UNIT-2

Lecture-13

Java Script

JavaScript is the premier client-side ***interpreted scripting language***. It‟s widely used in tasks ranging from the validation of form data to the creation of complex user interfaces. ***Dynamic HTML*** is a combination of the content formatted using HTML, CSS, Scripting language and DOM. By combining all of these technologies, we can create interesting and interactive websites.

History of JavaScript:

Netscape initially introduced the language under the name **LiveScript** in an early beta release of Navigator 2.0 in 1995, and the focus was on form validation. After that, the language was renamed JavaScript. After Netscape introduced JavaScript in version 2.0 of their browser, Microsoft introduced a clone of JavaScript called **JScript** in Internet Explorer 3.0.

What a JavaScript can do?

JavaScript gives web developers a programming language for use in web pages & allows them to do the following:

- JavaScript gives HTML designers a programming tool
- JavaScript can be used to validate data
- JavaScript can read and write HTML elements
- Create pop-up windows
- Perform mathematical calculations on data
- React to events, such as a user rolling over an image or clicking a button
- Retrieve the current date and time from a user‟s computer or the last time a document was modified
- Determine the user‟s screen size, browser version, or screen resolution
- JavaScript can put dynamic text into an HTML page
- JavaScript can be used to create cookies

Advantages of JavaScript:

- Less server interaction
- Immediate feedback to the visitors
- Increased interactivity
- Richer interfaces
- Web surfers don‟t need a special plug-in to use your scripts
- Java Script is relatively secure.

Limitations of JavaScript:

We cannot treat JavaScript as a full-fledged programming language. It lacks some of the important features like:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocess capabilities.
- If your script doesn‟t work then your page is useless.

Points to remember:

- JavaScript is case-sensitive
- Each line of code is terminated by a semicolon

- Variables are declared using the keyword **var**
- Scripts require neither a **main** function nor an **exit** condition. There are major differences between scripts and *proper* programs. Execution of a script starts with the first line of code & runs until there is no more code

JavaScript comments:

In JavaScript, each line of comment is preceded by two slashes and continues from that point to the end of the line.

//this is javascript comment

Block comments or Multiline comments: **/* */**

JavaScript is not the same as Java, which is a bigger programming language (although there are some similarities)

JavaScript and HTML Page

Having written some JavaScript, we need to include it in an HTML page. We can‟t execute these scripts from a command line, as the interpreter is part of the browser. The script is included in the web page and run by the browser, usually as soon as the page has been loaded. The browser is able to debug the script and can display errors.

Embedding JavaScript in HTML file:

If we are writing small scripts, or only use our scripts in few pages, then the easiest way is to include the script in the HTML code. The syntax is shown below:

```
<html>
<head>
<script language="javascript">
<!- -
Javascript code here
//- - >
</head>
<body>
……
</body>
</html>
```

Using External JavaScript in HTML file:

If we use lot of scripts, or our scripts are complex then including the code inside the web page will make the source file difficult to read and debug. A better idea is to put the JavaScript code in a separate file and include that code in the HTML file. By convention, JavaScript programs are stored in files with the **.js** extension.

```
<html>
<head>
<script language=”javascript” src=”sample.js”> </script>
</head>
<body>
……
</body>
</html>
```

POPUP BOXES IN JAVASCRIPT

alert(“string”) opens box containing the message

confirm(“string”) displays a message box with OK and CANCEL buttons

prompt(“string”) displays a prompt window with field for the user to enter a text string

Example:

```
<html>
<head>
<script language="javascript">
function show_alert()
{
alert("Hi! This is alert box!!");
}
</script>
</head>
<body>
<input type="button" onclick="show_alert()" value=“Display alert box" > </input>
</body>
</html>
```

Unit-2
Lecture-14

JavaScript Programming Elements

• Variables, datatypes, operators
• Statements
• Arrays
• Functions
• Objects in JavaScript
• Exception Handling
• Events
• Dynamic HTML with JavaScript

VARIABLES

Like any programming language, JavaScript has variables. A variable is a name assigned to computer memory location to store data. As the name suggests, the value of the variable can vary, as the program runs. We can create a variable with the **var** statement:

var <variablename> = <some value>;

Example:

var sum = 0;
var str;
We can initialize a variable like this:
str = “hello”;

Rules for variable names:

They must begin with a letter, digit or underscore character # We can‟t use spaces in names
Variable names are case sensitive # We can‟t use reserved word as a variable name.

Weakly Typed Language:

 Most high-level languages, including C and Java, are ***strongly typed*.** That is, a variable must be declared before it is used, and its type must be included in its declaration. Once a variable is declared, its type cannot be changed.

 As the JavaScript is***weakly typed*** language, data types are not explicitly declared.

Example: var num;
num = 3;
num = "San Diego";

First, when the variable ***num*** is declared, it is empty. Its data type is actually the type **undefined**. Then we assign it to the number 3, so its data type is **numeric**. Next we reassign it to the string "San Diego", so the variable‟s type is now **string.**

Example:

```
<html>
<body>
<script language=”javascript” type=”text/javascript”>
var s;
s = “Hello”;
alert(typeof s);
s = 54321;
alert(typeof s);
</script> </body> </html>
```

DATATYPES
• JavaScript supports five primitive data types:

number string boolean undefined null.
• These types are referred to as ***primitive types*** because they are the basic building blocks from which more complex types can be built.
• Of the five, only **number, string**, and **boolean** are real data types in the sense of actually storing data.
• Undefined and null are types that arise under special circumstances.

Numeric Data Type:
 These are numbers and can be integers (such as 2, 22 and 2000) or floating-point values (such as 23.42, -56.01, and 2E45).
 Valid ways to specify numbers in JavaScript
10 177.5 -2.71 .333333e77 -1.7E12 3.E-5 128e+100
 We can represent integers in one of the following 3 ways:
Decimal: The usual numbers which are having the base 10 are the decimal numbers
Octal: Octal literals begin with a leading zero, and they consist of digits from zero through seven. The following are all valid octal literals:
00 0777 024513600
HexaDecimal: Hexadecimal literals begin with a leading 0x, and they consist of digits from 0 through 9 and letters A through F. The following are all valid hexadecimal literals:
0x0 0XF8f00 0x1a3C5e7

Special Numeric Values: Special value	Result of	Comparisions
Infinity, -Infinity	Number too large or too small to represent	All infinity values compare equal to each other
NaN	Undefined Operation	NaN never compares equal to anything, even to itself

Operators in JavaScript
The operators in JavaScript can be classified as follows:
 Arithmetic operators
 Relational operators
 Logical operators
 Assignment operators

Arithmetic operators:

-

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Note: If the arguments of + are numbers then they are added together. If the arguments are strings then they are concatenated and result is returned.

Example:

```
<html>
<body>
<script language="JavaScript">
<!--
var a = 5;
++a;
alert("The value of a = " + a );
-->
</script>
</body>
</html>
```

String (+) Operator:

Example:

```
txt1="Welcome";
txt2="to L&T Infotech Ltd.!";
txt3=txt1 + " " + txt2;
(or)
txt1="Welcome ";
txt2="to CMRCET.!";
```

txt3=txt1 + txt2;

Relational operators/Comparison operators: Relational operators are used to compare quantities.

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Conditional Operator: Conditional operator is one the JavaScript"s comparison operator, which assigns a value to a variable based on some condition.

Syntax :

variablename=(condition)? value1 : value2;

Logical operators: Logical operators are used to combine two or more conditions.

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Example (Logical operators):

```
<html>
<head>
<title>Operator Example</title>
</head>
<body>
<script language="JavaScript">
<!--
var userID ;
var password;
userID = prompt("Enter User ID"," ");
password = prompt("Enter Password"," ");
```

```
if(userID == "user" && password == "secure")
alert("Valid login");
else
alert("Invalid login");
-->
</script>
</body>
</html>
```

Assignment Operators: are used to assign the result of an expression to a variable.

Operator	Example	It is Same as
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

The *typeof* operator

typeof operator is used to verify the type of the variable or value.
The ***typeof*** operator takes one parameter: the variable or value to check
Calling ***typeof*** on a variable or value returns one of the following values:
– “undefined” if the variable is of the Undefined type.

– “boolean” if the variable is of the Boolean type.

– “number” if the variable is of the Number type.

– “string” if the variable is of the String type.

– “object” if the variable is of a reference type or of the Null type

Example:

```
var s = “test string”;
alert(typeof s); //outputs “string”
alert(typeof 95); //outputs “number”
alert(typeof window); //outputs “object”
```

STATEMENTS

Programs are composed of two things : data and code (set of statements) which manipulates the data. Java script Statements can be divided into the following categories:

– Conditional Statements

– Looping Statements

– Jumping Statements

Conditional statements: Conditional statements are used to make decisions.
Various conditional statements in JavaScript:

- Various forms of**if**
- switch

Various forms of if:

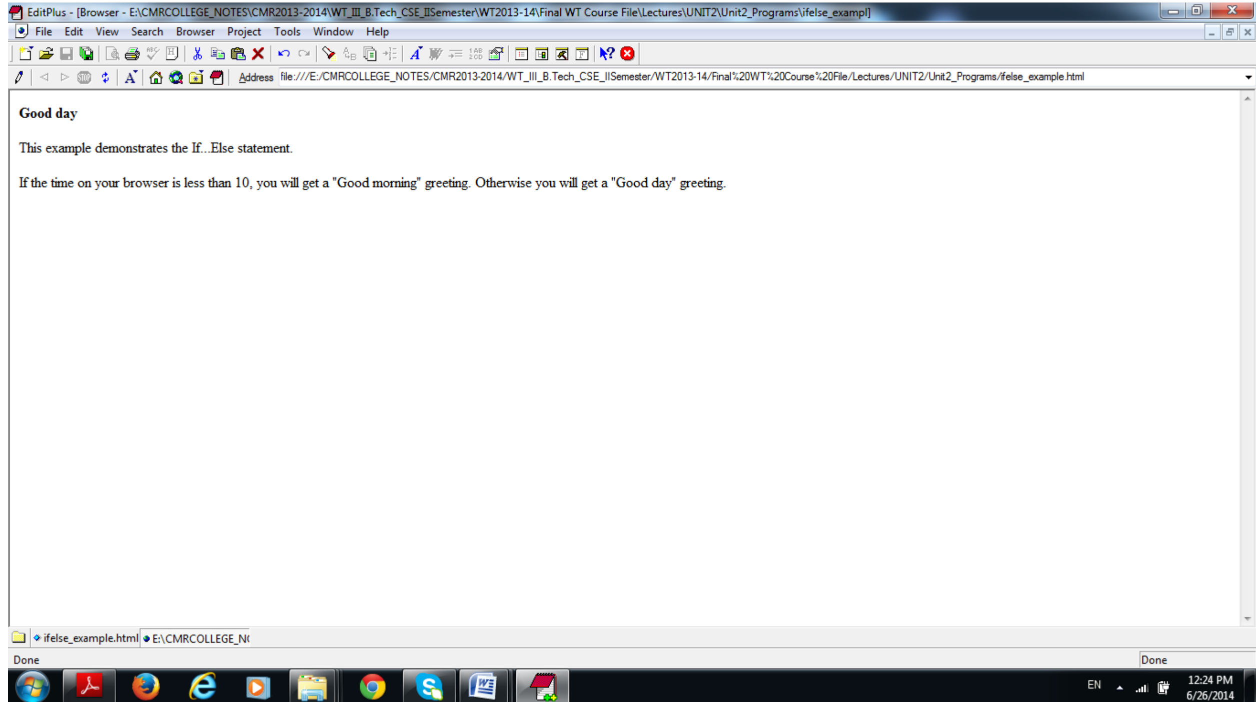
Simple if
if-else Statement
nested if
else-if ladder

Example:

```
<html>
<body>
<script language=“javascript”>
var d = new Date();
var time = d.getHours();
if (time < 10)
document.write("<b>Good morning</b>");
else
document.write("<b>Good day</b>");
</script>
<p> This example demonstrates the If...Else statement. </p>
<p> If the time on your browser is less than 10, you will get a "Good morning" greeting.
Otherwise you will get a "Good day" greeting. </p>
</body>
</html>
```

Output:

K.Yellaswamy ,AssistantProfessor|CMR College of Engineering & Technology
E-mail:toyellaswamy@gmail.com



switch statement:

A **switch** statement allows a program to evaluate an expression and attempt to match the expression's value to a case label. If a match is found, the program executes the associated statement.

Syntax

```
switch (expression)
{
case label1: block1
break;
case label2: block2
break;
….
default: def block;
}
```

Example:

```
<html>
<body>
<script language="“javascript”">
var d = new Date();
var theDay=d.getDay();
switch (theDay)
{
case 5: document.write("<b>Finally Friday</b>"); break;
```

K.Yellaswamy ,AssistantProfessor|CMR College of Engineering & Technology
E-mail:toyellaswamy@gmail.com

```
case 6: document.write("<b>Super Saturday</b>"); break;
case 0: document.write("<b>Sleepy Sunday</b>"); break;
default: document.write("<b>I'm really looking forward to this weekend!</b>");
}
</script>
<p>This JavaScript will generate a different greeting based on what day it is. Note that
Sunday=0, Monday=1, Tuesday=2, etc.</p>
</body> </html>
```

Looping statements: Loops are used to execute certain statements repeatedly. The various loops in JavaScript are:

- **for** loop
- **while** loop
- **do-while** loop

for loop syntax:

```
for(initialization; condition; loop variable update)
{
Set of statements
}
```

Example (for loop):

```
<html>
<body>
<script language="“javascript”">
for (var i = 1; i <= 6; i++)
{
document.write("<h" + i + ">This is header " + i + "</h" + i + ">");
}
</script>
</body>
</html>
```

for- in loop:

The **for-in** loop has a special use to enumerate all the properties contained within an object. This loop is rarely used in regular JavaScript.

Example: Display **window** object properties

```
var i, a = “”;
for( i in window)
a += i + “…”;
alert(a);
```

while loop: executes the statements as long as the condition is true.

Syntax: while(condition)

```
{
Set of statements
}
```

Example:

```
<html>
<body>
<script language="“javascript”">
```

K.Yellaswamy ,AssistantProfessor|CMR College of Engineering & Technology
E-mail:toyellaswamy@gmail.com

```
var i=0;
while (i<=10)
{
document.write("The number is " + i);
document.write("<br />");
i=i+1;
}
</script>
</body>
</html>
```

do-while syntax:

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

Jumping Statements:

break statement:

break statement is used to terminate a loop, switch, or label statement.
When we use break without a label, it terminates the innermost enclosing while, do-while, for, or switch immediately and transfers control to the following statement.
When we use break with a label, it terminates the specified labeled statement

Syntax:

– **break;**

– **break *label*;**

continue statement:

When we use **continue** without a label, it terminates the current iteration of the innermost enclosing while, do-while or for statement and continues execution of the loop with the next iteration.
When we use **continue** with a label, it applies to the looping statement identified with that label.

Syntax:

– **continue;**

– **continue *label*;**

WORKING WITH ARRAYS

An array is an ordered set of data elements which can be accessed through a single variable name. In many programming languages arrays are contiguous areas of memory which means that the first element is physically located next to the second and so on. In JavaScript, an array is slightly different because it is a special type of object and has functionality which is not normally available in other languages.

Basic Array Functions: The basic operations that are performed on arrays are creation, addition of elements (inserting elements), accessing individual elements, removing elements.
Creating Arrays: We can create arrays in several ways:

- var arrayObjectName = [element0, element1, ..., element N];
- var arrayObjectName = new Array(element0, element1, ..., element N);
- var arrayObjectName = new Array(arrayLength);

Ex:

- var colors = ["Red", "Green", "Blue"];
- var colors = new Array("Red", "Green", "Blue");
- var colors = Array("Red", "Green", "Blue");
- var thirdArray = [,,,,];
- var fourthArray = [,,35,,,16,,23,];

Note: JavaScript arrays can hold mixed data types as the following example shows:
var a = [“Monday”, 34, 45.7, “Tuesday”];
Accessing Array Elements:
Array elements are accessed through their ***index***. The **length** property can be used to know the length of the array. The index value runs from **0** to **length-1**.
Example:

```
<script language="javascript">
var a = [1,2,3];
var s = "";
for(var i=0;i<a.length;i++)
{
s += a[i] + " ";
}
alert(s);
</script>
```

Adding elements to an array:
What happens if we want to add an item to an array which is already full? Many languages struggle with this problem. But JavaScript has a really good solution: ***the interpreter simply extends the array and inserts the new item.***
Ex: var a = [“vit”,”svecw”,”sbsp”];
a[3] = “bvrit”;
a[10] = “bvrice”; //this extends the array and the values of elements a[4] to a[9] will be **undefined**.

Modifying array elements:
Array element values can be modified very easily.
Ex: To change a[1] value to "vdc" simply write:
a[1] = "vdc";
Searching an Array:
To search an array, simply read each element one by one & compare it with the value that we are looking for.
Removing Array Members:
JavaScript doesn"t provide a builtin function to remove array element. To do this, we can use the following approach:
- read each element in the array
- if the element is not the one you want to delete, copy it into a temporary array
- if you want to delete the element then do nothing
- increment the loop counter
- repeat theprocess
- finally store the temporary array reference in the main array variable

Note: The statement **delete a[0]** makes the value of a[0] **undefined**
OBJECT-BASED ARRAY FUNCTIONS:
In JavaScript, an array is an object. So, we can use various member functions of the object to manipulate arrays.
concat()
The **concat()** method returns the array resulting from concatenating argument arrays to the array on which the method was invoked. The original arrays are unaltered by this process.
Syntax: array1.concat(array2 [, array3,…arrayN]);
Example:
```
<script language="javascript">
var a = [1,2,3];
var b = ["a","b"];
alert(a.concat(b));
</script>
```
join()
join() method allows to join the array elements as strings separated by given specifier.The original array is unaltered by this process.
Syntax: arrayname.join(separator);

Example:

```
<script language="javascript">
var a = [1,2,3];
alert(a.join("#"));
</script>
```

push()

push() function adds one or more elements to the end of an array and returns the last element added.

Syntax: arrayname.push(element1 [, element2, ..elementN]);

Example:

```
<script language="javascript">
var a = [1,2,3];
alert(a.push(4,5)); //displays 5
alert(a); //displays 1,2,3,4,5
</script>
```

pop()

pop() removes the last element from the array and returns that element

Syntax: arrayname.pop();

reverse()

reverse() method transposes the elements of an array: the first array element becomes the last and the last becomes the first. The original array is altered by this process.

Syntax: arrayname.reverse();

Example:

```
<script language="javascript">
var a = [1,2,3];
a.reverse();
alert(a);
</script>
```

shift()

shift() removes the first element of the array and in doing so shortens its length by one. It returns the first element that is removed.

Ex: var a = [1, 2, 3];

```
var first = a.shift();
alert(a); // 2,3
alert(first); //1
```

unshift()

unshift() adds one or more elements to the front of an array.

Syntax: arrayname.unshift(element1 [, element2, ..elementN]);

Example:

var a = ["x","y","z"];
a.unshift("p","q");
alert(a); //**p,q,x,y,z**

slice()

slice() returns a “slice” (subarray) of the array on which it is invoked. The method takes two arguments, the *start* and *end* index, and returns an **array** containing the elements from index *start* up to but not including index *end*. If we specify only first parameter, then array containing the elements from *start* to the end of the array are returned. The original array is unaltered by this process.

Syntax: arrayname.slice(startindex , endindex);

Example:

var a = [1, 2, 3, 4, 5];
a.slice(2); // returns [3, 4, 5]
a.slice(1, 3); // returns [2, 3]

splice()

The **splice()** method can be used to add, replace, or remove elements of an array in place. Any elements that are removed are returned. It takes a variable number of arguments, the first two arguments are mandatory. The original array is altered by this process.

Syntax : arrayname.splice(*start*, *deleteCount*, *replacevalues*);

The first argument *start* is the index at which to perform the operation.

The second argument is *deleteCount*, the number of elements to delete beginning with index *start*. If we don‟t want to delete any elements specify this value as 0.

Any further arguments represented by *replacevalues* (that are comma-separated, if more than one) are inserted in place of the deleted elements.

Example:

var myArray = [1, 2, 3, 4, 5];
myArray.splice(3,2,"a","b"); // returns 4,5
alert(a); //**1,2,3,a,b**

sort()

sort() method sorts the array into lexicographic order. Elements which are not text are converted into strings before the sort operation is performed. This means, for example,732 will be placed before 80 in the sorted array. Original array is altered by this process.

Example1:
var myArray = ["cse","ece","eee"];
myArray.sort();
alert(myArray);
Example2:
var a = [80,732,450];
a.sort();
alert(a); //450,732,80
****

STRINGS

String is a set of characters enclose in a pair of single quotes or double quotes. In JavaScript using string object, many useful string related functionalities can be done. Some commonly used methods of string object are concatenating two strings, converting the string to uppercase or lowercase, finding the substring of a given string and so on.

PROPERTY:

length

This property of string returns the length of the string

Example: “cmr”.length //gives 3

METHODS:

charAt(index)

This method returns the character specified by index

Example: alert(“cmrcet”.charAt(2)**);**

indexOf(substring [, offset])

This method returns the index of substring found in the main string. If the substring is not found returns **-1**. By default the indexOf() function starts index **0**, however, an optional offset may be specified, so that the search starts from that position.

Example: “cmrcet”.indexOf(“sv”);
“Department of CSE”.indexOf(“CS”,5);

lastIndexOf(substring [,offset])

This method returns the index of substring found in the main string (i.e. last occurence). If the substring is not found returns **-1**. By default the lastIndexOf() function starts at index **string.length-1**, however, an optional offset may be specified, so that the search starts from that position in backwards.

Example: “cmrcet”.lastIndexOf(“cet”); //returns
“cmrcet”.lastIndexOf(“cet”,6); //returns

str1.concat(str2 [,str3 ..strN])
This method is used to concatenate strings together. For example, s1.concat(s2) returns the concatenated string of s1 and s2. The original strings don"t get altered by this operation.
substring(*start* [,*end*] **)**
This method returns the substring specified by ***start*** and ***end*** indices (upto ***end*** index, but not the character at ***end*** index). If the ***end*** index is not specified, then this method returns substring from ***start*** index to the end of the string.
Example: "vitsvecw".substring(3,6); //returns **sve**
"vitsvecw".substring(3); //returns **svecw**
substr(index [,length])
This method returns substring of specified number of characters (**length**), starting from **index**. If the length is not specified it returns the entire substring starting from **index**.
Example: "vitsvecw".substr(3,2); //returns **sv**
"vitsvecw".substr(3); //returns **svecw**
toLowerCase( )
returns the string in lower case. The original string is not altered by this operation.
Example:

```
<script language="javascript">
var s="CMRcet";
alert(s.toLowerCase());
alert(s); /
</script>
```

toUpperCase( )
returns the string in upper case. The original string is not altered by this operation.
Example:

```
<script language="javascript">
var s="Cmrcet";
alert(s.toUpperCase()); //displays
alert(s);
</script>
```

split(separator [,limit])
Splits the string based on the ***separator*** specified and returns that array of substrings. If the **limit** is specified only those number of substrings will be returned
Example1:

```
<script language="javascript">
var s="vit#svecw#bvrice#sbsp";
var t =s.split("#");
alert(t);
</script>
```

Example2:
```
<script language="javascript">
var s="cse#ece#mech#it";
var t =s.split("#");
alert(t); //displays cse,ece,mech,it
</script>
```

Write Javascript that determines whether the given string is palindrome or not

```
<html>
<body>
<script language="javascript">
var s = prompt("enter any string");
var n = s.length;
var flag=true;
for(var i=0; i<n;i++)
{
if(s.charAt(i) != s.charAt(n-1-i))
{
flag=false; break;
}
}
if(flag) alert("palendrome");
else alert("not palendrome");
</script>
</body>
</html>
```

STRING METHODS USED TO GENERATE HTML:

string.anchor(“anchorname”) string.link(url)
string.blink( ) string.big( )
string.bold( ) string.small( )
string.fixed( ) string.strike( )
string.fontcolor(colorvalue) string.sub( )
string.fontsize(integer 1 to 7) string.sup( )
string.italics( )

Example:
```
<script language="javascript">
var s = "Test".bold(); //s value is: <b>Test</b>
document.write(s);
document.write("<br>");
document.write("CMRCET".italics());
</script>
```

anchor ("*name*")	Creates a named anchor specified by the <**A**> element using the argument name as the value of the corresponding attribute.	var x = "Marked point".anchor("marker"); // <A NAME="marker">Marked point</A>
big()	Creates a <**BIG**> element using the provided string.	var x = "Grow".big(); // <BIG>Grow</BIG>
blink()	Creates a blinking text element enclosed by <**BLINK**> out of the provided string despite Internet Explorer's lack of support for the <**BLINK**> element.	var x = "Bad Netscape".blink(); // <BLINK>Bad Netscape</BLINK>
bold()	Creates a bold text element indicated by <**B**> out of the provided string.	var x = "Behold!".bold(); // <B>Behold!</B>
fixed()	Creates a fixed width text element indicated by <**TT**> out of the provided string.	var x = "Code".fixed(); // <TT>Code</TT>

fontcolor (color)	Creates a <**FONT**> tag with the color specified by the argument color. The value passed should be a valid hexadecimal string value or a string specifying a color name.	var x = "green".font("green");
		// <FONT COLOR="green">Green</FONT>
		var x = "Red".font("#FF0000");
		// <FONT COLOR="#FF0000">Red</FONT>

Fontsize (*size*)	Takes the argument specified by size that should be either in the range 1–7 or a relative +/– value of 1–7 and creates a <**FONT**> tag.	var x = "Change size".font(7);
		// <FONT SIZE="7">Change size</FONT>
		var x = "Change size".font("+1");
		// <FONT SIZE="+1">Change size</FONT>

italics()	Creates an italics element <I>.	var x = "Special".italics(); // <I>Special</I>

Link (location)	Takes the argument location and forms a link with the <**A**> element using the string as the link text.	var x = "click here".location("http://www.pint.com/");
		// <A HERF="http://www.pint.com">
		// click here</A>

small()	Creates a <**SMALL**> element out of the provided string.	var x = "Shrink".small(); // <SMALL>Shrink</SMALL>
strike()	Creates a <**STRIKE**> element out of the provided string.	var x = "Legal".strike(); // <STRIKE>Legal</STRIKE>
Sub()	Creates a subscript element specified by <**SUB**> out of the provided string.	var x = "test".sub() // _{test}
Sup()	Creates a superscript element specified by <**SUP**> out of the provided string.	var x = "test".sup() // ^{test}

FUNCTIONS

A function is a piece of code that performs a specific task. The function will be executed by an event or by a call to that function. We can call a function from anywhere within the page (or even from other pages if the function is embedded in an external **.js** file). JavaScript has a lot of builtin functions.

Defining functions:

JavaScript function definition consists of the **function** keyword, followed by the name of the function.

A list of arguments to the function are enclosed in parentheses and separated by commas. The statements within the function are enclosed in curly braces { }.

Syntax:

```
function functionname(var1,var2,...,varX)
{
some code
```

}

Parameter Passing:

Not every function accepts parameters. When a function receives a value as a parameter, that value is given a name and can be accessed using that name in the function. The names of parameters are taken from the function definition and are applied in the order in which parameters are passed in.

• Primitive data types are passed by value in JavaScript. This means that a copy is made of a variable when it is passed to a function, so any manipulation of a parameter holding primitive data in the body of the function leaves the value of the original variable untouched.

• Unlike primitive data types, composite types such as arrays and objects are passed by reference rather than value.

Examining the function call:

In JavaScript parameters are passed as arrays. Every function has two properties that can be used to find information about the parameters:

***functionname*.arguments**

This is an array of parameters that have been passed

***functionname*.arguments.length**

This is the number of parameters that have been passed into the function

Example:

```
<html>
<body>
<script language="javascript">
function fun(a,b){
var msg = fun.arguments[0]+".."+fun.arguments[1]; //referring a,b values
alert(msg);
}
```

```
fun(10,20); //function call
fun("abc","vit"); //function call
</script>
</body>
</html>
```

Returning values

The **return** statement is used to specify the value that is returned from the function. So functions that are going to return a value must use the **return** statement.

Example:

```
function prod(a,b)
{
x=a*b; return x;
}
```

Scoping Rules:

Programming languages usually impose rules, called scoping, which determine how a variable can be accessed. JavaScript is no exception. In JavaScript variables can be either ***local*** or ***global.***

global

Global scoping means that a variable is available to all parts of the program. Such variables are declared outside of any function.

local

Local variables are declared inside a function. They can only be used by that function.

GLOBAL FUNCTIONS

>**parseInt()**	>Converts the string argument to an integer and returns the value. If the string cannot be converted, it returns **NaN**. Like **parseFloat ()**, this method should handle strings starting with numbers and peel off what it needs, but other mixed strings will not be converted.	>var x;
		x = parseInt("-53");
		// x is -53
		
		x = parseInt("33.01568");
		// x is 33
		
		x = parseInt("47.6k-red-dog");
		// x is 47
		
		x = parseInt("a567.34");
		// x is NaN
		
		x = parseInt("won't work");
		// x is NaN

>**unescape ()**	>Takes a hexadecimal string value containing some characters of the form %xx and returns the ISO-Latin-1 ASCII equivalent of the passed values.	>Var aString="O%27Neill%20%26%20Sons";
		aString = unescape(aString);
		// aString = "O'Neill & Sons"
		aString = unescape("%64%56%26%23");
		// aString = "dV&#"

Method	Description	Example

>**escape()**	>Takes a string and returns a string where all non-alphanumeric characters such as spaces, tabs, and special characters have been replaced with their hexadecimal equivalents in the form %xx.	`>var aString="O'Neill & Sons";`
		`// aString = "O'Neill & Sons"`
		`aString = escape(aString);`
		`// aString="O%27Neill%20%26%20Sons"`

>**eval()**	>Takes a string and executes it as JavaScript code.	`>var x;`
		`var aString = "5+9";`
		`x = aString;`
		`// x contains the string "5+9"`
		`x = eval(aString);`
		`// x will contain the number 14`

>**isFinite()**	>Returns a Boolean indicating whether its number argument is finite.	`>var x;`
		`x = isFinite('56');`
		`// x is true`
		`x = isFinite(Infinity)`
		`// x is false`

>**isNaN()**	>Returns a Boolean indicating whether its number argument is **NaN**.	`>var x;`
		`x = isNaN('56');`
		`// x is False`
		`x = isNaN(0/0)`
		`// x is true`
		`x = isNaN(NaN);`
		`// x is true`

>**parseFloat ()**	>Converts the string argument to a floating-point number and returns the value. If the string cannot be converted, it returns **NaN**. The method should handle strings starting with numbers and peel off what it needs, but other mixed strings will not be converted.	`>var x;`
		`x = parseFloat("33.01568");`
		`// x is 33.01568`
		`x = parseFloat("47.6k-red-dog");`
		`// x is 47.6`
		`x = parseFloat("a567.34");`
		`// x is NaN`
		`x = parseFloat("won't work");`
		`// x is NaN`

EXCEPTION HANDLING

Runtime error handling is vitally important in all programs. Many OOP languages provide a mechanism for handling with general classes of errors. The mechanism is called *Exception Handling*.

An exception in object-based programming language is an object, created dynamically at run-time, which encapsulates an error and some information about it. In Java Script it returns an error object when an error is generated at browser while the code is executing.

try-catch block

The block of code that might cause the exception is placed inside **try** block. **catch** block contains statements that are to be executed when the exception arises.

```
try { statement one
statement two
statement three
} catch(Error) { //Handle errors here } finally { // execute the code even regardless of above
catches are matched }
try{
alert(„This is code inside the try clause‟);
ablert („Exception will be thrown by this code‟);
}
catch(exception)
{
alert("Internet Explorer says the error is " + exception.description);
}
```

throw statement

The **throw** statement allows us to create an exception. If we use this statement together with the **try...catch** statement, we can control program flow and generate accurate error messages.

Syntax

throw(exception)

The exception can be a string, integer, Boolean or an object

Example:

```
<html> <body> <script type="text/javascript"> var x=prompt("Enter a number between 0 and
10:",""); try { if(x>10) { throw "Error! The value is too high";
} else if(x<0) { throw "Error! The value is too low";
} else if(isNaN(x)) { throw "Error! The value is not a number"; } } catch(er) { alert(er); }
</script> </body> </html>
```

Document Object Model (DOM)

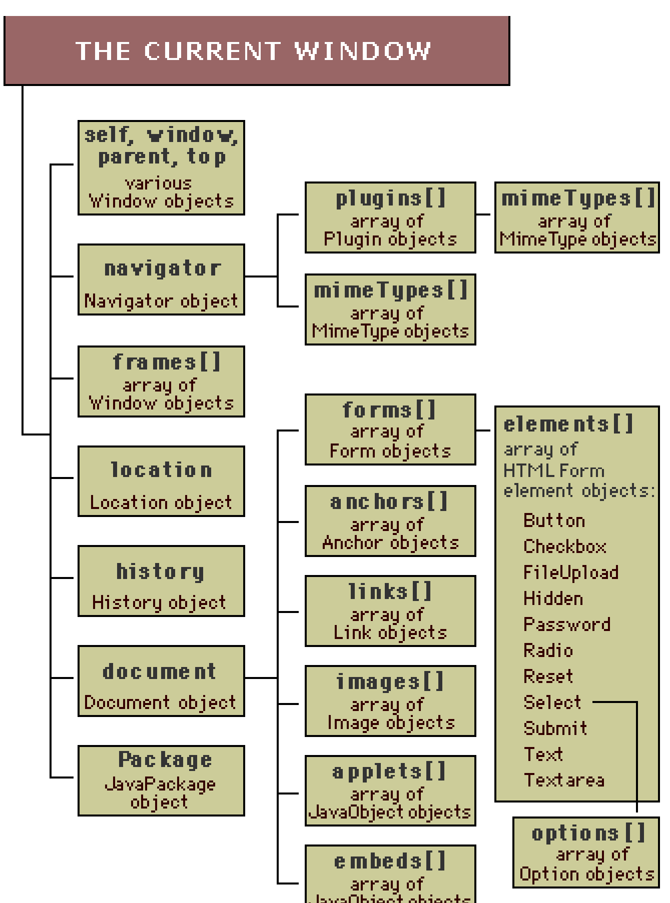
*The **DOM** defines what properties of a document can be retrieved and changed, and the methods that can be performed.* **(or)**

*The Browser **DOM** specifies how JavaScript (and other programming languages) can be used to access information about a document. Then you can perform calculations and decisions based on the values retrieved from the document using JavaScript*

Example: We can retrieve properties such as the value of the **height** attribute of any image, the **href** attribute of any link, or the length of a password entered into a text box in a form. Meanwhile, methods allow us to perform actions such as **reset()** or **submit()** methods on a form that allows you to reset or submit a form.

DOM Hierarchy

- The objects in the web page follow a strict hierarchy, where the **window** object is the very top level. Because **window** is the top level "root" object it can be omitted in the address syntax. For instance, the **window.document.bgColor** property, which stores the value of the window‟s current background color, can be addressed simply as **document.bgColor**
- Several of the DOM objects have properties that contain an array of elements in that web page. For example, with **document.images[],** the **images[]** array is a property of the document object that will store the URL address of each image contained on that web page.The URL of the first image in the HTML code is stored in the array at **document.images[0]**


THE CURRENT WINDOW
self, window, parent, top
various Window objects
navigator
Navigator object
plugins[]
array of Plugin objects
mimeTypes[]
array of MimeType objects
mimeTypes[]
array of MimeType objects
frames[]
array of Window objects
location
Location object
history
History object
document
Document object
Package
JavaPackage object
forms[]
array of Form objects
elements[]
array of HTML Form element objects:
Button
Checkbox
FileUpload
Hidden
Password
Radio
Reset
Select
Submit
Text
Textarea
anchors[]
array of Anchor objects
links[]
array of Link objects
images[]
array of Image objects
applets[]
array of JavaObject objects
embeds[]
array of JavaObject objects
options[]
array of Option objects

BUILT-IN OBJECTS IN JAVASCRIPT
Javascript has many built-in objects which posses the capability of performing many tasks. Hence sometimes Javascript is referred to as an ***Object based programming language***.
Now we will discuss few commonly used objects of javascript along with their attributes & behaviors.
THE WINDOW OBJECT
PROPERTIES:
frames[]
array of frames stored in the order in which they are defined in the document
frames.length
number of frames
self
current window
opener
the window (if any) which opened the current window
parent
parent of the current window if using a frameset
status
message in the status bar
defaultStatus
default message for the status bar
name
the name of the window if it was created using the open( ) method and a name was specified
location
this object contains the full URL address of the document that is currently loaded in the browser, and assigning a new value to this will load the specified URL into the browser.
A typical URL address may comprise these parts:
Protocol: // host / pathname ? #hash
We can use the following properties of **location** object to extract individual pieces of information from URL
window.location.href
window.location.protocol
window.location.host
window.location.pathname
window.location.hash

history

The **window.history** object contains history (i.e. array of URL addresses previously visited during a browser session). For security reasons, these are not directly readable but they are used to navigate back to previous pages. The back( ) and forward( ) methods of the window.history object emulate the browser"s Back and Forward buttons. More flexible navigation is often provided by the window.history.go( ) method.

Example: window.history.go(1) goes forward to the next page in the history
window.history.go(-2) goes backward by 2 pages in the history
window.history.go(0) causes the browser to reload the current document.

Example:

```
<html>
<head>
<script language="javascript">
function fun()
{
var n = prompt("enter any number");
window.history.go(n);
}
</script>
</head>
<body>
<form>
<h1>CMRCET</h1>
<h2>Medchal</h2>
<input type="button" value="navigate" onclick=fun()>
</form>
</body>
</html>
```

onload

This object can be used to specify the name of a function to be called immediately after a document has completely loaded in the browser

Example:

```
<html>
<head>
<script language="javascript">
```

```
window.onunload=fun;
function fun(){
alert("number of frames = " + window.frames.length);
}
</script>
</head>
<frameset rows="30%,30%,*">
<frame name="row1" src="page1.html">
<frame name="row2" src="page2.html">
<frame name="row3" src="page3.html">
</frameset>
</html>
```

onunload

This object can be used to specify the name of a function to be called when the user exits the web page.

METHODS:

alert("string")

opens box containing the message

confirm("string")

displays a message box with OK and CANCEL buttons

prompt("string")

displays a prompt window with field for the user to enter a text string

blur( )

remove focus from current window

focus( )

give focus to current window

scroll(x,y)

move the current window to the chosen x,y location

open("URL", "name", "options string")

The open() method has 3 arguments:

- URL to load in the pop-up window
- Name for the pop-up
- List of options

Example:
newWin = window.open(address,"newWin", "status=0, width=100, height=100, resizable=0");
The open( ) method can take the following options:
toolbar = [1|0] location = [1|0] directories = [1|0] status = [1|0] menubar = [1|0]
scrollbars = [1|0] resizable = [1|0] width = pixels height = pixels
Many of the attributes of a browser window are undesirable in a pop-up window. They can be switched on and off individually

close( )
This shuts the current window

Note: Because **window** is the top level "root" object, it can be omitted in the address syntax. Therefore we can refer its properties directly
Example: window.document.bgColor (or) document.bgColor
window.alert( ) (or) alert( )

THE DOCUMENT OBJECT

A document is a web page that is being either displayed or created. The document has a number of properties that can be accessed by JavaScript programs and used to manipulate the content of the page.

PROPERTIES:

bgColor

Background color of the document

Example: write a javascript that designs 3 buttons "red", "green", and "yellow". When ever the button is clicked, the document color should change accordingly

```
<html>
<head>
<script language="javascript">
function changecolor(s)
{
window.document.bgColor=s;
}
</script>
</head>
<body>
<form>
<input type="button" value="red" onclick="changecolor('red')">
<input type="button" value="green" onclick="changecolor('green')">
<input type="button" value="yellow" onclick="changecolor('yellow')">
</form>
</body> </html>
```

fgColor
Foreground color of the document
title
Title of the current document
location
This object contains the full URL address of the document that is currently loaded in the browser, and assigning a new value to this will load the specified URL into the browser.
Example:

```
<html>
<body>
<script language="javascript">
document.title="cmrcet";
function fun(){
document.location="page1.html";
}
</script>
<input type="button" value="change url" onclick="fun()">
</body>
</html>
```

lastModified
Object that provides information about date and time when a webpage was last modified. This data is usually supplied to **document.lastModified** from the HTTP file header that is sent by the web server
Example:

```
<html>
<body>
<script language="javascript">
window.status = "Last updated " + document.lastModified;
</script>
<h1> CMRCET</h1>
CMRCET was established in the year 2002.
</body>
</html>
```

linkColor, vlinkColor,alinkColor
These can be used to set the colors for the various types of links
forms[] array of forms on the current page

forms.length
the number of form objects on the page
links[]
array of links in the current page in the order in which they appear in the document
anchors[]
an array of anchors. Any named point inside an HTML document is an anchor. Anchors are create using <a name=…>. These will be commonly used for moving around inside a large page. The anchors property is an array of these names in the order in which they appear in the HTML document. Anchors can be accessed like this: **document.anchors[0]**
images[]
an array of images
applets[]
an array of applets
cookie
object that stores information about cookie
Methods:
write("string")
write an arbitrary string to the HTML document
writeln("string")
write a string to the HTML document and terminate it with a newline character. HTML pages can be created on the fly using JavaScript. This is done using the write or writeln methods of the document object.
Example:

```
document.write("<body>");
document.write("<h1>CMRCET</h1>");
document.write("<form>");
```

clear()
clear the current document
close()
close the current document
getElementById()
Returns the reference of the form control given by its Id

DATE OBJECT
This object is used to obtain the date and time. This date and time is based on computer‟s local time (system‟s time) or it can be based on GMT. This GMT is also known as UTC i.e. Universal Coordinated Time. This is basically a world time standard.
Following are the commonly used methods of
Date object:

Method	Meaning
getTime()	It returns the number of milliseconds. This value is the difference between the current time and the time value from 1st January 1970
getDate()	Returns the current date based on computers local time
getUTCDate()	Returns the current date obtained from UTC
getDay()	Returns the current day. The day number is from 0 to 6 i.e. from Sunday to Saturday
getUTCDay()	Returns the current day based on UTC. The day number is from 0 to 6
getHours()	Returns the hour value ranging from 0 to 23
getUTCHours()	Returns the hour value ranging from 0 to 23, based on UTC timing zone
getMinutes	Returns the minute value ranging from 0 to 59
getUTCMinutes()	Returns the minute value ranging from 0 to 59, based on UTC
getSeconds()	Returns the seconds value ranging from 0 to 59
getUTCSeconds()	Returns the seconds value ranging from 0 to 59, based on UTC
getMilliseconds()	Returns the milliseconds value ranging from 0 to 999, based on local time
getUTCMilliseconds()	Returns the milliseconds value ranging from 0 to 999, based on UTC
setDate(value)	This is used to set the Date
setHour(hr,min,sec,ms)	This is used to set the Hour

Example:

```
<html>
<head>
<title>Date Object</title>
</head>
<body>
<script type="text/javascript">
var d = new Date();
document.write("The Date is: "+d.toString()+"<br>");
document.write("Today date is: "+d.getDate()+"<br>");
document.write("UTC date is: "+d.getUTCDate()+"<br>");
document.write("Minutes: "+d.getMinutes()+"<br>");
document.write("UTC Minutes: "+d.getUTCMinutes()+"<br>");
```

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

MATH OBJECT

For performing the mathematical computations there are some useful methods available from math object.

- sqrt(num)
- abs(num)
- ceil(num)
- floor(num)
- log(num)
- pow(a,b)
- min(a,b)
- max(a,b)
- sin(num)
- cos(num)
- tan(num)
- exp(num)
- asin(value)
- acos(value)
- atan(value)

random() - returns a psuedorandom number between 1 to 1

round(value)

In addition to the above methods, it has several properties (Numeric constants) like:

Math.E	Euler constant
Math.PI	3.14159
Math.SQRT_2	The square root of 2
Math.SQRT1_2	The square root of ½
Math.LN2	Log of 2
Math.LN10	Log of 10

Example:

```
<html>
<body>
<script language="javascript">
var n = prompt("enter any number");
alert("square root is "+Math.sqrt(n));
</script>
</body>
</html>
```

Exercise1:

1.Write a JavaScript program that generates the following table for the given value of n

Number	Square
1	1
2	4
3	9

4	16

```
<html>
<body>
<table border=1>
<tr> <th>Number <th>Square </tr>
<script language="javascript">
var n = prompt("enter n");
for(i=1;i<=n;i++)
{
document.write("<tr><td>"+i+"<td>"+(i*i)+"</tr>");
}
</script>
</table>
</body>
</html>
```

FORM OBJECT

The **window.document.forms** object contains an array of all the forms in a HTML document, indexed in the order in which they appear in the HTML code.

For example, **window.document.forms[0]** addresses the first form to appear in the HTML code of a web page.

If the **id** attribute of the **<form>** element has been assigned a value, then the form can be addressed by name.

For example, a form named **reg** can be addressed as **document.forms.reg**

All the attributes assigned in the **<form>** tag can be accessed as properties of that form object.

Example:

```
<html>
<head>
<script language="javascript">
window.onload = fun;
function fun()
{
var msg = "Form name: " + document.forms.reg.id;
msg += "\nMethod: " + document.forms.reg.method;
msg += "\nAction: " + document.forms.reg.action;
window.alert(msg);
}
</script>
</head>
<body>
<form id="reg" method="post" action=mailto:abc@xyz.com>
Name: <input type="text" size=10> <br>
Age: <input type="text" size=5> <br>
```

</form>
</body>
</html>

Form elements:

The elements of the form are held in the array **window.document.forms[].elements[]**
The properties of the form elements can be accessed and set using Javascript.
The elements of the form are held in the array **window.document.forms[].elements[]**
The properties of the form elements can be accessed and set using Javascript.

Example1:

```
<html>
<head>
<script language="javascript">
function fun(){
var msg = "Element type: " + document.forms.reg.elements[0].type;
msg += "\nElement value: " + document.forms.reg.elements[0].value;
window.alert(msg);
}
</script>
</head>
<body>
<form id="reg">
<input type="button" value="click" name="btn1" onClick="fun()">
</form>
</body>
</html>
```

Example2:

```
<!--It is useful to change the label that is displayed on a button by its value attribute if that
button performs dual actions
-->
<html>
<head>
<script language="javascript">
var running=false; var num=0;
var tim;
function startstop(){
running = !running; count();
document.forms[0].btn1.value = (running) ? "stop" : "start";
}
function count(){
if(running){
num++;
window.status = "seconds elapsed: " + num;
tim = setTimeout("count()", 1000);
}
else{
num=0; clearTimeout(tim);
}
}
```

```
</script>
</head>
<body>
<form>
<input type="button" value="start" name="btn1" onClick="startstop()">
</form>
</body>
</html>
```

form Object properties, methods

Properties:

name

the name of the form

method

submission method in numeric form. 0 = GET, 1 = POST

action

the action attribute of the form

target

if specified this is the target window for responses to the submission of the form

elements[]

an array containing the form elements in the order in which they are declared in the document

length

the number of elements in the form

Methods:

submit()

submits the form

reset()

resets the form i.e. form controls will be set to default values

Radio Buttons

- Radio Buttons allow to select one option from a list of options.
- Radio Buttons can be created using <input type=”radio”>
- In the browser DOM the radio button group creates a **document.form** object with the given name

Example: Radio Buttons

```
<html>
<head>
<script language="javascript">
function radio_info()
{
```

```
var msg="1st radio value = " + document.forms[0].rbnBranch[0].value;
msg += "\n 2nd radio value = " + document.forms[0].rbnBranch[1].value;
msg += "\n 3rd radio value = " + document.forms[0].rbnBranch[2].value;
msg += "\n 4th radio value = " + document.forms[0].rbnBranch[3].value;
alert(msg);
}
</script>
</head>
<body>
<form id="form1">
The branches in our college are: <br>
<input type="radio" name="rbnBranch" value="cse" selected>CSE<br>
<input type="radio" name="rbnBranch" value="it">IT<br>
<input type="radio" name="rbnBranch" value="ece">ECE<br>
<input type="radio" name="rbnBranch" value="eee">EEE<br><br>
<input type="button" value="Get Radio Info" onclick="radio_info()">
</form>
</body>
</html>
```

Radio Polling: The important feature of a radio button is whether it is checked or not. This can be ascertained from the radio button‟s **checked** property. It will return **true**, if the button is checked otherwise **false.**

```
<html>
<head>
<script language="javascript">
function radio_info()
{
var msg="1st radio status = " + document.forms[0].rbnBranch[0].checked;
msg += "\n 2nd radio status = " + document.forms[0].rbnBranch[1].checked;
msg += "\n 3rd radio status = " + document.forms[0].rbnBranch[2].checked;
msg += "\n 4th radio status = " + document.forms[0].rbnBranch[3].checked;
alert(msg);
}
</script>
</head>
```

```
<body>
<form id="form1">
The branches in our college are: <br>
<input type="radio" name="rbnBranch" value="cse" selected>CSE<br>
<input type="radio" name="rbnBranch" value="it">IT<br>
<input type="radio" name="rbnBranch" value="ece">ECE<br>
<input type="radio" name="rbnBranch" value="eee">EEE<br><br>
<input type="button" value="Get Radio Info" onclick="radio_info()">
</form>
</body>
</html>
```

Check Boxes

Example:

```
<html>
<head>
<script language="javascript">
function info()
{
var msg="1st radio status = " + document.forms[0].branch[0].checked;
msg += "\n 2nd radio status = " + document.forms[0].branch[1].checked;
msg += "\n 3rd radio status = " + document.forms[0].branch[2].checked;
msg += "\n 4th radio status = " + document.forms[0].branch[3].checked;
alert(msg);
}
</script>
</head>
<body>
<form id="form1">
The branches in our college are: <br>
<input type="checkbox" name="branch" value="cse">CSE<br>
<input type="checkbox" name="branch" value="it">IT<br>
<input type="checkbox" name="branch" value="ece">ECE<br>
<input type="checkbox" name="branch" value="eee">EEE<br><br>
<input type="button" value="Get Info" onclick="info()">
</form>
</body>
</html>
```

Option Lists

The <option> tag can be used in the <select> tag to specify various options in the drop down menu list.All menu items are stored in the <select> object‟s **options[]** array. We can use the **selectedIndex** property of **option list** to identify the index of the selected item

Example:

```
<html>
<head>
<script language="javascript">
function get_selected(){
```

```
var s = document.forms[0].sltVehicle.selectedIndex;
document.forms[0].txtVehicle.value = document.forms[0].sltVehicle.options[s].text;
}
</script>
</head>
<body>
<form name="form1">
<select name="sltVehicle">
<option value="v"> Volvo </option>
<option value="s" selected> Saab </option>
<option value="m"> Mercedes </option>
<option value="a"> Audi </option>
</select>
<input type="button" value="Show Selected" onClick="get_selected()">
<input type="text" size=15 name="txtVehicle">
</form>
</body>
</html>
```

Exercise: Write a program that designs a Simple Calculator

```
<html>
<head>
<script language="javascript">
var exp="";
function fun(ch){
if(ch=='=') {
calc.txt1.value=eval(exp); exp = "";
}
else{
exp = exp + ch; calc.txt1.value=exp;
}
}
</script>
</head>
<body>
<form name="calc">
<table border=1>
<tr>
<th colspan=3>Simple calculator</th>
</tr>
<tr>
<th colspan=3><input type="text" name="txt1" size=15></th>
</tr>
<tr>
<td><input type="button" value="1" onclick="fun('1')"></td>
<td><input type="button" value="2" onclick="fun('2')"></td>
<td><input type="button" value="3" onclick="fun('3')"></td>
```

```
</tr>
<tr>
<td><input type="button" value="4" onclick="fun('4')"></td>
<td><input type="button" value="5" onclick="fun('5')"></td>
<td><input type="button" value="6" onclick="fun('6')"></td>
</tr>
<tr>
<td><input type="button" value="7" onclick="fun('7')"></td>
<td><input type="button" value="8" onclick="fun('8')"></td>
<td><input type="button" value="9" onclick="fun('9')"></td>
</tr>
<tr>
<td><input type="button" value="+" onclick="fun('+')"></td>
<td><input type="button" value="-" onclick="fun('-')"></td>
<td><input type="button" value="=" onclick="fun('=')"></td>
</tr>
</table>
</form>
</body>
</html>
```

The Browser / Navigator object

No two browser models will process our javascript in the same way. Its important that we find out which browser is being used to view our page. Then we can make a choice from our visitors:

- Redirect them to a non-scripted version of our site
- Present scripts that are tailored to suit each browser

For historical reasons, browser object is called **navigator** object
The **navigator** object has properties that provide information about the browser that is being used to view a document.

Properties:

Navigator.appCodeName□ The internal name for the browser. For both major products,this is Mozilla, which was the name of the original Netscape code source
navigator.appName□ public name of the browser
navigator.appVersion□ the version number, platform on which the browser is running
navigator.userAgent□ appCodeName + appVersion
navigator.platform□ platform in which browser in running
navigator.plugins[] □ array containing details of installed plugins
navigator.mimeTypes□ array of all supported MIME types. Useful to make sure that the browser can handle our data

Methods:

navigator.javaEnabled() □ This method returns true or false depending upon whether java is enabled or not in the system

Example:

```
<html>
<head>
<script language="javascript">
```

```
function fun()
{
if(navigator.javaEnabled()) window.location = "appletpage.html";
else window.location = "nonapplet.html";
}
</script>
</head>
<body>
Vishnu Institute of Technology was established in the year 2008 with four branches. For further
details visit our <a href="javascript:fun()" > website </a>
</body>
</html>
```

COOKIES

- Cookies are small bits of key-value pair information that a Web server sends to a browser through HTTP response and that the browser later returns unchanged through HTTP Request when visiting the same Web site or domain.

- A cookie is a small piece of information that is passed back and forth in the HTTP request and response. The cookie sent by a servlet to the client will be passed back to the server when the client requests another page from the same application.

- Cookies are tiny files that can be written by javascript to store small amounts of data on the local hard drive. There are limitations to the use of cookies that restrict their size to 4 kilobytes and web browsers are not required to retain more than 20 cookies per web server. Typically a cookie may often retain user data for use across web pages or on subsequent visits to a web site

- Depending on the *maximum age* of a cookie, the Web browser either maintains the cookie for the duration of the browsing session (i.e., until the user closes the Web browser) or stores the cookie on the client computer for future use. When the browser requests a resource from a server, cookies previously sent to the client by that server are returned to the server as part of the request formulated by the browser. Cookies are deleted automatically when they *expire* (i.e., reach their maximum age).

Benefits of Cookies:

- **Identifying a user during an e-commerce session**

- **Remembering usernames and passwords :** Cookies let a user log in to a site automatically, providing a significant convenience for users of unshared computers.

- **Customizing sites:** Sites can use cookies to remember user preferences.

- **Focusing advertising:** Cookies let the site remember which topics interest certain users and show advertisements relevant to those interests.

- **Security Issue:** Browsers generally only accept 20 cookies per site and 300 cookies total, and since browsers can limit each cookie to 4 kilobytes, cookies cannot be used to fill up someone's disk or launch other denial-of-service attacks.

History:
Cookies were originally invented by Netscape to give 'memory' to web servers and browsers. The HTTP protocol, which arranges for the transfer of web pages to your browser and browser requests for pages to servers, is *state-less*, which means that once the server has sent a page to a browser requesting it, it doesn't remember a thing about it. So if you come to the same web page a second, third, hundredth or millionth time, the server once again considers it the very first time you ever came there.
This can be annoying in a number of ways. The server cannot remember if you identified yourself when you want to access protected pages, it cannot remember your user preferences, it cannot remember anything. As soon as personalization was invented, this became a major problem.
Cookies were invented to solve this problem. There are other ways to solve it, but cookies are easy to maintain and very versatile.

How cookies work?
A cookie is nothing but a small text file that's stored in your browser. It contains some data:
1. A name-value pair containing the actual data
2. An expiry date after which it is no longer valid
3. The domain and path of the server it should be sent to

As soon as you request a page from a server (which was requested earlier & the server sent cookie to the client), the cookie is added to the HTTP header. Server side programs can then read out the information and give response accordingly. So every time you visit the site the cookie comes from, information about you is available. This is very nice sometimes, at other times it may somewhat endanger your privacy.
Cookies can be read by JavaScript too. They're mostly used for storing user preferences.
name-value
Each cookie has a ***name-value pair*** that contains the actual information. The name of the cookie is for your benefit, you will search for this name when reading out the cookie information.
Expiry date
Each cookie has an ***expiry date*** after which it is trashed. If you don't specify the expiry date the cookie is trashed when you close the browser. This expiry date should be in UTC (Greenwich) time.
Domain and path
Each cookie also has a ***domain*** and a ***path***. The domain tells the browser to which domain the cookie should be sent. If you don't specify it, it becomes the domain of the page that sets the cookie.
document.cookie
Cookies can be created, read and erased by JavaScript. They are accessible through the property document.cookie. Though you can treat document.cookie as if it's a **string**, it isn't really, and you have only access to the *name-value* pairs.

If I want to set a cookie for this domain with a name-value pair 'ppkcookie1=testcookie' that expires in seven days from the moment I write this sentence, I do
document.cookie='ppkcookie1=testcookie; expires=Thu, 2 Aug 2001 20:47:11 UTC; path=/'

Event Handling

An ***event*** is defined as “something that takes place” and that is exactly what it means in web programming as well.

An ***event handler*** is JavaScript code that is designed to run each time a particular event occurs.

Syntax for handling the events:

<tag Attributes **event=“handler”**>

Table: JavaScript Events

Event	Handler	Description
blur	onBlur	The input focus is moved from the object
change	onChange	The value of a field in a form has been changed by the user entering or deleting data
click	onClick	The mouse is clicked over an element of a page
dblclick	onDblClick	A form element or link is clicked twice in rapid succession
dragdrop	onDragDrop	A system file is dragged with a mouse and dropped onto the browser
focus	onFocus	Input focus is given to an element. The reverse of blur
keydown	onKeyDown	A key is pressed but not released
keypress	onKeyPress	A key is pressed
keyup	onKeyUp	A pressed key is released
load	onLoad	The page is loaded by the browser
mousedown	onMouseDown	A mouse button is pressed
mousemove	onMouseMove	The mouse is moved
mouseout	onMouseOut	The mouse pointer moves off an element
mouseover	onMouseOver	The mouse pointer moves over an element
mouseup	onMouseUp	The mouse button is released
move	onMove	A window is moved,

		maximized or restored either by the user or by a script
resize	onResize	A window is resized by the user or by script
submit	onSubmit	A form is submitted (the **submit** button is clicked)
unload	onUnload	The user leaves the webpage

Table: Objects and Event Handlers

Object	Event Handlers			
window	onload	onunload	onblur	onfocus
link	onclick	onmouseout	onmouseover	
image	onabort	onerror	onload	
form	onreset	onsubmit		
text	password	onblur	onchange	onfocus
textarea	onblur	onchange	onfocus	
button	onclick			
reset	onclick			
submit	onclick			
radio	onclick			
checkbox	onclick			
select	onblur	onchange	onfocus	
fileupload	onblur	onchange	onfocus	

Handling Events

There are two ways to set and execute the JavaScript event handler for an HTML tag:

– Set the event handler property inside HTML

– Set the event handler property inside JavaScript

Set the event handler property inside HTML: Example

```
<img src="img.jpg" id="SampleImage" onmouseover="changeimage(1)"
onmouseout="changeimage(2)" >
```

Set the event handler property inside JavaScript: Example

```
<img src="img.jpg" id="SampleImage" >
<script type="text/javascript" language="javascript">
var img = document.getElementById("SampleImage");
img.onmouseover = changeimageover;
img.onmouseout = changeimageout;
function changeimageover() {
img.src = "flower.jpg";
}
function changeimageout()
{
img.src = "img.jpg";
}
</script>
```

Example:

```
<html>
<head>
<script language="javascript">
function change(v)
```

```
{
var i = document.getElementById("mouse");
if(v==1) i.src="over.gif";
else i.src="out.gif";
}
</script>
</head>
<body>
<form name="form1">
<h1>Demonstrating Rollover Buttons</h1>
<img id="mouse" src="out.gif" width=100 height=100 onMouseOver=change(1)
onMouseOut=change(0)>
</form>
</body>
</html>
```

Regular Expressions & Pattern Matching

A Regular expression is a way of describing a pattern in a piece of text. It"s an easy way of matching a string to a pattern.

We could write a simple regular expression and use it to check, quickly, whether or not any given string is a properly formatted user input. This saves us from difficulties and allows us to write clean and tight code.

For instance, a script might take "***name***" data from a user and have to search through it checking that no digits have been entered. This type of problem can be solved by reading through the string one character at a time looking for the target pattern. Although it seems like a straightforward approach, it is not (Efficiency & speed matters, so any code that does has to be written carefully). The usual approach in scripting languages is to create a pattern called a ***regular expression,*** which describes a set of characters that may be present in a string.

Creating Regular Expressions:

A regular expression is a JavaScript object. We can create regular expressions in one of two ways.

- Static regular expressions
  Ex: var regex = /fish/fowl/ ;
- Dynamic regular expressions
  Ex: var regex = new RegExp("fish|fowl");

Note: If performance is an issue for our script, then we should try to use static expressions whenever possible. If we don"t know what we are going to be searching until runtime (for instance, the pattern may depend on user input) then we create dynamic patterns

A regular expression pattern is composed of simple characters, such as /abc/, or a combination of simple and special characters, such as /ab*c/ or /Chapter (\d+)\.\d*/.

Using Simple Patterns:

• Simple patterns are constructed of characters for which we want to find a direct match.

• For example, the pattern /abc/ matches character combinations in strings only when exactly the characters 'abc' occur together and in that order.

• Such a match would succeed in the strings

"Hi, do you know your abc's?“
"The latest airplane designs evolved from slabcraft."
• In both cases the match is with the substring 'abc'.

• There is no match in the string "Grab crab" because it does not contain the substring 'abc'.

Using special characters:
• When the search for a match requires something more than a direct match, such as finding one or more b's, or finding white space, the pattern includes special characters.

• For example, the pattern /ab*c/ matches any character combination in which a single 'a' is followed by zero or more 'b's (* means 0 or more occurrences of the preceding item) and then immediately followed by 'c'. In the string "cbbabbbbcdebc," the pattern matches the substring 'abbbbc'

Working with Regular Expressions

Regular expression patterns in java script must begin and end with forward slashes.

. ---Matches single character

\ ---identifies the next charcter as a literal value

^ ----Matches characters at begining of a string

$ ----Matches characters at the end of the string

() ----specifies required characters to include in pattern match

[] ----Specifies alternate characters allowed in a pattern match

[^] ---Specifies characters to exclude in a pattern match

~ ----identifies a possible range of characters to match

| ---Specifies alternate sets of characters to include

^ circumflex operator

Java script character class escape characters:

\w Alphanumeric character
\D Alphabetic characters
\d Numeric chacters

\S All pritable characters
\s white space characters
\W Any character that is not an alphanumeric character
\b Backspace character

Predefined character classes

Name	EquilanetPattern	Matches
\d	[0-9]	A digit
\D	[^0-9]	Not a digit
\w	[A-Za-z_0-9]	A word character(Alphanumeric)
\W	[^A-Za-z_0-9]	Not a word character
\s	[\r\t\n\f]	A white space character
\S	[^\r\t\n\f]	Not a White space character