CHAPTER 4
JavaScript

Overview of JavaScript

- JavaScript is a programming language commonly used in web development.
- JavaScript was originally developed at Netscape by Brendan Eich as a means to add dynamic and interactive elements to websites , was initially named Mocha but soon after was renamed LiveScript .
- LiveScript became a joint venture of Netscape and Sun Microsystems , and its name again was changed and know as JavaScript, in late 1995.
- JavaScript can be divided into three parts: the core , client side , and server side.
- The core part including its operators, expressions , statements , and subprograms, also called heart of the language.
- Client-side JavaScript is a collection of objects that support the control of a browser and interactions with users.
- Server-side JavaScript is a collection of objects that make the language useful on a Web server – for example , to support communication with a database management system .
- But here , we introduce only about client - side JavaScript .
- Client – side JavaScript is an Html embedded scripting language.

Is JavaScript and Java are same ?

Although JavaScript's name seems close to Java , JavaScript and Java are actually very different. One important difference is support for object – orientation programming . JavaScript object model is quite different from that of Java and C++. Also JavaScript does not support the object – oriented software development paradigm.

Some points that differentiate Java and JavaScript are given below:

- **Java** is an object – oriented programming language where **JavaScript** is an object – based programming language.
- **Java** is a strongly typed language , all types are known at compile time, and operand types are checked for compatibility but JavaScript is a weakly typed language where variable need not be declared and are dynamically typed.
- **Java** have classes and define objects but **JavaScript** do not have classes , its objects serve both as objects and as models of objects.
- Object – oriented language support class – based inheritance Where **JavaScript** support a technique that can be used to simulate some of the aspects of inheritance called prototype – based inheritance , is done using prototype objects.

JavaScript Objects

- Objects are collections of properties , which correspond to the members of classes in Java and C++ . Each property is either a data property or a function or method property.

- Data properties appear in two categories: primitive values and references to other objects. Also , variables that refer to objects are often called objects rather then references.
- JavaScript uses nonobject types for some of its simplest types; these nonobject types are called primitives.
- In JavaScript almost "everything" is an object .
  - Booleans can be objects (if defined with the new keyword)
  - Numbers can be objects (if defined with the new keyword)
  - Strings can be objects (if defined with the new keyword)
  - Dates are always objects
  - Maths are always objects
  - Regular expressions are always objects
  - Arrays are always objects
  - Functions are always objects
  - Objects are always objects
- Objects are variables too. But object can contain many values.

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>
<script>
// Create an object:
var person = {
    firstName: "Ramesh",
    lastName: "Karki",
    age: 20,
    eyeColor: "blue"
};
// Display some data from the object:
document.getElementById("demo").innerHTML =person.firstName+ " is" + person.age + "
years old.";
</script>
</body>
</html>
```

General Syntactic Characteristics /Lexical structure

A programming language's lexical structure specifies set of some basic rules about how a code should be written in it. Rules like what variable names looks like, the delimiter characters for comments, and how one program statement is separated from the next. It is the lowest-level syntax of a language.

i. **Character Set - Unicode** character set is used in JavaScript codes.

ii. **Case Sensitive – JavaScript** is case-sensitive language. While , while , WHILE all are different in JavaScript. It means that language keywords, variables, function names, and other identifiers must always be typed with a consistent capitalization of letters.

iii. **Whitespace and Line Breaks - JavaScript** is ignorant to spaces that appear between tokens in codes. Line breaks are also not considered by JS. This lets you freely use any number of spaces and newlines.

iv. **Comment** – There are two types of comment . First one is single line comment (i.e. double forward slash(//)) and second one is multiple line comment(i.e. followed by forward slash and * pair(/* and */).

v. **Identifier and Reserved words** – A JavaScript **identifier** must begin with a letter, an underscore (_), or a dollar sign ($). Digits are not allowed as the first character. The identifiers reserved by the language for its use are known as "**reserved words**". Example: var, typeof , case , continue etc.

vi. **Optional Semicolon** – Semicolons are used in JavaScript to separate statements from each other. But you can also omit the semicolon between two statements if those statements are written on separate lines.

vii. All JavaScript scripts will be embedded in HTML documents

- Either directly, as in

```
<script type = "text/javaScript">
-- JavaScript script - -
</script>
```

- Or indirectly, as a file specified in the src attribute of <script>, as in

```
<script type = "text/javaScript"    src = "myScript.js">
</script>
```

viii. Scripts are usually hidden from browsers that do not include JavaScript interpreters by putting then in special comments.

```
<!--
--JavaScript script --
//-->
```

- Also hides it from HTML validators

Browsers and HTML-JavaScript Documents

HTML-JavaScript Documents form different execution method . If HTML document does not include scripts, the browser reads the lines of the document and renders its window according to the tags , attributes , and content it finds. When a JavaScript encountered in the document , the browser uses its JavaScript Interpreter to "execute" the script. The output form of the script becomes the next markup to be rendered and when the end of the script is reached , the browser goes back to reading the HTML document and displaying its content.

There are two different ways to embed JavaScript in an HTML document : **implicitly** and **Explicitly**. In **explicit** embedding , the JavaScript code physically resides in the HTML document. In **implicit** embedding , JavaScript can be placed in its own file, separate from the HTML document.

Uses of JavaScript

- The main objective of JavaScript was to provide programming capability at both server and the client ends of a web connection.
- Client-side JavaScript can serve as an alternative in which computational capability resides on the server and is requested by the client.
- In particular, although server-side software supports file operations , database access, and networking , client-side JavaScript supports none of these.
- JavaScript is used to make interactions with users through form elements, such as buttons menus , they can be used to trigger computations and provide feedback to the user.
- To load data in the background from the server and loaded on to the page side without reloading.
- Client side validation of form elements instead of sending data to server every time.

JavaScript Variables

- In javascript variables are defined using *var* keyword.
- We don't need to define data type as in C, C++, Java etc.

  Example:

  ```
  var str=5; //Here str is an integer type
  var str=5.0; //Here str is float type var
  str="hello"; //Here str is string type
  ```

- As like above we can declare any type variables using *var* keyword.

Primitives and Operations

JavaScript have five primitive types: Number, String, Boolean, Undefined, and Null.

i. **Numeric Literal:** We don't need to define numeric values types (i.e. int, float, etc). The Number type values are represented internally in double-precision floating-point form. Literals numbers in a script can have the forms of either integer or floating-point values. Integer literals are strings of digits. Floating-point literals can have decimal points, exponents, or both.

ii. **String literal:** A string literal is a sequence of zero or more characters defined within single quotes(') Or double quotes ("). String Literals can include characters specified with escape sequences, such as \n and \t . The backslash (\ ) escape character turns special characters into string characters.

Code	Results	description
\'	'	Single-quote
\"	"	Double-quote
\\	\	backslash

\b	Backspace	
\f	Form feed	
\n	New line	
\r	Carriage return	
\t	Horizontal Tabulator	
\v	Vertical Tabulator	

The sequence \" inserts a double-quote in string.

Example:

```
<html>
<body>
    <h2>JavaScript Strings</h2>
    <p>The escape sequence \\ inserts a backslash in a
    string.</p> <p id="demo"></p>
    <p>The escape sequence \" inserts a double-quote in a string.</p>
    <p id="demo1"></p>
    <p>The escape sequence \' inserts a single-quote in a
    string.</p> <p id="demo2"></p>
<script>
    var x = "The character \\ is called backslash.";
    document.getElementById("demo"). nnerHTML = x;

    var z = "We are the so-called \"Vikings\" from the north.";
    document.getElementById("demo").innerHTML = z;

    var y = 'It\'s alright.';
    document.getElementById("demo2"). nnerHTML = y;

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

Here, string can be defined as an objects with the keywords *new*.

Example:

```
<html>
<body>
    <p id="demo"></p>
<script>
    var x = "John";       // x is a string
    var y = new String("John"); // y is an object
    document.getElementById("demo").innerHTML=typeof x + "<br> " + typeof y;
</script>
</body>
</html>
```

Strings Method:

In JavaScript length of a string is found with build-in ***length*** property.

Some methods and properties are:

- **indexOf()** – this method returns the index of (the position of) the first occurrence of a specified text in a string.
- **lastIndexOf()** – returns the index of the last occurrence of a specified text in a string.(**Both indexOf()**, and **lastIndexOf** ()return -1 if the text is not found).
- **search()** – method searches a string for a specified value and returns the position of the match.
- **slice()** – extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters: the start position, and the end position (end not included).
- **charAt** – returns the character from the specified index. Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character in a string, called ***stringName***, is *stringName.length* – 1.
- **charCodeAt()** – returns Unicode of the character at the specified index in a string.
- **concat()** – combine the text of two string and returns new string.
- **replace()** – replaces a specified value with another value in a string. By default, the replace() function replaces **only the first**
- match. **trim()** – is used to remove white space from both the end
- of strings. **trimLeft()** – is used to remove white space from the start of given string.
- **trimRight()** – is used to remove white spaces from the end of the given string.
- **split()** – is used to splits a String object into an array of strings by separating the string into substrings.

  syntax: *string.split([separator],[limit]);*

  separator – specifies the character to use for separating the string. If *separator* is omitted, the array returned contains one element consisting of the entire string.

  limit – integer specifying a limit on the number of splits to be found.
- **toLowerCase()** – returns the calling string value converted to lower case.
- **toUpperCase()** – returns the calling string value converted to upper case.
- **toString()** – is used to returns given string itself. ***(Syntax:*** *string.toString())*

Example:

```
<html>
<body>
<h2>JavaScript String Methods</h2>
        <p id="demo"></p>
        <p id="demo1"></p>
        <p id="demo2"></p>
        <p id="demo3"></p>
        <p id="demo4"></p>
        <p id="demo5"></p>
        <button onclick="myFunction()">Replace</button>
```

```
<p id="forButton">Please visit Microsoft!</p>
<button onclick="myFunction1()">Trim</button>
<button onclick="myFunction2()">To lower</button>
<button onclick="myFunction3()">To upper</button>
<button onclick="myFunction4()">Try it</button>
<p id="demo6"></p>
<p id="demo7"></p>
<script>
//illustrating indexOf() method
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
document.getElementById("demo").innerHTML = pos;

//illustrating lastIndexOf() method
var str1 = "Please locate where 'locate' occurs!"; var
pos1 = str1.lastIndexOf("locate");
document.getElementById("demo1").innerHTML = pos1;

//illustrating search() method
var str2 = "Please locate where 'locate' occurs!"; var
pos2 = str2.search("locate");
document.getElementById("demo2").innerHTML = pos2;

//illustrating slice() method
var str3 = "Apple, Banana, Kiwi";
var res = str3.slice(7,13); //extracts position 7 to 13
document.getElementById("demo3").innerHTML = res;

//the position is counted from end of the string, if -ve
var res1= str3.slice(-12,-6);
document.getElementById("demo4").innerHTML = res1;

//the method will slice out the rest of string
var res2= str3.slice(7);
document.getElementById("demo5").innerHTML = res2;

//illustrating replace() method
function myFunction() {
 var strLine = document.getElementById("forButton")
.innerHTML;//replaces only first match(case-sensitive too) var
txt = strLine.replace("Microsoft","our website");
document.getElementById("forButton ").innerHTML = txt; }

//illustrating trim() method
function myFunction1(){
var str4="                    Welcome";
alert(str4.trim());
}
//illustrating split() method
//illustrating toLowerCase() method
function myFunction2(){
var str5="Hello World";
alert(str5.toLowerCase());
```

```
}
//illustrating toUpperCase() method
function myFunction3(){
var str6="Hello World";
alert(str5.toUpperCase()); }
//illustrating concat() method
var text1 = "Hello";
var text2 = "World!";
var text3 = text1.concat(" ",text2);
document.getElementById("demo6").innerHTML = text3;
</script>
</body>
</html>
```

Example: (charAt(), charCodeAt(), and split())

```
<!DOCTYPE html>
<html>
<head>
        <title></title>
</head>
<body>
<script type="text/javascript">
        function func() {
                var str = 'Have a good Day.'+"<br>";
                var array = str.split(" ");
                 document.write(array);
        }
        func(); //function called
        function Function() {
                var str = 'Have a good Day.';
                var array = str.split(" ",3); //having limit too
                 document.write(array);
        }
        Function(); //function called

        //Illustrating charAt() method
        var str = new String( "This is string" ); document.writeln("<br
        />str.charAt(0) is:" + str.charAt(0)); document.writeln("<br
        />str.charAt(1) is:" + str.charAt(1)); document.writeln("<br
        />str.charAt(2) is:" + str.charAt(2)); document.writeln("<br
        />str.charAt(3) is:" + str.charAt(3)); document.writeln("<br
        />str.charAt(4) is:" + str.charAt(4)); document.writeln("<br
        />str.charAt(5) is:" + str.charAt(5));
        //Illustrating charCodeAt() method
        var str1 = new String( "This is string" );
        document.write("<br />str1.charCodeAt(0) is:" + str1.charCodeAt(0));
        document.write("<br />str1.charCodeAt(1) is:" + str1.charCodeAt(1));
        document.write("<br />str1.charCodeAt(2) is:" + str1.charCodeAt(2));
        document.write("<br />str1.charCodeAt(3) is:" + str1.charCodeAt(3));
        document.write("<br />str1.charCodeAt(4) is:" + str1.charCodeAt(4));
        document.write("<br />str1.charCodeAt(5) is:" + str1.charCodeAt(5));
</script>
</body>
</html>
```

iii. Others Primitive Types

- The three other primitive types are Null, Undefined and Boolean.
- The reserved word *null* is the only the value of type Null, which indicates no value.
- A variable is *null* if it has not been explicitly declared or assigned a value, also that will cause runtime error.
- The only value of type Undefined is *undefined* i.e. reserved.
- If a variable has been explicitly declared, but not assigned a value, it has the value *undefined*.
- The only values of type Boolean are *true* and *false*.
- These values are usually computed as the result of evaluating a relational or Boolean expression.

iv. Numeric Operators

- In JavaScript , numeric operators are similar to operators used in other programming language .
- But sign(+) is used for both addition and concatenation of character of string.
- Here, binary operators (+) for addition, (-) for subtraction, (*) for multiplication, (/) for division, and (%) for modulus.
- Also unary operators are plus(+), negate(-), decrement(--), and increment(++).
- Increment(++) and Decrement(--) can be either prefix or postfix.
- Operator (**) is defined as exponentiation operator.

Example:

```
<!DOCTYPE html>
<html>
<head>
 <title>Arithmetic operators</title>
</head>
<body>
 <p id="demo"></p>
 <script type="text/javascript">
  function arithmeticOperation(){
    var a=5;
    var b=6;
    var c='9';
    var x=a+c;
    var z=a+b;
    var y=a**5;
    document.getElementById('demo').innerHTML="The sum is
"+z+"<br>"+"The value of x is "+x+"<br>The value of y is "+y;
  }
  arithmeticOperation();
 </script>
</body>
</html>
```

Note: Do other example yourself.

v. The typeof operator

The typeof operator returns the type of its single operand. This typeof operator is used to produces "number", "string", or "boolean" if the operand is of primitive type Number, String, or Boolean respectively. If the operand is an object or null, typeof produces "object".

Example:

```
<!DOCTYPE html>
<html>
<head>
 <title>Arithmetic operators</title>
</head>
<body>
 <p id="demo"></p>
 <p id="demo1"></p>
 <p id="demo2"></p>
 <p id="demo3"></p>
 <script type="text/javascript">
  function Operation(){
   var a=5;
   var c=(++a)*3;
   var b=(a++)+"3";
   var car = {type:"Fiat", model:"500", color:"white"};
   document.getElementById('demo').innerHTML=typeof a;
   document.getElementById('demo1').innerHTML=typeof c;
   document.getElementById('demo2').innerHTML=typeof b;
   document.getElementById('demo3').innerHTML=typeof car;
  }
  Operation();
 </script>
</body>
</html>
```

vi. The *Date* object

- JavaScript facilitate about the current date and time in a program.
- In JavaScript , date and time are available through Date object and its rich collection of methods.
- A Date object is created with the new operator and the Date constructor, which has several forms.
- Some Date methods(Get methods and Set methods) are given within the example. So, understand yourself.
- Note, JavaScript counts month from 0 to 11. January is 0.December is 11.

Example:

```
<!DOCTYPE html>
<html>
<head>
 <title>Date object</title>
</head>
<body>
 <p id="demo"></p>
 <p id="demo1"></p>
```

```
<p id="demo2"></p>
<p id="demo3"></p>
<p id="demo4"></p>
<p id="demo5"></p>
<p id="demo6"></p>
<p id="demo7"></p>
<p id="demo8"></p>
<script type="text/javascript">
 //creates a new date object with the current date and time
 var d = new Date();
 document.getElementById("demo").innerHTML = d;

 //getTime() method returns the number of milliseconds since January 1, 1970
 var da = new Date();
 document.getElementById("demo1").innerHTML = da.getTime();

 //getFullYear() method returns the year of a date as a four digit number
 var daa = new Date();
 document.getElementById("demo2").innerHTML= daa.getFullYear();

 //getMonth() method returns the month of a date as a number (0-11)
 var daaa = new Date();
 document.getElementById("demo3").innerHTML = daaa.getMonth() + 1;

 //gtDate() method returns the day of a date as a number (1-31)
 var daaaa = new Date();
 document.getElementById("demo4").innerHTML = daaaa.getDate();

 //getHours() method returns the hours of a date as a number (0-23)
 var daaaaa = new Date();
 document.getElementById("demo5").innerHTML = daaaaa.getHours();

 //getMinutes() method returns the minutes of a date as a number (0-59)
 var daaaaaa = new Date();
 document.getElementById("demo6").innerHTML = daaaaaa.getMinutes();

 //getSeconds() method returns the seconds of a date as a number (0-59)
 var daaaaaaa = new Date();
 document.getElementById("demo7").innerHTML = daaaaaaa.getSeconds();

 //getDay() method returns the weekday of a date as a number (0-6)
 var daaaaaaaa = new Date();
 document.getElementById("demo8").innerHTML = daaaaaaaa.getDay();
</script>
</body>
</html>
```

Screen Output and Keyboard Input

- A JavaScript is interpreted when the browser finds the script or a reference to a separate script file in the body of the HTML document.
- The window in which the browser displays an HTML document is modeled with the *window* object.
- The window object includes two properties, *document* and *window*.

- The *document* property refers to the Document object.
- The *window* property is self-referential; it refers to the Window object.
- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using *innerHTML*.
  - Writing into an HTML output using , *document.write()*.
  - Writing into an alert box, using *window.alert()*.
  - Writing into the browser console, using *console.log()*.
  - ***confirm*** method – opens a dialog window in which the method displays its string parameter, along with two buttons: *OK* and *Cancel*. Returns Boolean value true for *OK* and false for *Cancel*.
  - ***prompt*** method – creates a dialog window that contains a text box used to collect a string of input from the user, which prompt returns as its value.

Example:

```
<!DOCTYPE html>
<html>
<head>
 <title>Screen output and keyboard input</title>
 <style type="text/css">
  #btn{
            border-radius: 5px;
            height: 30px;
            width: 70px;
   }
   #btn:hover{
            height: 50px;
            width: 100px;
   }
 </style>
</head>
<body>
   <button onclick="myFunction()" id="btn">click</button>
  <p id="demo"></p>
  <button onclick="myFunction1()"
  id="btn">Prompt</button> <p id="demo1"></p>
 <script type="text/javascript">
  function myFunction(){
            var num1=45;
            var num2=89;
            var sum=num1+num2;
    document.getElementById('demo').innerHTML="The sum of num1 and num2 is
"+sum;
  }
            var num1=50;
            var num2=100;
            var sum=num1+num2;
            document.write(sum);
            function myFunction1(){
            var str1=prompt("Enter first number ","");
            var str2=prompt("Enter second number ","");
            var sum1=str1+str2;
```

```
        window.alert("The sum is "+sum1);
    }
</script> </body></html>
```

Function

- A JavaScript function is a block of reusable code designed to perform a particular task. This eliminates the need of writing the same code again and again.
- A JavaScript function is defined with the *function* keyword, followed by a **name**, followed by parentheses **()**.
- The parentheses may include parameter names separated by commas: **(*parameter1*, *parameter2*, ...) Syntax**:

```
function functionName(parameter1, parameter2,.....)
{
    //code to be executed
}
```

Function Invocation:

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs(when a user clicks a button).
- When it is invoked (called)from JavaScript code.
- Automatically (self invoked)

Example:

```
<!DOCTYPE html>
<html>
<body>
        <h2>JavaScript Functions</h2>
        <p id="demo"></p>
        <p id="demo1"></p>
        <button onclick="btnClick()">change</button>
        <p id="demo2">The color of text will change if you click Change button</p>
<script>
        var x = myFunction(4, 3);
        document.getElementById("demo").innerHTML = x;
        function myFunction(a, b) {
         return a * b;
        }
        document.getElementById("demo1").innerHTML =
        "The temperature is " + toCelsius(77) + " Celsius";
        function toCelsius(fahrenheit) {
        return (5/9) * (fahrenheit-32);
        }
        function btnClick(){
        document.getElementById('demo2').style.color="red";
        }
</script>
</body>
</html>
```

Array

- The **Array** object lets you store multiple values in a single variable.
- It stores a fixed-size sequential collection of elements of the same type.

Syntax:

var array_Name=[item1,item2,.....]; (good way to create)**OR**

var array_Name=*new* Array(item1, item2,item3,....); (bad way to create)

- Array element can be access by referring to the **index** number.
- Array indexes start with 0. [0] is the first element.[1] is the second element.

Array Properties and Methods:

- ***length*** – property returns the length of an array(the number of array elements).
- ***toString()*** – method returns an array to string of(comma separated) array values.
- ***join()*** – method joins all array elements into a string. It behave like toString() method, but in addition you can specify separator. ***pop()*** – method removes last element of an array.(popping) ***push()*** – method adds an element to an array at end.(pushing) ***shift()*** – method removes first element and shift all other element to lower index.
- ***unshift()*** – method adds new element to an array at beginning.
- ***delete*** – operator is used to delete an elements. But it may leave undefined holes in the array. So prefer pop() or shift() methods.
- ***splice()*** – method can be used to add new element to an array, also returns an array with deleted elements.
- ***concat()*** - method creates a new array by merging(concatenating) existing arrays.
- ***slice()*** – method slices out a piece of an array into a new array.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Array Methods</h2>
    <p id="demo"></p>
    <p id="demo1"></p>
    <p id="demo2"></p>
    <p id="demo3"></p>
    <p id="demo4"></p>
    <p id="demo5"></p>
    <p id="demo6"></p>
    <p id="demo7"></p>
    <p id="demo8"></p>
    <p id="demo9"></p>
    <p id="demo10"></p>
<script>
    var fruits = ["Banana", "Orange", "Apple", "Mango"];//array var
    fruitsLen=fruits.length; //length property
    document.getElementById("demo").innerHTML = "The length of array fruits
    is "+fruitsLen;
```

```
    document.getElementById("demo1").innerHTML = fruits.toString();
    document.getElementById("demo2").innerHTML = fruits.join("*");
    fruits.push("Kiwi");
    document.getElementById("demo3").innerHTML = fruits;
    document.getElementById("demo4").innerHTML = fruits.pop();
    document.getElementById("demo5").innerHTML = fruits;
    var vehicle=["car","bus","motorcycle","van"];
    document.getElementById('demo6').innerHTML=vehicle[0];
    document.getElementById("demo7").innerHTML = "Removed item is:
    "+"<b>"+vehicle.shift()+"</b>";
    document.getElementById("demo8").innerHTML = "Array after removing:
    "+"<b>"+vehicle+"</b>";
    document.getElementById("demo9").innerHTML = "Returns length of array:
    "+"<b>"+vehicle.unshift("tractor")+"</b>";
    document.getElementById("demo10").innerHTML = "Array after adding item:
    "+"<b>"+vehicle+"</b>";
</script> </body> </html>
```

Array Sorts:

- ***sort()*** – method sorts an array alphabetically. i.e. By default, the sort() function sorts values as **strings**. To sort numbers ***compare function*** is needed.
- ***reverse()*** – method reverses the elements in an array.
- ***Math.max.apply()*** – is used to find highest number in an array.
- ***Math.min.apply()*** – is used to find lowest number in an array.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Array Sort</h2>
    <button onclick="myFun()">Index</button>
    <p id="indexDemo">we can use index to access each
    element.</p> <button onclick="myFunction()">sort</button> <p
    id="demo"></p>
    <button onclick="myFunction1()">reverse</button>
    <p id="demo1"></p>
    <button onclick="myFunction2()">Number sort Ascending Order</button>
    <p id="demo2"></p>
    <button onclick="myFunction3()">Number sort Descending
    Order</button> <p id="demo3"></p>
    <p>The highest number is <span id="demo4"></span>.</p>
<script>
    var fruits = ["Banana", "Orange", "Apple", "Mango"];
    document.getElementById("demo").innerHTML = fruits;
    function myFun(){
    document.getElementById('indexDemo').innerHTML="The element at index 0 is
    "+fruits[0];
}
    function myFunction() {
     fruits.sort();
    document.getElementById("demo").innerHTML = fruits;
    }
    var vehicle=["car","bus","motorcycle","van"];
```

```
document.getElementById("demo1").innerHTML =
vehicle; function myFunction1(){
// First sort the
array vehicle.sort();
// Then reverse it: vehicle.reverse();
document.getElementById("demo1").innerHTML =
vehicle;
}
var points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo2").innerHTML =
points; //ascending order
function myFunction2() {
points.sort(function(a, b){return a - b}); //Compare function
document.getElementById("demo2").innerHTML = points;
}
document.getElementById("demo3").innerHTML =
points; function myFunction3() {
points.sort(function(a, b){return b - a});
document.getElementById("demo3").innerHTML = points;
}
document.getElementById("demo4").innerHTML =
myArrayMax(points); function myArrayMax(arr) {
return Math.max.apply(null, arr);
}
</script>
</body>
</html>
```

Control Statements

Control Expressions

- The expressions upon which statement flow control can be based include primitive values, relational expressions, and compound expressions.
- The result of evaluating a control expression is one of the Boolean values true or false.

Operation	Operator
Is equal to	==
Is not equal to	!=
Is less than	<
Is greater than	>
Is less than or equal to	<=
Is greater than or equal to	>=
Is strictly equal to	===
Is strictly not equal to	!==

Conditional (if....else , if....elseif....else and switch) Statements

- Conditional statements are used to perform different actions based on different conditions.
- Use ***if*** to specify a block of code to be executed, if a specified condition is true.
- Use ***else*** to specify a block of code to be executed, if the same condition is false.
- Use elseif to specify a new condition to test, if the first condition is false.
- Use ***switch*** to specify many alternative blocks of code to be executed.

Example: if....elseif....else

```
<html>
<body>
        <p>Click the button to get a time-based greeting:</p>
       <button onclick="myFunction()">Try it</button>
      <p id="demo"></p>
<script>
        function myFunction() {
         var greeting;
         var time = new Date().getHours();
        if (time < 10) {
        greeting = "Good morning";
        } else if (time < 20) {
        greeting = "Good day";
        } else {
         greeting = "Good evening";
        }
        document.getElementById("demo").innerHTML = greeting;
        }
</script>
</body>
</html>
```

Example: Switch statement

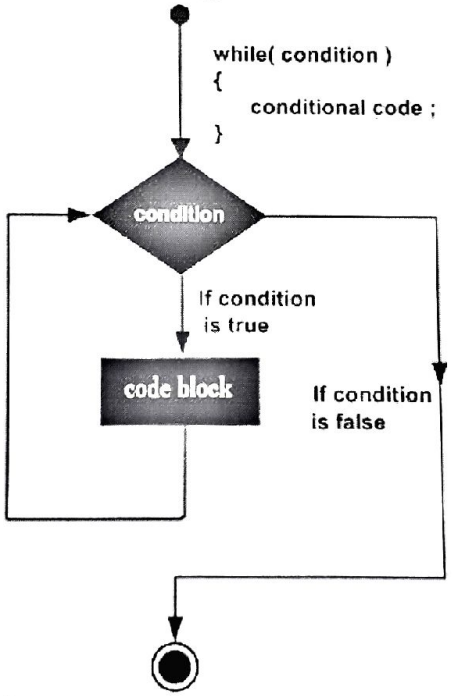
```
<!DOCTYPE html>
<html>
<body>
        <p id="demo"></p>
<script>
        var day;
       switch (new Date().getDay()) {
        case 0:
        day = "Sunday";
         break;
        case 1:
         day = "Monday";
        break;
        case 2:
        day = "Tuesday";
         break;
        case 3:
```

```
                day = "Wednesday";
                break;
                case 4:
                day = "Thursday";
                break;
                case 5:
                day = "Friday";
                break;
                case 6:
                day = "Saturday";
                }
                document.getElementById("demo").innerHTML = "Today is " + day;
        </script>
        </body>
        </html>
```

Loop Statement

While loop

Flow chart:



Syntax:

```
while(expression){
//statement(s) to be executed if expression is true
}
```

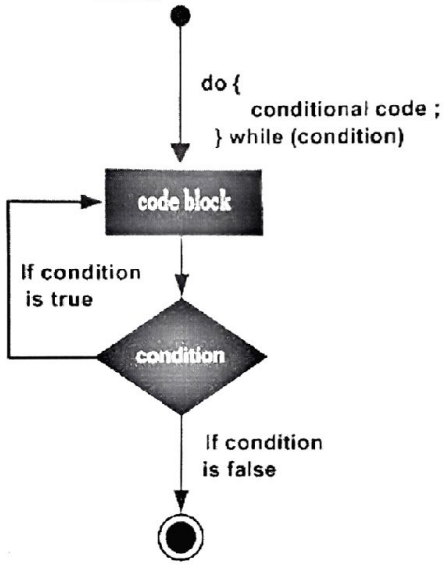
Example:

```
<html>
 <body>
  <script type="text/javascript">
    var count = 0;
```

```
        document.write("Starting Loop<br /> ");
        while (count < 10){
        document.write("Current Count : " + count + "<br />");
        count++;
        }
        document.write("Loop stopped!");
    </script>
    <p>Set the variable to different value and then
  try...</p> </body>
</html>
```

do…..while loop

Flow chart:



Syntax:

```
do{
    //statement(s) to be executed
}while(expression)
```

Example:

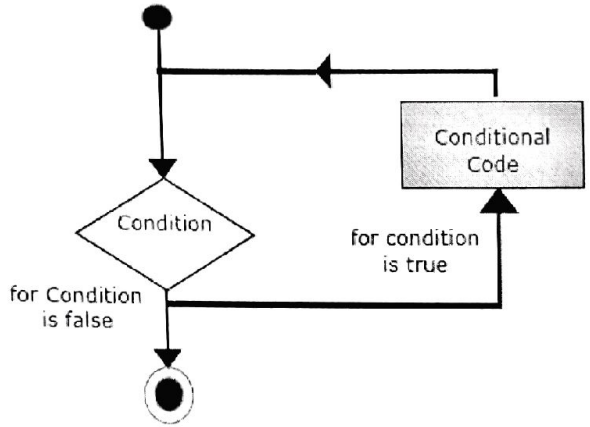
```
<html>
  <body>
    <script type = "text/javascript">
        var count = 0;
        document.write("Starting Loop" + "<br />");
        do {
          document.write("Current Count : " + count + "<br />");
          count++;
        }
        while (count < 5);
        document.write ("Loop stopped!");
    </script>
  </body>      </html>
```

for loop

It includes the following three important parts :

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the
- loop. The **iteration statement** where you can increase or decrease your counter.

Flow chart:



Syntax:

for(initialization; test-condition; iteration-statement){
//statement(s) to be executed if expression is true
}

Example:

```
<html>
  <body>
    <script type = "text/javascript">
        var count;
        document.write("Starting Loop" + "<br />");
        for(count = 0; count < 10; count++) {
          document.write("Current Count : " + count );
          document.write("<br />");
        }
          document.write("Loop stopped!");
        </script>
      <p>Set the variable to different value and then
  try...</p> </body>
 </html>
```

for...in loop

The **for...in** loop is used to loop through an object's properties.

Syntax:

```
for(variable_Name in object){
    //statement or block to be executed
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Loops</h2>
    <p>The for/in statement loops through the properties of an
    object.</p> <p id="demo"></p>
<script>
    var txt = "";
    var person = {fname:"John", lname:"Doe", age:25};
    var x;
    for (x in person) {
    txt += person[x] + " ";
}
     document.getElementById("demo").innerHTML =
 txt; </script>
 </body>
 </html>
```

JavaScript Object Constructors

- Sometimes we need a "blueprint" for creating many objects of the same "type".
- The way to create an "object type", is to use an object constructor function.

Example:

```
<!DOCTYPE html>
<html>
<head>
 <title></title>
</head>
<body>
        <p id="demo"></p>
 <script>
        // Constructor function for Person
        objects function Person(first, last, age,
        eye) { this.firstName = first;
         this.lastName =
        last; this.age = age;
        this.eyeColor = eye;
         }
        // Create two Person objects
        var myFather = new Person("John", "Doe", 50, "blue");
        var myMother = new Person("Sally", "Rally", 48,
        "green"); // Display age
        document.getElementById("demo").innerHTML = "My father is " +
        myFather.age + ". My mother is " + myMother.age + ".";
</script>
```

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

this keyword

- In a function definition, this refers to the "owner" of the function.
- The JavaScript *this* keyword refers to the object it belongs to.
- It has different values depending on where it is used.
  - In a method, ***this*** refers to the **owner object**.
  - Alone, ***this*** refers to the **global object**.
  - In a function, ***this*** refers to the **global object**.
  - In a function, in strict mode, ***this*** is ***undefined***.
  - In an event, ***this*** refers to the **element** that received the event.
  - Methods like **call()** and **apply()** can refer ***this*** to **any object**.

Example:

```
<html>
<body>
    <h2>The JavaScript <b>this</b> Keyword</h2>
    <p>In this example, <b>this</b> represents the <b>person</b> object.</p>
    <p>Because the person object "owns" the fullName method.</p> <p
    id="demo"></p>
    <p>In this example <strong>this</strong> refers to person2, even if it is a method of
    person1:</p>
    <p id="demo1"></p>
    <button onclick="this.style.display='none'">Click to Remove Me!</button>
<script>
    // Create an object:
    var person = {
    firstName: "John",
    lastName : "Doe", id :
    5566, fullName :
    function() {
    return this.firstName + " " + this.lastName;
    }
    };
    // Display data from the object:
    document.getElementById("demo").innerHTML =
    person.fullName(); //using method call()
    var person1 = {
    fullName: function() {
    return this.firstName + " " + this.lastName;
    }
    }
    var person2 = {
    firstName:"Ram",
    lastName: "Prasad",
    }
    var x = person1.fullName.call(person2);
    document.getElementById("demo1").innerHTML = x;
</script>
</body></html>
```

JavaScript HTML DOM

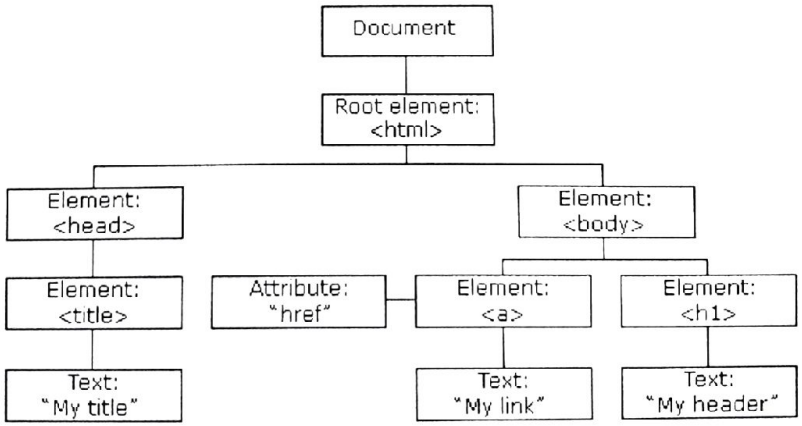
- DOM stands for Document Object Model has been under development by the W3C since the mid-1990s. DOM 3 is the current approved version.
- The DOM defines a standard for accessing documents.
- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The original motivation for the standard DOM was to provide a specification that would allow Java programs and JavaScript scripts that deal with HTML documents to be portable among various browsers.
- The DOM is an application programming interface(API) that defines an interface between HTML documents and application programs.
- The W3C DOM standard is separated into 3 different parts:
  - Core DOM - standard model for all document types
  - XML DOM - standard model for XML documents
  - HTML DOM - standard model for HTML documents

  But here we study only about HTML DOM in this chapter.

  The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:
  - The HTML elements as **objects**
  - The **properties** of all HTML elements
  - The **methods** to access all HTML elements
  - The **events** for all HTML elements

  In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

  The **HTML DOM** model is constructed as a tree of **Objects**:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes

- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

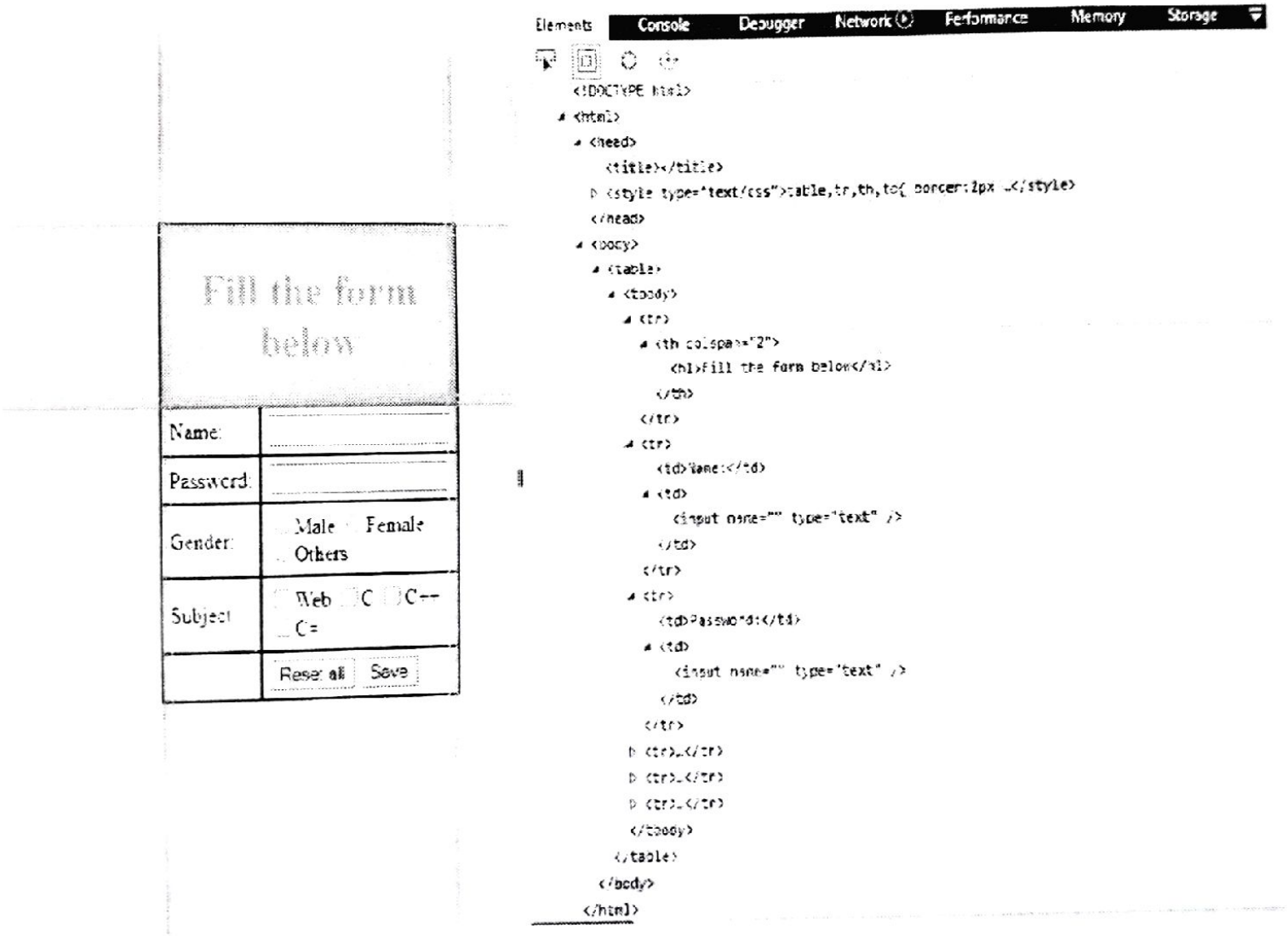


figure: The DOM structure of table.html with IE9

Event and Event Handling

- **Events** are action that are detected by JavaScript.
- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute.
- Examples of HTML events:
  -
  -
  -
  -
  -
  -
  -
- When an event occur then this action get response from script , this process is called **event handling**.

  In following example, the content of the ***<h1>*** element is changed when a user clicks on it:

```
<!DOCTYPE html>
<html>
<body>
  <h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>
```

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

Some example of JavaScript event type are given below:

onclick event type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse.

Example:

```
<html>
 <head>
   <script type = "text/javascript">
            function sayHello() {
            alert("Hello World")
            }
            document.getElementById("myBtn").onclick = displayDate;
            function displayDate() {
            document.getElementById("demo").innerHTML = Date();
            }
   </script>
 </head>
 <body>
   <p>Click the following button and see result</p>
   <button id="myBtn">Try it</button>
   <p id="demo"></p>
   <form>
     <input type = "button" onclick = "sayHello()" value = "Say Hello" />
   </form>
 </body>
</html>
```

onsubmit event type

onsubmit is an event that occurs when you try to submit a form.

Example:

```
<!DOCTYPE html>
<html>
<head>
   <title></title>
</head>
<body>
<form method="post" action="test9.php" target="_blank">
   <br><input type="text" name="firstName"> <br><input
   type="text" name="secondName"> <br><input
   type="submit" name="submit">
</form>
</body>
</html>
```

Note: When submit button is clicked ; form will be submitted and have action on php script named *"test9.php"* i.e. given below:

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title></title>
</head>
<body>
    Welcome <?php echo $_POST["firstName"]; ?><br>
    Your address is <?php echo $_POST["address"];?>
    <br>Hello <?php echo $_POST["firstName"]; ?>
</body>
</html>
```

onmouseover and onmouseout event type

The onmouseover event triggers when you bring your mouse over any element and the onmouseout triggers when you move your mouse out from that element.

Example:

```
<!DOCTYPE html>
<html>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>
<script>
function mOver(obj) {
 obj.innerHTML = "Release Me";
}
function mOut(obj) {
 obj.innerHTML = " Thank You ";
}
</script>
</body>
</html>
```

onmousedown and onmouseup event type

The ***onmousedown***, ***onmouseup***, and ***onclick*** events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered. Example:

```
<!DOCTYPE html>
<html>
<body>
    <div onmousedown="mDown(this)" onmouseup="mUp(this)"
    style="background-
    color:#D94A38;width:90px;height:20px;padding:40px;"> Click Me</div>
<script>
    function mDown(obj) {
    obj.style.backgroundColor = "#1ec5e5";
    obj.innerHTML = "Release Me";
}
    function mUp(obj) {
    obj.style.backgroundColor="#D94A38";
    obj.innerHTML="Thank You";
}
```

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

onchange event type

The onchange event is often used in combination with validation of input fields.

Example:

```
<html>
<head>
<script>
    function myFunction() {
    var x = document.getElementById("fname");
    x.value = x.value.toUpperCase();
}
</script>
</head>
<body>
    Enter your name: <input type="text" id="fname" onchange="myFunction()">
    <p>When you leave the input field, the input text transfer to upper case.</p>
</body>
</html>
```

onblur and onfocus event type

The ***onblur*** event occurs when an object loses focus. The onblur event is most often used with form validation code (e.g. when the user leaves a form field).

The ***onfocus*** event occurs when an element gets focus. The onfocus event is most often used with <input>, <select>, and <a>. Example:

```
<!DOCTYPE html>
<html>
<body>
    <p>This example uses the HTML DOM to assign an "onfocus" event to an input
    element.</p>
    <!-- When click for input it get focus but when click out the input field it looses
    focus i.e. blur. -->
    Enter your name: <input type="text" id="fname">
<script>
    document.getElementById("fname").onfocus = function() {myFunction()};
    function myFunction() {
    document.getElementById("fname").style.backgroundColor = "red";
    }
    document.getElementById("fname").onblur = function() {myFunction1()};
    function myFunction1() {
    document.getElementById("fname").style.backgroundColor = "yellow";
    }
</script>
</body>
</html>
```

There are a lot of event types that are used by JS . Some of them are listed below:

Attribute	Value	Description
Offline	script	Triggers when the document goes offline
Onabort	script	Triggers on an abort event
onafterprint	script	Triggers after the document is printed
onbeforeonload	script	Triggers before the document loads
onbeforeprint	script	Triggers before the document is printed
oncanplay	script	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	script	Triggers when media can be played to the end, without stopping for buffering
onchange	script	Triggers when an element changes
oncontextmenu	script	Triggers when a context menu is triggered
ondblclick	script	Triggers on a mouse double-click
ondrag	script	Triggers when an element is dragged
ondragend	script	Triggers at the end of a drag operation
ondragenter	script	Triggers when an element has been dragged to a valid drop target
ondragleave	script	Triggers when an element is being dragged over a valid drop target
ondragover	script	Triggers at the start of a drag operation
ondragstart	script	Triggers at the start of a drag operation
ondrop	script	Triggers when dragged element is being dropped
ondurationchange	script	Triggers when the length of the media is changed
onemptied	script	Triggers when a media resource element suddenly becomes empty.
onended	script	Triggers when media has reach the end
onerror	script	Triggers when an error occur
onformchange	script	Triggers when a form changes
onforminput	script	Triggers when a form gets user input
onhaschange	script	Triggers when the document has change
oninput	script	Triggers when an element gets user input
oninvalid	script	Triggers when an element is invalid

onkeydown	script	Triggers when a key is pressed
onkeypress	script	Triggers when a key is pressed and released
onkeyup	script	Triggers when a key is released
onload	script	Triggers when the document loads
onloadeddata	script	Triggers when media data is loaded
onloadedmetadata	script	Triggers when the duration and other media data of a media element is loaded
onloadstart	script	Triggers when the browser starts to load the media data
onmessage	script	Triggers when the message is triggered
onmousemove	script	Triggers when the mouse pointer moves
onmousewheel	script	Triggers when the mouse wheel is being rotated
onoffline	script	Triggers when the document goes offline
onoine	script	Triggers when the document comes online
ononline	script	Triggers when the document comes online
onpagehide	script	Triggers when the window is hidden
onpageshow	script	Triggers when the window becomes visible
onpause	script	Triggers when media data is paused
onplay	script	Triggers when media data is going to start playing
onplaying	script	Triggers when media data has start playing
onpopstate	script	Triggers when the window's history changes
onprogress	script	Triggers when the browser is fetching the media data
onratechange	script	Triggers when the media data's playing rate has changed
onreadystatechange	script	Triggers when the ready-state changes
onredo	script	Triggers when the document performs a redo
onresize	script	Triggers when the window is resized
onscroll	script	Triggers when an element's scrollbar is being scrolled
onseeked	script	Triggers when a media element's seeking attribute is no longer true, and the seeking has Ended
onseeking	script	Triggers when a media element's seeking attribute is true, and the seeking has begun
onselect	script	Triggers when an element is selected
onstalled	script	Triggers when there is an error in fetching media data

onstorage	script	Triggers when a document loads
onsuspend	script	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate	script	Triggers when media changes its playing position
onundo	script	Triggers when a document performs an undo
onunload	script	Triggers when the user leaves the document
onvolumechange	script	Triggers when media changes the volume, also when volume is set to "mute"
onwaiting	script	Triggers when media has stopped playing, but is expected to resume

Moving Elements and Element Positioning

We can move HTML elements in JavaScript as follows:

```
<html>
<head>
<script>
function moveDiv(obj){
  if(obj.align=="center"){
    obj.align="right";
  }
  else{
    obj.align="center";
  }
}
</script>
</head>
<body>
  <div id="myDiv" onclick="moveDiv(this)" align="center">
  Hello, World      </div>
</body>
</html>
```

The position property sets or returns the type of positioning method used for an element (static, relative, absolute or fixed).

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
  #myDIV {
   border: 1px solid black;
   background-color: lightblue;
   width: 300px;
  height: 300px;
  position: relative;
   top: 20px;
  }
</style>
</head>
```

```
<body>
  <p>Click the "Try it" button to change the position property of the DIV
  element:</p> <button onclick="myFunction()">Try it</button>

  <div id="myDIV">
        <p>This DIV element is placed 20 pixels from the top of its original
        position.</p> <p>Click the button to set the position property to "absolute" and
        see what happens.</p>
        <p>It will then be placed 20 pixels from the top the page.</p>
  </div>
<script>
        function myFunction() {
         document.getElementById("myDIV").style.position = "absolute";
}
</script>
</body>
```

Changing colors and fonts

Using JavaScript you can change any properties of CSS for any elements of HTML. The following example is for changing font color and its size.

Example:

```
<!DOCTYPE html>
<html>
<head>
        <title>Changing Color and Font size of
 content</title> <script type="text/javascript">
        function myFunction(){
        var x=document.getElementById('demo');
        x.style.color="Red";
        x.style.fontSize="50px";
        x.style.fontWeight="bold";
        x.style.fontStyle="italic";
   }
  </script>
</head>
<body>
        <p id="demo">Change me please!</p>
        <button onclick="myFunction()">Change</button>
</body>
</html>
```

Dynamic content and stacking elements

- The z-index attribute determines which element is in front and which are covered by the front element
- The JavaScript property associated with the z-index attribute is zIndex
- z-index can be changed dynamically (by changing zIndex)
- An image element can have an onclick attribute, so images can be clicked to trigger event handlers
- Anchors can also trigger event handlers when they are clicked

Example:

```
<!DOCTYPE html>
<html lang = "en">
  <head>
```

```
<title> Dynamic stacking of images </title>
<meta charset = "utf-8" />
<script type = "text/javascript">

        function toTop(obj){
         var topp = "airplane1";
        var domTop = document.getElementById(topp).style;
        var domNew = document.getElementById(obj).style;
        domTop.zIndex = "0";
        domNew.zIndex = "10";
        topp = obj;
}
 </script>
 <style type = "text/css">
        .plane1 {
        position: absolute;
        top: 0;
        width: 300px;
        height: 400px;
        left: 0;
        z-index: 0;
}
        .plane2 {
        position: relative;
        top: 50px;
        left: 110px;
        z-index: 0;}
        .plane3 {
        position: absolute;
        top: 100px;
         left: 220px;
        z-index: 0;}
  </style>
 </head>
 <body>
        <p>
        <img class ="plane1" id = "airplane1" src = "bas2.jpg" alt = "(Picture of an
        airplane)" onclick = "toTop('airplane1')" />
        <img class ="plane2" id = "airplane2" src = "karnali.jpg" alt = "(Picture of an
        airplane)" onclick = "toTop('airplane2')" />
        <img class ="plane3" id = "airplane3" src = "bas2.jpg" alt = "(Picture of an
        airplane)" onclick = "toTop('airplane3')" /> </p>

 </body>
</html>
```

Locating the mouse cursor and reacting to a mouse click

- The *clientX* property returns the horizontal coordinate (according to the client area) of the mouse pointer when a mouse event was triggered.(similarly we can use *clientY* for vertical). In current window.
- The *screenY* property returns the vertical coordinate (according to the users computer screen) of the mouse pointer when an event was triggered(similarly we can use *screenX* for horizontal coordinates).

Example:

```
<!DOCTYPE html>
<html>
<head>
        <style>
        div {
        width: 500px;
        height: 300px;
        border: 1px solid black;
        text-align: center;
        }
</style>
</head>
<body>
        <div onclick="showCoords(event)"><p id="demo"> </p></div>
        <p><strong>Tip:</strong> Try to click different places in the div.</p>
<script>
        function showCoords(event) {
         var cX = event.clientX;
         var sX = event.screenX;
         var cY = event.clientY;
        var sY = event.screenY;
        var coords1 = "client - X: " + cX + ", Y coords: " + cY; var coords2 = "screen - X:
        " + sX + ", Y coords: " + sY; document.getElementById("demo").innerHTML =
        coords1 + "<br>" + coords2; }

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

Dragging and dropping elements

Drag and drop elements allows us for dragging and dropping elements from one position to another.

Steps:

1. First make an element ***draggable*** , set ***draggable*** attribute to ***true***.
2. Then specify , what should happen when element is dragged. ***ondragstart*** attribute calls a function (***drag(event)***), that specifies what data to be dragged and the ***dataTransfer.setData()*** method sets the data type and the value of the dragged data.
3. Then also specify where to drop dragged data (i.e. specified by ***ondragover*** event). Also to allow a ***drop*** , we must prevent the default handling of the element.
4. ***preventDefault()*** method is used to prevent the browser default handling of the data.
5. When the dragged data is dropped , a drop event occurs. The ***ondrop*** attribute calls a function , ***drop(event).***
6. Append the dragged element into the drop element.

Example:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
    #div1 {
            width: 350px;
            height: 71px;
            padding: 10px;
            border: 1px solid #aaaaaa;
            }
</style>
<script>
            function allowDrop(ev) {
             ev.preventDefault();
            }
            function drag(ev) {
             ev.dataTransfer.setData("text", ev.target.id);
            }
            function drop(ev) {
            ev.preventDefault();
            var data = ev.dataTransfer.getData("text");
            ev.target.appendChild(document.getElementById(data));
            }
</script>
</head>
<body>
            <p>Drag the image into the rectangle:</p>
            <div id="div1" ondrop="drop(event)"
            ondragover="allowDrop(event)"></div> <br>
            <img id="drag1" src="karnali.jpg" draggable="true" ondragstart="drag(event)"
            width="336" height="70">
</body>
</html>
```

Display and Visibility in JavaScript

The **display** property sets or returns the element's display type. The **visibility** property sets or returns whether an element should be visible. Both properties allows author to show or hide an element. However, if you set ***display: none***, it hides the entire element, while ***visibility: hidden*** means that the contents of the element will be invisible, but the element stays in its original position and size.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Display and Visibility Properties</title>
</head>
<body>
  <p id="hideShow"> click HIDE button to hide and SHOW button to show</p>
```

```
<button onclick="hideDisplay()">Hide</button>
<button onclick="showDisplay()">SHOW</button>
<p id="hideShow1"> click HIDDEN button to hide and VISIBLE button to show</p>
<button onclick="hideVisibility()">HIDDEN</button> <button
onclick="showVisibility()">VISIBLE</button>
<script type="text/javascript">
        var disp=document.getElementById('hideShow').style;
        function hideDisplay(){
        disp.display="none";
        }
        function showDisplay(){
        disp.display="block";
        }

        var visi=document.getElementById('hideShow1').style;
        function hideVisibility(){
        visi.visibility="hidden";
        }
        function showVisibility(){
        visi.visibility="visible";
        }
</script>
</body>
</html>
```

Regular Expression(Pattern Matching)

A regular expression is an object that describes a pattern of characters. The JavaScript **RegExp** class represents regular expressions, and both String and **RegExp** define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

Syntax:

var pattern=new RegExp(pattern, attributes);

or simply

var pattern=/pattern/attributes;

Here is the description of the parameters −

- **pattern** − A string that specifies the pattern of the regular expression or another regular expression.
- **attributes** − An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multi-line matches, respectively.

Brackets[]:

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Sr.No.	Expression & Description
1	[...] Any one character between the brackets.

2	**[^...]** Any one character not between the brackets.
3	**[0-9]** It matches any decimal digit from 0 through 9.
4	**[a-z]**It matches any character from lowercase **a** through lowercase **z**.
5	**[A-Z]**It matches any character from uppercase **A** through uppercase **Z**.
6	**[a-Z]**It matches any character from lowercase **a** through uppercase **Z**.

Quantifiers:

Sr.No.	Expression & Description
1	**p+** It matches any string containing one or more p's.
2	**p*** It matches any string containing zero or more p's.
3	**p?** It matches any string containing at most one p.
4	**p{N}** It matches any string containing a sequence of **N** p's
5	**p{2,3}** It matches any string containing a sequence of two or three p's.
6	**p{2, }** It matches any string containing a sequence of at least two p's.
7	**pS** It matches any string with p at the end of it.
8	**^p** It matches any string with p at the beginning of it.

Examples:

Following examples explain more about matching characters.

Sr.No.	Expression & Description
1	**[^a-zA-Z]** It matches any string not containing any of the characters ranging from **a** through **z** and **A** through Z.
2	**p.p** It matches any string containing **p,** followed by any character, in turn followed by another p.

3	**^.{2}$** It matches any string containing exactly two characters.
4	**(.*)** It matches any string enclosed within and .
5	**p(hp)*** It matches any string containing a **p** followed by zero or more instances of the sequence **hp**.

Literal Characters:

Sr.No.	Character & Description
1	Alphanumeric Itself
2	**\0** The NUL character (\u0000)
3	**\t** Tab (\u0009
4	**\n** Newline (\u000A)
5	**\v** Vertical tab (\u000B)
6	**\f** Form feed (\u000C)
7	**\r** Carriage return (\u000D)
8	**\xnn** The Latin character specified by the hexadecimal number nn; for example, \x0A is the same as \n
9	**\uxxxx** The Unicode character specified by the hexadecimal number xxxx; for example, \u0009 is the same as \t
10	**\cX** The control character ^X; for example, \cJ is equivalent to the newline character \n

Metacharacters:

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

Sr.No.	Character & Description
1	**.** a single character
2	**\s** a whitespace character (space, tab, newline)
3	**\S** non-whitespace character
4	**\d** a digit (0-9)

5	**\D** a non-digit
6	**\w** a word character (a-z, A-Z, 0-9, _)
7	**\W** a non-word character
8	**[\b]** a literal backspace (special case).
9	**[aeiou]** matches a single character in the given set
10	**[^aeiou]** matches a single character outside the given set
11	**(foo|bar|baz)** matches any of the alternatives specified

Modifiers:

Several modifiers are available that can simplify the way you work with **regexps,** like case sensitivity, searching in multiple lines, etc.

Sr.No.	Modifier & Description
1	**i** Perform case-insensitive matching.
2	**m** Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a string boundary
3	**g** Performs a global match that is, find all matches rather than stopping after the first match.

Example: **global** is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs global matching, i.e., whether it was created with the "g" attribute. Returns "TRUE" if the "g" modifier is set, "FALSE" otherwise.

```
<html>
  <head>
    <title>JavaScript RegExp global Property</title>
  </head>
  <body>
    <script type = "text/javascript">
      var re = new RegExp( "string" );
      if ( re.global ) {
        document.write("Test1 - Global property is set");
      }
      else {
        document.write("Test1 - Global property is not set");
      }
      re = new RegExp( "string", "g" );
      if ( re.global ) {
        document.write("<br />Test2 - Global property is set");
      }
      else {
```

```
        document.write("<br />Test2 - Global property is not set");
      }
    </script>
  </body>
</html>
```

test() method: It searches a string for a pattern, and returns true or false, depending on the result.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Regular Expressions</h2>
    <p>Search for an "e" in the next paragraph:</p>
    <p id="p01">The best things in life are free!</p>
    <p id="demo"></p>
<script>
    text = document.getElementById("p01").innerHTML;
    document.getElementById("demo").innerHTML = /e/.test(text);
</script>
</body>
</html>
```

exec() Method: It searches a string for a specified pattern, and returns the found text as an object. If no match is found, it returns an empty *(null)* object.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Regular Expressions</h2>
    <p id="demo"></p>
    <script>
    var obj = /t/i.exec("The best things in life are free!");
    document.getElementById("demo").innerHTML =
    "Found " + obj[0] + " in position " + obj.index + " in the text: " +
obj.input; </script>
</body>
</html>
```

Error Handling

There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

- **Syntax error** – also called **parsing errors,** occur at compile time in traditional programming languages and at interpret time in JavaScript. When syntax error occurs , only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.
- Runtime error – also called **exceptions,** occur during execution (after compilation/interpretation). Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

- Logical error – Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions. You can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

Example: You can use **finally** block which will always execute unconditionally after the try/catch.

```
<html>
  <head>
    <script type = "text/javascript">
        function myFunc() {
          var a = 100;
          try {
            alert("Value of variable a is : " + a );
          }
          catch ( e ) {
            alert("Error: " + e.description );
          }
          finally {
            alert("Finally block will always execute!" );
          }
        }
    </script>
  </head>
  <body>
    <p>Click the following to see the result:</p>
    <form>
      <input type = "button" value = "Click Me" onclick = "myFunc();"
    /> </form>
  </body>
</html>
```

The throw statement

You can use **throw** statement to raise your built-in exceptions or your customized exceptions.

Example:(validate input number)

```
<html>
<body>
  <p>Please input a number between 5 and 10:</p>
  <input id="demo" type="text">
  <button type="button" onclick="myFunction()">Test Input</button>
```

```
  <p id="p01"></p>
<script>
  function myFunction() {
  var message, x;
  message = document.getElementById("p01");
  message.innerHTML = "";
  x = document.getElementById("demo").value;
  try {
   if(x == "") throw "empty"; //isNaN
   , is Not a Number if(isNaN(x))
   throw "not a number";
   x = Number(x);
   if(x < 5) throw "too low";
   if(x > 10)  throw "too high";
  }
  catch(err) {
   message.innerHTML = "Input is " + err;
  }
 }
</script>
</body>
</html>
```

The onerror() method

The **onerror** event handler was the first feature to facilitate error handling in JavaScript. The **error** event is fired on the window object whenever an exception occurs on the page.

The **onerror** event handler provides three pieces of information to identify the exact nature of the error :

- **Error message** – The same message that the browser would display for the given error
- **URL** – The file in which the error occurred
- **Line number** – The line number in the given URL that caused the error

Example:

```
<html>
  <head>
    <script type = "text/javascript">
        window.onerror = function (msg, url, line) {
          alert("Message : " + msg );
          alert("url : " + url );
          alert("Line number : " + line );
        }
    </script>
  </head>
  <body>
    <p>Click the following to see the result:</p>
```

```
    <form>
      <input type = "button" value = "Click Me" onclick = "myFunc();"
    /> </form>
  </body>
</html>
```

Type conversion in PHP

JavaScript variables can be converted to a new variable and another data type:

- By the use of a JavaScript function
- **Automatically** by JavaScript itself

Converting Numbers to Strings

The global method ***String()*** can convert numbers to strings. Similarly ***boolean*** , and ***date*** can be converted to string.

Example:

```
<!DOCTYPE html>
<html>
<body>
    <h2>The JavaScript String() Method</h2>
    <p>The String() method can convert a number to a
    string.</p> <p id="demo"></p>
<script>
    var x = 123;
    document.getElementById("demo").innerHTML =
    String(x) + "<br>" +
    String(123) + "<br>" +
    String(100 + 23);
</script>
</body>
</html>
```

Converting Strings to Numbers

Example:

```
<script>
    var x = "123";
    document.getElementById("demo").innerHTML =
    Number(x) ;
    var y="5"; //y is a string
    var z=+y; //z is a number
    document.getElementById('demo1').innerHTML=z;
    var a="4"; //a is string
    var b=6; //b is number
    var c=a+b; //+ sign concatenate "4" and 6 =46 not =10
    document.getElementById('demo2').innerHTML=c;
</script>
```