What is JavaScript ?

JavaScript started life as LiveScript, but Netscape changed the name, possibly because of the excitement being generated by Java.to JavaScript. JavaScript made its first appearance in Netscape 2.0 in 1995 with a name *LiveScript*.

JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages.

The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers

Client-side JavaScript:

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need no longer be static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism features many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user explicitly or implicitly initiates.

Advantages of JavaScript:

The merits of using JavaScript are:

- **Less server interaction:** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity:** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces:** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations with JavaScript:

We can not treat JavaScript as a full fledged programming language. It lacks the following important features:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript can not be used for Networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocess capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Syntax

A JavaScript consists of JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.

You can place the <script> tag containing your JavaScript anywhere within you web page but it is preferred way to keep it within the <head> tags.

The <script> tag alert the browser program to begin interpreting all the text between these tags as a script. So simple syntax of your JavaScript will be as follows

```
<script ...>
  JavaScript code
</script>
```

The script tag takes two important attributes:

- **language:** This attribute specifies what scripting language you are using. Typically, its value will be *javascript*. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to *"text/javascript"*.

So your JavaScript segment will look like:

```
<script language="javascript" type="text/javascript">
  JavaScript code
</script>
```

Your First JavaScript Script:

Let us write our class example to print out "Hello World".

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
   document.write("Hello World!")
//-->
```

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

We added an optional HTML comment that surrounds our Javascript code. This is to save our code from a browser that does not support Javascript. The comment ends with a "//-->". Here "//" signifies a comment in Javascript, so we add that to prevent a browser from reading the end of the HTML comment in as a piece of Javascript code.

Next, we call a function *document.write* which writes a string into our HTML document. This function can be used to write text, HTML, or both. So above code will display following result:

```
Hello World!
```

Case Sensitivity:

JavaScript is a case-sensitive language. This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So identifiers *Time*, *TIme* and *TIME* will have different meanings in JavaScript.

NOTE: Care should be taken while writing your variable and function names in JavaScript.

Comments in JavaScript:

JavaScript supports both C-style and C++-style comments, Thus:

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Example:

```
<script language="javascript" type="text/javascript">
<!--

// This is a comment. It is similar to comments in C++

/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming
 */
```

```
//-->
</script>
```

JavaScript Placement in HTML File

There is a flexibility given to include JavaScript code anywhere in an HTML document. But there are following most preferred ways to include JavaScript in your HTML file.

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in and external file and then include in <head>...</head> section.

In the following section we will see how we can put JavaScript in different ways:

JavaScript in <head>...</head> section:

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows:

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
   alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

JavaScript in <body>...</body> section:

If you need a script to run as the page loads so that the script generates content in the page, the script goes in the <body> portion of the document. In this case you would not have any function defined using JavaScript:

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
```

```
</script>
<p>This is web page body </p>
</body>
</html>
```

This will produce following result:

```
Advertisements

Hello World
This is web page body
```

JavaScript in <body> and <head> sections:

You can put your JavaScript code in <head> and <body> section altogether as follows:

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
   alert("Hello World")
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

This will produce following result:

```
Advertisements

Hello World
```

JavaScript in External File

As you begin to work more extensively with JavaScript, you will likely find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The *script* tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using *script* tag and its *src* attribute:

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.......
</body>
</html>
```

To use JavaScript from an external file source, you need to write your all JavaScript source code in a simple text file with extension ".js" and then include that file as shown above.

For example, you can keep following content in filename.js file and then you can use *sayHello* function in your HTML file after including filename.js file:

```
function sayHello() {
   alert("Hello World")
}
```

JavaScript Variables and DataTypes

JavaScript Data Types:

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

- Numbers eg. 123, 120.50 etc.
- Strings of text e.g. "This text string" etc.
- Boolean e.g. true or false.

JavaScript also defines two trivial data types, *null* and *undefined*, each of which defines only a single value.

JavaScript Variables:

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows:

```
<script type="text/javascript">
<!--
var money;
var name;
//-->
</script>
```

You can also declare multiple variables with the same **var** keyword as follows:

```
<script type="text/javascript">
<!--
var money, name;
//-->
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or later point in time when you need that variable as follows:

For instance, you might create a variable named *money* and assign the value 2000.50 to it later. For another variable you can assign a value the time of initialization as follows:

```
<script type="text/javascript">
<!--
var name = "Ali";
var money;
money = 2000.50;
//-->
</script>
```

Note: Use the **var** keyword only for declaration or initialization. once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is *untyped* language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript alert(), confirm(), and prompt() functions
Alert Box
An alert box is often used if you want to make sure information comes through to the user.
When an alert box pops up, the user will have to click "OK" to proceed.
An alert box can be displayed using JavaScript's alert() function.
Syntax:
alert("*Text to display in the alert box*");
Example:

<input type="button" onclick="alert('Hi, I am an alert box!');" value="Click here for a message" />

Displaying a confirmation box
A confirmation box is used to let the user make a choice. A confirmation box will appear will with an "OK" button and a "Cancel" button. Different actions will occur depending on what button the user clicks. You can specify this course of action with conditional logic.
A confirmation box can be displayed using Javascript's confirm() function.
Syntax:
confirm("*Text to display in the confirmation box*");
Example:
<html> <head> <script type="text/javascript"> function confirmGetMessage() { //display a confirmation box asking the visitor if they want to get a message var theAnswer = confirm("Get a message?"); //if the user presses the "OK" button display the message "Javascript is cool!!" if (theAnswer){ alert("Javascript is cool."); } //otherwise display another message else{ alert("Here is a message anyway."); } } </script> </head> <body> <form> <input type="button" onclick="confirmGetMessage()" value="Click me for some reason" /> </form> </body> </html>
Output:

Displaying a prompt box
A prompt box is used to get data from the user. A prompt box will appear with an "OK" button and a "Cancel" button. Different actions will occur depending on what button the user clicks. If the user clicks the "OK" button, the value entered into the prompt box will be set. If the user clicks the "Cancel" button, a **null** value (empty string) will be set, or a default value if you set it as the second argument in the function.
A prompt box can be displayed using Javascript's prompt() function.
Syntax:
prompt("*Text to display in the prompt box*, "*Default value*");
Example:
<html> <head> <script type="text/javascript"> function promptMessage() { if (favColor != null){ alert("Your favorite color is " + favColor); } else { alert("You did not specify your favorite color"); } } </script> </head> <body> <form> <input type="button" onclick="promptMessage()" value="Click here to specify your favorite color" /> </form> </body> </html>

JavaScript Operators

What is an operator?

Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. JavaScript language supports following type of operators.

- Arithmetic Operators
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

The Arithmatic Operators:

There are following arithmatic operators supported by JavaScript language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denumerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Note: Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

The Comparison Operators:

There are following comparison operators supported by JavaScript language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Logical Operators:

There are following logical operators supported by JavaScript language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

The Assignment Operators:

There are following assignment operators supported by JavaScript language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Note: Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

JavaScript if...else Statements

While writing a program, there may be a situation when you need to adopt one path out of the given two paths. So you need to make use of conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain **if..else** statement.

JavaScript supports following forms of **if..else** statement:

- if statement
- if...else statement
- if...else if... statement.

if statement:

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax:

```
if (expression){
   Statement(s) to be executed if expression is true
}
```

Here JavaScript *expression* is evaluated. If the resulting value is *true*, given *statement(s)* are executed. If *expression* is *false* then no statement would be not executed. Most of the times you will use comparison operators while making decisions.

Example:

```
<script type="text/javascript">
<!--
var age = 20;
if( age > 18 ){
   document.write("<b>Qualifies for driving</b>");
}
//-->
</script>
```

if...else statement:

The **if...else** statement is the next form of control statement that allows JavaScript to execute statements in more controlled way.

Syntax:

```
if (expression){
   Statement(s) to be executed if expression is true
}else{
   Statement(s) to be executed if expression is false
}
```

Here JavaScript *expression* is evaluated. If the resulting value is *true*, given *statement(s)* in the *if* block, are executed. If *expression* is *false* then given *statement(s)* in the *else* block, are executed.

Example:

```
<script type="text/javascript">
<!--
var age = 15;
if( age > 18 ){
   document.write("<b>Qualifies for driving</b>");
}else{
   document.write("<b>Does not qualify for driving</b>");
}
//-->
</script>
```

if...else if... statement:

The **if...else if...** statement is the one level advance form of control statement that allows JavaScript to make correct decision out of several conditions.

Syntax:

```
if (expression 1){
   Statement(s) to be executed if expression 1 is true
}else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}else{
   Statement(s) to be executed if no expression is true
}
```

There is nothing special about this code. It is just a series of *if* statements, where each *if* is part of the *else* clause of the previous statement. Statement(s) are executed based on the true condition, if non of the condition is true then *else* block is executed.

Example:

```
<script type="text/javascript">
<!--
var book = "maths";
if( book == "history" ){
   document.write("<b>History Book</b>");
}else if( book == "maths" ){
   document.write("<b>Maths Book</b>");
}else if( book == "economics" ){
   document.write("<b>Economics Book</b>");
}else{
  document.write("<b>Unknown Book</b>");
}
//-->
</script>
```

JavaScript Switch Case

You can use multiple *if...else if* statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated *if...else if* statements.

Syntax:

The basic syntax of the **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

```
switch (expression)
{
  case condition 1: statement(s)
                    break;
  case condition 2: statement(s)
                    break;
   ...
  case condition n: statement(s)
                    break;
  default: statement(s)
}
```

The **break** statements indicate to the interpreter the end of that particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain **break** statement in *Loop Control* chapter.

Example:

Following example illustrates a basic while loop:

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
  case 'A': document.write("Good job<br />");
            break;
  case 'B': document.write("Pretty good<br />");
            break;
  case 'C': document.write("Passed<br />");
            break;
  case 'D': document.write("Not so good<br />");
```

```
            break;
  case 'F': document.write("Failed<br />");
            break;
  default:  document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
```

This will produce following result:

```
Entering switch block
Good job
Exiting switch block
```

Example:

Consider a case if you do not use **break** statement:

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
  case 'A': document.write("Good job<br />");
  case 'B': document.write("Pretty good<br />");
  case 'C': document.write("Passed<br />");
  case 'D': document.write("Not so good<br />");
  case 'F': document.write("Failed<br />");
  default:  document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
```

This will produce following result:

```
Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
```

Form

Let's say you have a form like this:

```
<form name="aa">
<input type="text" size="10" value="" name="bb"><br>
<input type="button"
value="Click Here"onclick="alert(document.aa.bb.value)">
</form>
```

Notice that we gave the names to the form and the element. So JavaScript can gain access to them.

onBlur

If you want to get information from users and want to check each element (ie: user name, password, email) individually, and alert the user to correct the wrong input before moving on, you can use `onBlur`. Let's see how `onblur` works:

```
<html><head><script>
function emailchk()
{
var x=document.feedback.email.value
if (x.indexOf("@")==-1)
{
        alert("It seems you entered an invalid email address.")
        document.feedback.email.focus()
}
}
</script></head><body>

<form
name="feedback">
Email:<input type="text" size="20" name="email"
onblur="emailchk()"><br>
Comment: <textarea name="comment" rows="2" cols="20"></textarea><br>
<input type="submit" value="Submit">
</form>
</body></html>
```

If you enter an email address without the `@`, you'll get an alert asking you to re-enter the data. What is: `x.indexOf(@)==-1`? This is a method that JavaScript can search every character within a string and look for what we want. If it finds it will return the position of the char within the string. If it doesn't, it will return -1. Therefore, `x.indexOf("@")==-1` basically means: "if the string doesn't include @, then:

```
alert("It seems you entered an invalid email address.")
document.feedback.email.focus()
```

What's `focus()`? This is a method of the text box, which basically forces the cursor to be at the specified text box. `onsubmit` Unlike `onblur, onsubmit` handler is inserted inside the `<form>` tag, and not inside any one element. Lets do an example:

```
<script>
<!--
function validate()
{
if(document.login.userName.value=="")
```

```
{
        alert ("Please enter User Name")
        return false
}
if(document.login.password.value=="")
{
        alert ("Please enter Password")
        return false
}
}
//-->
</script>
<form name="login" onsubmit="return validate()">
<input type="text" size="20" name="userName">
<input type="text" size="20" name="password">
<input type="submit" name="submit" value="Submit">
</form>
```

Note:
`if(document.login.userName.value=="")`. This means "If the box named userName of the form named login contains nothing, then...". return false. This is used to stop the form from submitting. By default, a form will return true if submitting. `return validate()` That means, "if submitting, then call the `function validate()`".

Protect a file by using Login

Let's try an example

```
<html><head>
<SCRIPT Language="JavaScript">
function checkLogin(x)
{
if ((x.id.value != "Sam")||(x.pass.value !="Sam123"))
{
        alert("Invalid Login");
        return false;
}
else
        location="main.htm"
}
</script>
</head><body>
<form>
<p>UserID:<input type="text" name="id"></p>
<p>Password:<input type="password" name="pass"></p>
<p><input type="button" value="Login" onClick="checkLogin(this.form)"></p>
</form>
</body></html>
```

`||` means "or", and ,`!=` indicates "not equal". So we can explain the script: "If the id does not equal 'Sam', or the password does not equal 'Sam123', then show an alert ('Invalid Login') and stop submitting. Else, open the page 'main.htm'".

Link

In most cases, a form can be repaced by a link:

```
<a href="JavaScript:window.location.reload()">Click to reload!</a>
```

More examples:

```
<a href="#" onClick="alert('Hello, world!')">Click me to say Hello</a><br>
<a href="#" onMouseOver="location='main.htm'">Mouse over to see Main
Page</a>
```

Date

Let's see an example:

```
<HTML><HEAD><TITLE>Show
Date</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
var x= new Date();
document.write (x);
</SCRIPT>
</BODY></HTML>
```

To activate a Date Object, you can do this: `var x=new Date()`. Whenever you want to create an instance of the date object, use this important word: new followed by the object name().

Dynamically display different pages

You can display different pages according to the different time. Here is an example:

```
var banTime= new Date()
var ss=banTime.getHours()
if (ss<=12)
        document.write("<img src='banner1.gif'>")
else
        document.write("<img src='banner2.gif'>")
```

Date object		
Methods		
getDate getTime getTimezoneOffset	getDay getMonth getYear	getSeconds getMinutes getHours

Window

Open a window

To open a window, simply use the method "window.open()":

```
<form>
<input type="button" value="Click here to see"
onclick="window.open('test.htm')">
</form>
```

You can replace `test.htm` with any URL, for example, with `http://www.yahoo.com`.

Size, toolbar, menubar, scrollbars, location, status

Let's add some of attributes to the above script to control the size of the window, and show: toolbar, scrollbars etc. The syntax to add attributes is:

```
pen("URL","name","attributes")
```

For example:

```
<form>
<input type="button" value="Click here to see"
onclick="window.open('page2.htm','win1','width=200,height=200,menubar')">
</form>
```

Another example with no attributes turned on, except the size changed:

```
<form>
<input type="button" value="Click here to see"
onclick="window.open('page2.htm','win1','width=200,height=200')">
</form>
```

Here is the complete list of attributes you can add:

width	height	toolbar
location	directories	status
scrollbars	resizable	menubar

Reload

To reload a window, use this method:

```
window.location.reload()
```

Close Window

Your can use one of the codes shown below:

```
<form>
<input type="button" value="Close Window" onClick="window.close()">
</form>
<a href="javascript:window.close()">Close Window</a>
```

Loading

The basic syntax when loading new content into a window is:

```
window.location="test.htm"
```

This is the same as

```
<a href="test.htm>Try this </a>
```

Let's provide an example, where a confirm box will allow users to choose between going to two places:

```
<script>
<!--
function ss()
```

```
{
var ok=confirm('Click "OK" to go to yahoo, "CANCEL" to go to hotmail')
if (ok)
location="http://www.yahoo.com"
else
location="http://www.hotmail.com"
}
//-->
</script>
```

Remote Control Window

Let's say you have opened a new window from the current window. After that, you will wonder how to make a control between the two windows. To do this, we need to first give a name to the window.Look at below:

```
aa=window.open('test.htm','','width=200,height=200')
```

By giving this window a name "aa", it will give you access to anything that's inside this window from other windows. Whenever we want to access anything that's inside this newly opened window, for example, to write to this window, we would do this:
aa.document.write("This is a test.").

Now, let's see an example of how to change the background color of another window:

```
<html><head><title></title></head>
<body>
<form>
<input type="button" value="Open another page"
onClick="aa=window.open('test.htm','','width=200,height=200')">
<input type="radio" name="x" onClick="aa.document.bgColor='red'">
<input type="radio" name="x" onClick="aa.document.bgColor='green'">
<input type="radio" name="x" onClick="aa.document.bgColor='yellow'">
</form>
</body></html>
```

opener

Using "opener" property, we can access the main window from the newly opened window.

Let's create Main page:

```
<html>
<head>
<title></title>
</head>
<body>
<form>
<input type="button" value="Open another page"
onClick="aa=window.open('test.htm','','width=100,height=200')">
</form>
</body>
</html>
```

Then create Remote control page (in this example, that is test.htm):

```
<html>
<head>
<title></title>
<script>
function remote(url){
window.opener.location=url
}
</script>
</head>
<body>
<p><a href="#" onClick="remote('file1.htm')">File
1</a></p>
<p><a href="#" onClick="remote('file2.htm')">File
2</a></p>
</body>
</html
```

Accessing Values

Having read the Objects and Properties page, you should now know how to find out the values of form elements through the DOM. We're going to be using the name notation instead of using numbered indexes to access the elements, so that you are free to move around the fields on your page without having to rewrite parts of your script every time. A sample, and simple, form may look like this:

```
<form name="feedback" action="script.cgi" method="post" onSubmit="return
checkform()">
<input type="text" name="name">
<input type="text" name="email">
<textarea name="comments"></textarea>
</form>
```

Validating this form would be considerably simpler than one containing radio buttons or select boxes, but any form element can be accessed. Below are the ways to get the value from all types of form elements. In all cases, the form is called feedback and the element is called field.

Text Boxes, <textarea>s and hiddens

These are the easiest elements to access. The code is simply

```
document.feedback.field.value
```

You'll usually be checking if this value is empty, i.e.

```
if (document.feedback.field.value == '') {
        return false;
}
```

That's checking the value's equality with a null String (two single quotes with nothing between them). When you are asking a reader for their email address, you can use a simple » address validation function to make sure the address has a valid structure.

Select Boxes

Select boxes are a little trickier. Each option in a drop-down box is indexed in the array options[], starting as always with 0. You then get the value of the element at this index. It's like this:

```
document.feedback.field.options
[document.feedback.field.selectedIndex].value
```

You can also change the selected index through JavaScript. To set it to the first option, execute this:

```
document.feedback.field.selectedIndex = 0;
```

Check Boxes

Checkboxes behave differently to other elements — their value is always on. Instead, you have to check if their Boolean checked value is true or, in this case, false.

```
if (!document.feedback.field.checked) {
        // box is not checked
        return false;
}
```

Naturally, to check a box, do this

```
document.feedback.field.checked = true;
```

Radio Buttons

Annoyingly, there is no simple way to check which radio button out of a group is selected — you have to check through each element, linked with Boolean AND operators . Usually you'll just want to check if *none* of them have been selected, as in this example:

```
if (!document.feedback.field[0].checked &&
!document.feedback.field[1].checked &&
!document.feedback.field[2].checked) {
        // no radio button is selected
        return false;
}
```

You can check a radio button in the same way as a checkbox.

JavaScript Event Handlers

An event handler executes a segment of a code based on certain events occurring within the application, such as onLoad, onClick. JavaScript event handers can be divided into two parts: interactive event handlers and non-interactive event handlers. An interactive event handler is the one that depends on the user interactivity with the form or the document. For example, onMouseOver is an interactive event handler because it depends on the users action with the mouse. On the other hand non-interactive event handler would be onLoad, because this event handler would automatically execute JavaScript code without the user's interactivity. Here are all the event handlers available in JavaScript:

Event Handler	**Used In**
onAbort	image
onBlur	select, text, text area
onChange	select, text, textarea
onClick	button, checkbox, radio, link, reset, submit, area
onError	image
onFocus	select, text, testarea
onLoad	windows, image
onMouseOver	link, area
onMouseOut	link, area
onSelect	text, textarea
onSubmit	form
onUnload	window

onAbort:
An onAbort event handler executes JavaScript code when the user aborts loading an image.

See Example:

```
<HTML>
<TITLE>Example of onAbort Event Handler</TITLE>
<HEAD>
</HEAD>

<BODY>
<H3>Example of onAbort Event Handler</H3>
<b>Stop the loading of this image and see what happens:</b><p>
<IMG SRC="reaz.gif" onAbort="alert('You stopped the loading the image!')">
</BODY>
</HTML>
```

Here, an alert() method is called using the onAbort event handler when the user aborts loading the image.

onBlur:
An onBlur event handler executes JavaScript code when input focus leaves the field of a text, textarea, or a select option. For windows, frames and framesets the event handler executes JavaScript code when the window loses focus. In windows you need to specify the event handler in the <BODY> attribute. For example:

<BODY BGCOLOR='#ffffff' onBlur="document.bgcolor='#000000'">
Note: On a Windows platform, the onBlur event does not work with <FRAMESET>.

```
<HTML>
<TITLE>Example of onBlur Event Handler</TITLE>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function valid(form){
   var input=0;
   input=document.myform.data.value;
            if (input<0){
               alert("Please input a value that is less than 0");
            }
}
</SCRIPT>
</HEAD>
<BODY>
<H3> Example of onBlur Event Handler</H3>
Try inputting a value less than zero:<br>
<form name="myform">
 <input type="text" name="data" value="" size=10 onBlur="valid(this.form)">
</form>
</BODY>
</HTML>
```

In this example, 'data' is a text field. When a user attempts to leave the field, the onBlur event handler calls the valid() function to confirm that 'data' has a legal value. Note that the keyword *this* is used to refer to the current object.

onChange:
The onChange event handler executes JavaScript code when input focus exits the field after the user modifies its text.

```
<HTML>
<TITLE>Example of onChange Event Handler</TITLE>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function valid(form){
   var input=0;
   input=document.myform.data.value;
   alert("You have changed the value from 10 to " + input );
}
</SCRIPT>
</HEAD>
```

```
<BODY>
<H3>Example of onChange Event Handler</H3>
Try changing the value from 10 to something else:<br>
<form name="myform">
 <input type="text" name="data" value="10" size=10 onChange="valid(this.form)">
</form>
</BODY>
</HTML>
```

In this example, 'data' is a text field. When a user attempts to leave the field after a change of the original value, the onChange event handler calls the valid() function which alerts the user about value that has been inputted.

onClick:

In an onClick event handler, JavaScript function is called when an object in a button (regular, radio, reset and submit) is clicked, a link is pushed, a checkbox is checked or an image map area is selected. Except for the regular button and the area, the onClick event handler can return false to cancel the action. For example:

```
<INPUT TYPE="submit" NAME="mysubmit" VALUE="Submit" onClick="return
confirm(`Are you sure you want to submit the form?')">
```

Note: On Windows platform, the onClick event handler does not work with reset buttons.

```
<HTML>
<TITLE>Example of onClick Event Handler</TITLE>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function valid(form){
   var input=0;
   input=document.myform.data.value;
   alert("Hello " + input + " ! Welcome...");
}
</SCRIPT>
</HEAD>
<BODY>
<H3> Example of onClick Event Handler </H3>
Click on the button after inputting your name into the text box:<br>
<form name="myform">
 <input type="text" name="data" value="" size=10>
<INPUT TYPE="button" VALUE="Click Here" onClick="valid(this.form)">
</form>
</BODY>
</HTML>
```

In this example, when the user clicks the button "Click Here", the onClick event handler calls the function valid().

onFocus:

An onFocus event handler executes JavaScript code when input focus enters the field either by tabbing in or by clicking but not selecting input from the field. For windows, frames and

framesets the event handler executes JavaScript code when the window gets focused. In windows you need to specify the event handler in the <BODY> attribute. For example:

<BODY BGCOLOR="#ffffff" onFocus="document.bgcolor='#000000'">
Note: On a Windows platform, the onFocus event handler does not work with <FRAMESET>.

```
<HTML>
<TITLE>Example of onFocus Event Handler</TITLE>
<HEAD></HEAD>
<BODY>
<H3>Example of onFocus Event Handler</H3>
Click your mouse in the text box:<br>
<form name="myform">
 <input type="text" name="data" value="" size=10 onFocus='alert("You focused the
textbox!!")'>
</form>
</BODY>
</HTML>
```

In the above example, when you put your mouse on the text box, an alert() message displays a message.

onLoad:

An onLoad event occurs when a window or image finishes loading. For windows, this event handler is specified in the BODY attribute of the window. In an image, the event handler will execute handler text when the image is loaded. For example:

```
<IMG NAME="myimage" SRC="http://rhoque.com/ad_rh.jpg" onLoad="alert('You loaded
myimage')">
<HTML>
<TITLE>Example of onLoad Event Handler</TITLE>
<HEAD>
<SCRIPT LANGUGE="JavaScript">
function hello(){
   alert("Hello there...\nThis is an example of onLoad.");
}
</SCRIPT>
</HEAD>
<BODY onLoad="hello()">
<H3>Example of onLoad Event Handler</H3>
</BODY>
</HTML>
```

The example shows how the function hello() is called by using the onLoad event handler.

onMouseOver:

JavaScript code is called when the mouse is placed over a specific link or an object or area from outside that object or area. For area object the event handler is specified with the <AREA> tag. For example:

```
<MAP NAME="mymap">
<AREA NAME="FirstArea" COORDS="0,0,49,25" HREF="mylink.html"
        onMouseOver="self.status='This will take you to mylink.html'; return true">
</MAP>

<HTML>
<TITLE>Example of onMouseOver Event Handler</TITLE>
<HEAD>
</HEAD>
<BODY>
<H3> Example of onMouseOver Event Handler </H3>
Put your mouse over <A HREF="" onMouseOver="window.status='Hello! How are you?' ;
return true;"> here</a>
and look at the status bar (usually at the bottom of your browser window).
</BODY>
</HTML>
```

In the above example when you point your mouse to the link, the text "Hello! How are you?" appears on your window's status bar.

onMouseOut:

JavaScript code is called when the mouse leaves a specific link or an object or area from outside that object or area. For area object the event handler is specified with the <AREA> tag.

```
<HTML>
<TITLE> Example of onMouseOut Event Handler </TITLE>
<HEAD>
</HEAD>
<BODY>
<H3> Example of onMouseOut Event Handler </H3>
Put your mouse over <A HREF="" onMouseOut="window.status='You left the link!' ; return
true;"> here</a> and then take the mouse pointer away.
</BODY>
</HTML>
```

In the above example, after pointing your mouse and leaving the link , the text "You left the link!" appears on your window's staus bar.

onReset:

A onReset event handler executes JavaScript code when the user resets a form by clicking on the reset button.

```
<HTML>
<TITLE>Example of onReset Event Handler</TITLE>
<HEAD></HEAD>
<BODY>
<H3> Example of onReset Event Handler </H3>
Please type something in the text box and press the reset button:<br>
<form name="myform" onReset="alert('This will reset the form!')">
 <input type="text" name="data" value="" size="20">
<input type="reset" Value="Reset Form" name="myreset">
```

```
</form>
</BODY>
</HTML>
```
In the above example, when you push the button, "Reset Form" after typing something, the alert method displays the message, "This will reset the form!"

onSelect:
A onSelect event handler executes JavaScript code when the user selects some of the text within a text or textarea field.

```
<HTML>
<TITLE>Example of onSelect Event Handler</TITLE>
<HEAD></HEAD>
<BODY>
<H3>Example of onSelect Event Handler</H3>
Select the text from the text box:<br>
<form name="myform">
<input type="text" name="data" value="Select This" size=20 onSelect="alert('This is an example of onSelect!!')">
</form>
</BODY>
</HTML>
```
In the above example, when you try to select the text or part of the text, the alert method displays the message, "This is an example of onSelect!!".

onSubmit:
An onSubmit event handler calls JavaScript code when the form is submitted.

```
<HTML>
<TITLE> Example of onSubmit Event Handler </TITLE>
<HEAD>
</HEAD>
<BODY>
<H3>Example of onSubmit Event Handler </H3>
Type your name and press the button<br>
<form name="myform" onSubmit="alert('Thank you ' + myform.data.value +'!')">
<input type="text" name="data">
<input type="submit" name ="submit" value="Submit this form">
</form>
</BODY>
```
In this example, the onSubmit event handler calls an alert() function when the button "Sumbit this form" is pressed.

onUnload:
An onUnload event handler calls JavaScript code when a document is exited.

```
<HTML>
<TITLE>Example 2.11</TITLE>
<HEAD>
<SCRIPT LANGUGE="JavaScript">
```

```
function goodbye(){
      alert("Thanks for Visiting!");
      }
</SCRIPT>
</HEAD>
<BODY onUnLoad="goodbye()">
<H3>Example of onUnload event handler</H3>
Look what happens when you try to leave this page...
</BODY>
</HTML>
```

In this example, the onUnload event handler calls the Goodbye() function as user exits a document.

Note: You can also call JavaScript code via explicit event handler call. For example say you have a function called myfunction(). You could call this function like this:

```
document.form.mybotton.onclick=myfunction
```

Notice that you don't need to put () after the function and also the event handler has to be spelled out in lowercase.

JavaScript Functions

A function is a group of reusable code which can be called anywhere in your programme. This eliminates the need of writing same code again and again. This will help programmers to write modular code. You can divide your big programme in a number of small and manageable functions.

Like any other advance programming language, JavaScript also supports all the features necessary to write modular code using functions.

You must have seen functions like *alert()* and *write()* in previous chapters. We are using these function again and again but they have been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section will explain you how to write your own functions in JavaScript.

Functions

Before we use a function we need to define that function. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces. The basic syntax is shown here:

```
<script type="text/javascript">
<!--
```

```
function functionname(parameter-list)
{
  statements
}
//-->
</script>
```

Example:

A simple function that takes no parameters called sayHello is defined here:

```
<script type="text/javascript">
<!--
function sayHello()
{
   alert("Hello there");
}
//-->
</script>
```

Calling a Function:

To invoke a function somewhere later in the script, you would simple need to write the name of that function as follows:

```
<script type="text/javascript">
<!--
sayHello();
//-->
</script>
```

Function Parameters:

Till now we have seen function without a parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters.

A function can take multiple parameters separated by comma.

Example:

Let us do a bit modification in our *sayHello* function. This time it will take two parameters:

```
<script type="text/javascript">
<!--
```

```
function sayHello(name, age)
{
   alert( name + " is " + age + " years old.");
}
//-->
</script>
```

Note: We are using + operator to concatenate string and number all together. JavaScript does not mind in adding numbers into strings.

Now we can call this function as follows:

```
<script type="text/javascript">
<!--
sayHello('Zara', 7 );
//-->
</script>
```