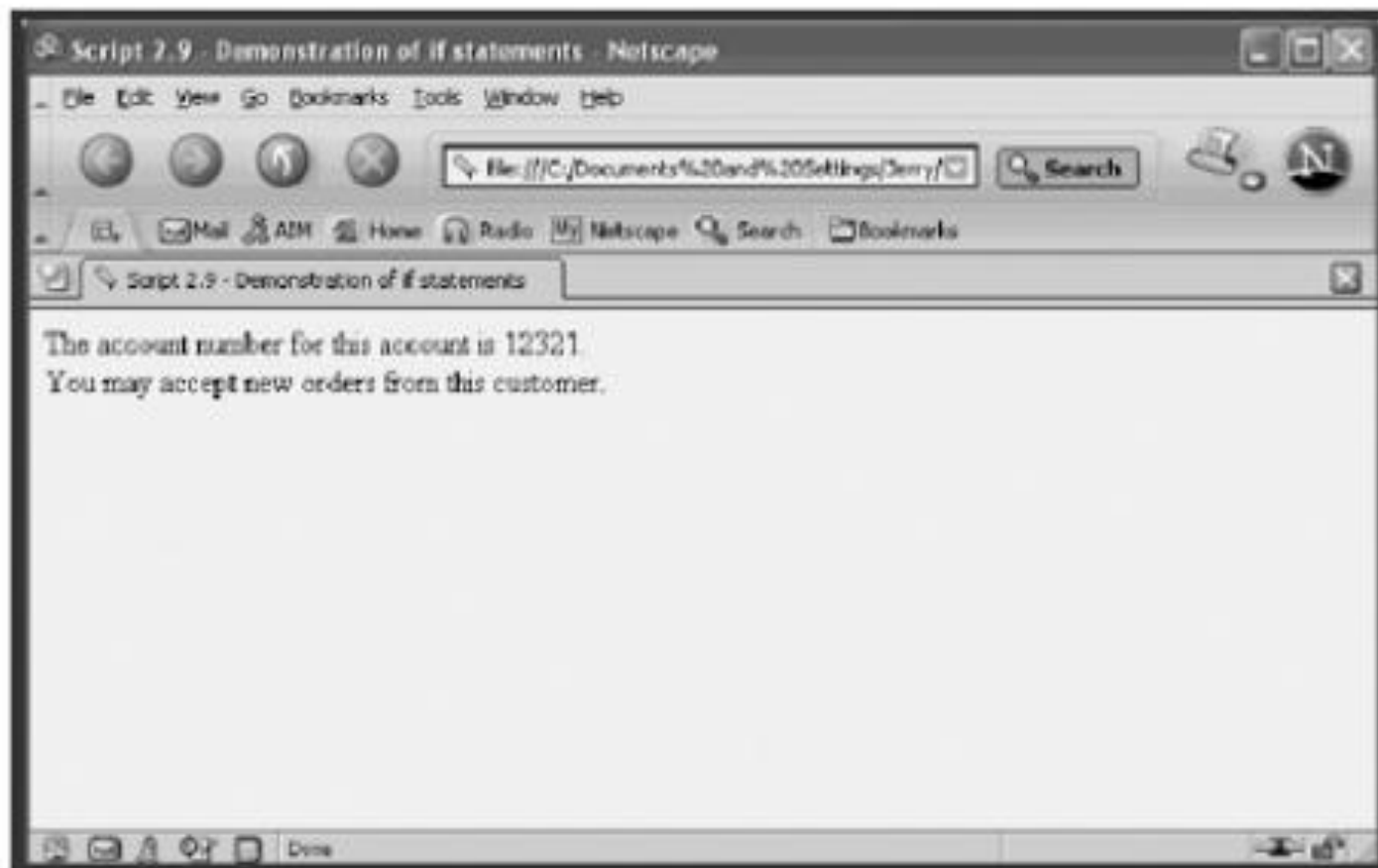
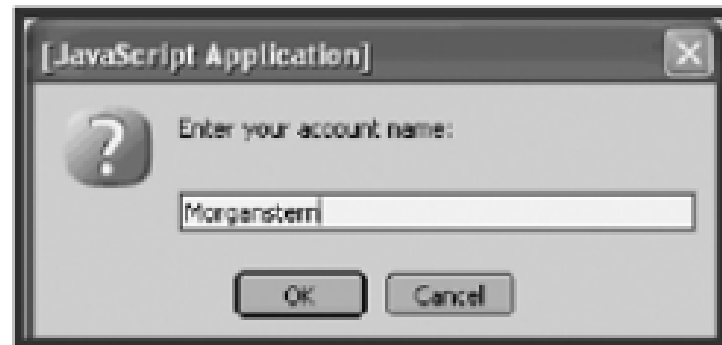


# JavaScript Statements

- These statements are the programming instructions, or logic, that you write to tell the scripts what you want them to do.
- **Using Conditional Statements** to Alter Script Flow
  - Conditional statements enable you to test for various conditions and take action based on the results of the test. Specifically, conditional statements execute when the tested condition proves to be true. Conditional statements include the if, if...else, and switch statements.



```
<HTML>
  <HEAD>
    <TITLE>Script 2.9 - Demonstration of if statements</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements
        var accountStatus = open; //assume all accounts are
open by default
        //Prompt the user for an account name
        var accountName = window.prompt("Enter your account
name:");
        //Test whether the supplied account name equals
Morganstern
        if (accountName == "Morganstern")
          document.write("The account number for this account
is 12321. <BR>");
        //Set accountStatus equal to warning if the account
name equals Davidson
        if (accountName == "Davidson") {
          document.write("The account number for this account
is 88844. <BR>");
          accountStatus = "warning";
        }
        //Display one of two messages based on the value of
accountStatus
        if (accountStatus != "warning") {
          document.write("You may accept new orders from this
customer.");
        }
        else {
          document.write("Do not accept new orders for this
account.");
        }
        // End hiding JavaScript statements -->
      </SCRIPT>
    </BODY>
  </HTML>
```



```
if (accountStatus != "warning") {  
    if (accountNumber > 10000) {  
        document.write("You may accept new orders from this  
customer.");  
    }  
    else {  
        document.write("Invalid account number. Notify bank  
security.");  
    }  
}  
Else {  
    document.write("This Account has been marked with a  
warning!");  
}
```

# The switch Statement

- The switch statement evaluates a series of conditional tests or cases and executes additional statements based on whether the case proves true. The syntax of the switch statement is shown here:

```
switch (expression) {  
    case label:  
        statements;  
        break;  
    .  
    .  
    .  
    case label:  
        statements;  
        break;  
    default:  
        statements;  
}
```

```
<HTML>
  <HEAD>
    <TITLE>Script 2.10 - Demonstration of the switch
statement</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements
```

```
    var accountStatus = "None";
    var accountName = window.prompt("Enter your account
name:");
    switch (accountName) {
        case "Morganstern":
            document.write("The account number for this
account is 12321.");
            accountStatus = "approved";
            break;
        case "Davidson":
            document.write("The account number for this
account is 88844.");
            accountStatus = "warning";
            break;
        default:
            document.write("The account is not registered in
this system.");
            accountStatus = "error";
    }
    if (accountStatus == "warning") {
        document.write(accountStatus);
        window.alert("Contact the on-duty supervisor");
    }
    // End hiding JavaScript statements -->
</SCRIPT>
</BODY>
</HTML>
```

- Adding Comments for Clarity
  - You can also add a comment at the end of script statements as demonstrated here.  
`document.write("Hello World"); //Write a message to the current Window`
- Alternatively, you may create multiline comments by placing them inside the `/*` and `*/` characters as demonstrated in the following example.

```
/* The following statement writes a message in the currently  
open browser window  
using the document object's write4() method. */  
document.write("Hello World");
```

# Optimizing Code with Looping Logic

- A loop is a series of statements that executes repeatedly, allowing you to perform iterative operations within your script. JavaScript and JScript statements that support looping logic include the for, while, do...while, label, break, and continue statements. The nice thing about loops is that they enable you to write just a few line of code and then to execute them repeatedly, making your scripts easier to write and maintain.



- The for Statement

```
for (expression; condition; increment) {  
    statements;  
}
```

```
<HTML>  
  <HEAD>  
    <TITLE>Script 2.12 - Demonstration of a for loop</TITLE>  
  </HEAD>  
  <BODY>  
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">  
      <!-- Start hiding JavaScript statements  
        for (i=0; i<5; i++) {  
          document.write("Watch as the variable grows with each  
iteration: ",i,"<BR>");  
        }  
      // End hiding JavaScript statements -->  
    </SCRIPT>  
  </BODY>  
</HTML>
```

Script 2.12 - Demonstration of a for loop - Netscape

File Edit View Go Bookmarks Tools Window Help



file:///E:/Chapter%202/Scripts/Script%202.12.h

Search



Mail AIM Home Radio Netscape Search Bookmarks

Script 2.12 - Demonstration of a for loop

Watch as the variable grows with each iteration: 0  
Watch as the variable grows with each iteration: 1  
Watch as the variable grows with each iteration: 2  
Watch as the variable grows with each iteration: 3  
Watch as the variable grows with each iteration: 4

Done

# • The while Statement

The while statement executes a loop as long as a condition is true. The syntax of the while statement is shown here:

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

```
<HTML>  
  <HEAD>  
    <TITLE>Script 2.13 - Demonstration of the while  
statement</TITLE>  
  </HEAD>  
  <BODY>  
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">  
      <!-- Start hiding JavaScript statements  
  
      var counter = 10;  
  
      document.write("<B>Watch me count backwards!</B><BR>");  
  
      while (counter > 0) {  
        counter;  
        document.write("counter = ", counter , "<BR>");  
      }  
  
      // End hiding JavaScript statements -->  
    </SCRIPT>  
  </BODY>  
</HTML>
```

Watch me count backwards!

counter = 9

counter = 8

counter = 7

counter = 6

counter = 5

counter = 4

counter = 3

counter = 2

counter = 1

counter = 0

- **The do...while Statement**

The do...while statement executes a loop until a condition becomes false. The syntax of the do...while statement is outlined below:

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

```
<HTML>
  <HEAD>
    <TITLE>Script 2.14 - Demonstration of the do...while
statement</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        var counter = 10;

        document.write("<B>Watch me count backwards!</B><BR>");
        do {
          counter;
          document.write("counter = ", counter , "<BR>");
        } while (counter > 0)

        // End hiding JavaScript statements -->
      </SCRIPT>
    </BODY>
  </HTML>
```

- The label Statement
  - The label statement lets you specify a reference point in your script. Typically, the label statement is associated with loops. A label statement can be referenced by either the break or continue statement.
- The break Statement
  - The break statement lets you terminate a label, switch, or loop. The script then continues processing with the statement that follows the label, switch, or loop statement that was broken out of.
- The continue Statement
  - The continue statement is similar to the break statement. However, instead of terminating the execution of the loop, it merely terminates the current iteration of the loop.

```
<HTML>
  <HEAD>
    <TITLE>Script 2.17 - Demonstration of the continue
statement</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

      var counter = 10;

      document.write("<B>Watch me count backwards!</B><BR>");

      CounterLoop:
```



```
while (counter > 0) {  
    counter;  
    switch (counter) {  
        case 8:  
            continue CounterLoop;  
        case 6:  
            continue CounterLoop;  
        case 4:  
            continue CounterLoop;  
        case 2:  
            continue CounterLoop;  
    }  
    document.write("counter = ", counter , "<BR>");  
}  
// End hiding JavaScript statements -->  
</SCRIPT>  
</BODY>  
</HTML>
```

# Manipulating Objects

- You can think of objects as being similar to variables except that objects can contain multiple values known as its properties. Objects also can contain built-in functions, which are collections of predefined script statements that are designed to work with the object and its data. Examples of browser objects include browser windows and form elements.
- There are a number of JavaScript statements that provide you with the ability to manipulate properties associated with objects. These statements include the **for...in** and **with** statements.
- The **for...in** statement enables you to access all the properties for a specified object, and the **with** statement provides easy access to specific object properties and methods.

# The for...in Statement

- The for...in statement is used to iterate through all the properties belonging to a specified object. The syntax of the for...in statement is listed below:

```
for (variable in object) {  
    statements;  
}
```

```
<HTML>
  <HEAD>
    <TITLE>Script 2.18 - Demonstration of the for...in statement
  </TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements


        for (i in navigator) {
          document.write(i, "<BR>");
        }

      // End hiding JavaScript statements -->
    </SCRIPT>
  </BODY>
</HTML>
```

Script 2.18 - Demonstration of the for...in statement - Microsoft Internet Explorer

File Edit View Favorites Tools Help


Back Forward Stop Reload Home Search Favorites Media Print Mail

Address  E:\Chapter 2\Scripts\Script 2.18.html

 Go [Links](#)

```
appCodeName  
appName  
appMinorVersion  
cpuClass  
platform  
plugins  
opsProfile  
userProfile  
systemLanguage  
userLanguage  
appVersion  
userAgent  
onLine  
cookieEnabled  
mimeTypes
```

 Done

 My Computer

# The with Statement

- The with statement is a convenient way of saving a few keystrokes when writing your scripts. It enables you to set a default object for a group of statements. The syntax of the with statement is listed below.

```
with (object) {  
    statements;  
}
```

```
<HTML>
  <HEAD>
    <TITLE>Script 2.19 - Demonstration of the with statement
  </TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        with (document) {
          write("The with statement saved me a few keystrokes.
<BR>");
          write("Its use is purely discretionary!");
        }

      // End hiding JavaScript statements -->
    </SCRIPT>
  </BODY>
</HTML>
```

# Streamlining Your Scripts with Functions

- A **function** is a collection of statements that performs a given task. Most functions are defined in the head section of HTML pages. Putting your functions in the head section is a good idea for two reasons:
  - The first is that you'll always know where your functions are because they're in a common location.
  - The second reason is that the head section always executes first, ensuring that functions are available when needed later in the script.
- If a JavaScript statement attempts to call a function before the browser has loaded it, an error will occur because the function is technically still undefined.



# What is difference between a function and a method.

- The answer is not much. Methods are simply functions that have been predefined. Methods are associated with specific objects, meaning that each object provides its own collection of methods that you can call from within your scripts.

# Defining a Function

- The first step in working with a function is to define it, which is accomplished using the syntax outlined below.

```
function FunctionName (p1, p2,...pn) {  
    statements;  
    return  
}
```

The function statement defines a new function with the name you specify. The function's name is followed by a pair of parentheses that may contain a list of optional arguments that the function can use as input. Commas separate multiple arguments. All function statements are placed within the curly braces {}. Functions can end with an optional return statement, which enables optional information to be returned to the statement that called the function.

```
function SayHello(visitorName) {  
    window.alert("Hello and welcome, " + visitorName);  
    return  
}
```

## Calling Functions

The following two function calls demonstrate this form of call. The first example calls a function named SayHello() without passing any arguments. The second example calls a function named SayGoodbye() and passes a single argument.

```
SayHello();  
SayGoodbye("William");
```

- You can make a call to a function in the form of an expression. This enables the function to return a value to the calling statement. For example, the following statement calls a function named `GetUserName()` and stores the result returned by the function in a variable named `UserName`.

```
---- UserName = GetUserName();
```

```
<HTML>
  <HEAD>
    <TITLE>Script 2.20 - A simple function</TITLE>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        function SayHello(visitorName) {
          window.alert("Hello and welcome, " + visitorName);
        }

      // End hiding JavaScript statements -->
    </SCRIPT>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        var name;

        name = window.prompt("What is your name?","");
        SayHello(name);

      // End hiding JavaScript statements -->
    </SCRIPT>
  </BODY>
</HTML>
```



```
<HTML>
  <HEAD>
    <TITLE>Script 2.21 - Returning values from
functions</TITLE>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        var name;
        var result;

        function SayHello() {
          name = window.prompt("What is your name?","");
          return name;
        }

        // End hiding JavaScript statements -->
      </SCRIPT>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        result = SayHello();

        window.alert("Hello and welcome, " + result);

        // End hiding JavaScript statements -->
      </SCRIPT>
  </BODY>
</HTML>
```

# Using Arrays

- An array is an indexed list of values that can be referred to as a unit. Arrays can contain any type of value. Before you can use an array, you must first declare it. The following example shows how to create an array that will hold 10 strings that contain information about an automobile inventory.
- The first statement uses the new keyword to create an Array object named Auto that will contain 10 entries. The rest of the statements assign string values to the array. As you can see, the array's index starts with 0 and goes to 9.



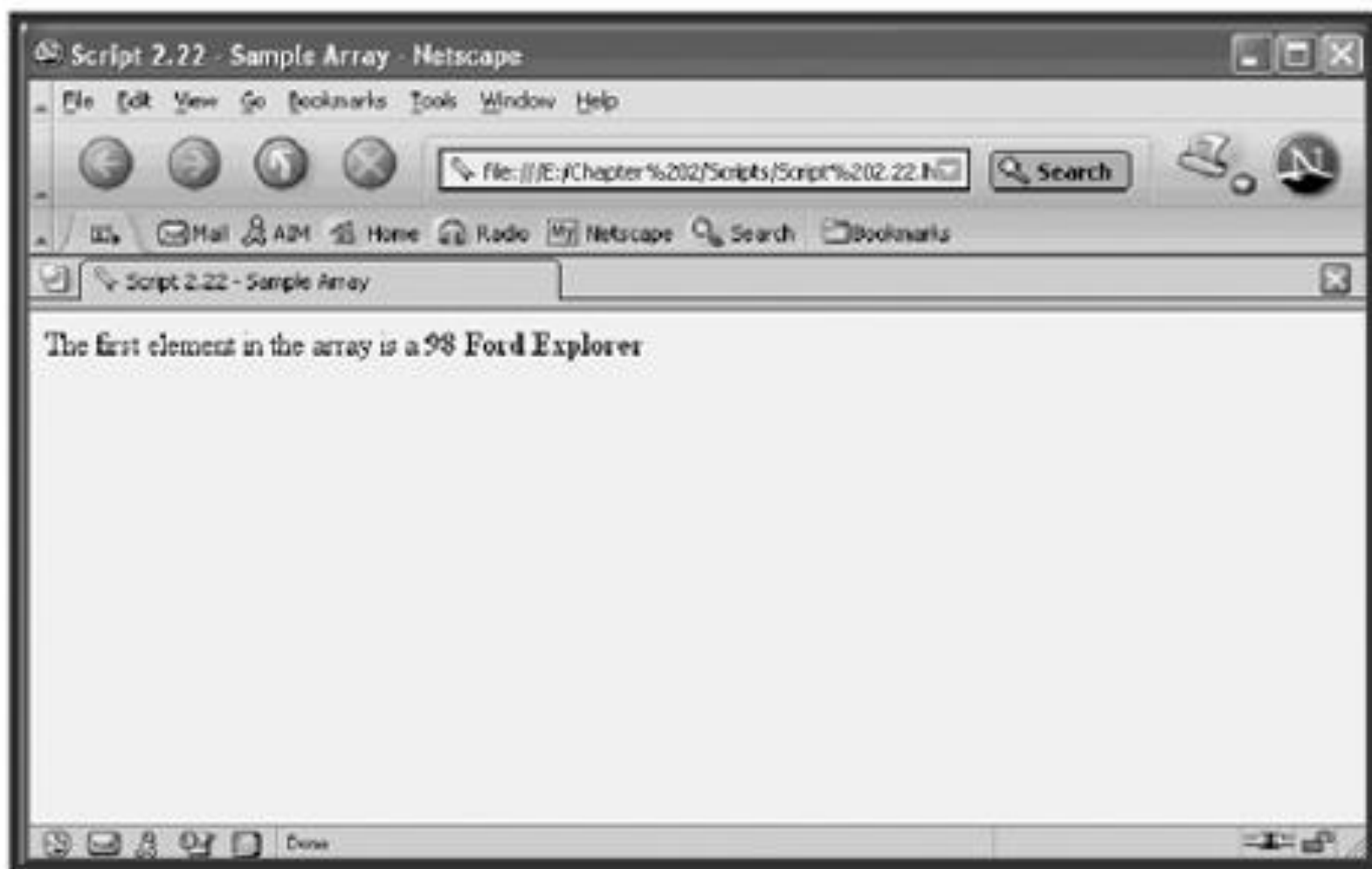
```
<HTML>
  <HEAD>
    <TITLE>Script 2.22 - Sample Array</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

      Auto = new Array(10);
      Auto[0] = "98 Ford Explorer";
      Auto[1] = "97 Ford Explorer";
      Auto[2] = "85 Plymouth Mustang";
      Auto[3] = "96 Plymouth Voyager";
```

```
Auto[4] = "90 Honda Civic";
Auto[5] = "97 Honda Civic";
Auto[6] = "96 Plymouth Neon";
Auto[7] = "98 Plymouth Neon";
Auto[8] = "92 Ford Explorer Wagon";
Auto[9] = "95 Honda Civic Hatchback";

document.write("The first element in the array is a
<B>", Auto[0], "</B>");

// End hiding JavaScript statements -->
</SCRIPT>
</BODY>
</HTML>
```



# Processing Arrays with for Loops

```
<HTML>
<HEAD>
<TITLE>Script 2.23 - Looping through an Array</TITLE>
</HEAD>
<BODY>
  <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
    <!-- Start hiding JavaScript statements

    var arrayLength;
    Auto = new Array(10);
    Auto[0] = "98 Ford Explorer";
    Auto[1] = "97 Ford Explorer";
    Auto[2] = "85 Plymouth Mustang";
    Auto[3] = "96 Plymouth Voyager";
    Auto[4] = "90 Honda Civic";
    Auto[5] = "97 Honda Civic";
    Auto[6] = "96 Plymouth Neon";
    Auto[7] = "98 Plymouth Neon";
    Auto[8] = "92 Ford Explorer Wagon";
    Auto[9] = "95 Honda Civic Hatchback";

    arrayLength = Auto.length;

    document.write("<B>Current Inventory Listing</B><BR>");

    for (var i = 0; i < arrayLength;i++) {
      document.write(Auto[i], "<BR>");
    }

    // End hiding JavaScript statements -->
  </SCRIPT>
</BODY>
</HTML>
```



# Dense Arrays

- You can also create dense arrays. A dense array is one that is populated at the time it is declared. This is a very efficient technique for creating small arrays. The following example shows you how to create a dense array named `animals` that consists of five entries. The array will have the following structure:

```
animals = new Array("mice", "dog", "cat", "hamster", "fish");
```

```
<HTML>
  <HEAD>
    <TITLE>Script 2.24 - A Dense Array</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        var arrayLength;

        Animals = new Array("mice", "dog", "cat", "hamster",
"fish");

        arrayLength = Animals.length;

        document.write("<B>List of animals in the Zoo</B><BR>");

        for (var i = 0; i < arrayLength;i++) {
          document.write(Animals[i], "<BR>");
        }

        // End hiding JavaScript statements -->
      </SCRIPT>
    </BODY>
  </HTML>
```





# Sorting Arrays

- Arrays have a `sort()` method you can use to sort their contents. The following example shows how to display a sorted list of the contents of an array. The script first defines a dense array with five entries. Next, it displays a heading, and it then uses the `sort()` method inside a `document.write()` statement to display the sorted list. Notice that I did not have to create a for loop to step iteratively through the array's index.

```
<HEAD>
  <TITLE>Script 2.25 - Sorting an Array</TITLE>
</HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        Animals = new Array("mice", "dog", "cat", "hamster",
"fish");

        document.write("<B>List of animals in the
Zoo</B><BR>");
        document.write(Animals.sort());

      // End hiding JavaScript statements -->
    </SCRIPT>
  </BODY>
</HTML>
```



# Populating Arrays with Dynamic Data

- You can create scripts that enable users to supply them with their data, as opposed to hard coding it into the script. For example, the following script prompts users to type their three favorite things and then creates an array to contain the list.

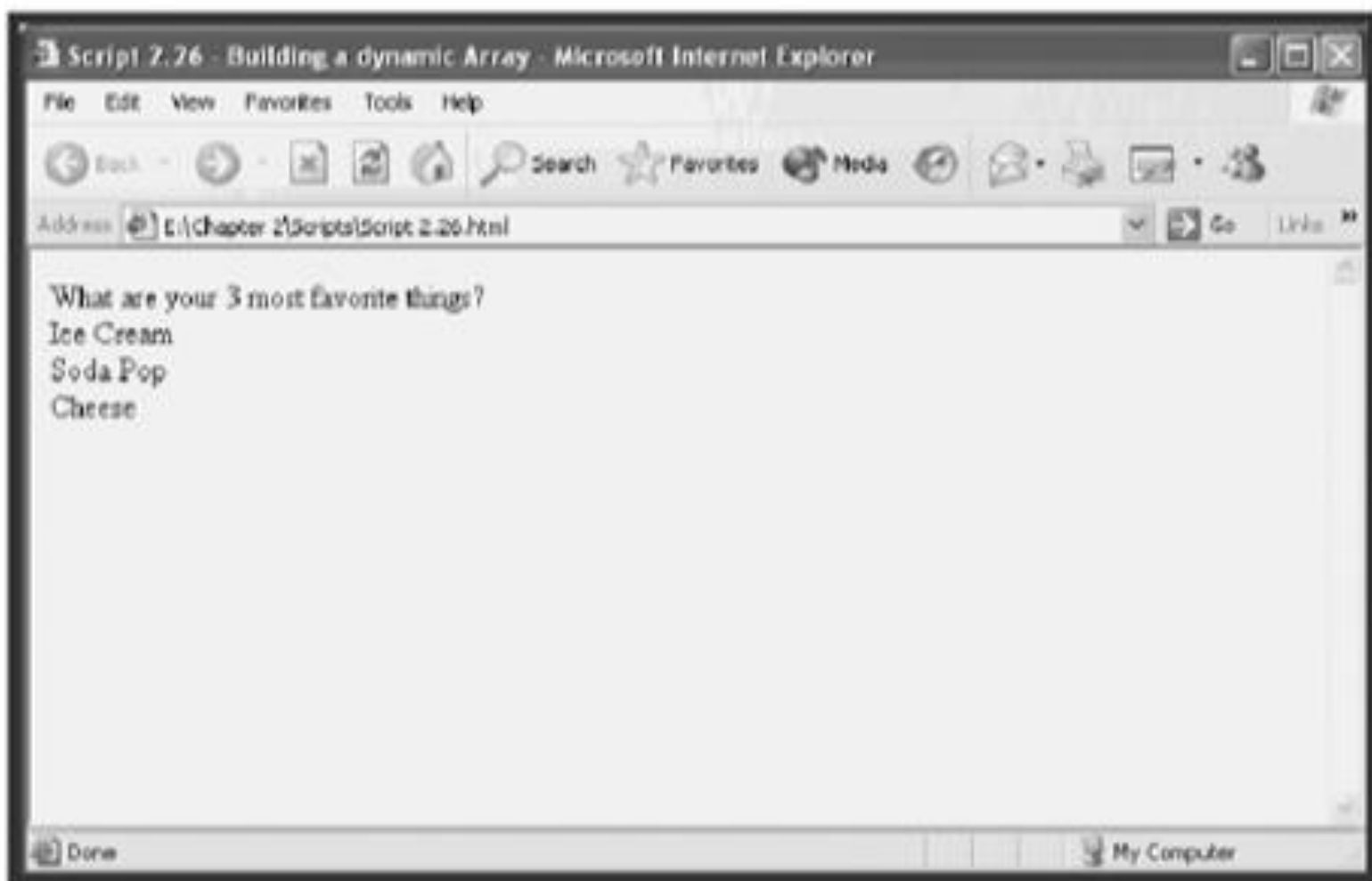
```
<HTML>
  <HEAD>
    <TITLE>Script 2.26 - Building a dynamic Array</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        document.write("What are your 3 most favorite things?
<BR>");

        NumberArray = new Array(3);

        for (var i = 0; i < 3;i++) {
          var yourNumber = prompt('I like: ', ' ');
          NumberArray[i] = yourNumber;
          document.write(NumberArray[i], "<BR>");
        }

        // End hiding JavaScript statements -->
      </SCRIPT>
    </BODY>
  </HTML>
```



- Another way to work with an array dynamically is by increasing its length during script execution. You can extend the size of an array by adding new array elements with an index number that is higher than the last index number in the array. For example, if I were to declare an array with an index of 50, I could later expand the array by adding a new array element with an index number of 99. This would increase the size of the array to 100.

# Object-Based Programming

- **Creating Custom Objects**

---Custom objects are created using the new keyword. For example, you can create a new instance of an Array object as demonstrated below.

```
Animals = new Array("mice", "dog", "cat", "hamster", "fish");
```

Built-in objects include the Array, Boolean, Date, Function, Math, Number, Object, and String objects.



# The Array Object

- To work with arrays, you must create an instance of an Array object.
- An array is an indexed list of values that can be referred to as a unit. You can work with array elements by referencing their names and index numbers. An array has a length property that tells you how large the array is, as well as an assortment of methods for manipulating its contents. These include methods to sort an array, to concatenate two arrays into a single array, to add or remove elements from the array, and to reverse the order of the array. The first entry in an array is assigned the index 0.

- The following script demonstrates how to build and display the contents of a multidimensional array:



```

<HTML>
  <HEAD>
    <TITLE>Script 2.27 - Creating a Multi-Dimensional
Array</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!--Start hiding JavaScript statements

        var arraySize

        //Part 1 - Prompt user for array size
        arraySize = window.prompt("How many cars do you plan to
list?","");
        MyArray = new Array(arraySize);

        //Part 2 - Populate the array with user supplied
information
        for (row=0; row < arraySize; row++ ) {
          MyArray[row] = new Array(3);
          MyArray[row][0] = window.prompt("Type of car?","");
          MyArray[row][1] = window.prompt("Color of car?","");
          MyArray[row][2] = window.prompt("Sale price?","");
        }

        //Part 3 - Display the contents of the array
        for (row=0; row < arraySize; row++ ) {
          for (column=0; column < 3; column++ ) {
            document.write(MyArray[row][column] + " ");
          }
          document.write("<BR>");
        }
      // End hiding JavaScript statements -->
    </SCRIPT>
  </BODY>
</HTML>

```

# The Boolean Object

- JavaScript allow you to treat Boolean values like objects

```
BooleanObject = new Boolean(value);
```

In this syntax, value must be either true or false. If you assign a Boolean object a value other than true or false, the value will be converted to a Boolean value. If a value of NaN, null, undefined, -0, or 0 is found, then the values assigned will be false; otherwise the value is converted to true.

```
TestObject1 = new Boolean();           //initial value is false
TestObject2 = new Boolean(-0);         //initial value is false
TestObject3 = new Boolean(false);      //initial value is false
TestObject4 = new Boolean(true);       //initial value is true
TestObject5 = new Boolean("Hello");    //initial value is true
TestObject6 = new Boolean("false");    //initial value is true
```

# The Date Object

- You must use the Date object to insert dates and times in your scripts. The Date object does not have any properties but does provide access to a large number of built-in methods.
- Date information is based on the number of millions of seconds that have passed since January 1, 1970.

```
Today = new Date();  
document.write(today);
```

```

<HTML>
  <HEAD>
    <TITLE>Script 2.29 - Using the Date Object</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        var arrayLength;

        TodaysDate = new Date();

        document.write("<B>Today's date is: </B>" + TodaysDate
+ "<P>");

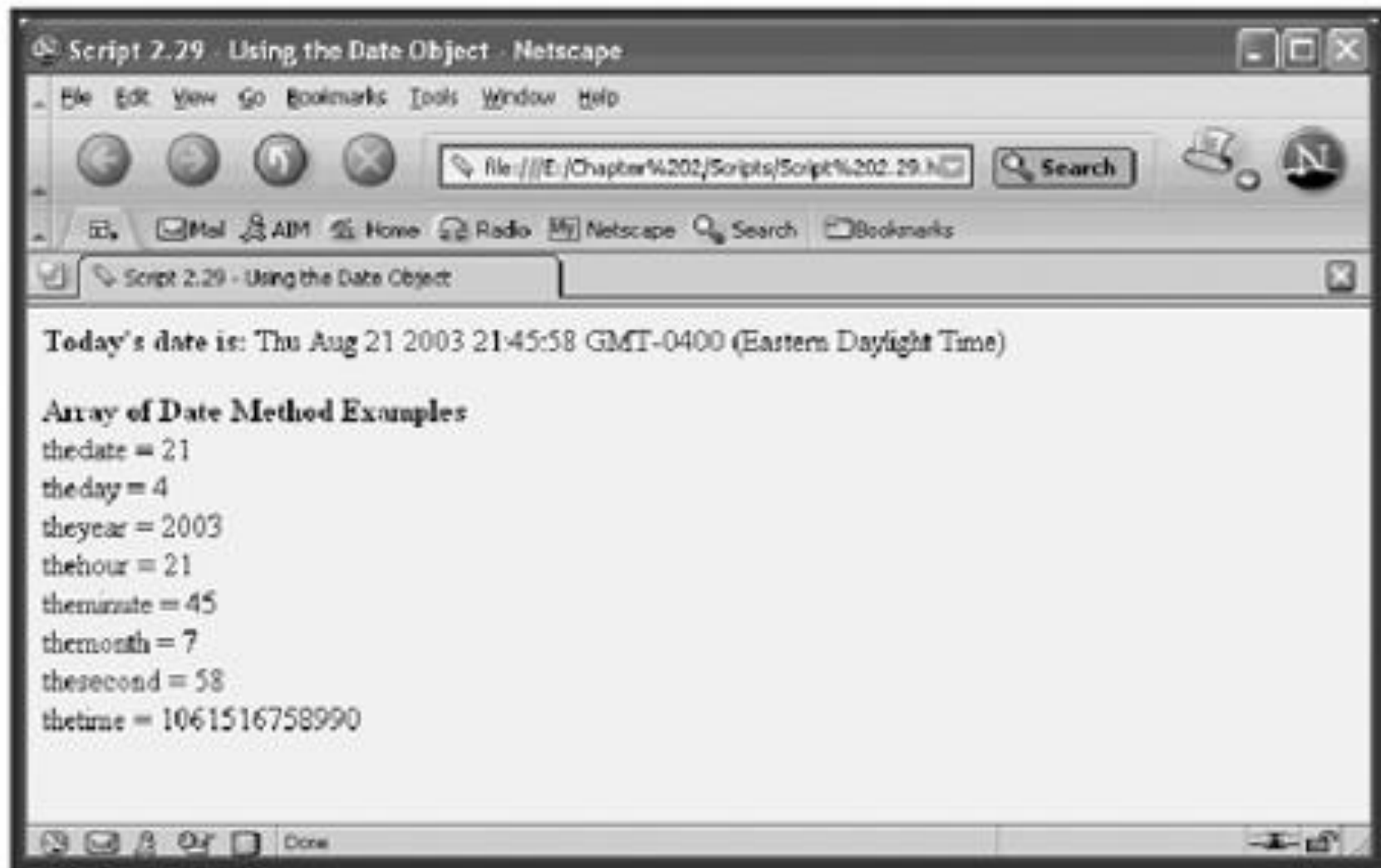
        MyDate = new Array(8);
        MyDate[0] = "thedata = "    + TodaysDate.getDate();
        MyDate[1] = "theday = "     + TodaysDate.getDay();
        MyDate[2] = "theyear = "    + TodaysDate.getFullYear();
        MyDate[3] = "thehour = "    + TodaysDate.getHours();
        MyDate[4] = "theminute = "  + TodaysDate.getMinutes();
        MyDate[5] = "themonth = "   + TodaysDate.getMonth();
        MyDate[6] = "thesecond = "  + TodaysDate.getSeconds();
        MyDate[7] = "thetime = "    + TodaysDate.getTime();

        arrayLength = MyDate.length;

        document.write("<B>Array of Date Method Examples</B>
<BR>");

        for (var i = 0; i < arrayLength; i++ ) {
          document.write(MyDate[i], "<BR>");
        }
        // End hiding JavaScript statements -->
      </SCRIPT>
    </BODY>
  </HTML>

```



# The Math Object

- The Math object provides mathematical constants in the form of properties. It also provides mathematical functions in the form of methods.

Property	Description
E	Euler's constant (2.718)
LN2	Natural logarithm of 2 (2.302)
LN10	Natural logarithm of 10 (.693)
LOG2E	Base 2 logarithm of e (.434)
LOG10	Base 10 logarithm of 10 (1.442)
PI	Ratio of the circumference of a circle to its diameter (3.141549)
SQRT1_2	Square root of ? (.707)
SQRT2	Square root of 2 (1.414)



Method	Description
<code>abs()</code>	Returns the absolute value
<code>cos()</code> , <code>sin()</code> , <code>tan()</code>	Trigonometric functions
<code>acos()</code> , <code>asin()</code> , <code>atan()</code>	Inverse trigonometric functions
<code>exp()</code> , <code>log()</code>	Exponential and natural logarithms
<code>ceil()</code>	Returns the lowest integer greater than or equal to the argument
<code>floor()</code>	Returns the highest integer less than or equal to the argument
<code>min(x, y)</code>	Returns either x or y, depending on which is lower
<code>max(x, y)</code>	Returns either x or y, depending on which is higher
<code>pow(x, y)</code>	Returns the value of $x^y$
<code>random()</code>	Returns a random number between 0 and 1
<code>round()</code>	Rounds the argument to the nearest integer
<code>sqrt()</code>	Returns the square of the argument

# The Number Object

- The Number object lets you treat numbers like objects and store numerical constants as properties. The values of these constants cannot be changed.
- You define an instance of the Number object using the new keyword as demonstrated-

```
MyNumber = new Number(99.99);
```

Property	Description
MAX_VALUE	Largest possible number
MIN_VALUE	Smallest possible number
NaN	Not a number
NEGATIVE_INFINITY	Positive infinity
POSITIVE_INFINITY	Negative infinity

# The String Object

- As you have already learned, the String object enables you to work with strings as objects.

Method	Description
<code>charAt()</code>	Returns the character at the specified position in the string where the index of a String object begins at zero
<code>concat()</code>	Combines two strings into one new string
<code>fromCharCode()</code>	Creates a string value based on the supplied code set
<code>indexOf()</code>	Returns the position of a specified substring
<code>lastIndexOf()</code>	Returns the last position of a specified substring
<code>slice()</code>	Creates a new string using a portion of the current string
<code>split()</code>	Organizes a string into an array
<code>substring()</code>	Returns the specified portion of a string
<code>toLowerCase()</code>	Returns the string in all lowercase characters
<code>toUpperCase()</code>	Returns the string in all uppercase characters

```
<HTML>
  <HEAD>
    <TITLE>Script 2.31 - Working with the String Object</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
      <!-- Start hiding JavaScript statements

        MyString1 = new String(" He is gone now!");

        MyString2 = new String("Once upon a time there was a
little boy " +
        "who cried wolf.");

        document.write("<B>MyString1 - </B>" + MyString1 +
"<BR>");
        document.write("<B>MyString2 - </B>" + MyString2 +
"<BR>");

        document.write("<B>MyString2.charAt(5) - </B>" +
MyString2.charAt(5) +
        "<BR>");
        document.write("<B>MyString2.charCodeAt(5) - </B>" +
        MyString2.charCodeAt(5) + "<BR>");
```

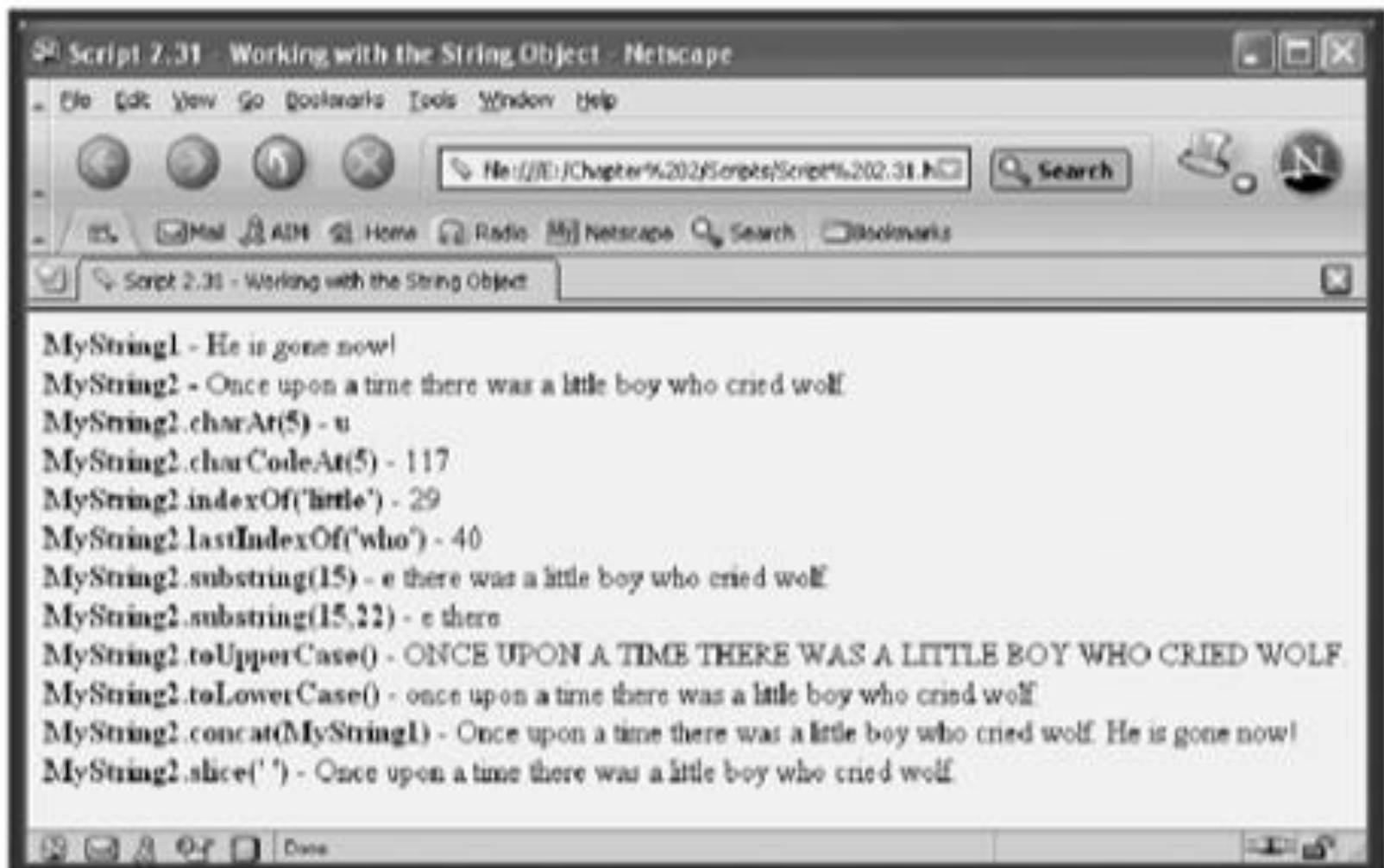
```
document.write("<B>MyString2.indexOf('little') - </B>" +
    MyString2.indexOf("little") + "<BR>");
document.write("<B>MyString2.lastIndexOf('who') - </B>" +
    MyString2.lastIndexOf("who") + "<BR>");
document.write("<B>MyString2.substring(15) - </B>" +
    MyString2.substring(15) + "<BR>");
document.write("<B>MyString2.substring(15,22) - </B>" +
    MyString2.substring(15,22) + "<BR>");

document.write("<B>MyString2.toUpperCase() - </B>" +
    MyString2.toUpperCase() + "<BR>");
document.write("<B>MyString2.toLowerCase() - </B>" +
    MyString2.toLowerCase() + "<BR>");

document.write("<B>MyString2.concat(MyString1) - </B>" +
    MyString2.concat(MyString1) + "<BR>");

document.write("<B>MyString2.slice(' ') - </B>" +
    MyString2.slice(" ") + "<BR>");

// End hiding JavaScript statements -->
</SCRIPT>
</BODY>
</HTML>
```



# Working with Browser-Based Objects

- When an HTML page loads a browser such as Netscape Communicator or Internet Explorer, the browser defines a collection of objects based on the content it finds on the page. These objects are created in a top-down order as the page is loaded. As the browser builds the object references, it does so in a hierarchical fashion.

# A Browser View of Objects

- When a browser loads an HTML page, it simultaneously creates a logical view of all relevant objects into a tree-like structure. These objects are related to one another and have parent-child and sibling relationships. For example, consider the following HTML page:

```
<HTML>
```

```
  <HEAD>
```

```
    <TITLE>Script 3.1 - A Typical HTML page</TITLE>
```

```
  </HEAD>
```

```
  <BODY>
```

```
    <P>Hello.</P>
```

```
    <P>Welcome to my Web site.</P>
```

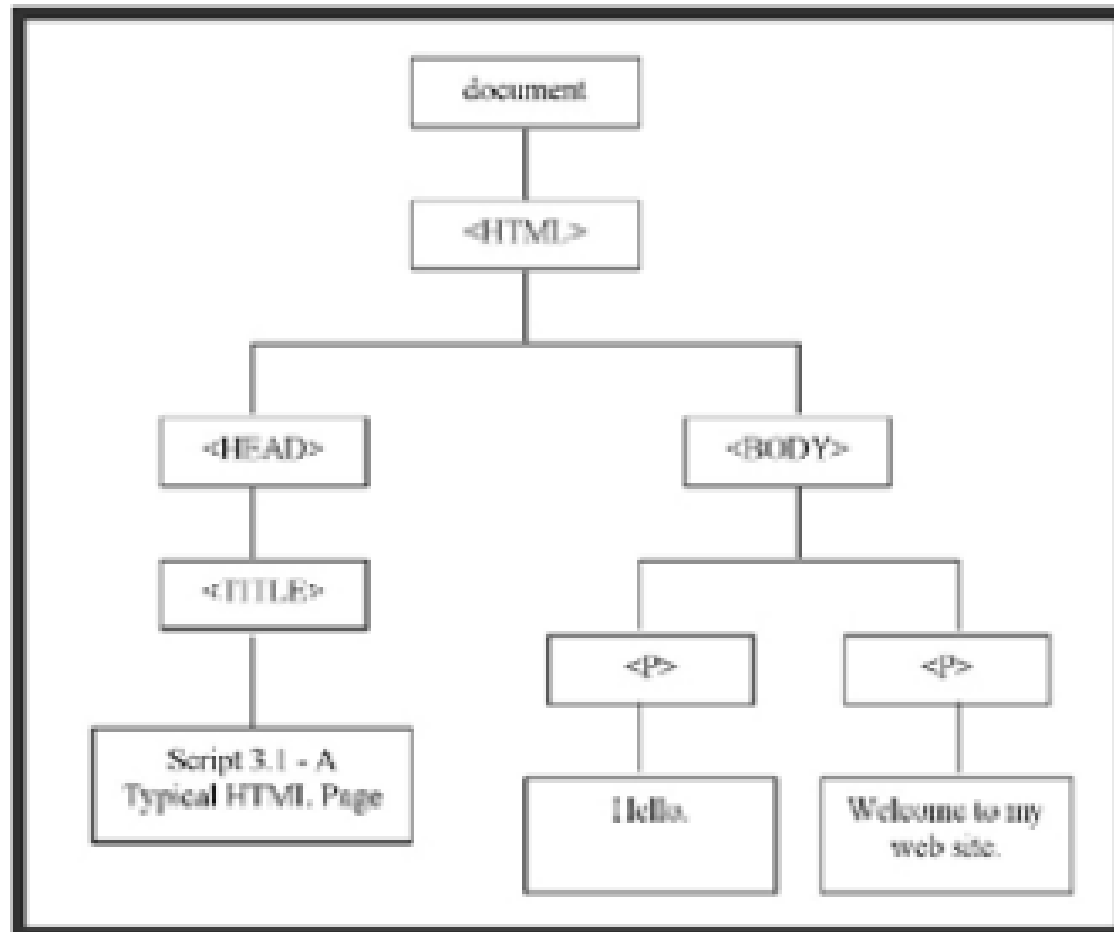
```
  </BODY>
```



When loaded by a browser, you see the output shown in Figure



The browser, on the other hand, sees the content of the HTML page a little differently, as depicted in Figure



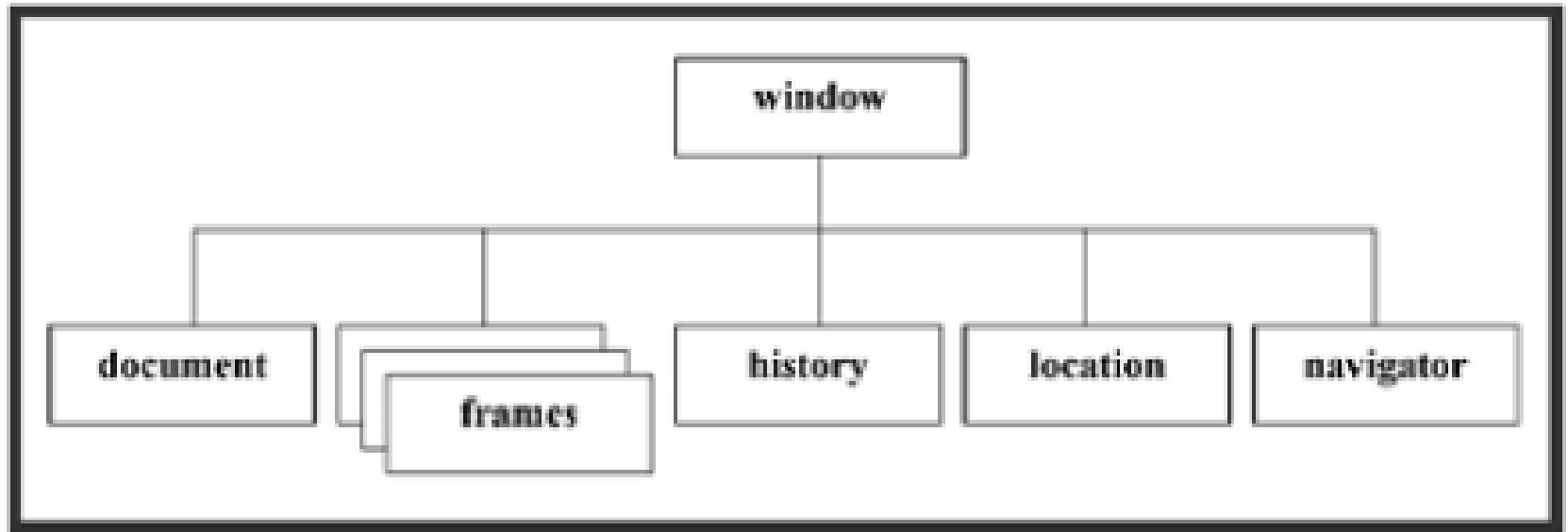
- As you can see, the document object resides at the top of this tree. Underneath it are the various HTML tags that define its contents, and under these tags is the content they contain.

# A Brief Overview of Browser Object Models

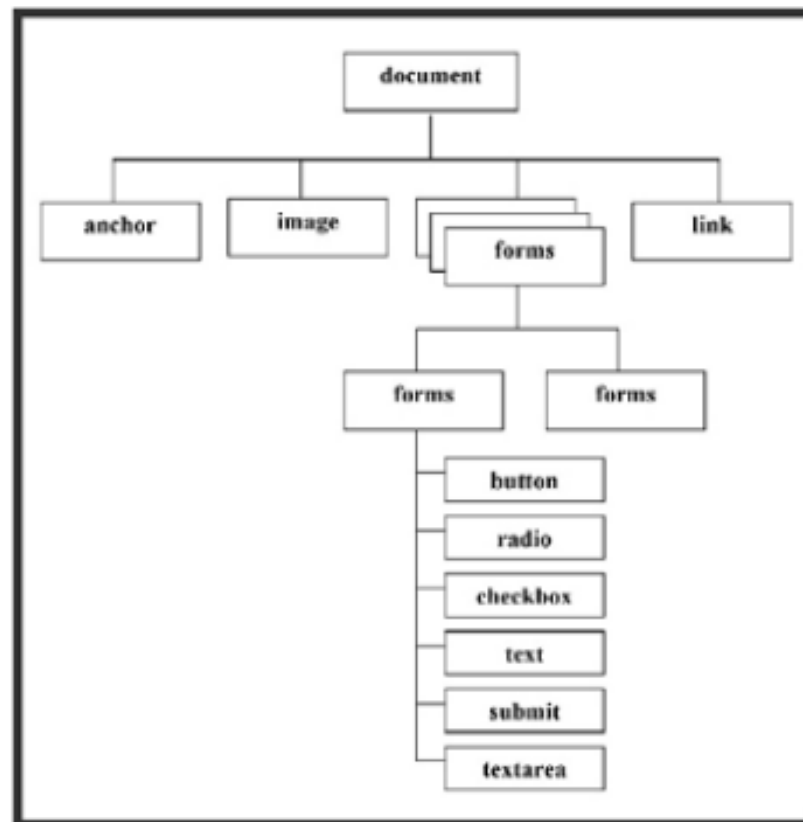
- In an effort to help bring stability to Web development, a group known as the World Wide Web Consortium developed a browser object standard called the Document Object Model (DOM). This model defines every element on an HTML page and provides the capability to access it. Both Microsoft and Netscape are working toward fully integrated support for the DOM in their browsers.

- The uppermost object supported by older browser object models is the window object. This object provides access to a number of lower-level or child objects, as listed below.
  - . **document object**- Provides the capability to access and manipulate the currently loaded HTML page.
  - frames collection** - Provides access to an indexed list of the frames defined in the browser window.
  - history object** - Provides methods that can be used to navigate through the document object's history list (e.g. to Web pages previously visited).
  - location object** - Provides access to information about the current URL as well as the capability to load a new URL.
  - navigator object**- Provides access to information about the browser being used to access the HTML page.

# The window object provides access to multiple child objects



- Of all these objects, the one that you will use the most is the document object. The document object provides access to a large number of additional objects and collections, as depicted in Figure



# Working with the Document Object Model

- Table provides a complete listing of DOM properties that you can use within your JavaScripts in order to access browser objects. These properties provide comprehensive control over all the content located on an HTML page by providing you with the capability to travel up and down the tree without having to hard code in references to specific HTML elements (e.g. via the NAME attribute).



Property	Description
<code>firstChild</code>	The first child node belonging to an object
<code>lastChild</code>	An object's last child node
<code>childNodes</code>	A collection (e.g. array) of child objects belonging to an object
<code>parentNode</code>	An object's parent object
<code>nextSibling</code>	The child node following the previous child node in the tree
<code>prevSibling</code>	The child node that comes before the current child node
<code>nodeName</code>	The name assigned to an object's HTML tag
<code>nodeType</code>	Identifies the type of HTML element (tag, attribute, or text) associated with the object
<code>nodeValue</code>	Retrieves the value assigned to a text node
<code>data</code>	Retrieves the value for the specified text node
<code>specified</code>	Determines whether an attribute has been specified
<code>attributes</code>	A collection (e.g. array) made up of all an object's attributes

```
<HTML>
```

```
    <HEAD>
```

```
        <TITLE>Script 3.2 - DOM Navigation and Access  
Example</TITLE>
```

```
    </HEAD>
```

```
    <BODY ID="bodyTag">
```

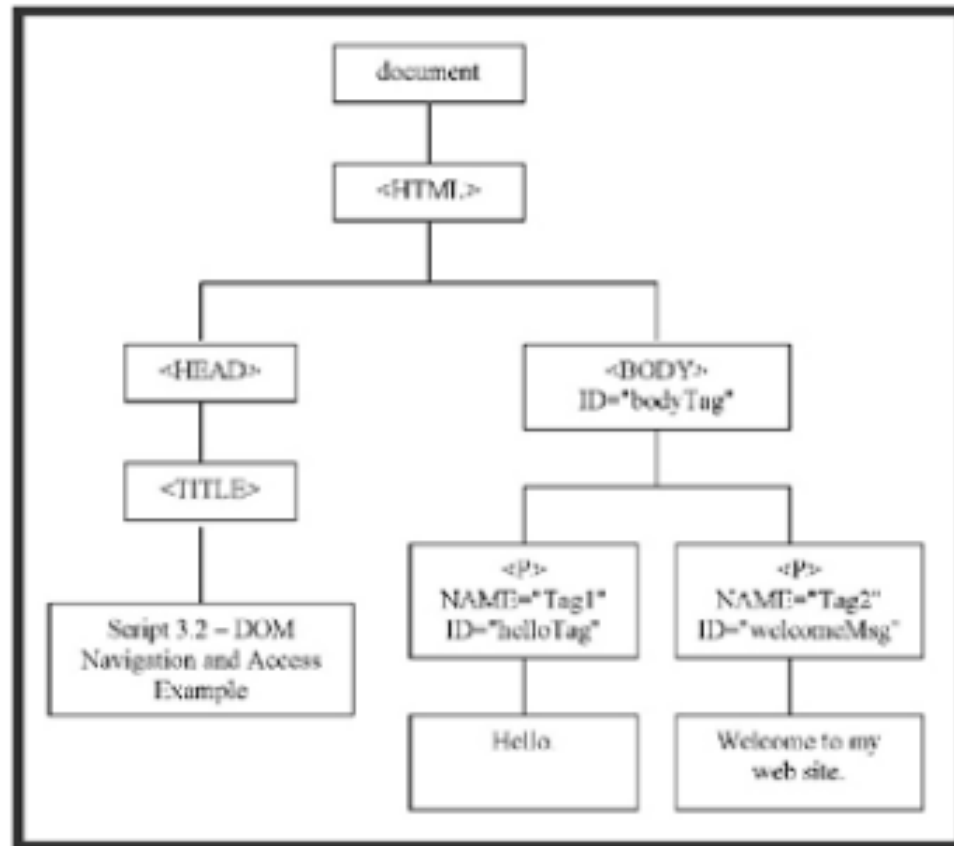
```
        <P ID="Tag1" Name="helloTag">Hello.</P>
```

```
        <p ID="Tag2" NAME="welcomeMsg"> Welcome to my Web  
site.</P>
```

```
    </BODY>
```

```
</HTML>
```

- Graphically, this script can be represented as shown in Figure. The document object sits at the root of the tree and has just one child object, documentElement. documentElement represents the HTML page's opening <html> tag.



```
<HTML>
```

```
  <HEAD>
```

```
    <TITLE>Script 3.3 - DOM Navigation and Access Example
```

```
</TITLE>
```

```
  </HEAD>
```

```
  <BODY ID="bodyTag">
```

```
    <P ID="Tag1" NAME="helloTag">Hello.</P>
```

```
    <P ID="Tag2" NAME="welcomeMsg"> Welcome to my Web  
site.</P>
```

```
  <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
```

```
    <!-- Start hiding JavaScript statements
```

```
      window.alert("The ID assigned to the 1st HTML tag is: " +  
        document.documentElement.firstChild.tagName);
```

```
      window.alert("The ID assigned to the 2nd HTML tag is: " +  
        document.documentElement.lastChild.tagName);
```

```
      window.alert("The ID assigned to the child of the 1st
```

HTML tag is: " +

```
document.documentElement.firstChild.firstChild.tagName);  
    window.alert("Click on OK to dynamically modify the  
value of Tag2");
```

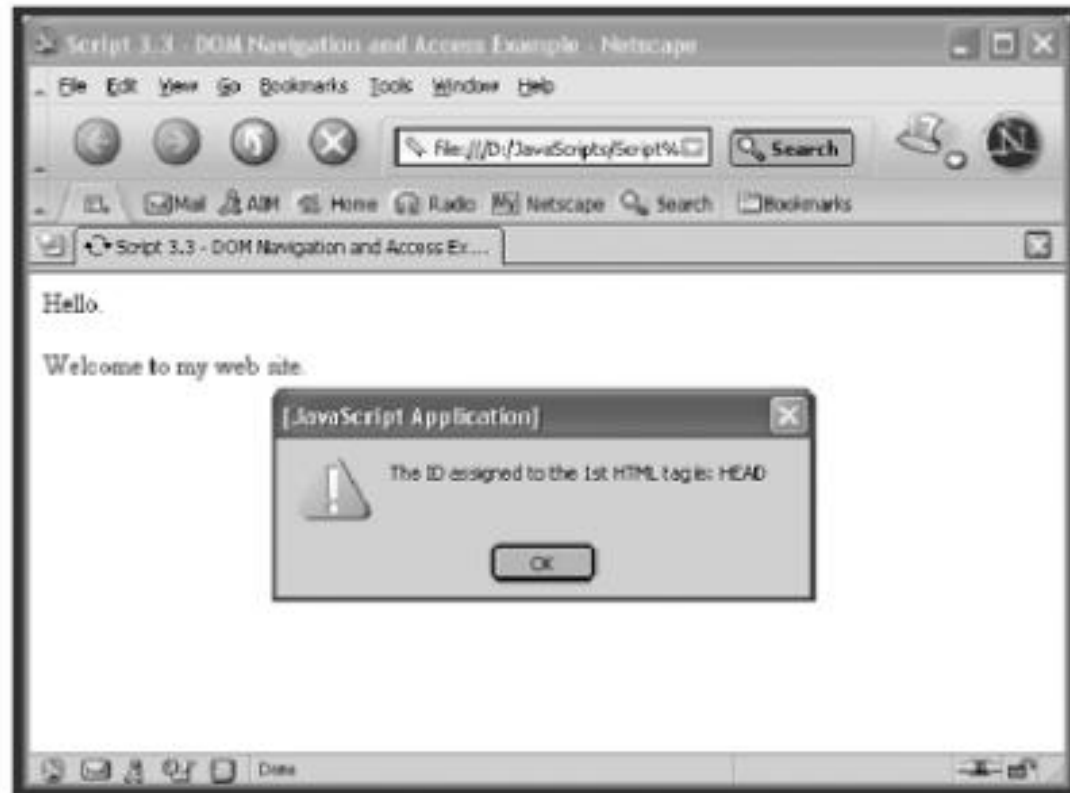
```
document.getElementById("Tag2").firstChild.nodeValue="Welcome  
to " +  
    "my new and improved Web site.";
```

```
    // End hiding JavaScript statements -->  
</SCRIPT>
```

```
</BODY>
```

```
</HTML>
```

- The first JavaScript statement displays the tag name of the first HTML tag that exists under the document object (e.g. the tag), as shown in Figure



- Take a look at the tree hierarchy shown in Figure and compare it to the components that make up this statement. Next, the JavaScript uses the `lastChild` property to reference the tag. Then the script references the tag, which happens to be the first tag located under the first tag in the script. The next statement displays a message stating that the script is about to rewrite the value assigned to the HTML page's second tag. Finally, the last statement in the script, shown below, rewrites a portion of the text displayed on the screen. It accomplishes this task by using the `GetElementById()` method to establish a reference to `Tag2`. This establishes a reference to the HTML page's second tag. The statement then references the `nodeValue` of the tag's `firstChild` (e.g. the tag's content) and assigns it a new value.

- Figure shows how the HTML page looks after its content has been changed dynamically.



The DOM provides the capability to reference any element located on an HTML page. It provides the capability to navigate the HTML page without having to reference hard coded tag names and to modify content dynamically. However, the price for this flexibility is complexity.



# The window Object

- The window object is the ancestor of all other objects on the page. Multiple windows can be opened at the same time. The window object has dozens of objects and methods associated with it, many of which you have already worked with. For example, you've used the **window.alert()** method to display messages in the alert dialog box, **window.prompt()** to retrieve user input, and **window.confirm()** to seek user permission before taking action.

- In the example, the browser opens a second window as soon as the Web page loads, using the window object's `open()` method. The window is named `window1`.

```
<HTML>
```

```
  <HEAD>
```

```
    <TITLE>Script 3.5 - Working with the Window Object - 2
```

```
</TITLE>
```

```
  </HEAD>
```

```
  <BODY>
```

```
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
```

```
    <!-- Start hiding JavaScript statements
```

```
      window1=window.open();
```

```
    // End hiding JavaScript statements -->
```

```
  </SCRIPT>
```

```
    <FORM NAME="form1">
```

```
      <INPUT TYPE="button" VALUE="Close the window"  
onClick="window1.close()">
```

```
      <INPUT TYPE="button" VALUE="Resize"  
onClick="window1.resizeTo(300,400)">
```

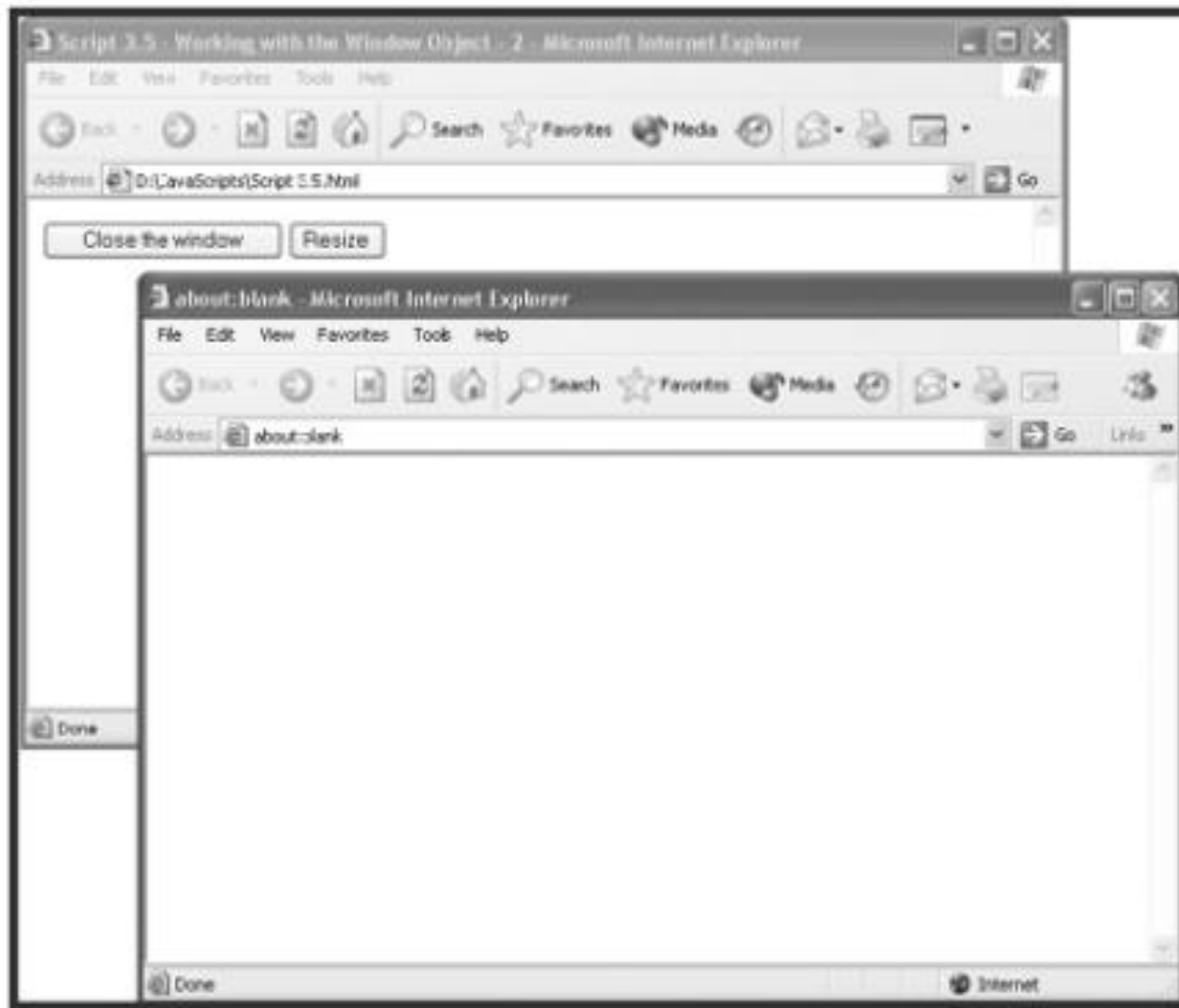
```
    </FORM>
```

```
  </BODY>
```

```
</HTML>
```

- The script contains a form that defines two buttons, each of which contains an onClick event handler. If you click the first button, labeled Close the Window, the window object's close() method is executed. In order for the browser to know which window you want to close, you must specify the window's name (in this case, it's window1).
- The second button in the form, Resize, executes the window object's resizeTo() method. This method tells the browser to change the size of the window1 window object to a pixel size of 300 by 400.

- Figure demonstrates what you'll see when you first load this page using Internet Explorer. Two windows are opened. The first window displays the form containing the script, and the second window is opened as a result of the script's `window1=window.open()` statement.



The window object's `open()` method enables you a great deal of control over the appearance of the window. For example, you can control whether the window has certain features such as a menu bar, toolbar, scroll bar, and status bar, as well as whether the window can be resized.

- Identifying major browser features



```
<HTML>
```

```
  <HEAD>
```

```
    <TITLE>Script 3.6 -Working with the Window Object - 3
```

```
  </TITLE>
```

```
  </HEAD>
```

```
  <BODY>
```

```
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
```

```
    <!-- Start hiding JavaScript statements
```

```
      window.open("", "Window1", "menubar=no,toolbar=yes," +
```

```
        "scrollbar=yes,resizable=yes,status=yes,
```

```
location=yes");
```

```
      // End hiding JavaScript statements -->
```

```
    </SCRIPT>
```

```
  </BODY>
```

By default, all the window features are enabled. To control them, you would rewrite the statement containing the open() method as follows:

```
window.open("", "Window1",
```

```
  "menubar=no,toolbar=yes,scrollbar=yes,resizable=yes,  
status=yes,location=yes");
```



- When written this way, the `open()` method accepts three sets of parameters.
- The first parameter is the URL that should be opened in the new window. In the preceding example, I left this parameter blank, causing the browser to open an empty window.
- The second parameter is the name you are assigning to the window. You must assign a name to the window so that you can reference it from other parts of your script.
- The final parameter is comprised of one or more arguments separated by commas. Each argument identifies a window feature and assigns it a value of yes or no. A value assignment of yes tells the browser to include that option when creating the window. A value of no tells the browser to eliminate the specified feature. By default, all features are enabled.

# The document Object

- The document object is the heart and soul of JavaScript. Each Web page can contain a single document object. You have already used it extensively to write output to the screen using the `document.write()` method.
- The properties of the document object depend on the content of the page's HTML. For example, if a Web page contains images, then the document properties for the page will contain an `images[]` array that lists every image on the page.

- The document object has a host of properties you can use to control the appearance of the page and to gather information about its contents. For example, the document object stores property values for the following arrays:
  - **anchors[]**. An array containing a list of all anchors in the document
  - **applets[]**. An array containing a list of all applets in the document
  - **embeds[]**. An array containing a list of all embedded objects in the document
  - **forms[]**. An array containing a list of all forms in the document
  - **images[]**. An array containing a list of all images in the document
  - **links[]**. An array containing a list of all links in the document
  - **plugins[]**. An array containing a list of all plug-ins in the document

- Other document object properties enable you to affect appearance:
  - **bgColor**. Specifies the document background color
  - **fgColor**. Specifies the color of document text
  - **linkColor**. Specifies the color of links
  - **alinkColor**. Specifies the color of active links
  - **vlinkColor**. Specifies the color of visited links

Here is a partial list of other useful document properties:

--**cookie** - Lets you get and set cookie values

--**lastModified** - A string that shows the date and time at which the document was last changed

--**referrer** - A string showing the URL the user came from

--**title** - A string containing the contents of the HTML tags

--**URL** - A string containing the document's URL

- The following example demonstrates the use of two document properties. The first statement in the script prints the title located in the HTML tags, using the `document.title` property. The second statement displays the last modification date and time for the page, using the `document.lastModified` property

```
<HTML>
```

```
    <HEAD>
```

```
        <TITLE>Script 3.7 - Setting the Modification Date and  
Time</TITLE>
```

```
    </HEAD>
```

```
    <BODY>
```

```
        <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
```

```
        <!-- Start hiding JavaScript statements
```

```
            document.write("<B>Document Title:</B> " +  
document.title + "<BR>");
```

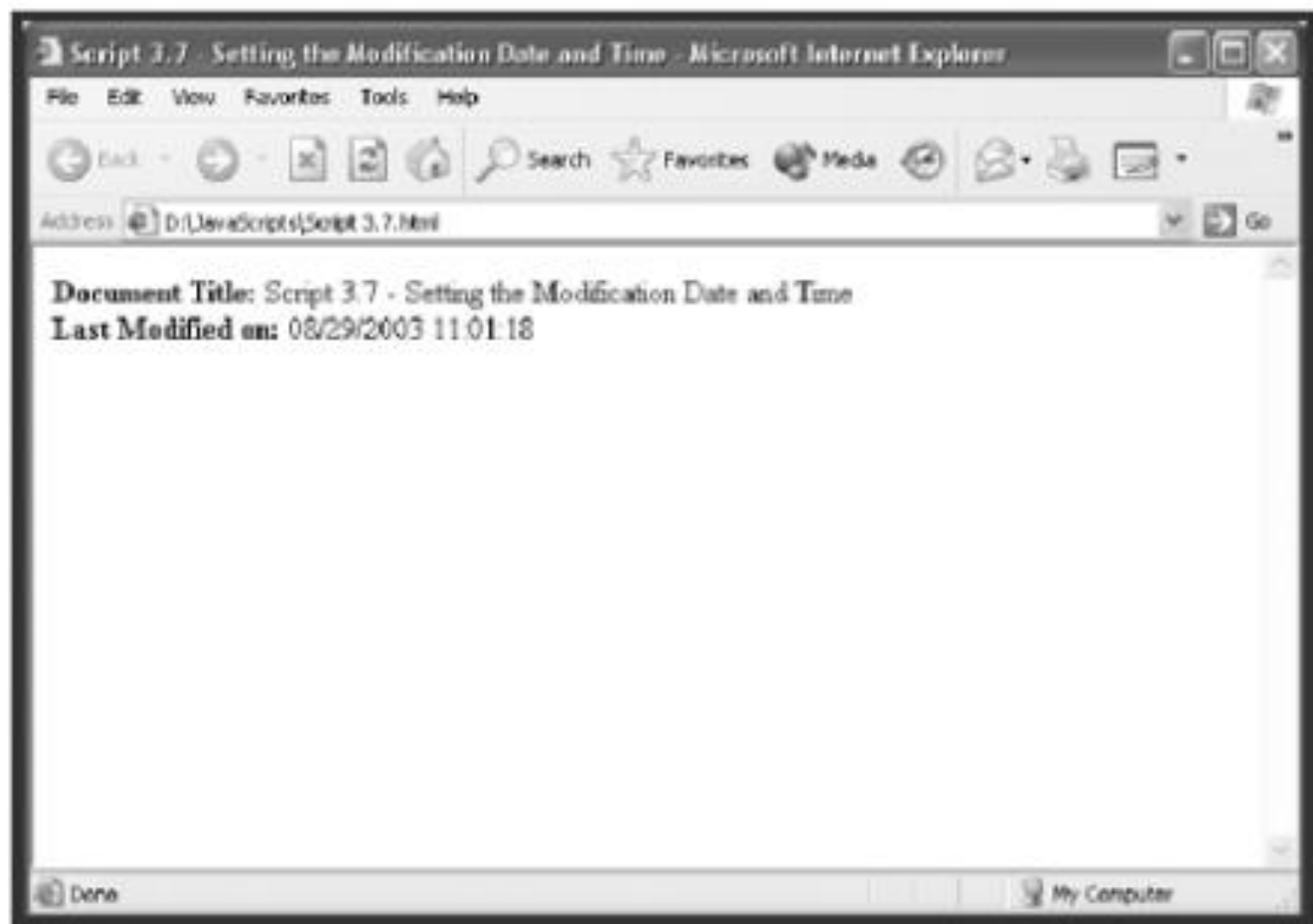
```
            document.write("<B>Last Modified on:</B> " +  
document.lastModified);
```

```
            // End hiding JavaScript statements -->
```

```
        </SCRIPT>
```

```
    </BODY>
```

```
</HTML>
```



# The form Object

- The document object's forms[] array maintains a list of every form on a page, starting with form[0] and incrementing by 1 for each additional form. You can refer to individual forms using their forms[] array index number or by name.
- For example, the following code creates a form called form1 that contains a single button:

```
<FORM NAME="form1">  
  <INPUT TYPE="button" NAME="backButton" VALUE="Back"  
onClick="window.back()" ">  
</FORM>
```



- Assuming that this is the only form on the page, you can refer to it either as `document.forms[0]` or as `document.form1`. Forms can contain many elements, such as text boxes and buttons. To manage these form elements, each form object contains its own array, named `elements[]`. The first element in the form is stored as `element[0]`, the second element is stored as `element[1]`, and so on.

# Few other important objects

- **The location Object** - The location object is relatively simple. Its properties contain information about its own URL. For example, `location.href` specifies the URL of the page, and `location.pathname` specifies just the path portion of the URL. If you make changes to the location object, the browser automatically attempts to load the URL again.

- **The history Object-** The history object maintains a list of URLs that the browser has visited since it began its current session. One useful property of this object is the length property, which you can access using the following syntax:

## **history.length**

The `history` object has three methods:

- The `back()` method loads the previously visited URL in the history list. This is the same as clicking on the browser's Back button.
- The `forward()` method loads the next URL in the history list. This has the same effect as clicking on the browser's Forward button.
- The `go()` method loads the specified URL in the history list. For example, `history.go(4)` loads the URL four entries ahead in the history list, `history.go(-4)` loads a URL four entries back in the history list, and `history.go(0)` reloads the current URL.

```
<HTML>
```

```
    <HEAD>
```

```
        <TITLE>Script 3.9 - Using History Object Methods for  
Navigation</TITLE>
```

```
    </HEAD>
```

```
    <BODY>
```

```
        <FORM NAME="form1">
```

```
            <INPUT TYPE="button" VALUE="Back"  
                onClick="history.back() ">
```

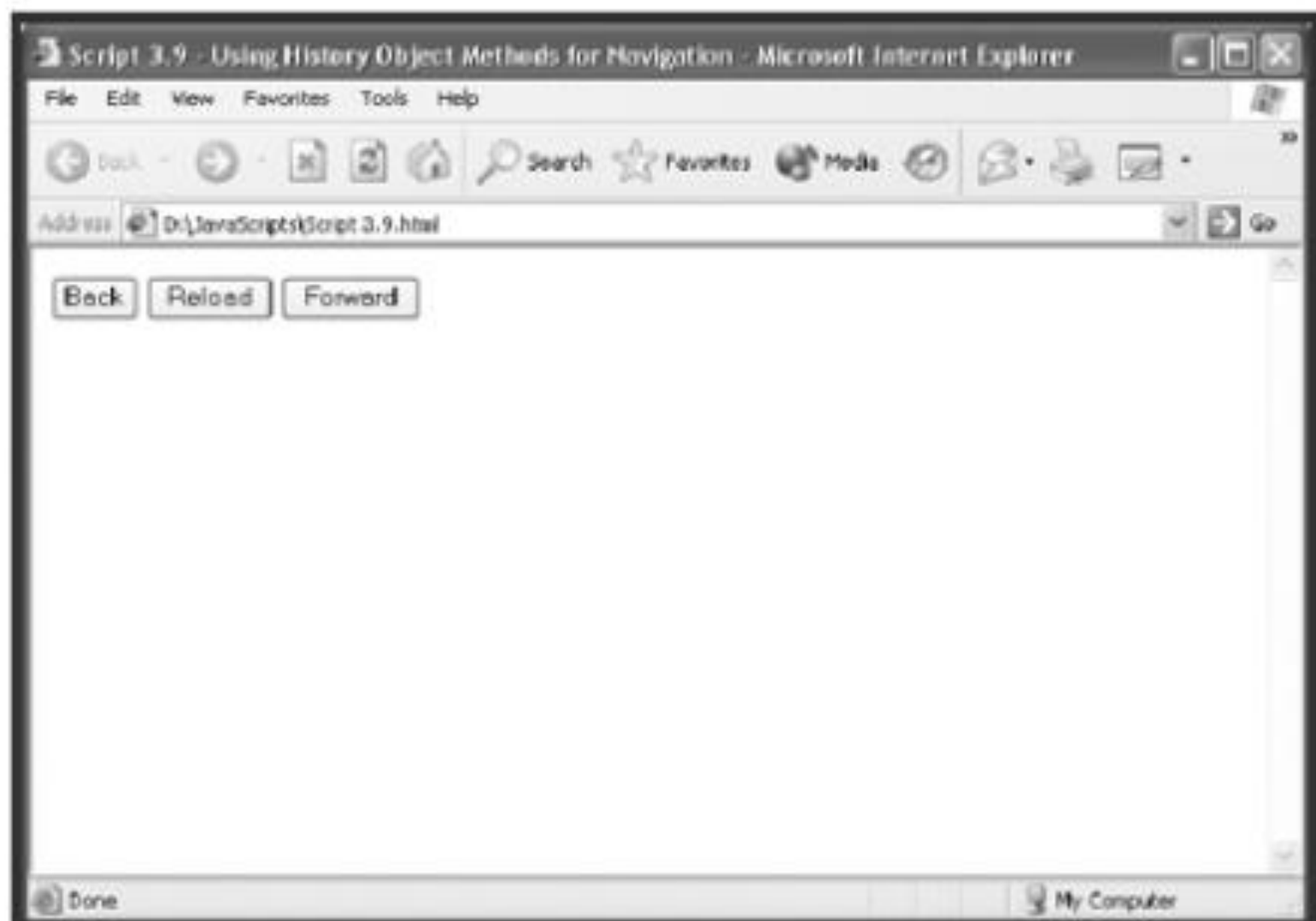
```
            <INPUT TYPE="button" VALUE="Reload"  
                onClick="history.go(0) ">
```

```
            <INPUT TYPE="button" VALUE="Forward"  
                onClick="history.forward() ">
```

```
        </FORM>
```

```
    </BODY>
```

```
</HTML>
```



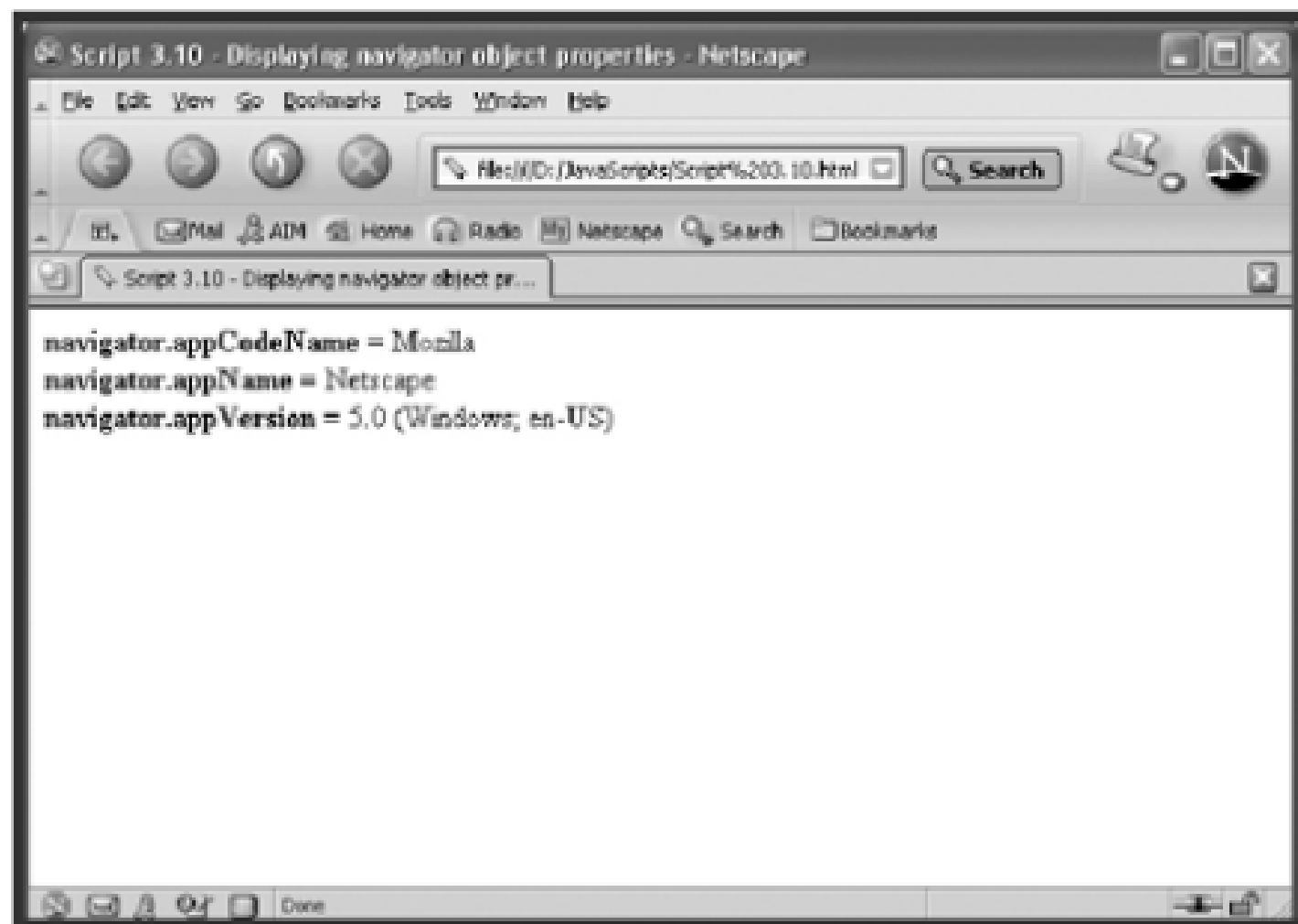
- **The navigator Object-** The navigator object is a top-level object like the window object. This means that the navigator object is not below any other object in the object hierarchy. It contains information about the browser that loaded the Web page. Some of the useful properties of the navigator object include the following:
  - `navigator.appCodeName`. Contains the code name of the browser
  - `navigator.appName`. Contains the name of the browser
  - `navigator.appVersion`. Contains the version number of the browser

```
<HTML>

  <HEAD>
    <TITLE>Script 3.10 - Displaying navigator object properties
  </TITLE>
  </HEAD>

  <BODY>
    <SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
  <!-- Start hiding JavaScript statements
      document.write("<B>navigator.appCodeName = </B>" +
navigator.appCodeName + "<BR>");
      document.write("<B>navigator.appName = </B>" + navigator.
appName + "<BR>");
      document.write("<B>navigator.appVersion = </B>" +
navigator.appVersion + "<BR>");
      // End hiding JavaScript statements -->
    </SCRIPT>
  </BODY>

</HTML>
```





# Creating Objects

- In addition to core JavaScript objects and browser-based objects, JavaScript provides you with the capability to build your own objects.
- Three basic steps are required to create an instance of an object:
  1. Define its structure
  2. Assign its properties
  3. Assign functions that will act as object methods

- There are two ways to create a new object. The first is to write a function that defines the format of the new object and then to use the new operator to instantiate a new object. For example, the following function defines a new object that contains two properties and one method:

```
function Customer(name, phone) {  
    this.name = name;  
    this.phone = phone;  
    this.ShowAlert = ShowAlert;  
}
```

```
function ShowAlert() {  
    window.alert("Customer: " + this.name + " - Phone: "  
+ this.phone)  
}
```

The first line defines a function that accepts two arguments. The next two lines use these arguments to establish property attributes. The last line assigns a method named `ShowAlert`. Notice that there is no associated argument with the defined method.

The `ShowAlert` method is itself just a function. As shown here, `ShowAlert()` displays an alert prompt that reveals the `name` and `phone` properties of a customer object:

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Script 3.11 - Creating a custom object</TITLE>
```

```
<SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
```

```
<!--Start hiding JavaScript statements
```

```
    function Customer(name, phone) {  
        this.name = name;  
        this.phone = phone;  
        this.ShowAlert = ShowAlert;  
    }
```

```
    function ShowAlert() {  
        window.alert("Customer: " + this.name + " - Phone: "  
+ this.phone);  
    }
```

```
// End hiding JavaScript statements -->  
</SCRIPT>
```

```
</HEAD>
```

```
<BODY>
```

```
<SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
```

```
<!--Start hiding JavaScript statements
```

```
    Customer1 = new Customer("Robert Robertson",  
8043334444);
```

```
    Customer1.ShowAlert();
```

```
// End hiding JavaScript statements -->
```

```
</SCRIPT>
```

```
</BODY>
```

```
</HTML>
```



## 2<sup>nd</sup> method to assign values to the defined object

<HTML>

<HEAD>

<TITLE>Script 3.12 - Another way to create a custom  
object</TITLE>

```
<SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
<!--Start hiding JavaScript statements
    function ShowAlert() {
        window.alert("Customer: " + this.name + " - Phone: "
+ this.phone);
    }
// End hiding JavaScript statements -->
</SCRIPT>
```

</HEAD>

<BODY>

```
<SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
<!--Start hiding JavaScript statements
    Customer1 = {name:"Robert Robertson", phone:8043334444,
ShowAlert:ShowAlert};
    Customer1.ShowAlert();
// End hiding JavaScript statements -->
</SCRIPT>
```

</BODY>

</HTML>

- **Deleting Your Object** -JavaScript provides a delete operator you can use to delete any objects you no longer want. You can add the following line of code to delete the Customer1 object you've just created:  
**delete Customer1;**
- **Expanding Object Definitions** -You can add additional properties to an object by any of three means:
  1. Rewrite the function to accommodate the new object property
  2. Use the prototype property to add an additional property to all instances of the object
  3. Add a new property to an individual object without affecting other objects

```
customer.prototype.nickname=null;
```

```
Customer1.nickname="Leeman";
```

- To add a new property to an individual object without adding the property to other instances of the object, type the name of the object, a period, and then the name of the new property and assign its value as shown in the following example. Here a new property, zipcode, has been created for the Customer1 object and assigned a value of 23116.

```
Customer1.zipcode = 23116;
```



# Handling Events

- **Defining Events** - In JavaScript, an event occurs within the confines of the browser. Events include such activities as mouse clicks, mouse movement, pressing keyboard keys, the opening and closing of windows and frames, and the resizing of windows. Browsers recognize events and perform default actions when those events occur.
- For example, when a user clicks on a link, the onClick event occurs, and the browser's default action is to load the Web page or image specified by the link.
- To make your Web pages really interactive, you must add another tool to your scripting skill set: event handlers.

- An event handler is a trap that recognizes the occurrence of a particular type of event. You can add code to your scripts that alters the browser's default response to events. For example, instead of automatically loading the URL specified by a link, you could display a confirmation dialog box that asks the user to agree to certain terms before proceeding.
- Each event is associated with a specific object. When an event occurs for a given object, its event handler executes (assuming that you have written one). For example, the click event occurs whenever a user clicks on a button, document, check box, link, radio option, reset button, or submit button.

- You place them within HTML tags that define the object. For example, you can define an event to occur whenever a Web page is loaded by placing an onLoad event handler inside the first HTML tag as shown here:

```
<BODY onLoad="window.alert('Web page loading: Complete.')">
```

In this example, an alert dialog appears when the page finishes loading. Notice that the event handler comes immediately after the tag and that its value is placed within quotation marks. You can use any JavaScript statement as the value for the event handler. You can even use multiple statements, provided that you separate them with semicolons. Alternatively, you can use a function call, which enables you to perform more complex actions in response to the event.

- **The event Object** -The event object is populated on every occurrence of an event. The information in its properties can be referenced by event handlers, giving your script access to detailed information about the event.
- **Types of Events** - JavaScript currently supports 24 different events and event handlers. These events can be broadly divided into a few categories:
  1. Window and frame events
  2. Mouse events
  3. Keyboard events
  4. Error events

A number of these events and their associated event handlers are demonstrated in the scripts that follow.

# Window and Frame Events

- Events that affect window and frame objects include the load, resize, unload, and move events. The event handlers for these events are `onLoad`, `onResize`, `onUnload`, and `onMove`. These event handlers are placed inside the tag. The following script uses the `alert()` method to demonstrate how to execute JavaScript statements in response to occurrences of these events:

```
<HTML>
```

```
  <HEAD>
```

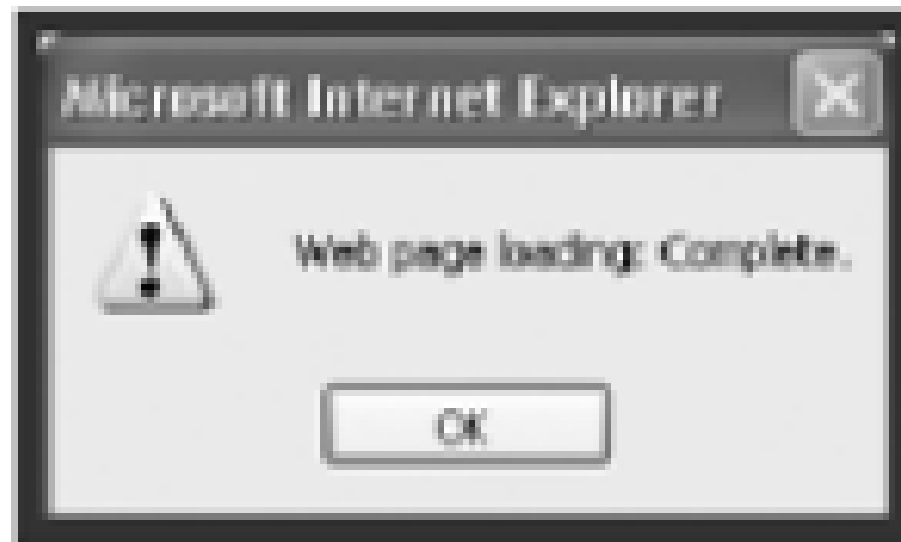
```
    <TITLE>Script 3.13 - onLoad, onResize, onUnload & onMove  
Example</TITLE>
```

```
  </HEAD>
```

```
  <BODY onLoad="window.alert('Web page loading: Complete.')"  
    onResize="window.alert('What is the matter with my current  
size?')"  
    onUnload="window.alert('Oh no, I am melting.....')"  
  </BODY>
```

```
</HTML>
```

- The first thing you will see when you run this script is a prompt notifying you that the Web page has finished loading. This message is triggered when the onLoad event handler executes in response to the load event.
- If you resize the window, the reload event will cause the onReload event handler to execute and display an alert message. Likewise, resizing the window results in a similar alert message. The unLoad event handler does not execute until you close the window or load it with another URL.





# Mouse Events

- Mouse events execute whenever you do something with the mouse. This includes any of the following:
  - The `MouseOver` event occurs when the pointer is moved over an object.
  - The `MouseOut` event occurs when the pointer is moved off an object.
  - The `MouseDown` event occurs when a mouse button is pressed.
  - The `MouseUp` event occurs when a mouse button is released.
  - The `MouseMove` event occurs whenever the mouse is moved.
  - The `Click` event occurs whenever you single-click on an object.
  - The `DblClick` event occurs whenever you double-click on an object.

- The following script demonstrates the use of the `onMouseOver/onMouseOut` and `onMouseDown/onMouseUp` events. This script defines two links. Clicking on either link instructs the browser to load the Web page at `www.microsoft.com`. By adding the `onMouseOver` and `onMouseOut` event handlers to the first HTML link tag, I instructed the browser to change the document's background to red when the pointer passes over the link. The `onMouseOut` event handler then changes the background to white when the pointer moves off the link.

- The onMouseDown and onMouseUp event handlers associated with the second link instruct the browser to change the document's background to red when the user clicks on the link (that is, when the user presses down on the mouse button) and to change the background color to white when the user releases the mouse button. Neither the onMouseDown nor the onMouseUp event handler alters the default action of the link. Therefore, if you click on the second link, the Web page at [www.microsoft.com](http://www.microsoft.com) is still loaded.

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Script 3.14  Mouse Event Handler Example</TITLE>
```

```
</HEAD>
```

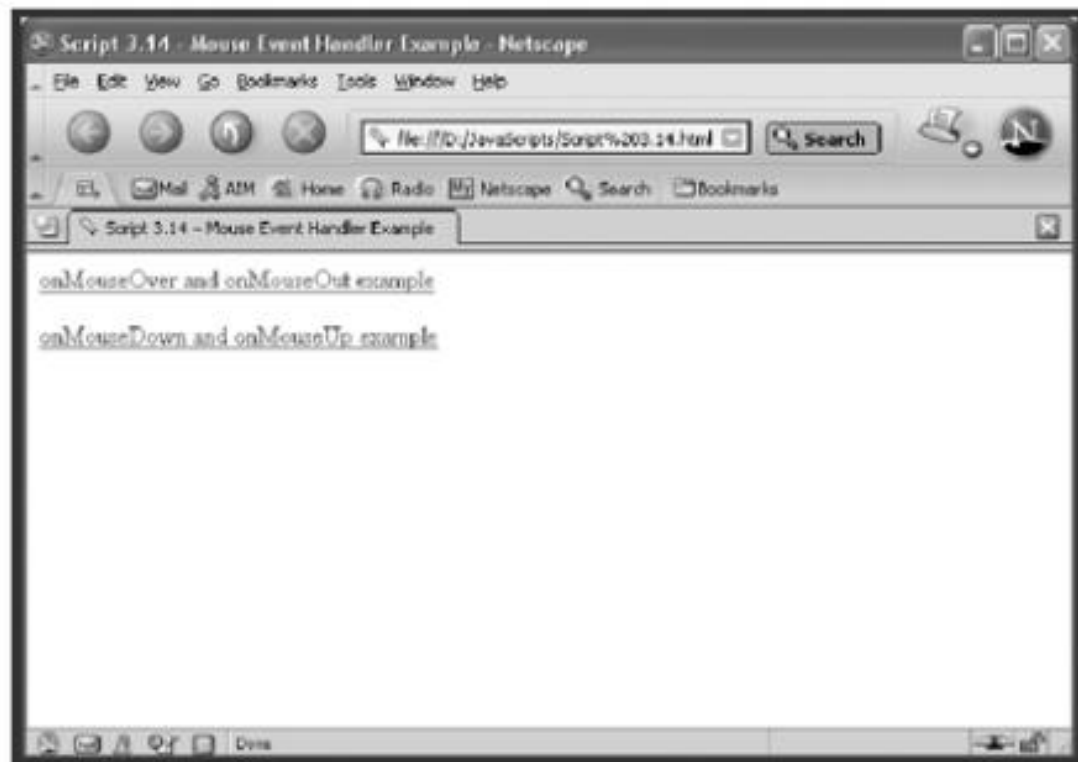
```
<BODY>
```

```
<A HREF="http://www.microsoft.com"  
  onMouseOver='document.bgColor="red"';  
  onMouseOut='document.bgColor="white"';>  
  onMouseOver and onMouseOut example</A><P>
```

```
<A HREF="http://www.microsoft.com"  
  onMouseDown='document.bgColor="red"';  
  onMouseUp='document.bgColor="white"';>  
  onMouseDown and onMouseUp example</A><P>
```

```
</BODY>
```

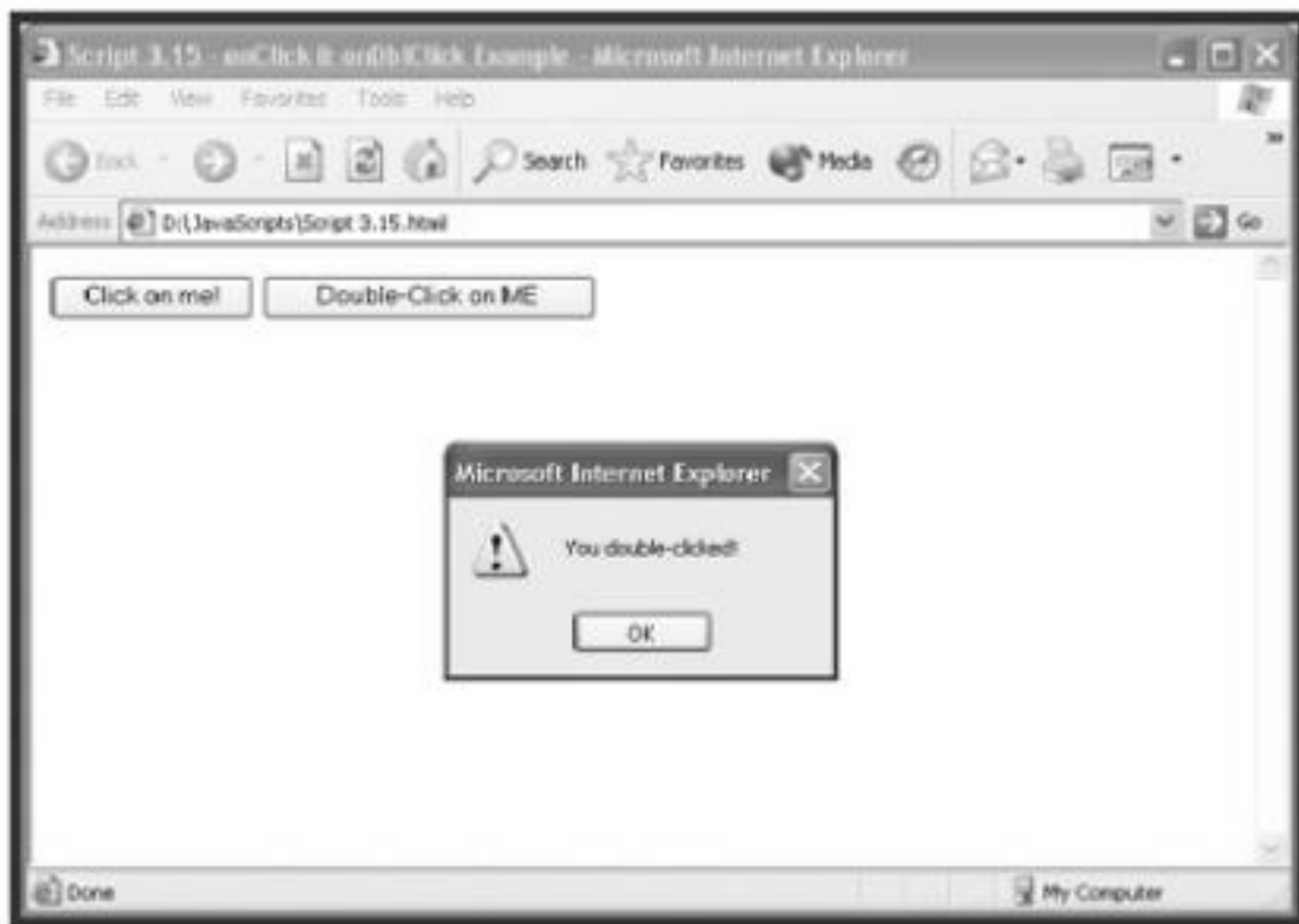
```
</HTML>
```



```
<HTML>

  <HEAD>
    <TITLE>Script 3.15 - onClick & onDb1Click Example</TITLE>
  </HEAD>
  <BODY>
    <FORM>
      <INPUT TYPE="button" VALUE="Click on me!"
        onClick="window.alert('You single-clicked.')">
      <INPUT TYPE="button" VALUE="Double-Click on ME"
        onDb1Click="window.alert('You double-clicked!')">
    </FORM>
  </BODY>

</HTML>
```



# Keyboard Events

- Keyboard events are like mouse events in that they occur whenever the user presses or releases a keyboard key. There are three keyboard events: `KeyDown`, `KeyUp`, and `KeyPress`. The following example demonstrates how you can use the `onKeyDown` event handler to trap keyboard information from Netscape Navigator.



<HTML>

<HEAD>

<TITLE>Script 3.16 - onKeyDown Example</TITLE>

<SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">

<!-- Start hiding JavaScript statements

var totalKeyStrokes = 0;

// End hiding JavaScript statements -->

</SCRIPT>

<SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">

<!-- Start hiding JavaScript statements

function CountKeyStrokes() {

totalKeyStrokes = ++totalKeyStrokes;

}

// End hiding JavaScript statements -->

</SCRIPT>

</HEAD>

<BODY>

<P>Type a few characters and then click on the  
button:</P>

<FORM>

<TEXTAREA ROWS="4" COLS="70"  
onKeyDown="CountKeyStrokes()"></TEXTAREA>  
<P></P>  
<INPUT TYPE="button" VALUE="Count Keystrokes" +  
onClick="window.alert('Total # of keystrokes = ' +  
totalKeyStrokes) ">  
</FORM>

</BODY>

</HTML>

- The script defines a simple form that consists of two elements: a TEXTAREA field and a button. The *onKeyDown* event handler is used to keep track of the total number of keystrokes typed by the user in the TEXTAREA field. The onKeyDown event is set to trigger each time the user types a new keystroke. It calls the CountKeyStrokes() function, which adds 1 to a variable called totalKeyStrokes. This variable stores a running total of the current number of keystrokes made by the user. When the user clicks on the form's button, a pop-up dialog is displayed that shows the total number of keystrokes entered so far by the user.



# Error Events

- An error event occurs whenever your JavaScripts run into an error. Error events automatically trigger the *onerror* event. By adding an *onerror* event handler to your scripts, you can intercept these errors and suppress them. After they are suppressed, you then can either attempt to fix them programmatically or display a customized error message.

Unlike other event handlers, the *onerror* event handler is spelled in all lowercase.

- The *onerror* event handler automatically receives three arguments when it is triggered: the error message itself, the URL of the Web page, and the line number in the script where the error occurred. You can use the **onerror** event handler to respond to the error. For example, you can display an alert prompt, redirect the user to another Web page, call a function, advise the user to upgrade the browser or get a specific plug-in, and so on. The *onerror* event handler works a little differently than other event handlers and has the following syntax.

```
window.onerror=FunctionName
```

- When used, the *onerror* event handler is set up in the head section of the HTML page, along with its associated function.
- The following script demonstrates how to use the *onerror* event handler to display error information using a function called ErrorTrap(). This function accepts three arguments that map to the three arguments passed to the *onerror* event handler. The function uses these arguments to format three lines of text that present the error information. To produce an error on the page, I deliberately added an s to the end of the word window in the HTML page's body tag. windows is a not a valid browser object.

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Script 3.17 - onError Example</TITLE>
```

```
<SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">
```

```
<!--Start hiding JavaScript statements
```

```
    function ErrorTrap(msg,url,line_no) {  
        document.write("<P>An error has occurred in this  
script.</P>");  
        document.write("Error = " + msg + " on line " +  
line_no + "<BR>");  
        document.write("URL = " + url + "<BR>");  
        return true;  
    }
```

```
    onerror=ErrorTrap;
```

```
    // End hiding JavaScript statements -->
```

```
</SCRIPT>
```

```
</HEAD>
```

```
<BODY onLoad="windows.alert('Hi')">
```

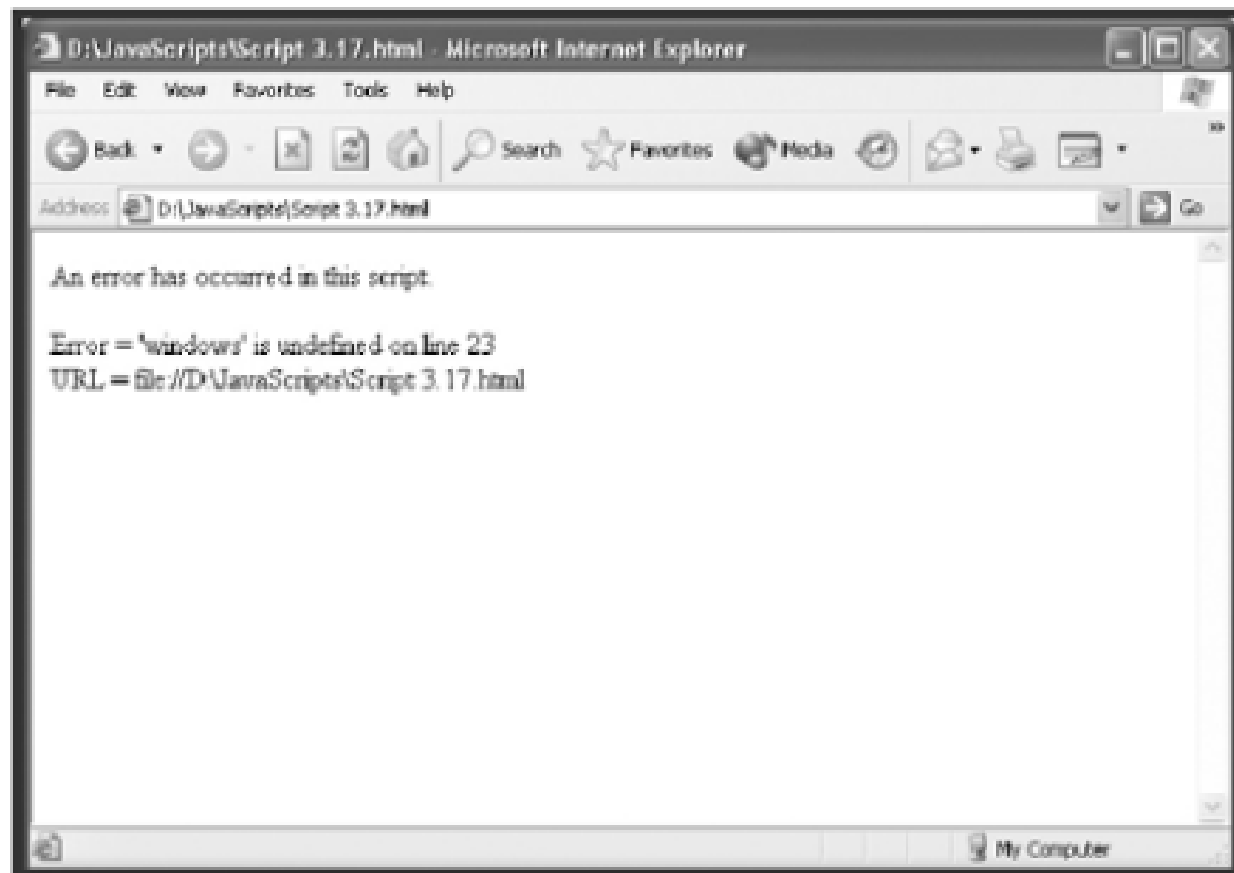
```
    Welcome to my Web page.
```

```
</BODY>
```

```
</HTML>
```



Figure shows the results of loading this script using Internet Explorer. As you can see, the message written to the window tells you what the error was, where in the Web page the error occurred, and where the page is located.



- The onerror event handler can also be used within HTML tags. By placing the onerror event handler within a particular HTML tag, you can define different actions for error events on an object-by-object basis. For example, the following JavaScript uses this technique to create an error message in the event that the specified graphic file cannot be found.

```
<HTML>
```

```
  <HEAD>
```

```
    <TITLE>Script 3.18 - Another onError Example</TITLE>
```

```
  </HEAD>
```

```
  <BODY>
```

```
    <IMG NAME="xxxx" SRC="xxxx.jpg"
```

```
    onError="window.alert('Unable to load image!')">
```

```
  </BODY>
```

```
</HTML>
```

```
<HEAD>
  <TITLE>Script 3.19 - Age Verification Example</TITLE>
</HEAD>

<BODY>
  <A HREF="http://www.ptpmagazine.com"
    onClick="return(window.confirm('You must be older than
30 to visit ' +
    'this site. Are you?'))"> ptpmagazine.com</A>
</BODY>

</HTML>
```

# Using JavaScript to Manage Forms

- Within the context of an HTML page, a form is created using the tags. Within the tags, individual form elements such as check boxes or text fields are defined.
- You can use the power of forms on your Web pages without using JavaScript. However, JavaScript provides powerful validation capabilities and enables you to interact with visitors as they fill out the forms.

The browser creates a `form` object for every form element defined by `<FORM>` tags. You can assign a name to each element using the optional `NAME=""` attribute for each element, or you can use the `forms[]` array, which contains an index listing of every form element on the page, beginning with an index of 0. In addition, each form contains its own `elements[]` array that contains an entry for each form element on that particular form. For example, the fourth form element on a form called `myForm` can be referenced as `myForm.elements[3]`.

# Form Properties and Methods

- The form object has a number of properties, many of which are objects in their own right. You can use these properties, listed below, to access and manipulate form contents.
- **action**. Created based on the `<FORM>` tag's `ACTION` attribute.
- **button**. An object that users can click to initiate an action.
- **checkbox**. An object that enables users to select or clear an option by clicking on it.
- **elements[]**. An array of all the form elements.
- **encoding**. Created based on the `<FORM>` tag's `ENCTYPE` attribute.
- **fileUpload**. An object used to specify a file that can be included as form data.
- **hidden**. An object that does not appear on the form but that you can programmatically fill with data.

- **length.** A property that specifies the number of elements in the `forms[]` array.
- **method.** Created based on the `<FORM>` tag's `METHOD` attribute.
- **name.** Created based on the `<FORM>` tag's `NAME` attribute.
- **password.** An object that masks any text that is typed into it.
- **radio.** An object that is organized into a collection of similar objects, only one of which can be selected.
- **reset.** A button object that enables you to clear and reset a form to its default values.
- **select.** An object that provides a list of selectable options.
- **submit.** A button object that enables you to submit the form for processing.
- **target.** Created based on the `<FORM>` tag's `TARGET` attribute.
- **text.** An object used to hold a single line of text.
- **textarea.** An object similar to the `text` object except that it accepts multiple lines of text.

The `form` object also has several methods that can be applied to it:



- **handleEvent()**. Enables you to simulate a specific event on the specified object such as `click` or `dblclick` events.
- **reset()**. Enables you to clear and reset a form to its default values programmatically.
- **submit()**. Enables you to submit a form programmatically for processing.

# Form Events

- In addition to event handlers supplied by individual form elements, the form object provides the following two event handlers:
  - The onReset event handler enables you to execute JavaScript statements or call a function before the form is reset.
  - The onSubmit event handler enables you to execute JavaScript statements or call a function before the form is submitted.

- These event handlers enable you to execute commands or call subroutines before performing the reset or submit actions requested by the visitor. As the examples that follow will demonstrate, this provides you with the opportunity to create warning messages that inform the visitor of the consequences of his actions or to validate form contents and assist the visitor in properly filling out the form, while at the same time making sure only valid data is provided.

- The following list identifies which events are associated with each form element and can be used as a reference to see what events each form element enables you to use when you are developing your forms:

Event	Form Elements Affected
Click	button, checkbox, radio, reset, submit
Blur	button, checkbox, fileUpload, password, radio, reset, select, submit, text, textarea
Focus	button, checkbox, fileUpload, password, radio, reset, select, submit, text, textarea
Select	fileUpload, password, text, textarea

# Other form Related Methods

- Most form elements provide a number of methods you can use to work with the elements programmatically. These methods enable you to invoke events without depending on the visitor. For example, you might create a form that includes several text fields and a reset button. In the event that the user decides to click on the reset button and start filling out the form all over again, you could add the `focus()` method to the reset button's `onClick` event and assist the visitor by placing the cursor back in the first text field. A list of methods supported by form elements includes:

- The `click()` method has the same affect as though the user clicked on the object.
- The `blur()` method moves the focus away from the object.
- The `focus()` method places the focus on the object.
- The `select()` method highlights the text in the form elements that contain text.

# Form Validation

- The following example creates a form for a bicycle shop that enables Internet customers to place online orders for new bikes. Customers are given a list of options to choose from and are asked to specify what method of payment they plan to use when they pick up their new bikes.

<HTML>

<HEAD>

<TITLE>Script 3.35- A form validation example</TITLE>

<SCRIPT LANGUAGE="JavaScript" TYPE="Text/JavaScript">

<!-- Start hiding JavaScript statements

```
function ValidateOrder() {  
    if (document.myForm.firstName.value.length < 1) {  
        window.alert("Missing First name! Please correct");  
        return;  
    }  
    if (document.myForm.lastName.value.length < 1) {  
        window.alert("Missing Last name! Please correct");  
        return;  
    }  
    if (document.myForm.address.value.length < 1) {  
        window.alert("Missing Address! Please correct");  
        return;  
    }  
}
```



```
option_selected = "no";
for (i=0; i < document.myForm.radio_option1.length; i++) {
    if (document.myForm.radio_option1[i].checked) {
        option_selected = "yes";
    }
}
if (option_selected == "no") {
    window.alert("Please specify a bike model!");
    return;
}
window.alert("Your order looks good. Click on Submit!");
}
```

```
// End hiding JavaScript statements -->
</SCRIPT>

</HEAD>

<BODY>
  <CENTER>
    <H2>Welcome to Jerry's Custom Bike Shop</H2>
    <H5>Payment due at pickup</H5>
  </CENTER>
  <FORM NAME="myForm" METHOD="post" ACTION="cgi-bin/myscript">
    <B>First name:</B> <INPUT NAME="firstName" TYPE="text"
SIZE="15"
    MAXLENGTH="20"><BR>
    <B>Last name:</B> <INPUT NAME="lastName" TYPE="text"
SIZE="15"
    MAXLENGTH="20">
    <P><B>Mailing Address:</B> <INPUT NAME="address"
TYPE="text"
    SIZE="30" MAXLENGTH="50"></P>
```

```
<P><B>Select the bike you wish to order:</B></P>
  10 Speed Deluxe: <INPUT NAME="radio_option1"
TYPE="radio"
  VALUE="10Speed"><BR>
  15 Speed Racer: <INPUT NAME="radio_option1"
TYPE="radio"
  VALUE="15Speed"><BR>
<P><B>Additional Comments: (Optional)</B></P>
<TEXTAREA NAME="myTextarea" TYPE="textarea" ROWS="4"
  COLS="40"></TEXTAREA>
<P><B>Please specify method of payment:</B>
<SELECT NAME="myList">
  <OPTION SELECTED VALUE="check">Personal Check
  <OPTION VALUE="creditCard">Credit Card
  <OPTION VALUE="moneyOrder">Money Order
</SELECT></P>
<INPUT TYPE="reset" VALUE="Reset Form">
<INPUT TYPE="submit" VALUE="Submit Order"
onClick="window.alert('Your' +
  ' bike will be ready in 5 days')">
  <INPUT TYPE="button" VALUE="Validate Order"
onClick="ValidateOrder()">
  </FORM>
</BODY>

</HTML>
```

- You can always use e-mail to retrieve the contents of your forms. The trick to making this work is to modify the tag as shown here:

```
<FORM NAME="myForm" ACTION="mailto:jl1f04@yahoo.com" ENCTYPE="text/plain">
```

Script 3.35- A form validation example - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media Print Mail

Address D:\JavaScripts\Script 3.35.html Go

## Welcome to Jerry's Custom Bike Shop

**Payment due at pickup**

First name:

Last name:

Mailing Address:

Select the bike you wish to order:

10 Speed Deluxe: ☐

15 Speed Racer: ☐

Additional Comments: (Optional)

Please specify method of payment:

Done My Computer