UNIT 4   INTRODUCTION TO JAVASCRIPT

4.0   INTRODUCTION

JavaScript is the programming language of the Web. It is used mainly for validating forms. JavaScript and Java can be related to each other. There exist many other differences between the two. The client interprets JavaScript, whereas in Java, one can execute a Java file only after compiling it. JavaScript is based on an object model. In this unit you will learn how to write JavaScript code and insert them into your HTML documents, and how to make your pages more dynamic and interactive.

Besides basic programming constructs and concepts, you will also learn about Object-based programming in JavaScript. We will also discuss some commonly used objects, message boxes and forms. One of the most important parts of JavaScript is Event handling that will allow you to write Event-driven code.

4.1   OBJECTIVES

After going through this unit you would be able to learn and use the following features of the JavaScript:

- Operators;
- Loop constructs;
- Functions;
- Objects such as Math object, Date object;
- Input and output boxes;

- Event handlers;
- Form object; and
- Form array.

4.2   JAVASCRIPT VARIABLES AND DATATYPES

Let us first see the skeleton of a JavaScript file.

```
<HTML>
  <HEAD>
    <TITLE>IGNOU </TITLE>
    <SCRIPT LANGUAGE = "JavaScript">
    </SCRIPT>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

JavaScript code should be written between the <SCRIPT> and </SCRIPT> tags. The value LANGUAGE = "JavaScript" indicates to the browser that Javascript code has been used in the HTML document. It is a good programming practice to include the Javascript code within the <HEAD> and </HEAD> tags.

Now let us start with variables. Variables store and retrieve data, also known as "values". A variable can refer to a value, which changes or is changed. Variables are referred to by name, although the name you give them must conform to certain rules. A JavaScript identifier, or name, must start with a letter or underscore ("_"); subsequent characters can also be digits (0-9). Because JavaScript is case sensitive, letters include the characters "A" through "Z" (uppercase) and the characters "a" through "z" (lowercase). Typically, variable names are chosen to be meaningful and related to the value they hold. For example, a good variable name for containing the total price of goods orders would be *total_price*.

4.2.1   Declaring Variables

You can declare a variable with the var statement:

var strname = *some value*

You can also declare a variable by simply assigning a value to the variable. But if you do not assign a value and simply use the variable then it leads to an error.
Strname = *some value*

You assign a value to a variable like this:

var strname = "Hello"

Or like this:

strname = "Hello"

The variable name is on the left hand side of the expression and the value you want to assign to the variable is on to the right side. Thus the variable "strname" shown above gets the value "Hello" assigned to it.

Life span of variables

When you declare a variable within a function, the variable can be accessed only

within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.

If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

4.2.2 Data Types

A value, the data assigned to a variable, may consist of any sort of data. However, JavaScript considers data to fall into several possible *types*. Depending on the type of data, certain operations may or may not be allowed on the values. For example, you cannot arithmetically multiply two string values. Variables can be of these types:

Data Types	Description
Number	3 or 7.987 are the examples of Integer and floating-point numbers. Integers can be positive, 0, or negative; Integers can be expressed in decimal (base 10), hexadecimal (base 16), and octal (base 8). A decimal integer literal consists of a sequence of digits without a leading 0 (zero). A leading 0 (zero) on an integer literal indicates it is in octal; a leading 0x (or 0X) indicates hexadecimal. Hexadecimal integers can include digits (0-9) and the letters a-f and A-F. Octal integers can include only the digits 0-7. A floating-point number can contain either a decimal fraction, an "e" (uppercase or lowercase), that is used to represent "ten to the power of" in scientific notation, or both. The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-"). A floating-point literal must have at least one digit and either a decimal point or "e" (or "E").
Boolean	True or False. The possible Boolean values are true and false. These are special values, and are not usable as 1 and 0. In a comparison, any expression that evaluates to 0 is taken to be false, and any statement that evaluates to a number other than 0 is taken to be true.
String	"Hello World!" Strings are delineated by single or double quotation marks. (Use single quotes to type strings that contain quotation marks.)
Object	MyObj = new Object();
Null	Not the same as zero – no value at all. A null value is one that has no value and means nothing.
Undefined	A value that is undefined is a value held by a variable after it has been created, but before a value has been assigned to it.

4.3 STATEMENTS AND OPERATORS

Assignment Operators		
Operator	Functionality	Example/Explanation
=	Sets one value equal to another	counter=0 Sets the counter to equal the number 0
+=	Shortcut for adding to the current value.	clicks += 2 Sets the variable named counter to equal the current value plus two.
-=	Shortcut for subtracting from the current value.	clicks -= 2 Sets the variable named counter to equal the current value minus two.
*=	Shortcut for multiplying the current value.	clicks *= 2 Sets the variable named counter to equal the current value multiplied by two.
/=	Shortcut for dividing the current value.	clicks /= 2 Sets the variable named counter to equal the current value divided by two.

Comparison Operators		

Description: The comparison operators compare two items and return a value of "true"if the condition evaluates to true, else they return false.

Operator	Functionality	Example/Explanation
==	Returns a true value if the items are the same	Counter == 10 Returns the value "true" if the counter's value is currently equal to the number 10
!=	Returns a true value if the items are **not** the same	Counter != 10 Returns the value "true" if the counter's value is any value except the number 10
>	Returns a true value if the item on the left is greater than the item on the right	counter>10 Returns the value "true" if the counter's value is larger than the number 10
>=	Returns a true value if the item on the left is equal to or greater than the item on the right	counter>=10 Returns the value "true" if the counter's value is equal to or larger than the number 10
<	Returns a true value if the item on the left is less than the item on the right	counter<10 Returns the value "true" if the counter's value is smaller than the number 10
<=	Returns a true value if the item on the left is equal to or less than the item on the right	counter<=10 Returns the value "true" if the counter's value is equal to or less than the number 10

Computational Operators		

Description: The computational operators perform a mathematical function on a value or values, and return a single value.

Operator	Functionality	Example/Explanation
+	Adds two values together	counter+2 Returns the sum of the counter plus 2
-	Subtracts one value from another	counter-2 Returns the sum of the counter minus 2
*	Multiplies two values	counter*10 Returns the result of the variable times 10
/	Divides the value on the left by the one on the right and returns the result	counter/2 Divides the current value of the counter by 2 and returns the result
++X	Increments the value, and then returns the result	++counter Looks at the current value of the counter, increments it by one, and then returns the result. If the counter has a value of 3, this expression returns the value of 4.
X++	Returns the value, and then increments the value	counter++ Returns the value of the counter, then increments the counter. If the counter has a value of 3, this expression returns the value of 3, then sets the counter value to 4.
--X	Decreases the value, and then returns the result	--counter Looks at the current value of the counter, decreases it by one, and then returns the result. If the counter has a value of 7, this expression returns the value of 6.
X--	Returns the value, and then decreases the value	counter-- Returns the value of the counter, then decreases the counter value. If the counter has a value of 7, this expression returns the value of 7, then sets the counter value to 6.

Logical Operators		
Description: The logical operators evaluate expressions and then return a true or false value based on the result.		
Operator	**Functionality**	**Example/ Explanation**
&&	Looks at two expressions and returns a value of "true" if the expressions on the left and right of the operator are both true	If day = 'friday' && date=13 then alert("Are You Superstitious?") Compares the value of the day and the value of the date. If it is true that today is a Friday and if it is also true that the date is the 13th, then an alert box pops up with the message "Are You Superstitious?"
||	Looks at two expressions and returns a value of "true" if either one -- but not both -- of the expressions are true.	if day='friday'&&date=13 then alert("Are You Superstitious?") else if day='friday'||date=13 then alert("Aren't you glad it isn't Friday the 13th?") Compares the value of the day and the value of the date. If it is true that today is a Friday and if it is also true that the date is the 13th, then an alert box pops up with the message "Are You Superstitious?" If both are not true, the script moves onto the next line of code... Which compares the value of the day and the value of the date. If either one -- but not both -- is true, then an alert box pops up with the message "Aren't you glad it isn't Friday the 13th?"

4.4   CONTROL STRUCTURES

JavaScript supports the usual control structures:

- The conditionals if, if...else, and switch;
- The iterations for, while, do...while, break, and continue;

4.4.1   Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have three conditional statements:

- if statement - use this statement if you want to execute a set of code when a condition is true
- if...else statement - use this statement if you want to select one of two sets of code to execute
- switch statement - use this statement if you want to select one of many sets of code to execute

```
if( myVariable == 2 ) {
   myVariable = 1;
} else {
   myVariable = 0;
}
```

If the value of myVariable in Figure 4.1 is 2 then the first condition evaluates to true and the value of myVariable is set to 1. If it is anything other than 2 then the else part gets executed.

Now let us see an example of a nested if statement in Figure 4.2.

```
if ( myVariable == 2 ) {
  myVariable = 1;
} else {
  If (myVariable == 5 ) {
    myVariable = 3;
  } else {
    myVariable = 4;
  }
}
```

Switch Statement

If there exist multiple conditions, the switch statement is recommended. This is because only one expression gets evaluated based on which control directly jumps to the respective case.

```
switch(myVar) {
  case 1:
    //if myVar is 1 this is executed
  case 'sample':
    //if myVar is 'sample' (or 1, see the next paragraph)
    //this is executed
  case false:
    //if myVar is false (or 1 or 'sample', see the next paragraph)
    //this is executed
  default:
    //if myVar does not satisfy any case, (or if it is
    //1 or 'sample' or false, see the next paragraph)
    //this is executed
}
```

As shown in Figure 4.3, depending on the value of "myvar", the statement of the respective case gets executed. If a case is satisfied, the code beyond that case will also be executed unless the break statement is used. In the above example, if myVar is 1, the code for case 'sample', case 'false' and 'default' will all be executed as well.

4.4.2 Loop Statements

A loop is a set of commands that executes repeatedly until a specified condition is met. JavaScript supports two loop statements: for and while. In addition, you can use the break and continue statements within loop statements. Another statement, for...in, executes statements repeatedly but is used for object manipulation.

• For Statement

A for loop repeats until a specified condition evaluates to false. The JavaScript for loop is similar to the Java and C for loops. A for statement looks as follows:

```
for ([initial-expression]; [condition]; [increment-expression]) {
  Statements
}
```

When a for loop executes, the following sequence of operations occur:

1. The initializing expression *initial-expression*, if any, is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity.
2. The *condition* expression is evaluated. If the value of *condition* is true, the loop

statements execute. If the value of *condition* is false, the loop terminates.

3. The update expression *increment-expression* executes.
4. The *statements* get executed, and control returns to step 2. Actually the syntax provides for a single statement; when enclosed in braces '{' and '}', any number of statements are treated as a single statement.

The following function contains a for loop that counts the number of selected options in a scrolling list (a *select* object that allows multiple selections). The for loop declares the variable *i* and initializes it to zero. It checks that *i* is less than the number of options in the *select* object, performs the succeeding if statement, and increments *i* by one after each pass through the loop.

```
<HTML>
<HEAD>
<TITLE>IGNOU </TITLE>
<SCRIPT LANGUAGE = "JavaScript">
function howMany(selectObject) {
        var numberSelected=0
        for (var i=0; i < selectObject.options.length; i++) {
                if (selectObject.options[i].selected==true)
                        numberSelected++
        }
        return numberSelected
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="selectForm">
    <P><B>Choose some music types, then click the button below:</B>
    <BR><SELECT NAME="musicTypes" MULTIPLE>
        <OPTION SELECTED> R&B
        <OPTION> Jazz
        <OPTION> Blues
        <OPTION> New Age
        <OPTION> Classical
        <OPTION> Opera
    </SELECT>
    <P><INPUT TYPE="button" VALUE="How many are selected?"
    onClick="alert ('Number of options selected: ' +
    howMany(document.selectForm.musicTypes))">
</FORM>                              •
</BODY>
</HTML>
```

• **While Statement**

The while statement defines a loop that iterates as long as a condition remains true. In the following example the control waits until the value of a text field becomes "go":

    while (Document.Form1.Text1.Value != "go") {*Statements* }

In a while loop the condition is evaluated first before executing the statements.

• **For In Statement**

This is a different type of loop, used to iterate through the properties of an object or the elements of an array. For example consider the following statement that loops through the properties of the Scores object, using the variable x to hold each property in turn:

For (x in Scores) {Statements}

- **Break Statement**

The break statement is used for terminating the current While or For loop and then
transferring program control to the statement just after the terminated loop. The
following function has a break statement that terminates the **while** loop when *i*
becomes equal to 3, and then returns the value 3 * *x*.

```
function testBreak(x) {
        var i = 0
        while (i < 6) {
                if (i == 3)
                        break
                i++
        }
        return i*x
}
```

- **Continue Statement**

A continue statement terminates execution of the block of statements in a while or for
loop and continues execution of the loop with the next iteration. In contrast to the
break statement, continue does not terminate the execution of the loop entirely.
Instead,

- In a while loop, it jumps back to the *condition*.
- In a for loop, it jumps back to the *increment-expression*.

The following example shows a While loop with a continue statement that executes
when the value of *i* becomes equal to three. Thus, *n* takes on the values one, three,
seven, and twelve.

```
i = 0
n = 0
while (i < 5) {
        i++
        if (i == 3) continue
        n += i
}
```

4.5   OBJECT-BASED PROGRAMMING

JavaScript is a very powerful object-based (or prototype-based) language. JavaScript
is not a full-blown OOP (Object-Oriented Programming) language, such as Java, but
it is an object-based language. Objects not only help you better understand how
JavaScript works, but in large scripts, you can create self-contained JavaScript
objects, rather than the procedural code you may be using now. This also allows you
to reuse code more often.

4.5.1 Functions

Functions are the central working units of JavaScript. Almost all the scripting code
uses one or more functions to get the desired result. If you want your page to provide
certain a user-defined functionality, then functions are a convenient way of doing so.
Therefore it is important that you understand what a function is and how it works.

First let us understand the basic syntax of a function; then we look at how to call it.
After that you must know how to pass arguments and why you need to do this.

Finally, you have to know how to return a value from a function. The following code shows the implementation of a function.

```
function example(a,b)
{
        number += a;
        alert('You have chosen: ' + b);
}
```

The function made above can be called using the following syntax.
Example(1,'house')

In fact, when you define the function Example, you create a new JavaScript command that you can call from anywhere on the page. Whenever you call it, the JavaScript code inside the curly brackets {} is executed.

Calling the Function

You can call the function from any other place in your JavaScript code. After the function is executed, the control goes back to the other script that called it.
alert('Example 1: the House');
example(1,'house');
(write more code)

So this script first generates an alert box, then calls the function and after the function is finished it continues to execute the rest of the instructions in the calling code.

Arguments

You can pass arguments to a function. These are variables, either numbers or strings, which are used inside the function. Of course the output of the function depends on the arguments you give it.

In the following example we pass two arguments, the number 1 and the string 'house':
example(1,'house');

When these arguments arrive at the function, they are stored in two variables, a and b. You have to declare these in the function header, as you can see below.
function example(a,b)

```
<HTML>
  <HEAD>
    <TITLE>IGNOU </TITLE>
    <SCRIPT Language = "JavaScript">
      function example(a, b)
      {
          var number;
          number += a ;
          alert('You have chosen: ' + b);
                        }
    </SCRIPT>
  </HEAD>
  <BODY>
  <FORM NAME="selectForm">
    <P><B>Click the button below:</B>
    <BR>
    <P><INPUT TYPE="button" VALUE="Click" onClick="example(1,'house')">
  </FORM>
```
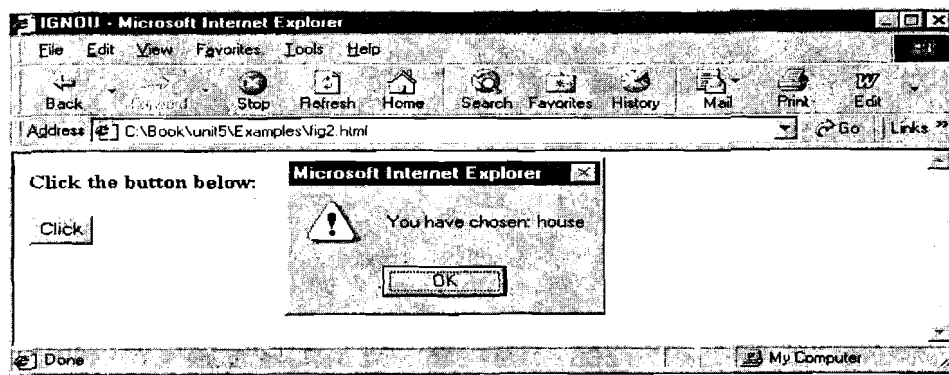
```
</BODY>
</HTML>
```

Figure 4.1: Using Functions

It adds 1 to number and displays "You have chosen: house". Of course, if you call the
function like example (5,'palace'), then it adds 5 to number and displays "You have
chosen: palace". The output of the function depends on the arguments you give it.

Returning a value

One more thing a function can do is to return a value. Suppose we have the following
function:

```
<HTML>
  <HEAD>
    <TITLE>IGNOU </TITLE>
    <SCRIPT Language = "JavaScript">
    function calculate(a,b,c)
    {
        d = (a+b) * c;
        return d;
    }
    </SCRIPT>
  </HEAD>
  <BODY>
  <SCRIPT Language = "JavaScript">
    var x = calculate(4,5,9);
    var y = calculate((x/3),3,5);
    alert('calculate(4,5,9) = ' +x + ' and ' + ' calculate((x/3),3,5) = ' + y);
  </SCRIPT>
  </BODY>
</HTML>
```
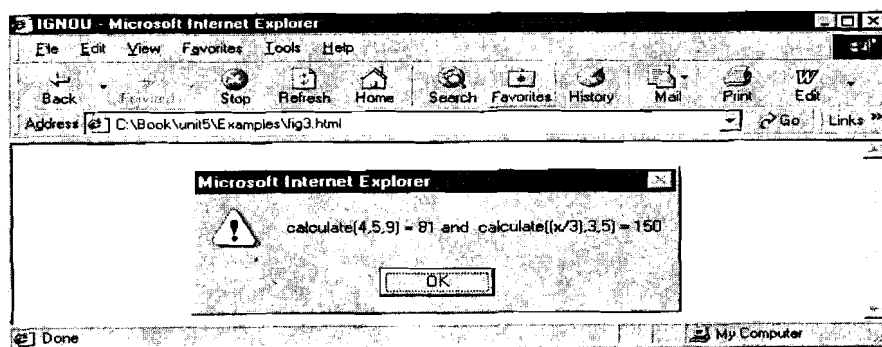

Figure 4.2: Using a Function that Returns a Value

99

The function shown in Figure 4.8 calculates a number from t

he numbers you pass to it. When it is done it returns the result of the calculation. The function passes the result back to the function that called it. When the function executes the return statement, control goes back to the calling program without executing any more code in the function, if there is any.

The calling of the function is done using the following two statements in the figure:

var x = calculate(4,5,9);
var y = calculate((x/3),3,5);

It means that you have declared a variable x and are telling JavaScript to execute calculate( ) with the arguments 4, 5 and 9 and to put the returned value (81) in x. Then you declare a variable y and execute calculate( ) again. The first argument is x/3, which means 81/3 = 27, so y becomes 150.Of course you can also return strings or even Boolean values (true or false). When using JavaScript in forms, you can write a function that returns either true or false and thus tells the browser whether to submit a form or not.

4.5.2 Executing Deferred Scripts

Deferred scripts do not do anything immediately. In order to use deferred commands, you must call them from outside the deferred script. There are three ways to call deferred scripts:

- From immediate scripts, using the function mechanism
- By user-initiated events, using event handlers
- By clicking on links or image-map zones that are associated with the script

Calling Deferred Code from a Script

A function is a deferred script because it does not do anything until an event, a function, a JavaScript link, or an immediate script calls it. You have probably noticed that you can call a function from within a script. Sometimes you are interested in calling a function from the same script, and in other cases you might want to call it from another script. Both of these are possible.

Calling a function from the same script is very simple. You just need to specify the name of the function, as demonstrated in Figure 4.9.

```
<HTML>
<HEAD>
<TITLE>Calling deferred code from its own script</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
makeLine(30)
function makeLine(lineWidth) {
  document.write("<HR SIZE=" + lineWidth + ">")
}
makeLine(10)
// -->
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

4.5.3 Objects

A JavaScript object is an instance of datatype. Object is given a unique name and the collection of properties of the corresponding object may be accessed using the dot syntax. As a quick introduction to the concept of JavaScript objects, here is the code that creates an instance of an object called myObj:

```
var myObj = new Object();
myObj.business = "Voice and Data Networking"; myObj.CEO = "IGNOU";
myObj.ticker = "CSCO";
```

After having run that code (in a scope situation that allowed myObj to be accessed as required), you could run this...
document.write("My Object ticker symbol is " + myObj.ticker + ".");
...and get a complete sentence in your HTML document.

- **Document Object**

The document object is a property of the window object. This object is the container for all HTML HEAD and BODY objects associated within the HTML tags of an HTML document

Document Object Properties

Property	Description
AlinkColor	The color of active links
BgColor	The background color of the web page. It is set in the <BODY> tag. The following code sets the background color to white. document.bgColor = "#FFFFFF"
Cookie	Used to identify the value of a cookie
Domain	The domain name of the document server
Embeds	An array containing all the plugins in a document
FgColor	The text color attribute set in the <body> tag
FileCreatedDate	Use this value to show when the loaded HTML file was created
FileModifiedDate	Use this value to show the last change date of the HTML file currently loaded
LastModified	The date the file was modified last
Layers	An array containing all the layers in a document
LinkColor	The color of HTML links in the document. It is specified in the <BODY> tag.
Title	The name of the current document as described between the header <TITLE> tags.
URL	The location of the current document
VlinkColor	The color of visited links as specified in the <BODY> tag

Document Object Methods

Method	Description
Clear	This is depreciated
Close	Closes an output stream that was used to create a document object
Contextual	It can be used to specify style of specific tags. The following example specified that text in blockquotes is to be blue: document.contextual(document.tags.blockquote).color = "blue"; Multiple styles may be specified in the contextual method to set the value of text in a <H3> tag that is underlined to the color blue, for example. document.contextual(document.tags.H3, document.tags.U).color = "blue";

ElementFromPoint(x, y)	Returns the object at point x, y in the HTML document.
GetSelection	Get the selected text (if any is selected)
open([mimeType])	Opens a new document object with the optional mime type.
write(expr1[,expr2...exprN])	Add data to a document. Writes the values passed to the write function to the document. For example document.write("<H3>") document.writeln("This is a Header") document.write("</H3>")
writeln(expr1[,expr2...exprN])	Adds the passed values to the document appended with a new line character.

- **Predefined Objects**

Let us consider some of the most frequently used predefined objects provided in Javascript.

- **Math object**

In most applications we need to perform calculations, whether it is accounting software or scientific software. Programmers are often typecast as good mathematicians. Every mathematician needs a calculator sometimes, or in the case of JavaScript, the Math object. If we want to calculate "2.5 to the power of 8" or "Sin0.9" in your script, then JavaScript's virtual calculator is what you need. The Math object contains a number of manipulating functions:

The Math object

Methods	Description
Math.abs(x)	Return absolute value of x
Math.acos(x)	Return arc cosine of x in radians
Math.asin(x)	Return arc sine of x in radians
Math.atan(x)	Return arc tan of x in radians
Math.atan2(x, y)	Counterclockwise angle between x axis and point (x,y)
Math.ceil(x)	Rounds a number up
Math.cos(x)	Trigonometric cosine of x (x in radians)
Math.exp(x)	Exponential method e^x
Math.floor(x)	Rounds a number down
Math.log(x)	Natural logarithm of x (base e)
Math.max(a, b)	Returns the larger of two values
Math.min(a, b)	Returns the smaller of two values
Math.pow(x, y)	Returns x^y
Math.round(x)	Rounds x to the closest integer
Math.sin(x)	Trigonometric sine of x (x in radians)
Math.sqrt(x)	Square root of x
Math.tan(x)	Trigonometric tangent of x (x in radians)
Properties	**Description**
Math.E	Euler's constant (~ 2.718)
Math.LN10	Natural logarithm of 10 (~ 2.302)
Math.LN2	Natural logarithm of 2 (~ 0.693)
Math.LOG10E	Base 10 logarithm of Euler's constant (~ 0.0434)
Math.LOG2E	Base 2 logarithm of Euler's constant (~1.442)
Math.PI	The ratio of a circle's circumference to its diameter (~3.141)
Math.SQRT1_2	Square root of 0.5 (~ 0.707)
Math.SQRT2	Square root of 2.0 (~ 1.414)

Let us have Javascript perform some mathematical calculations:

//calculate e^5

Math.exp(5)

```
//calculate cos(2PI)
Math.cos(2*Math.PI)
```

The "with" statement

If you intend to invoke Math multiple times in your script, a good statement to remember is "with." Using it you can omit the "Math." prefix for any subsequent Math properties/methods:

```
with (Math){
var x= sin(3.5)
var y=tan(5)
var result=max(x,y)
}
```

- **Date Object**

The Date object is used to work with dates and times.

Creating a Date Instance

You should create an instance of the Date object with the "new" keyword. The following line of code stores the current date in a variable called "my_date":

```
var my_date=new Date( )
```

After creating an instance of the Date object, you can access all the methods of the object from the "my_date" variable. If, for example, you want to return the date (from 1-31) of a Date object, you should write the following:

```
my_date.getDate( )
```

You can also write a date inside the parentheses of the Date( ) object. The following line of code shows some of the date formats available.

new Date("Month dd, yyyy hh:mm:ss"), new Date("Month dd, yyyy"), new Date(yy,mm,dd,hh,mm,ss), new Date(yy,mm,dd), new Date(milliseconds)

Here is how you can create a Date object for each of the ways above:

var my_date=new Date("October 12, 1988 13:14:00"), var my_date=new Date("October 12, 1988"), var my_date=new Date(88,09,12,13,14,00), var my_date=new Date(88,09,12), var my_date=new Date(500)

- **Some Date Methods**

Methods	Explanation
Date( )	Returns a Date object
GetDate( )	Returns the date of a Date object (from 1-31)
GetDay( )	Returns the day of a Date object (from 0-6. 0=Sunday, 1=Monday, etc.)
GetMonth( )	Returns the month of a Date object (from 0-11. 0=January, 1=February, etc.)
GetFullYear( )	Returns the year of the Date object (four digits)
GetHours( )	Returns the hour of the Date object (from 0-23)
GetMinutes( )	Returns the minute of the Date object (from 0-59)
GetSeconds( )	Returns the second of the Date object (from 0-59)

Examples:
Date

Returns the current date, including date, month, and year. Note that the getMonth method returns 0 in January, 1 in February etc. So add 1 to the getMonth method to

display the correct date.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JAVASCRIPT">
var d = new Date()
document.write("date = ")
document.write(d.getDate( ))
document.write(".")
document.write(d.getMonth() + 1)
document.write(".")
document.write(d.getFullYear())
document.write("time = ")
document.write(d.getHours())
document.write(".")
document.write(d.getMinutes() + 1)
document.write(".")
document.write(d.getSeconds())
</SCRIPT>
</BODY>
</HTML>
```
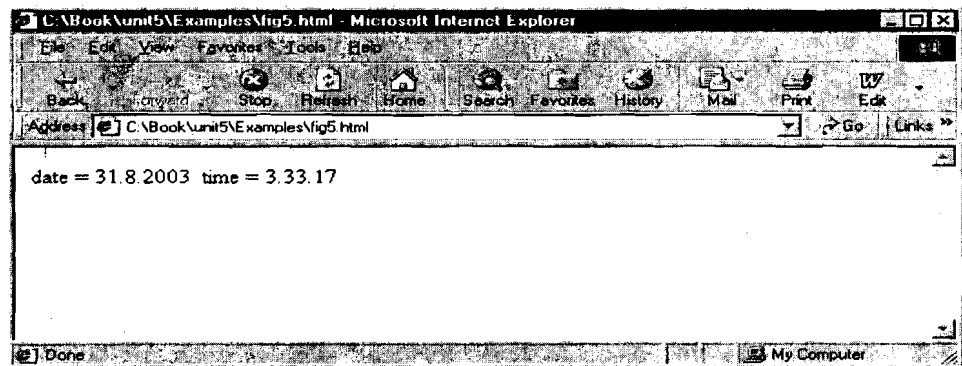

Figure 4.3: Using the Date Object

Time

Returns the current time for the timezone in hours, minutes, and seconds as shown in Figure 5.11. To return the time in GMT use getUTCHours, getUTCMinutes etc.

- **Array Object**

An Array object is used to store a set of values in a single variable name. Each value is an element of the array and has an associated index number. You can refer to a particular element in the array by using the name of the array and the index number. The index number starts at zero.

You create an instance of the Array object with the "new" keyword.

var family_names=new Array(5)

The expected number of elements goes inside the parentheses, in this case it is 5.

You assign data to each of the elements in the array like this:

family_names[0]="Sharma" family_names[1]="Singh" family_names[2]="Gill" family_names[3]="Kumar" family_names[4]="Khan"

The data can be retrieved from any element by using the index of the array element

you want:

Mother=family_names[0] father=family_names[1]
The Most Commonly Used Array Methods

Methods	Explanation
Length	Returns the number of elements in an array. This property is assigned a value when an array is created
reverse( )	Returns the array reversed
slice( )	Returns a specified part of the array
Sort( )	Returns a sorted array

- **History Object**

The History object is a predefined JavaScript object which is accessible through the history property of a window object. The window.history property is an array of URL strings, which reflect the entries in the History object. The History object consists of an array of URLs, accessible through the browser's Go menu, which the client has visited within a window. It is possible to change a window's current URL without an entry being made in the **History** object by using the location.replace method. The History object contains 4 properties and 3 methods as summarized here:

Property	Summary for History Object
Current	Specifies the URL of the current history entry.
Next	Specifies the URL of the next history entry.
Previous	Specifies the URL of the previous history entry.
Length	Reflects the number of entries in the history list.

Method	Summary for History Object
Back( )	Loads the previous URL in the history list.
Forward( )	Loads the next URL in the history list.
Go( )	Loads a URL from the history list.

- **Location Object**

The Location object is part of a Window object and is accessed through the window.location property. It contains the complete URL of a given Window object, or, if none is specified, of the current Window object. Syntax : All of its properties are strings representing different portions of the URL, which generally takes the following form:

<protocol>//<host>[:<port>]/<pathname>[<hash>][<search>]
You can create a Location object by simply assigning a URL to the location property of an object:

Syntax: window.location = "file:///C:/Projects"

| Property	Description
Hash	The **hash** property is a string beginning with a hash (#), that specifies an anchor name in an HTTP URL
Host	The **host** property is a string comprising the **hostname** and **port** strings.
hostname	The **hostname** property specifies the server name, subdomain and domain name (or IP address) of a URL.
Href	The **href** property is a string specifying the entire URL, and of which all other **link** properties are substrings.
pathname	The **pathname** property is a string portion of a URL specifying how a particular resource can be accessed.
Port	The **port** property is a string specifying the communications port that the server uses.
protocol	The **protocol** property is the string at the beginning of a URL, up to and Including the first colon (:), which specifies the method of access to the

Property	Description
	URL.
search	The **search** property is a string beginning with a question mark that specifies any query information in an HTTP URL.

Some Common Methods

Method	Description
reload	The reload method forces a reload of the windows current document, i.e. the one contained in the Location.href
Replace	The replace method replaces the current history entry with the specified URL. After calling the replace method to change the history entry, you cannot navigate back to the previous URL using the browser's Back button.

Check Your Progress 1

1. Write a JavaScript code block using arrays and generate the current date in words. This should include the day, the month and the year.
2. Write JavaScript code that converts the entered text to upper case.
3. Write JavaScript code to validate Username and Password. Username and Password are stored in variables.
4. Design the following Web page
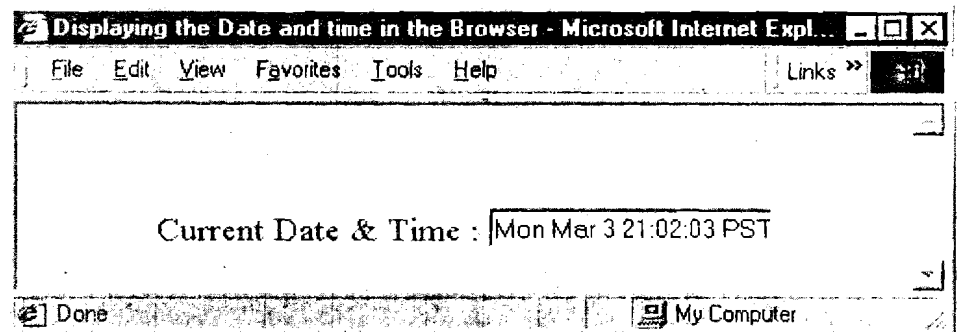
4.6   MESSAGEBOX IN JAVASCRIPT

In this section, we cover the topic of dialog boxes. This topic is important for understanding the way parameters are passed between different windows. This mechanism is a key component of some of the new capabilities of Internet Explorer 5.5 and 6.0, such as print templates. You use dialog boxes all over. The alert box is probably the most popular one.

4.6.1   Dialog Boxes

In this the user cannot switch to the window without closing the current window. This kind of dialog box is referred to as a modal dialog box. You create a modal dialog box with showModalDialog().

Syntax: showModalDialog ("Message")

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JAVASCRIPT">

function fnOpenModal(){
 window.showModalDialog("test.htm")
}
```

```
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME = IGNOU>
<INPUT TYPE="button" VALUE="Push Me" onclick="fnOpenModal()">
</BODY>
</HTML>
```

4.6.2 Alert Boxes

Alert boxes can be utilized for a variety of things, such as to display when an input field has not been entered properly, to display a message on document open or close, or to inform someone that they have clicked a link to leave your site. Whatever the use, the construction is essentially the same.
Syntax : alert("message")

The following example will generate a simple alert box based on clicking either a link or a form button.

```
<html>
<title>Codeave.com(JavaScript: Alert Box)</title>
<body bgcolor="#ffffff">
<!-- Example of a form button that will open an alert box -->
<form>
<input type="button" value="Open Alert Box"
onClick='alert("Place your message here... \n Click OK to continue.")'>
</form>
<p>
<!-- Example of a link that will open an alert box -->
<a href='javascript:onClick=alert("Place your message here... \n Click OK to
continue.")'>
Open Alert Box</a>

</body>
</html>
```
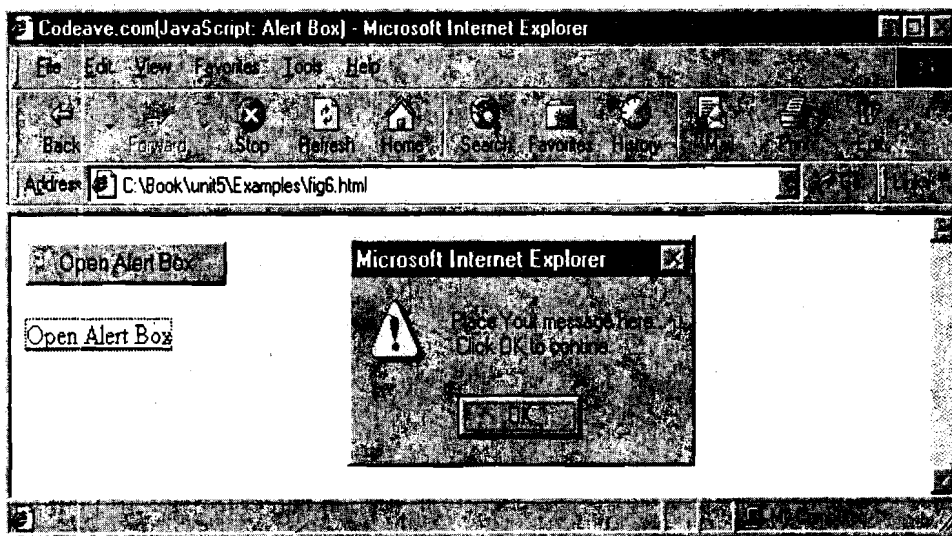

Figure 4.4: Using an Alert Box

4.6.3 Confirm Boxes

The JavaScript confirm alert box differs from a regular alert box in that it provides two choices to the user, OK and Cancel. Typically, you'll see confirmation boxes utilized on links where the destination is outside the domain of the page you are currently on or to alert you of potentially objectionable material. The following example will display a confirmation alert box when either a link or form button is clicked. Once either OK or Cancel are selected an alert box will follow to display which was chosen.

```
<html>
<body bgcolor="#FFFFFF">
<title>CodeAve.com(JavaScript: Confirm Alert Box)</title>
<script language="JavaScript">
<!--
function confirm_entry()
        {
        input_box=confirm("Click OK or Cancel to Continue");
        if (input_box==true)
                {
                // Output when OK is clicked
                alert ("You clicked OK");
                }
        else

                {
                // Output when Cancel is clicked
                alert ("You clicked cancel");
                }
        }
-->
</script>
Click <a href="JavaScript:confirm_entry()">here</a>
<p>
<form onSubmit="confirm_entry()">
<input type="submit" >
</form>
</body>
</html>
```

4.6.4 Prompt Boxes

The prompt box allows the user to enter information. The benefits of using a prompt are fairly limited and the use of forms would often be preferred (from a user perspective).

The prompt box title text cannot be changed and the same applies to button text. You can have 2 lines of text using \n where the new line starts (please note that the opera browser up to version 7 will only display 1 line of text)

The text entry field is similar to a form input type="text". The 2 buttons are OK and Cancel. The value returned is either the text entered or null.

The syntax for the prompt is

prompt("your message",""); (script tags omitted)
"your message","" the ,"" holds the default text value "" = empty.

4.7   JAVASCRIPT WITH HTML

4.7.1   Events

Events are actions that can be detected by JavaScript. An example would be the onMouseOver event, which is detected when the user moves the mouse over an object. Another event is the onLoad event, which is detected as soon as the page is finished loading. Usually, events are used in combination with functions, so that the function is called when the event happens. An example would be a function that would animate a button. The function simply shifts two images. One image that shows the button in an "up" position, and another image that shows the button in a "down" position. If this function is called using an onMouseOver event, it will make it look as if the button is pressed down when the mouse is moved over the image.

4.7.2   Event Handlers

An event handler executes a segment of a code based on certain events occurring within the application, such as onLoad or onClick. JavaScript event handlers can be divided into two parts: interactive event handlers and non-interactive event handlers. An interactive event handler is the one that depends on user interaction with the form or the document.

For example, onMouseOver is an interactive event handler because it depends on the user's action with the mouse. An example of a non-interactive event handler is onLoad, as it automatically executes JavaScript code without the need for user interaction. Here are all the event handlers available in JavaScript.

- **onLoad and onUnload**

onLoad and onUnload are mainly used for popups that appear when the user enters or leaves the page. Another important use is in combination with cookies that should be set upon arrival or when leaving your pages.

For example, you might have a popup asking the user to enter his name upon his first arrival to your page. The name is then stored in a cookie. Furthermore, when the visitor leaves your page a cookie stores the current date. Next time the visitor arrives at your page, it will have another popup saying something like: "Welcome Ajay, this page has not been updated since your last visit 8 days ago".

Another common use of the onLoad and onUnload events is in making annoying pages that immediately open several other windows as soon as you enter the page.

- **OnFocus and onBlur**

The onFocus event handler executes the specified JavaScript code or function on the occurrence of a focus event. This is when a window, frame or form element is given the focus. This can be caused by the user clicking on the current window, frame or form element, by using the TAB key to cycle through the various elements on screen, or by a call to the window.focus method. Be aware that assigning an alert box to an object's onFocus event handler will result in recurrent alerts as pressing the 'OK' button in the alert box will return focus to the calling element or object.

The onFocus event handler uses the Event object properties.

type - this property indicates the type of event.
target - this property indicates the object to which the event was originally sent.

The following example shows the use of the onFocus event handler to replace the default string displayed in the text box. Note that the first line is HTML code and that

the text box resides on a form called 'myForm'.

Syntax : onfocus = script

```
<HTML>
<HEAD>

</HEAD>
<BODY>
<FORM NAME = "myform" >
<input type="text" name="myText" value="Give me focus" onFocus =
"changeVal()">
</BODY>
<SCRIPT LANGUAGE="JAVASCRIPT">

s1 = new String(myform.myText.value)
function changeVal() {
s1 = "I'm feeling focused"
document.myform.myText.value = s1.toUpperCase()
}
</SCRIPT>
</HTML>
```
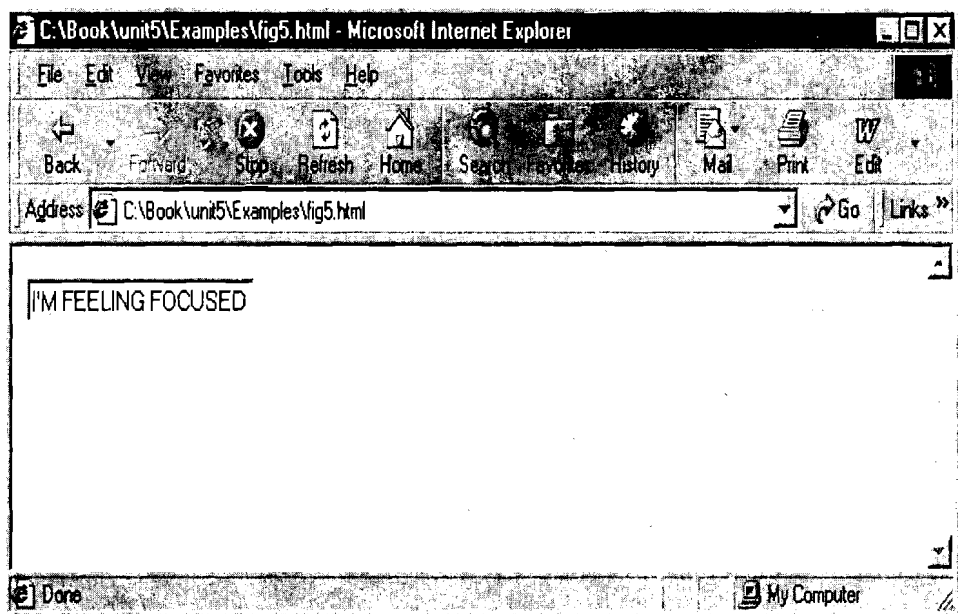

Figure 4.5: Using the onFocus Method

---

onblur = script

The onBlur event handler executes the specified JavaScript code or function on the occurrence of a blur event. This is when a window, frame or form element loses focus. This can be caused by the user clicking outside of the current window, frame or form element, by using the TAB key to cycle through the various elements on screen, or by a call to the window.blur method.

The onBlur event handler uses the Event object properties.

type - this property indicates the type of event.
target - this property indicates the object to which the event was originally sent.

The following example shows the use of the onBlur event handler to ask the user to check that the details given are correct. Note that the first line is HTML code.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<FORM NAME = "myform" >
Enter email address <INPUT TYPE="text" VALUE="" NAME="userEmail"
onBlur=addCheck( )>
</BODY>
<SCRIPT LANGUAGE="JAVASCRIPT">
function addCheck() {
alert("Please check your email details are correct before submitting")
}
</SCRIPT>
</HTML>
```

- **OnError**

The onError event handler executes the specified JavaScript code or function on the occurrence of an error event. This happens when an image or document causes an error during loading. A distinction must be made between a browser error, when the user types in a non-existent URL, for example, and a JavaScript runtime or syntax error. This event handler will only be triggered by a JavaScript error, not a browser error.

Apart from the onError handler triggering a JavaScript function, it can also be set to onError="null", which suppresses the standard JavaScript error dialog boxes. To suppress JavaScript error dialogs when calling a function using onError, the function must return true (Example 2 below demonstrates this).

There are two things to bear in mind when using window.onerror. First, this only applies to the window containing window.onerror, not any others, and secondly, window.onerror must be spelt all lower-case and contained within <script> tags; it cannot be defined in HTML (this obviously does not apply when using onError with an image tag, as in example 1 below).

The onFocus event handler uses the Event object properties.

type - this property indicates the type of event.
target - this property indicates the object to which the event was originally sent.

The first example suppresses the normal JavaScript error dialogs if a problem arises when trying to load the specified image, while Example 2 does the same, but applied to a window, by using a return value of true in the called function, and displays a customized message instead.

Syntax : Object.onError = [function name]

Example 1

```
<IMG NAME="imgFaculty" SRC="dodgy.jpg onError="null">
```
Example 2

```
<script type="text/javascript" language="JavaScript">
s1 = new String(myForm.myText.value)
```

```
window.onerror=myErrorHandler
function myErrorHandler() {
alert('A customized error message')
return true
}
</script>
<body onload=nonexistantFunc( )>
```

Check Your Progress 2

Design a Web page that displays a welcome message whenever the page is loaded and an Exit message whenever the page is unloaded.

4.8 FORMS

Each form in a document creates a form object. Since a document can contain more than one form, Form objects are stored in an array called forms.

4.8.1 Forms Array

Using the forms[ ] array we access each Form object in turn and show the value of its name property in a message box. Let us have a look at an example that uses the forms array. Here we have a page with three forms on it.

```
<HTML>
<HEAD>
    <SCRIPT LANGUAGE=JavaScript>
    function window_onload()
    {
      var numberForms = document.forms.length;
      var formIndex;
      for (formIndex = 0; formIndex < numberForms; formIndex++)
      {
        alert(document.forms[formIndex].name);
      }
    }
    </SCRIPT>
</HEAD>
<BODY LANGUAGE=JavaScript onLoad="window_onload()">
    <FORM NAME="form1">
        <P>This is inside form1</P>
    </FORM>
    <FORM NAME="form2">
        <P>This is inside form2</P>
    </FORM>
    <FORM NAME="form3">
        <P>This is inside form3</P>
    </FORM>
</BODY>
</HTML>
```

Within the body of the page we define three forms. Each form is given a name, and contains a paragraph of text. Within the definition of the <BODY> tag, the window_onload( ) function is connected to the window object's onLoad event handler.

```
<BODY LANGUAGE=JavaScript onLoad="return
```

window_onload( )">

This means that when the page is loaded, our window_onload() function will be called. The window_onload( ) function is defined in a script block in the HEAD of the page. Within this function we loop through the forms[ ] array. Just like any other JavaScript array, the forms[ ] array has a length property, which we can use to determine how many times we need to loop. Actually, as we know how many forms there are, we could just write the number in. However, here we are also demonstrating the length property, since it is then easier to add to the array without having to change the function. Generalizing your code like this is a good practice to follow.

The function starts by getting the number of Form objects within the forms array and stores it in the variable numberForms.

```
function window_onload( )
{
  var numberForms = document.forms.length;
```

Next we define a variable, formIndex, to be used in our for loop. After this comes the for loop itself.

```
var formIndex;
for (formIndex = 0; formIndex < numberForms; formIndex++)
{
  alert(document.forms[formIndex].name);
}
```

Remember that since the indices for arrays start at zero, our loop needs to go from an index of 0 to an index of numberForms - 1. We do this by initializing the formIndex variable to zero, and setting the condition of the for loop to formIndex < numberForms.

Within the for loop's code, we pass the index of the desired form (that is, formIndex) to document.forms[ ], which gives us the Form object at that array index in the forms array. To access the Form object's name property, we put a dot at the end and the name of the property, name.

- **Form Object**

Form is a property of the document object. This corresponds to an HTML input form constructed with the FORM tag. A form can be submitted by calling the JavaScript submit method or clicking the form SUBMIT button. Some of the form properties are:

- Action - This specifies the URL and CGI script file name the form is to be submitted to. It allows reading or changing the ACTION attribute of the HTML FORM tag.
- Button – An object representing a GUI control.
- Elements - An array of fields and elements in the form.
- Encoding - This is a read or write string. It specifies the encoding method the form data is encoded in, before being submitted to the server. It corresponds to the ENCTYPE attribute of the FORM tag. The default is "application/x-www-form-urlencoded". Other encoding includes text/plain or multipart/form- data.
- Length - The number of fields in the elements array, that is, the length of the elements array.
- Method - This is a read or write string. It has the value "GET" or "POST".
- Name - The form name. Corresponds to the FORM Name attribute.

- Password – An object representing a password field.
- Radio – An object representing a radio button field.
- Reset - An object representing a reset button.
- Select - An object representing a selection list.
- Submit - An object representing a submit button.
- Target - The name of the frame or window to which the form submission response is sent by the server.
  Corresponds to the FORM TARGET attribute.
- Text - An object representing a text field.
- Textarea - An object representing a text area field.

Some of the **Form Element Properties** are:

- Name – It provides access to an element's name attribute. It applies to all form elements.
- Type – Identifies the object's type. It applies to all form elements.
- Value - Identifies the object's value. It applies to all, button, checkbox, hidden, password, radio, reset, submit, text or textarea.
- Checked - Identifies whether the element is checked. It applies to checkbox and radio.
- Default checked - Identifies whether the element is checked by default. It applies to checkbox and radio.
- Default value – Identifies the object's default value. It applies to password, submit and textarea.
- Length - Identifies the length of a select list. It applies to select.
- Options – An array that identifies the options supported by the select list. It applies to select.

Syntax : <FORM Name = "myform" Action = "mailto:subscribe@abc.com" Method = POST Enctype = "multipart/form-data" onsubmit = "return check( )" >

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE = "javascript">
function check ()
        { if (document.myform.email.value == "") {
                alert("please enter your email-id");
                return false; }
        }
</SCRIPT>
</HEAD>
<BODY>
Enter your Email to subscribe : <p>
<FORM Name = "myform" Action = "mailto:subscribe@abc.com" Method = POST
Enctype = "multipart/form-data" onsubmit = "return check()" >
        <Input type = "text" Name = "email">
        <Input type = "submit" value = "submit">
</FORM>
</BODY>
</HTML>
```

The above code is used for obtaining the user's email-id. In case the user clicks the submit button without entering the email-id in the desired text field, he sees a message box indicating that the email-id has not been entered.

Case Study

Design a Web page with appropriate functionality to accept an order for a fast food outlet. It should check if the user has entered a valid name and email-id. It should also calculate the value of the order.

```
<HTML>
<HEAD>
 <TITLE>Donald Duck</TITLE>
<SCRIPT Language="JavaScript">
var m;
function chk_name()
{
if( document.form1.txt_name.value == "")
{
alert("Please enter your name");
document.form1.txt_name.focus();
}
}
function chk_email()
{
var str = document.form1.txt_email.value ;
var i;
if ( document.form1.txt_email.value == "")
{
alert("Please enter your Email-ID");
document.form1.txt_email.focus();
}
i = str.indexOf("@");
if ( i< 0)
{
alert ("Please enter a valid Email-Id");
}
}

function mainitem(F1)
{ var z=" ";
            for(j=0;j<3;j++)
            {
                    for(i=0;i<F1.elements[j].length;i++)
                    {
                            if (F1.elements[j][i].selected)
                                {
                                var y=F1.elements[j].options[i].value;
                                z=z+"\n"+y;
                                F1.elements[3].value=z;
                                }
                    }
            }
        m=z;
}
function cal(F1)
{ var d=0;
        for(j=0;j<3;j++)
        {
                for(i=0;i<F1.elements[j].length;i++)
                {
        if (F1.elements[j][i].selected)
```

115

```
                                    {
                                    var y=F1.elements[j].options[i].value;
                                    s=new String(y);
                                    var a=s.indexOf(">");
                                    var b=s.substring(a+1,a+3);
                            c=parseInt(b);
                            d=d+c;
                            }
                    }
                }
                p="Total cost of the selected items="+d;
                m=m+"\n"+p;
F1.elements[3].value=m;
}

function clr(F1)
{
F1.elements[3].value=" ";
}
</SCRIPT>
</HEAD>
<BODY>
<h2><font color="blue"><center> Welcome to the World renowned online Fast Food
Center </font>
<font color="red"> Donald Duck ! </center></font></h2>
<Form name="form1" ACTION = "mailto:dlc@ignou.ac.in" METHOD = POST>
Select the Menu Items of your choice and then click on the Total Cost to find the bill
amount-
<BR><BR>

<Table >
<TR valign=top ><td>
Major dishes :<BR>
<select name="s1" MULTIPLE onBlur="mainitem(this.form)">
<option value="Onion cheese capsicum Pizza->300" selected> Onion cheese
capsicum Pizza
<option value="Onion mushroom Pizza->200"> Onion mushroom Pizza
<option value="Chicken Tikka Pizza->460"> Chicken Tikka Pizza
<option value="Cheese Pizza->150"> Cheese Pizza
</select>
<BR><BR></td>
<td> </td><td> </td>
<td>
Soups :<BR>
<select name="s2" MULTIPLE onBlur="mainitem(this.form)">
<option value="Tomato Soup->70"> Tomato Soup
<option value="Sweet corn Soup->80">Sweet corn Soup
<option value="Sweet n Sour soup->90">Sweet n Sour soup
<option value="Mixed veg soup->50">Mixed veg soup
</select>
<BR><BR></td>
<td> </td><td> </td>
<td>
Miscellaneous :<BR>
<select name="s3" onBlur="mainitem(this.form)">
<option value=" ">'Desserts'
<option value="Milkshakes->35">Milkshakes
```

```
<option value="Soft drinks->20">Soft drinks
<option value="Ice cream sodas->25">Softy
</select>
<BR><BR></td>
<td> </td><td> </td>
</TR>
</Table>
<Table>
<TR valign=top><td>
The items selected form the Menu are :
<TEXTAREA Name="TA1" Rows=10 Cols=50>
</TEXTAREA><BR><BR></td>
<td> </td><td> </td>
<td><BR>
<input type="button" Value="Total Cost" onClick="cal(this.form)">
<input type="button" Value="Clear" onClick="clr(this.form)">
</td>
</TR>
</Table>
<HR noshade>
<h2><font color="red"> Personal Details </center></font></h2>
<Table >
<TR valign=top ><td>
Name:<BR>
<Input Type = "Text" Name = "txt_name" Onblur = "chk_name()">
<BR><BR></td>
<td> </td><td> </td>
<td>
Contact Address :<br>
<TEXTAREA Name="TA2" Rows=3 Cols=10>
</TEXTAREA>
<BR><BR></td>
<td> </td><td> </td>
<td>
Email :<BR>
<Input Type = "Text" Name = "txt_email" Onblur = "chk_email()">
<BR><BR></td>
<td> </td><td> </td>

</TR>
</Table>
<Table>
<TR valign=top ><td>
Phone :<BR>
<Input Type = "Text" Name = "txt_phone">
<BR><BR></td>
<td> </td><td> </td>
<td>
<BR>
<Input Type = "submit" Name = "btnsubmit" Value = "        Submit        ">
<BR><BR></td>
<td> </td><td> </td>
</TR>
</Table>
</Form>
</BODY>
</HTML>
```
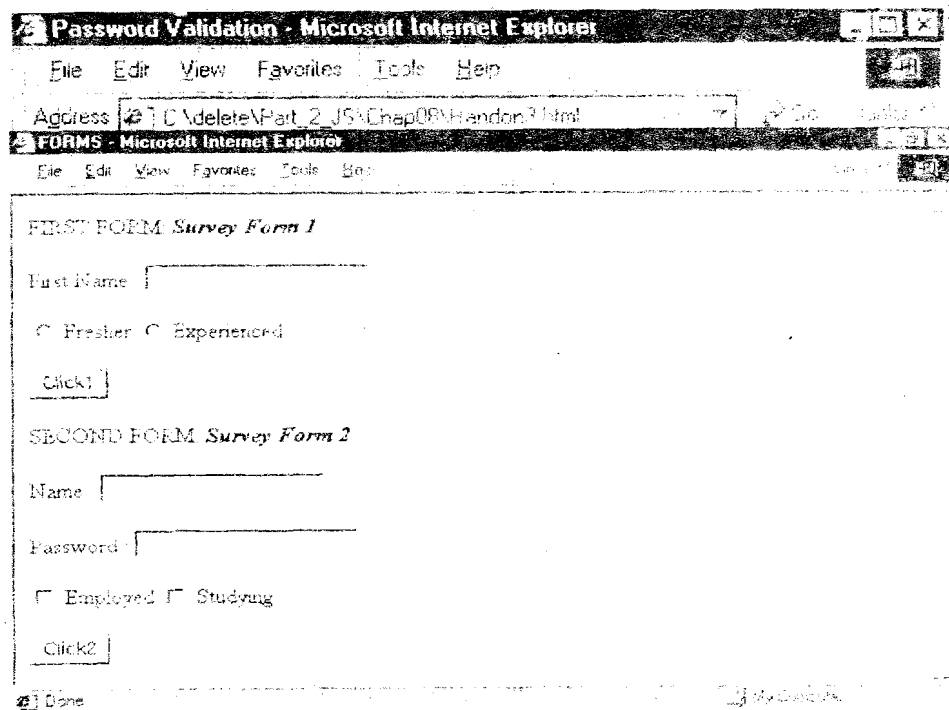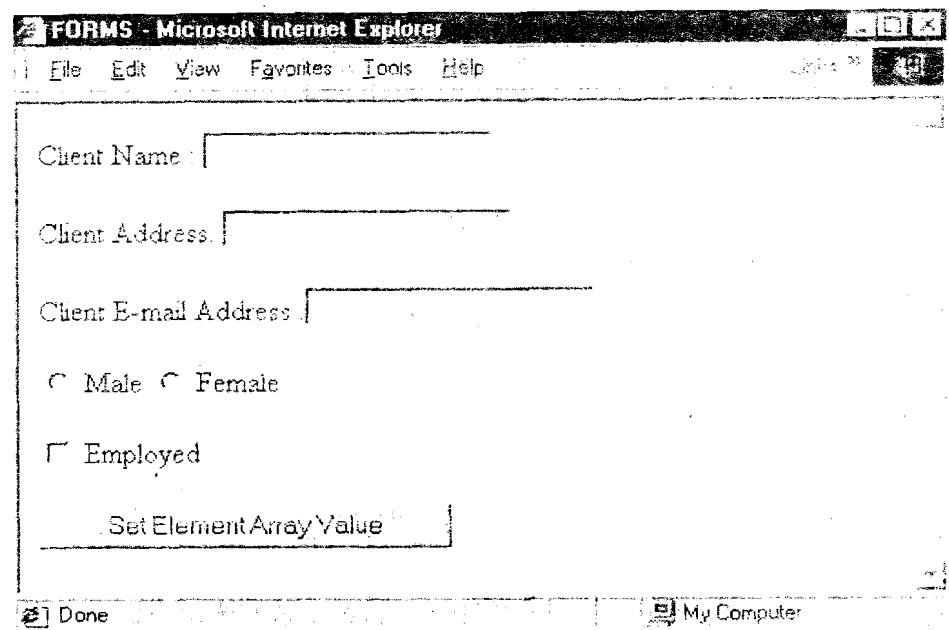
117

Check Your Progress 3

1.   Design the following Web page.

2.   Design the following Web page.

3.   Design the following Web page in such a way that selecting any option
      from the radio button displays the appropriate result in the Result box.

4.   Suppose you have an order form that updates automatically each time you
enter or change a quantity, the cost for that item and the total cost are updated. Each
field must be validated. Design the following web page. (Case Study)

4.9   SUMMARY

In this unit we have learned about the major features of JavaScript. Besides normal
programming constructs, datatypes and variables, we have also discussed the most
commonly used objects. These are: Document, Math, Date, History, Location etc. The
Submit and Reset objects are used to submit the information to the server and reset
the information entered, respectively. Finally, we discussed a very important object,
the Form object.

4.10   SOLUTIONS/ ANSWERS

Check Your Progress 1

1.

```
<HTML>
<HEAD>
<TITLE> Date validations </TITLE>
<SCRIPT>
        var monthNames = new Array(12);
        monthNames[0]= "January";
        monthNames[1]= "February";
        monthNames[2]= "March";
        monthNames[3]= "April";
        monthNames[4]= "May";
        monthNames[5]= "June";
        monthNames[6]= "July";
        monthNames[7]= "August";
        monthNames[8]= "September";
        monthNames[9]= "October";
        monthNames[10]= "November";
        monthNames[11]= "December";

        var dayNames = new Array(7);
        dayNames[0]= "Sunday";
        dayNames[1]= "Monday";
        dayNames[2]= "Tuesday";
        dayNames[3]= "Wednesday";
        dayNames[4]= "Thursday";
        dayNames[5]= "Friday";
        dayNames[6]= "Saturday";

        function customDateString (m_date)
        {
                var daywords = dayNames[m_date.getDay()];
                var theday = m_date.getDate();
                var themonth = monthNames[m_date.getMonth()];
                var theyear = m_date.getYear();
                return daywords + ",  " + themonth + " " + theday + ", " + theyear;
        }
</SCRIPT>
</HEAD>

<BODY>
<H1>WELCOME!</H1>
<SCRIPT>
        document.write(customDateString( new Date()))
</SCRIPT>
</BODY>
</HTML>
```

2.

```
<HTML>
<HEAD>
<SCRIPT  language="JavaScript">
```

```
        function checkData(column_data)
        {
                if (column_data != "" && column_data.value !=
column_data.value.toUpperCase( ))
                {
                        column_data.value = column_data.value.toUpperCase( )
                }
        }
</SCRIPT>
</HEAD>

<BODY>
<FORM>
        <INPUT TYPE="text" NAME="collector" SIZE=10
onChange="checkData(this)">
        <INPUT TYPE="text" NAME="dummy" SIZE=10>
</FORM>
</BODY>
</HTML>
```

3.

```
<HTML>
<HEAD>
<TITLE>Password Validation</TITLE>
<SCRIPT language="JavaScript">
<!--
var userpassword = new Array(4);
userpassword[0]= "Ajay";
userpassword[1]= "ajay";
userpassword[2]= "Rohan";
userpassword[3]= "rohan";

function checkOut()
{
        var flag =0;
        var flag1 =0;
        var i =0;
        var j =0;

for (x=0; x<document.survey.elements.length; x++)
        {
        if (document.survey.elements[x].value == "")
        {
        alert("You forgot one of the required fields. Please try again")
        return;
        }
        }
        var user= document.survey.elements[0].value
        var password = document.survey.elements[1].value
        while(i<=3)
        {
        if(userpassword[i] == user)
        {
                j=i;
                j++;
                if(userpassword[j] == password)
```

```
                    {
                    flag=1;
                    break;
                    }
            }
            i+=2;
            }
if(flag == 0)
            {
            alert("Please enter a valid user name and password")
            return;
            }
            else
            {
            alert("Welcome !!\n" +document.forms[0].Username.value);
            }
return;
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<FORM ACTION="" method="POST" NAME="survey"
onSubmit="return checkOut(this.form)">
<INPUT TYPE="TEXT" NAME="Username" SIZE="15" MAXLENGTH="15">
User Name
<BR><INPUT TYPE="PASSWORD" NAME="Pasword" SIZE="15">Password
<BR><INPUT TYPE="SUBMIT" VALUE="Submit"><INPUT TYPE="RESET"
VALUE="Start Over">
</FORM>
</BODY>
</HTML>
```

4.

```
<HTML>
<HEAD>
<TITLE>Displaying the Date and time in the Browser</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
function begin(form)
                {
                form_name = form;
                time_out=window.setTimeout("display_date()",500)
                }

function display_date()
                {
                form_name.date.value=new Date();
                time_out=window.setTimeout("display_date()",1000)
                }
function display_clock()
{
document.write('<FONT COLOR=red SIZE=+1><FORM
NAME=time_form><CENTER><BR><BR>Current Date & Time :')
```

```
document.write(' <INPUT NAME=date size=19
value=""</FORM></FONT></CENTER>')
begin(document.time_form)
}
display_clock()
</SCRIPT>
</BODY>
</HTML>
```

Check Your Progress 2

1.

```
<HTML>
  <HEAD>
    <TITLE>Creating and Using User Defined Functions</TITLE>
      <SCRIPT LANGUAGE="JavaScript">
          var name = "";
          function hello( )
          {
              name = prompt('Enter Your Name:','Name');
              alert('Greetings ' + name + ', Welcome to my page!');
          }

          function goodbye( )
          {
              alert('Goodbye ' + name + ', Sorry to see you go!');
          }
      </SCRIPT>
  </HEAD>
  <BODY onLoad="hello( );" onUnload="goodbye( );">
      <IMG SRC="Images\Pinkwhit.gif ">
  </BODY>
</HTML>
```

Check Your Progress 3

1.

```
<HTML>
<HEAD>
      <TITLE>FORMS</TITLE>
              <!-- This code allows to access the Form objects Elements Array //-->
  <SCRIPT Language="JavaScript">
function Ver(form1)
{
      v=form1.elements.length;
if (form1.elements[3].name=="Button1")
{
              alert('First form name : '+document.forms[0].name);
              alert('No. of Form elements of ' +document.forms[0].name + ' = '+v);
```

```
                }
                else if (form1.elements[4].name=="Button2")
                {
                alert('Second form name : '+document.forms[1].name);
                alert('No. of Form elements of ' +document.forms[1].name + ' = '+v);
                }
                for(i=0;i<v;i++)
                alert(form1.elements[i].name+ ' is at position '+i);
        }
    </SCRIPT>
</HEAD>

<BODY>
<FORM Name="Survey Form 1">
        FIRST FORM: <i><b>Survey Form 1 </b></i><BR><BR>
        First Name : <Input Type=Text Name="Text1" Value=""><BR><BR>
                <Input Type="radio" Name="Radio1" Value=""> Fresher
                <Input Type="radio" Name="Radio1" Value="">
Experienced<BR><BR>
                <Input Type="Button" Name="Button1" Value="Click1"
onClick="Ver(form)">
</FORM>

<FORM Name="Survey Form 2">
                SECOND FORM: <i><b> Survey Form 2 </b></i><BR><BR>
        Name : <Input Type="Text" Name="Text2" Value=""> <BR><BR>
                Password : <Input Type="Password" Name="Pass2" Value="">
<BR><BR>
                <Input Type="CheckBox" Name="Check1" Value="" > Employed
                <Input Type="CheckBox" Name="Check2" Value="" > Studying
<BR><BR>
                <Input Type="Button" Name="Button2" Value="Click2"
onClick="Ver(form)">
</FORM>
</BODY>
</HTML>
```

2.

```
<HMTL>
 <HEAD>
        <TITLE>FORMS</TITLE>

        <!-- This code checks the Checkbox when the button is clicked //-->
    <SCRIPT Language='JavaScript'>
    function Chk(f1)
{
 f1.Check.checked=true;
 alert(" The Checkbox just got checked ");
 f1.Check.checked=false;
```

124

```
fl.Radio[0].checked=true;
fl.Radio[1].checked=false;
alert(" The Radio button just got checked ");
}
  </SCRIPT>
 </HEAD>



<BODY>
<FORM>
        Client Name : <Input Type=Text Name="Text" Value=""><BR><BR>
        Client Address: <Input Type=Text Name="Text1" Value=""> <BR><BR>
        Client E-mail Address :<Input Type=Text Name="Text2"
Value=""><BR><BR>
        <Input Type="radio" Name="Radio" Value=""> Male
        <Input Type="radio" Name="Radio" Value=""> Female<BR><BR>
        <Input Type="CheckBox" Name="Check" Value=""> Employed <BR><BR>
<Input Type="Button" Name="Bt" Value="Set Element Array Value"
onClick="Chk(this.form)">
</FORM>
</BODY>
</HTML>


3.


<HTML>
<HEAD>
<TITLE> Working with Radio Buttons </TITLE>
<SCRIPT LANGUAGE="JavaScript">
function calculate(form)
        {
    if(form.elements[2].checked)
        {
                form.result.value = form.entry.value * form.entry.value;
        }
    else
        {
                form.result.value = form.entry.value * 2;
        }
        }
</SCRIPT>
</HEAD>
<BODY>
<FORM >
<CENTER><BR>
<B>Value:</B>
<INPUT  TYPE="text"  NAME="entry"  VALUE=0>
<BR><BR>
<SPACER Size= 190>
<B>Action:<B><BR>
```

```
<SPACER Size = 225>
<INPUT TYPE="radio" NAME="action1" VALUE="twice"
   onClick="calculate(this.form);">Double<BR>
<SPACER Size = 225>
<INPUT TYPE="radio" NAME="action1" VALUE="square"
           onClick="calculate(this.form);">Square <BR><BR>
<B>Result:</B>
<INPUT TYPE=text NAME="result" onFocus = "this.blur();">
</CENTER>
</FORM>
</BODY>
</HTML>
```

4.   (Case Study)
```
<HTML>
<HEAD><TITLE>Order Form</TITLE>
<SCRIPT>
// function to calculate the total cost field
function Total()
{ var tot = 0;
tot += (240 * document.order.qty1.value);
tot += (270 * document.order.qty2.value);
tot += (300 * document.order.qty3.value);
tot += (600 * document.order.qty4.value);
document.order.totalcost.value = tot; }
// function to update cost when quantity is changed
function UpdateCost(number, unitcost)
{ costname = "cost" + number; qtyname = "qty" + number;
var q = document.order[qtyname].value;
document.order[costname].value = q * unitcost;
Total();
}

// function to copy billing address to shipping address
function CopyAddress()
{ if (document.order.same.checked)
{ document.order.shipto.value = document.order.billto.value;
}
}
//global variable for error flag
var errfound = false;

//function to validate by length
function ValidLength(item, len)
```

```javascript
{ return (item.length >= len); }

//function to validate an email address
function ValidEmail(item)
{ if (!ValidLength(item, 5)) return false;
if (item.indexOf ('@', 0) == -1)
return false;
return true;
}


// display an error alert
function error(elem, text)
{
// abort if we already found an error
 if (errfound) return;
window.alert(text);
elem.select();
elem.focus();
errfound = true;
}
// main validation function
function Validate()
{
errfound = false;
if (!ValidLength(document.order.name1.value,6))
error(document.order.name1,"Invalid Name");
if (!ValidLength(document.order.phone.value,10))
error(document.order.phone,"Invalid Phone");
if (!ValidLength(document.order.billto.value,30))
error(document.order.billto,"Invalid Billing Address");
if (!ValidLength(document.order.shipto.value,30))
error(document.order.shipto,"Invalid Shipping Address");
if (!ValidEmail(document.order.email.value))
error(document.order.email, "Invalid Email Address");
if (document.order.totalcost.value == "")
error(document.order.qty1, "Please Order at least one item.");
if (document.order.payby.selectedIndex != 1)
{
if (!ValidLength(document.order.creditno.value,2))
error(document.order.creditno,"Invalid Credit/Check number");
}
return !errfound; /* true if there are no errors */ }
</SCRIPT> </HEAD>
<BODY>
```

```
<H1>Online Cake Order Form</H1>
<FORM NAME="order" onSubmit="return Validate();">
<B>Name:</B>
<INPUT TYPE="text" NAME="name1" SIZE=20>
 <B>Phone: </B>
<INPUT TYPE="text" NAME="phone" SIZE=15>
<B>E-mail address:</B>
<INPUT TYPE="text" NAME="email" SIZE=20><BR><BR>
<B>Shipping Address:</B>
<BR>
<TEXTAREA NAME="shipto" COLS=40 ROWS=4 onChange="CopyAddress();">
Enter your shipping address here. </TEXTAREA>
<B>Products to Order:</B><BR>
Qty: <INPUT TYPE="TEXT" NAME="qty1" VALUE="0" SIZE=4 onChange =
"UpdateCost(1, 240);">
Cost: <INPUT TYPE="TEXT" NAME="cost1" SIZE=6> (Rs.240 ) Vanilla cake
<BR>
Qty: <INPUT TYPE="TEXT" NAME="qty2" VALUE="0" SIZE=4 onChange =
"UpdateCost(2, 270);">
Cost: <INPUT TYPE="TEXT" NAME="cost2" SIZE=6> (Rs.270 ) Strawberry cake
<BR>
Qty: <INPUT TYPE="TEXT" NAME="qty3" VALUE="0" SIZE=4 onChange =
"UpdateCost(3, 300);">
 Cost: <INPUT TYPE="TEXT" NAME="cost3" SIZE=6> (Rs.300 ) Black Forest
cake <BR>
Qty: <INPUT TYPE="TEXT" NAME="qty4" VALUE="0" SIZE=4 onChange =
"UpdateCost(4, 600);">
Cost: <INPUT TYPE="TEXT" NAME="cost4" SIZE=6> (Rs.600 ) Chocolate cake
<HR>  <B>Total  Cost:</B>  <INPUT  TYPE="TEXT"  NAME="totalcost"
SIZE=8><HR> <B>
Method of Payment</B>:
<SELECT NAME="payby"> <OPTION VALUE="check" SELECTED>
Check or Money Order
<OPTION VALUE="cash">Cash or Cashier's Check
<OPTION VALUE="credit">Credit Card (specify number)
 </SELECT><BR> <B>Credit Card or Check Number:</B>:
<INPUT TYPE="TEXT" NAME="creditno" SIZE="20"><BR>
<INPUT TYPE="SUBMIT" NAME="submit" VALUE="Send Your Order">
<INPUT TYPE="RESET" VALUE="Clear Form">
</FORM>
</BODY>
</HTML>
```

4.11 FURTHER READINGS

1. JavaScript Programmers Reference, Cliff Wootton. Publisher: Wrox Press Inc.
    Year 1999.

2. Beginning JavaScript, Paul Wilton. Publisher: Wrox Press Inc. 1st Edition

3. JavaScript: The Definitive Guide, David Flanagan. Publisher: O'Reilly. 4th Edition
    2001.

4. The JavaScript Bible, Danny Goodman. Publisher: John Wiley & Sons Inc.
    Edition 2001.