Module 4: JavaScript

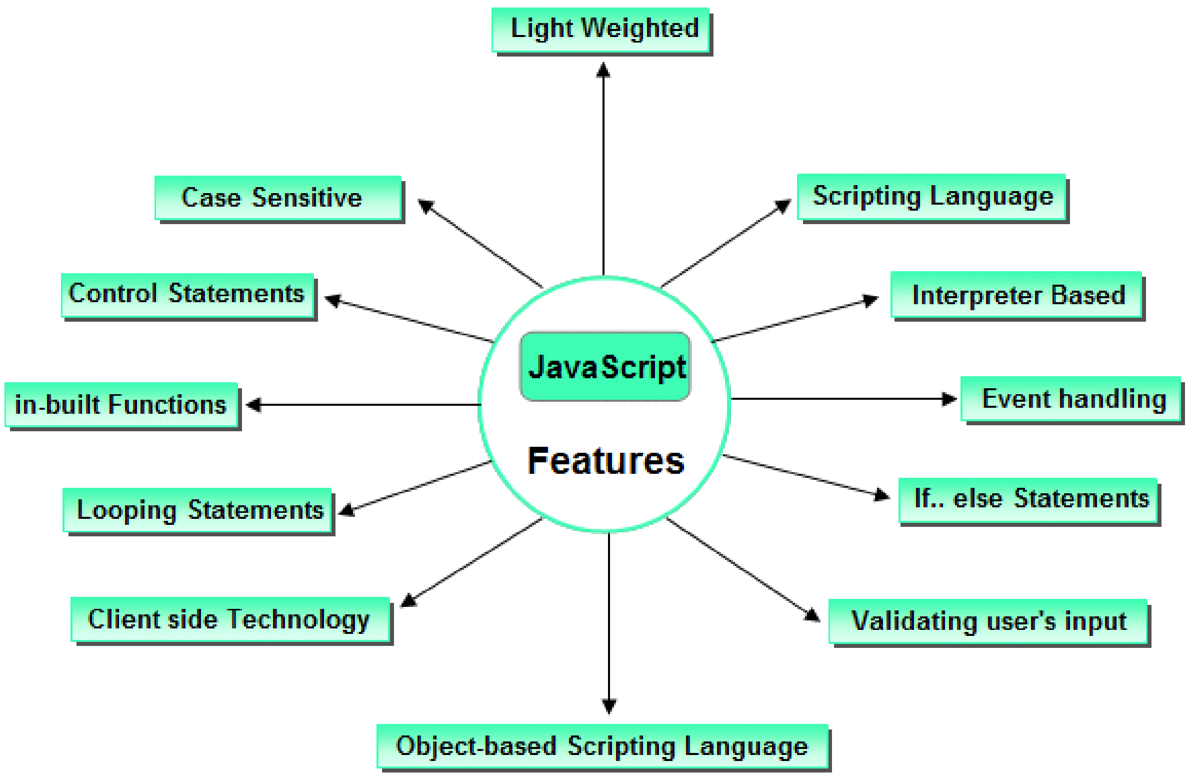
JavaScript is an object-oriented scripting language that is predominantly used for adding dynamic components of various types to websites. It is first implemented by **Netscape** (with help from Sun Microsystems). JavaScript was created by **Brendan Eich** at Netscape in 1995 for the purpose of allowing code in webpages.

User can only design a web page Using HTML. By using JavaScript user can run any logic on web browser like addition of two numbers, check any condition, looping statements (for, while), decision making statement (if-else) at client side.

JavaScript is a part of the basic HTML page. It is contained inside the <script>....</script> tags. By default, scripts in a page will be executed immediately while the page loads into the browser. User can execute a script when a user triggers an event. User can place the script in the head section or body section of the HTML page or in a separate file also. The extension of the file is .js.

Features of JavaScript

JavaScript is a client side technology. The main features are



- JavaScript is a light weighted object-based (it provides predefined objects) scripting language and it is not java.
- Giving the user more control over the browser and OS.
- Handling dates and time.

- JavaScript is interpreter based scripting language.
- JavaScript is case sensitive.
- Every statement in JavaScript must be terminated with semicolon (;).
- Most of the JavaScript control statements syntax is same as syntax of control statements in C language.
- Ability to create new functions within scripts. Declare a function in JavaScript using **function** keyword.

It is mainly used for:

- Validate user input in an HTML form before sending the data to a server (Client-side validation)
- Displaying Dynamic drop-down menus and popup windows or dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)
- Displaying clocks (data and time).
- Build forms that respond to user input without accessing a server.
- Change the appearance of HTML documents and dynamically write HTML into separate Windows.
- Open and close new windows or frames.
- Manipulate HTML "layers" including hiding, moving, and allowing the user to drag them around a browser window.
- Build small but complete client side programs.

Comment

Comments are used to deliver messages. It is used to add information about the code, warnings or suggestions so that the end user or other developer can easily interpret the code. There are two types of comments are in JavaScript

- Single-line Comment
- Multi-line Comment

<u>Single-line Comment</u>

It is represented by double forward slashes (//). It can be used before any statement.

```
<script>
// It is single line comment
document.write ("Hello JavaScript");
```

```
</script>
```

Multi-line Comment

It can be used to add single as well as multi line comments. It is represented by forward slash (/) with asterisk (*) then asterisk with forward slash.

Example

```
<script>
/* It is multi line comment.
It will not be displayed */
document.write("Javascript multiline comment");
</script>
```

JavaScript Variables

JavaScript variables are containers for storing data values. All JavaScript variables must be identified with unique names. These unique names are called identifiers. JavaScript identifiers are case-sensitive. The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

var keyword is used before variable name to declare any variable. This keyword is optional

Syntax

```
var x;
```

```
<script>
var a=10;
var b=20;
var c=a+b;
document.write(c);
```

```
</script>
```

Types of Variable in JavaScript

- ***Local Variable*** ➔ A variable which is declared inside block or function is called local variable. It is accessible within the function or block only.

```
<script>
function abc(){
var x=10;  //local variable
}
</script>
```

```
<script>
If(10<13){
var y=20;//javascript local variable
}
</script>
```

- ***Global Variable***➔ A global variable is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable.

```
<script>
var value=10;//global variable
function a(){
alert(value);
}
function b(){
alert(value);
}  </script>
```

Declaring global variable through window object

```
window.value=20;
```

```
function m(){
window.value=200; //declaring global variable by window object
}
```

```
function n(){
alert(window.value); //accessing global variable from other function
}
```

Example of JavaScript

```
<!doctype html>
<html>
<head>
<script>
function add(){
var a,b,c;
a=Number(document.getElementById("first").value);
b=Number(document.getElementById("second").value);
c= a + b;
document.getElementById("answer").value= c;
}
</script>
</head>
<body>
<input id="first">
<input id="second">
<button onclick="add()">Add</button>
<input id="answer">
</body>
</html>
```

Code Explanation

- **no=Number(document.getElementById("first").value);**

  This code is used for receive first input value form input field which have id **first**.
- **no=Number(document.getElementById("second").value);**

  This code is used for receive first input value form input field which have id **second**.
- **document.getElementById("answer").value= fact;**

  This code is used for receive calculated value of factorial and display in input field which have id **answer**

- **<button onclick="add()">Add</button>**

  This code is used for call add function when button clicked

JavaScript Operators

JavaScript Operators use either value or variable to compute some task. JavaScript has following types of operators,

1. Arithmetic Operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Conditional Operator (Ternary Operator)
6. Bitwise Operators
7. Miscellaneous Operators
    - typeof
    - delete
    - instanceof
    - new
    - this
    - in

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on numbers (literals or variables). JavaScript arithmetic operator takes operand (as a values or variable) and returns the single value. In the following examples, x = 10, y = 5 and result variable is used to store the result of the operation.

Operator	Description	Example	Results
+	Addition	result = x + y	result = 15
-	Subtraction	result = x - y	result = 5
*	Multiplication	result = x * y	result = 50
/	Division	result = x / y	result = 2
%	Modulus	result = x % y	result = 0

++	Increment	result = x++
		result = x
		result = ++x
			result = 10
			result = 11
			result = 12

--	Decrement	result = x--
		result = x
		result = --x
			result = 12
			result = 11
			result = 10

JavaScript Assignment Operators

JavaScript assignment operators are used to assign values to JavaScript variables.

Operator	Example	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

Operator	Sign	Description	Example	Equivalent to	Results
Assignment	=	Assign value from one operand to another operand value.	result = x	result = x	result = 17
Addition	+=	Addition of operands and finally assign to left operand.	result += x	result = result + y	result = 22
Subtraction	-=	Subtraction of operands and finally assign to left operand.	result -= y	result = result - y	result = 17
Multiplication	*=	Multiplication of operands and finally assign to left operand.	result *= y	result = result * y	result = 85
Division	/=	Division of operands and finally assign to left operand.	result /= y	result = result / y	result = 17
Modulus	%=	Modulus of operands and finally assign to left operand.	result %= y	result = result % y	result = 2
Bitwise AND	&=	AND operator compare two bits values return a results of 1, If both bits	result &= y	result = result & y = 2 & 5 = 0000 0010 &	result = 0

		are 1. otherwise return 0.		0000 0101 = 0000 0000 = 0	
Bitwise OR	|=	OR operator compare two bits values and return result of 1, If the bits are complementary. Otherwise return 0.	result |= y	result = result | y = 2 | 5 = 0000 0010 | 0000 0101 = 0000 0111 = 7	result = 7
Bitwise XOR	^=	EXCLUSIVE OR operator compare two bits values and return a results of 1, If any one bits are 1 or either both bits one.	result ^= y	result = result ^ y = 7 ^ 5 = 0000 0111 ^ 0000 0101 = 0000 0010 = 2	result = 2
Shift Left	<<=	Shift left operator move the bits to a left side.	result <<= y	result = result <<= y = 2 <<= 5 = 0000 0010 <<= 0100 0000 = 64	result = 64
Shift Right	>>=	Shift left operator move the bits to a left side.	result >>= y	result = result >>= y = 2 >>= 5 = 0100 0000 >>= 0000 0010 = 2	result = 2

String Operators

The + operator can also be used to add (concatenate) strings. The += assignment operator can also be used to add (concatenate) strings.

```
 txt1 = "John";
 txt2 = "Doe";
 txt3 = txt1 + " " + txt2;
```

Comparison and Logical Operators

JavaScript comparison operator determines the two operands satisfy the given condition. Comparison operator returns either true or false.

Operator	**Description**
==	equal to

===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Operator	Sign	Description
Equal	==	If both operands are equal, returns true.
Identical equal	===	If both operands are equal and/or same data type, returns true.
Not equal	!=	If both operands are not equal, returns true.
Identical not equal	!==	If both operands are not equal and/or same data type, returns true.
Greater than	>	If left operand larger than right operand, return true.
Less then	<	If left operand smaller than right operand, return true.
Greater than, equal	>=	If left operand larger or equal than right operand, return true.
Less than, equal	<=	If left operand smaller or equal than right operand, return true.

JavaScript logical operators return Boolean result based on operands.

Operator	Sign	Description
Logical AND	&&	Return true if both are must be true, otherwise return false.
Logical OR	||	Evaluate both operands, Return true if either both or any one operand true, Return false if both are false.
Logical NOT	!	Return the inverse of the given value result true become false, and false become true.

JavaScript bitwise operators

JavaScript bitwise operators evaluate and perform specific bitwise (32 bits either zero or one) expression.

Operator	Sign	Description
Bitwise AND	&	Return bitwise AND operation for given two operands.
Bitwise OR	|	Return bitwise OR operation for given two operands.
Bitwise XOR	^	Return bitwise XOR operation for given two operands.
Bitwise NOT	~	Return bitwise NOT operation for given operand.
Bitwise Shift Left	<<	Return left shift of given operands.
Bitwise Shift Right	>>	Return right shift of given operands.
Bitwise Unsigned Shift Right	>>>	Return right shift without consider sign of given operands.

JavaScript Type Operators

Operator	**Description**
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

JavaScript typeof operator returns valid data type identifiers as a string of given expression. **typeof** operator return six possible values: "string", "number", "boolean", "object", "function", and "undefined".

Syntax:

```
typeof expression
typeof(expression)
```

Example :

```
var name = 'Opal Kole';
var age = 48;
var married = true;
var experience = [2010, 2011, 2012, 2013, 2014];
var message = function(){ console.log("Hello world!"); }
var address;

typeof name;       // Returns "string"
typeof age;        // Return "number"
typeof married;    // Return "boolean"
typeof experience;  // Return "object"
typeof message;    // Return "function"
typeof address;    // Return "undefined"
```

JavaScript delete operator deletes object property or removes specific element in an array. It will return false if element not exist, array element undefined etc. otherwise returns true.

Syntax:

```
delete expression;
delete array;                                    // delete array
delete array[index];
```

Example 1:

```
var address = "63 street Ct.";
delete address;                       // Returns false, Using var keyword you can't delete
add = "63 street Ct.";
delete add;                           // Returns true, explicit declare you can delete
```

Example 2:

```
var myObj = new Object();
myObj.name = "Opal Kole";
myObj.age = 48;
myObj.married  = true;
delete myObj.name;                                  // delete object property
delete myObj["count"];                              // delete object property

var experience = [2010, 2011, 2012, 2013, 2014];    // array elements
delete experience[2];                 // delete 2nd index from array elements
console.log(experience);              // [2010, 2011, undefined Ã— 1, 2013, 2014]
```

JavaScript *instanceof* indicates a boolean result. Returns true, if the object is an instance of specific class.

Syntax:

```
Object instanceof class
```

JavaScript *new* operator is used to create an instance of the object.

Syntax:

```
var myObj = new Object;
var myObj = new Object( );            // or you can write
          // Object - required, for constructor of the object.
var arr = new Array( [ argument1, argument2, ..., ..., argumentN ] );
          // argument - optional, pass any number of argument in a object.
```

Example :

```
var myObj = new Object(); // or you can write: var myObj = new Object;
myObj.name = "Opal Kole";
myObj.address = "63 street Ct.";
myObj.age = 48;
myObj.married  = true;
console.log(myObj);
// Object {name: "Opal Kole", address: "63 street Ct.", age: 48, married: true}
```

JavaScript *this* operator represents the current object.

```
Syntax:
this["propertyname"]
this.propertyname
```

Example :

```
function employee(name, address, age, married) {
  this.name = name;
  this.address = address;
  this.age = age;
  this.married = married;
}
var myObj = new employee("Opal Kole", "63 street Ct.", 48, true);
console.log(myObj);
// employee {name: "Opal Kole", address: "63 street Ct.", age: 48, married: true}
```

JavaScript *in* operator returns a boolean result if specified property exist in an object.

```
Syntax:
property in object
```

Example :

```
<script>
        var myObj = new Object();          // or you can write: var myObj = new Object;
        myObj.name = "Opal Kole";
        myObj.address = "63 street Ct.";
        myObj.age = 48;
        myObj.married  = true;

        document.writeln("name" in myObj);      // Returns true
        document.writeln("birthdate" in myObj);  // Returns false
                                 // birthdate propery not in myObj
        document.writeln("address" in myObj);    // Returns true
        document.writeln("age" in myObj);       // Returns true
        document.writeln("married" in myObj);    // Returns true
</script>
```

JavaScript Data Types

JavaScript is a *loosely typed* or a *dynamic* language. i.e; No need to declare the type of a variable ahead of time. The type will get determined automatically while the program is being processed. JavaScript variables can hold many **data types**: numbers, strings, arrays, objects and more.

```
var length = 16;                       // Number
var lastName = "Johnson";               // String
var cars = ["Saab", "Volvo", "BMW"];       // Array
var x = {firstName:"John", lastName:"Doe"};    // Object
```

Primitive datatype:

A **primitive** (primitive value, primitive data type) is data that is not an object and has no methods. There are 6 primitive data types: string, number, boolean, null, undefined, symbol. All primitives are **immutable** (cannot be changed). JavaScript primitives data types are,

- Number
- String
- Boolean
- Symbol
- Null
- Undefined

and Object data types are,

- Array
- JSON
- Function, Object and Properties.

JavaScript Variables are declared using **var** statement. Javascript will store undefined value if **var** is not specified.

Boolean type: Boolean represents a logical entity and can have two values: true, and false. It is used to determine a condition/expression is true.

```
var val1 = true;

var val2 = false;
```

Null type: - The Null type has exactly one value: null. JavaScript Null is used to specify the variable is empty.

```
var str = null;
```

Undefined type:- A variable that has not been assigned a value has the value undefined. JavaScript uninitialized variables value are undefined. Uninitialized variable value is equal to null.

Number type:- JavaScript has only one Number (numeric) data types. Number data type can store normal integer, floating-point values. A floating-point represent a decimal integer with either decimal points or fraction expressed.

String type: - JavaScript string data type represent textual data surrounding to single/double quotes. Each element in the String occupies a position in the String. The first element is at index 0, the next at index 1, and so on. The length of a String is the number of elements in it.

```
var name = 'Hello, I am run this town.!';             // Single quote

var name = "Hello, I am run this town.!";             // Double quote

var name = "Hello, I'm run this town.!";// Single quote use inside string

var name1 = "";                                       // empty string
```

String Methods and Properties
<u>String Length</u> :- The length property returns the length of a string:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Result is 26
<u>Finding a String in a String</u>:-The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

Result is 7
The lastIndexOf() method returns the index of the last occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Result is 21
Both the indexOf(), and the lastIndexOf() methods return -1 if the text is not found. JavaScript counts positions from zero. 0 is the first position in a string, 1 is the second, 2 is the third. .Both methods accept a second parameter as the starting position for the search.
<u>Searching for a String in a String: -</u>The search() method searches a string for a specified value and returns the position of the match:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

Result is 7
<u>Extracting String Parts</u>
There are 3 methods for extracting a part of a string:

- slice(start, end)
- substring(start, end)
- substr(start, length)

The slice() Method
slice() extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters: the starting index (position), and the ending index (position).
Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7,13);
```

Result is Banana
If a parameter is negative, the position is counted from the end of the string.
Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12,-6);
```

Result is Banana

If the second parameter is omitted, the method will slice out the rest of the string:

Example

```
var res = str.slice(7);
```

Result is Banana, Kiwi

or, counting from the end:

Example

```
var res = str.slice(-12);
```

Result is Banana, Kiwi

The substring() Method

substring() is similar to slice(). The difference is that substring() cannot accept negative indexes.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7,13);
```

Result is Banana

If the second parameter is omitted, substring() will slice out the rest of the string.

The substr() Method

substr() is similar to slice(). The difference is that the second parameter specifies the length of the extracted part.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7,6);
```

Result is Banana

If the first parameter is negative, the position counts from the end of the string. The second parameter cannot be negative, because it defines the length. If the second parameter is omitted, substr() will slice out the rest of the string.

<u>Replacing String Content</u>

The **replace()** method replaces a specified value with another value in a string:

Example

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft","Saintgits");
```

result is Please visit Saintgits!

The replace() method can also take a regular expression as the search value. By default, the replace() function replaces only the first match. To replace all matches, use a regular expression with a g flag (for global match):

Example

```
str = "Please visit Microsoft and Microsoft!";
var n = str.replace(/Microsoft/g,"Saintgits");
```

Result is Please visit Saintgits and Saintgits!

The replace() method does not change the string it is called on. It returns a new string.

<u>Converting to Upper and Lower Case</u>

A string is converted to upper case with **toUpperCase()**:

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper
Result is  HELLO WORLD!
```

A string is converted to lower case with **toLowerCase()**:

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower
Result is   hello world!
```

The concat() Method

concat() joins two or more strings:

Example

```
var text1 = "Hello";
var text2 = "World";
text3 = text1.concat(" ",text2);
Result is  Hello World
```

The **concat()** method can be used instead of the plus operator.

Example

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ","World!");
```

Extracting String Characters

There are 2 **safe** methods for extracting string characters:

- charAt(position)
- charCodeAt(position)

The charAt() Method

The **charAt()** method returns the character at a specified index (position) in a string:

Example

```
var str = "HELLO WORLD";
str.charAt(0);           // returns H
```

The charCodeAt() Method

The **charCodeAt()** method returns the unicode of the character at a specified index in a string:

```
Example
var str = "HELLO WORLD";
str.charCodeAt(0);        // returns 72
```

Accessing a String as an Array

```
var str = "HELLO WORLD";
str[0];                // returns H
```

Symbol:- JavaScript Symbol data type new (Currently ECMAScript 6 Drafted) used for identifier unique object properties.

```
Symbol([description])
```

```
var s1 = Symbol();
var s1 = Symbol('name');
```

JavaScript Number Methods

***The toString() Method* :-** toString() returns a number as a string**.** All number methods can be used on any type of numbers (literals, variables, or expressions)

```
Example

var x = 123;
x.toString();          // returns 123 from variable x
(123).toString();       // returns 123 from literal 123
(100 + 23).toString();   // returns 123 from expression 100 + 23
```

The toFixed() Method
toFixed() returns a string, with the number written with a specified number of decimals:
Example

```
var x = 9.656;
x.toFixed(0);         // returns 10
x.toFixed(2);         // returns 9.66
x.toFixed(4);         // returns 9.6560
x.toFixed(6);         // returns 9.656000
```

The toPrecision() Method
toPrecision() returns a string, with a number written with a specified length:

```
Example
var x = 9.656;
x.toPrecision();       // returns 9.656
x.toPrecision(2);      // returns 9.7
x.toPrecision(4);      // returns 9.656
x.toPrecision(6);      // returns 9.65600
```

The valueOf() Method
valueOf() returns a number as a number.

Example

```
var x = 123;
x.valueOf();          // returns 123 from variable x
(123).valueOf();      // returns 123 from literal 123
(100 + 23).valueOf(); // returns 123 from expression 100 + 23
```

<u>Converting Variables to Numbers</u>

There are 3 JavaScript methods that can be used to convert variables to numbers:

- The Number() method
- The parseInt() method
- The parseFloat() method

These methods are not **number** methods, but **global** JavaScript methods.

Global Methods

JavaScript global methods can be used on all JavaScript data types. These are the most relevant methods, when working with numbers:

Method	Description
Number()	Returns a number, converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns an integer

The Number() Method

Number() can be used to convert JavaScript variables to numbers:

Example

```
x = true;
Number(x);      // returns 1
x = false;
Number(x);      // returns 0
x = new Date();
Number(x);      // returns 1404568027739
x = "10"
Number(x);      // returns 10
x = "10 20"
Number(x);      // returns NaN
```

The parseInt() Method

parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

Example

```
parseInt("10");       // returns 10
parseInt("10.33");    // returns 10
parseInt("10 20 30"); // returns 10
parseInt("10 years"); // returns 10
parseInt("years 10"); // returns NaN
```

If the number cannot be converted, NaN (Not a Number) is returned.

The parseFloat() Method

parseFloat() parses a string and returns a number. Spaces are allowed. Only the first number is returned:

Example

```
parseFloat("10");       // returns 10
parseFloat("10.33");    // returns 10.33
parseFloat("10 20 30"); // returns 10
parseFloat("10 years"); // returns 10
parseFloat("years 10"); // returns NaN
```

Find sum of two number using JavaScript

Example

```
<!doctype html>
<html><head><script>
function add(){
var a,b,c;
a=Number(document.getElementById("first").value);
b=Number(document.getElementById("second").value);
c= a + b;
document.getElementById("answer").value= c;
}
</script></head>
<body>
<input id="first">
<input id="second">
<button onclick="add()">Add</button>
<input id="answer"></body> </html>
```

Java Objects

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers

Example

```
Math.PI;          // returns 3.141592653589793
```

Math.round()

Math.round(x) returns the value of x rounded to its nearest integer:

Example

```
Math.round(4.7);   // returns 5
Math.round(4.4);   // returns 4
```

Math.pow()
Math.pow(x,y) returns the value of x to the power of y:
Example

```
Math.pow(8,2);     // returns 64
```

Math.sqrt()
Math.sqrt(x) returns the square root of x:
Example

```
Math.sqrt(64);     // returns 8
```

Math.abs()
Math.abs(x) returns the absolute (positive) value of x:
Example

```
Math.abs(-4.7);    // returns 4.7
```

Math.floor()
Math.floor(x) returns the value of x rounded **down** to its nearest integer:
Example

```
Math.floor(4.7);   // returns 4
```

Math.sin()
Math.sin(x) returns the sine (a value between -1 and 1) of the angel x (given in radians).
Angle in radians = Angle in degrees x PI / 180.
Example

```
Math.sin(90 * Math.PI / 180);     // returns 1 (the sine of 90 degrees)
```

Math.cos()
Math.cos(x) returns the cosine (a value between -1 and 1) of the angel x (given in radians).
Angle in radians = Angle in degrees x PI / 180.
Example

```
Math.cos(0 * Math.PI / 180);     // returns 1 (the cos of 0 degrees)
```

Math.min() and Math.max()
Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments:
Example

```
Math.min(0, 150, 30, 20, -8, -200);  // returns -200
```

Example

```
Math.max(0, 150, 30, 20, -8, -200);  // returns 150
```

Math.random()
Math.random() returns a random number between 0 (inclusive), and 1 (exclusive):
Example

```
Math.random();     // returns a random number
```

Math Object Methods

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians

atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
atan2(y,x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns the value of x rounded up to its nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns the value of x rounded down to its nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x,y,z,...,n)	Returns the number with the highest value
min(x,y,z,...,n)	Returns the number with the lowest value
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Returns the value of x rounded to its nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

JavaScript Dates

A JavaScript date can be written as a string:
Sun Oct 16 2016 23:27:56 GMT+0530 (India Standard Time)
or as a number:
1476640676634
Dates written as numbers, specifies the number of milliseconds since January 1, 1970, 00:00:00.

Displaying Dates

Example

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Date();
</script>
```

Creating Date Objects

A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds. Date objects are created with the **new Date()** constructor. There are **4 ways** of initiating a date:

new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(), creates a new date object with the **current date and time**:

Example

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

new Date(**date string**), creates a new date object from the **specified date and time**:

Example

```
<script>
var d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
</script>
```

new Date(**number**), creates a new date object as **zero time plus the number**.
Zero time is 01 January 1970 00:00:00 UTC. The number is specified in milliseconds:

Example

```
<script>
var d = new Date(86400000);
document.getElementById("demo").innerHTML = d;
</script>
```

new Date(**7 numbers**), creates a new date object with the **specified date and time**:
The 7 numbers specify the year, month, day, hour, minute, second, and millisecond, in that order:

Example

```
<script>
var d = new Date(99,5,24,11,33,30,0);
document.getElementById("demo").innerHTML = d;
</script>
```

Example

```
<script>
var d = new Date(99,5,24);
document.getElementById("demo").innerHTML = d;
</script>
```

JavaScript counts months from 0 to 11. January is 0. December is 11.

JavaScript Date Input

There are generally 4 types of JavaScript date input formats:

Type	**Example**

ISO Date	"2015-03-25" (The International Standard)
Short Date	"03/25/2015" or "2015/03/25"
Long Date	"Mar 25 2015" or "25 Mar 2015"
Full Date	"Wednesday March 25 2015"

JavaScript Date Output

JavaScript will (by default) output dates in full text string format:
Wed Mar 25 2015 05:30:00 GMT+0530 (India Standard Time)

JavaScript Date Methods

Date Get Methods

Get methods are used for getting a part of a date. Here are the most common (alphabetically):

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

Date Set Methods

Set methods are used for setting a part of a date. Here are the most common (alphabetically):

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)

setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

JavaScript Array Object

JavaScript array is a collection of values that are belonging to a same data type. It store a list of values in single variable. Each and every element is going to be stored in the array with the unique index starting from zero. Through the indexes user can store the data or elements into the array or user can get the elements from the array. **new** keyword is used to declare an array in JavaScript. new Array(n) is used to create an array, where n was the number of slots in the array.

Creating an Array

Syntax:

var *array-name* = [*item1*, *item2*, ...];

Example

```
var cars = ["Saab", "Volvo", "BMW"];
```

Example

```
var cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

Using the JavaScript new Keyword

Syntax:

```
Array_name = new array(); //create array object with the zero size.
Or
Array_name = new array (n);  //create an array with n elements (size)
```

Example

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Access the Elements of an Array

var name = cars[0]; // accesses the value of the first element in cars

cars[0] = "Opel"; // modifies the first element in cars

Access the Full Array

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

Example of array in JavaScript

```
<html>
<head>
<script type="text/javascript">
function arrayex()
{
myarray = new Array(5);
myarray[0] = 10;
myarray[1] = 20;
myarray[2] = 30;
myarray[3] = 40;
myarray[4] = 50;
sum = 0;

for (i=0; i<myarray.length; i++)
{
sum = sum + myarray[i];
}
alert(sum)
}
</script>
</head>
<body>
<input type="button" onclick="arrayex()" value="click here">
</body>
</html>
```

Function used in Array

Function	Discription
concat()	To concats the elements of one array at the end of another array and returns an array.
sort()	To sort all elements of an array.
reverse()	To reverse elements of an array.
slice()	To extract specified number of elements starting from specified index without deleting them from array.

splice()	It will extract specified number of elements starting from specified index and deletes them from array.
push()	To push all elements in array at top.
pop()	To pop the top elements from array.

Control Structures

Control structures are used to control the flow of execution of a running program. They are divided into:

1. Conditional statements (also called conditional logic)
   - If....else
   - Switch
2. Looping control
   - Do..while
   - While
   - For
3. Branch logic
   - Break
   - Continue
   - Return

Conditional statements are used to perform different actions based on different conditions. In JavaScript the following conditional statements are available:

- 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 else if 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

The if statement is used in JavaScript to execute the code if condition is true or false. There are three forms of if statement.

- If Statement
- If else statement
- if else if statement

<u>The if Statement</u>

Simple if will execute the block of statements only if the particular condition is true. The condition is interpreted as a Boolean value.

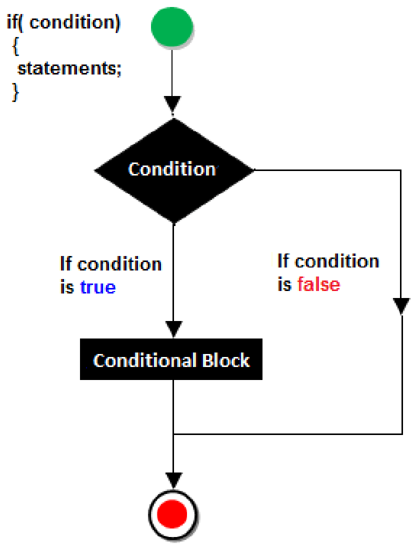
Syntax

```
        if (condition) {
            block of code to be executed if the condition is true
        }
```

Uppercase letters (If or IF) will generate a JavaScript error.

Example

```
        if (hour < 18) {
            greeting = "Good day";
        }
```

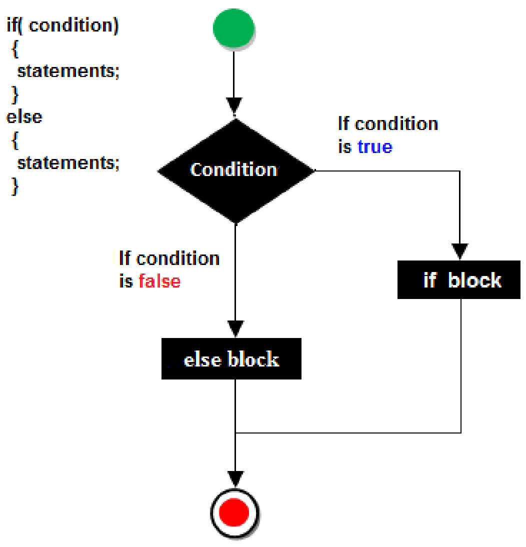


if-else statement

It can be used to execute one block of statement among two blocks. The condition is evaluated and the result is interpreted as Boolean value. If the condition is true, the true statement block is executed. If the condition is false, the false statement block is executed.

Syntax:

```
if (condition) {
  true statement block
} else {
  false statement block
}
```



```
<script>
var a=40;
if(a%2==0)
{
document.write("a is even number");
}
else{
document.write("a is odd number");
}
</script>
```

If...else if statement

It evaluates the content only if expression is true from several expressions.

```
if(expression1)
{
//content to be evaluated if expression1 is true
}
else if(expression2)
{
//content to be evaluated if expression2 is true
}
else
{
//content to be evaluated if no expression is true
}
```

Example:

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

Switch Statement

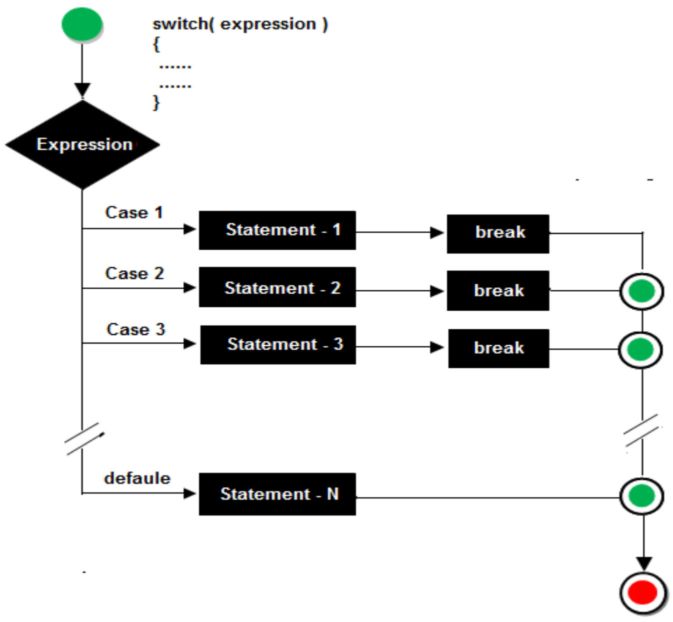
This is used to give an expression to evaluate and several different statements to execute based on the value of the expression. It is just like else if statement. The switch statement evaluates an expression. The value of the expression is then compared with the values of each case in the structure. If there is a match, the associated block of code is executed. If nothing matches, a default condition will be used. The switch statement is often used together with a break or a default keyword (or both). The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

Note:

Default code to be executed if above values are not matched

In switch statement all the cases will be evaluated if we do not use break statement.

```
switch(expression){
case value1:
statement;
break;
case value2:
statement;
break;
......
default:
statement;
}
```



```
switch(grade){
case 'A':
result="A Grade";
break;
case 'B':
result="B Grade";
break;
case 'C':
result="C Grade";
break;
default:
result="No Grade";
}
document.write(result);
```

```
<script type="text/javascript">var
dateobj=new Date()var
today=dateobj.getDay()
switch(today){
case 1:
   window.location="monday.htm";
   break
case 2:
   window.location="tuesday.htm"
   break
case 3:
   window.location="wednesday.htm"
   break
case 4:
   window.location="thursday.htm"
   break
case 5:
  window.location="friday.htm"
  break
case 6:
  window.location="saturday.htm"
  break
case 0:
  window.location="sunday.htm"
  break
}
</script>
```

JavaScript Loops

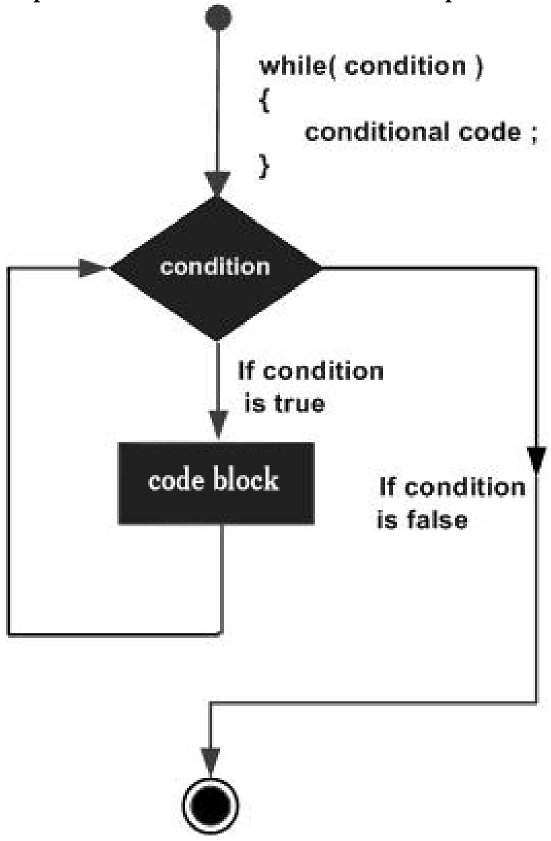
JavaScript loops are used to repeatedly run a block of code - until a certain condition is met.

Loop statements are

1. While loop
2. For loop
3. Do-while loop

While Loop

It executes a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.



Syntax

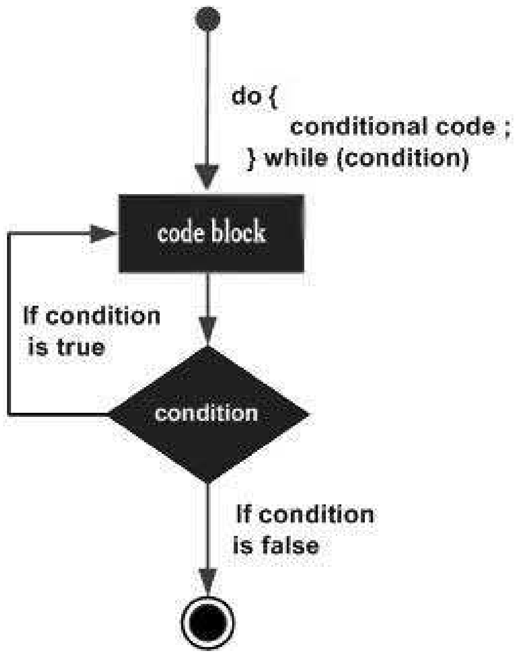
```
while (expression){
  Statement(s) to be executed if expression is true
}
```

Example

```
<script type="text/javascript">
<!--
 var count = 0;
 document.write("Starting Loop ");
 while (count < 10){
 document.write("Current Count : " + count + "<br />");
 count++; }
 document.write("Loop stopped!");
 //--> </script>
```

do...while Loop

It is similar to the while loop except that the condition check happens at the end of the loop. The loop will always be executed at least once, even if the condition is false.



Syntax

```
do{
  Statement(s) to be executed;
} while (expression);
```

Example

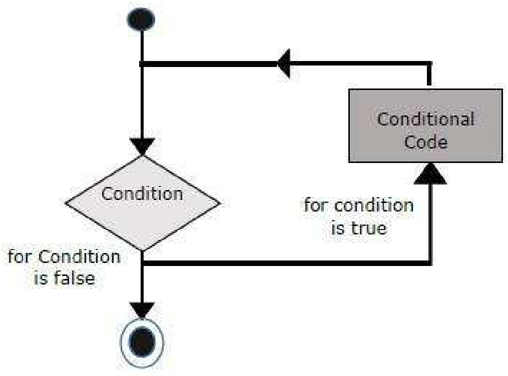
```
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop" + "<br />");
do{
document.write("Current Count : " + count +
"<br />");
count++;}while (count < 5);
document.write ("Loop stopped!");
//--> </script>
```

For Loop

'**for**' loop includes the following three parts :

- **Loop initialization:** it is used to initialize counter to starting value or give initial values to loop control variable. The initialization statement is executed before the loop begins.
- **Test statement**: it 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.
- **Iteration statement:** used to increase or decrease the counter or loop control variable.

Separate the three sections by semicolons.



Syntax

```
for (initialization; test condition; iteration
statement){
  Statement(s) to be executed if test
condition is true
}
```

Example

```
<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>
```

for...in loop

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

Syntax

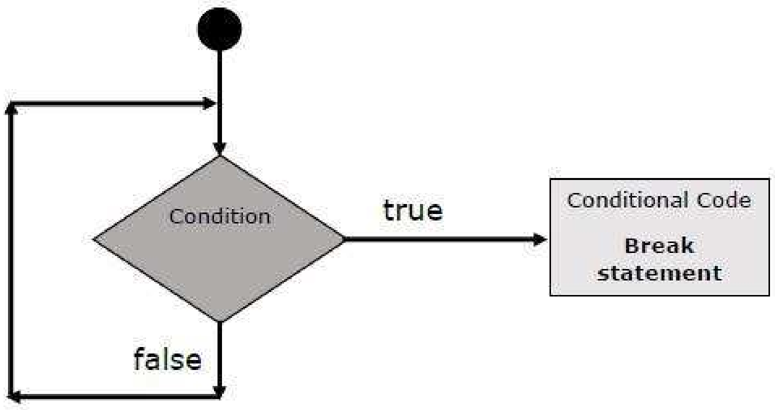
```
for (variablename in object){

  Statement or block to execute

}
```

In each iteration, one property from object is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

Example

```
<script type="text/javascript">
      <!--
        var aProperty;
        document.write("Navigator Object Properties<br /> ");
        for (aProperty in navigator) {
          document.write(aProperty);
          document.write("<br />");
        }
        document.write ("Exiting from the loop!");
      //-->
    </script>
```

JavaScript provides break and continue statements that are used to immediately come out of any loop or to start the next iteration of any loop respectively. The break statement, which was briefly introduced with the switch statement, is used to exit a loop early, breaking out of the enclosing curly braces.



```
var x = 1;
while (x < 20)
      {
        if (x == 5){
          break; // breaks out of loop completely
        }
        x = x + 1;
        document.write( x + "<br />");
      }
```

The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a **continue** statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

```
var x = 1;
while (x < 10)
        {
          x = x + 1;

          if (x == 5){
            continue; // skip rest of the loop body
          }
          document.write( x + "<br />");
        }
```

JavaScript – Functions

A function is a group of reusable code designed to perform a particular task which can be called anywhere in the program. This eliminates the need of writing the same code again and again. Functions allow a programmer to divide a big program into a number of small and manageable functions.

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.Function names can contain letters, digits, underscores, and dollar signs.

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...***)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

function ***name*** **(*parameter1, parameter2, parameter3*) {**
   code to be executed
}

A function is composed of a sequence of statements called the *function body*. Values can be *passed* to a function, and the function will *return* a value. Function **parameters** are the **names** listed in the function definition. Function arguments are the real **values** received by the function when it is invoked. Inside the function, the arguments (the parameters) behave as local variables.

Example:

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

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)

Function is invoked by its name.
Example:

```
<html>
  <head>
   <script type="text/javascript">
      function sayHello()
      {
        document.write ("Hello there!");
      }
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>

    <form>
      <input type="button" onclick="sayHello()" value="Say Hello">
    </form>
    <p>Use different text in write method and then try...</p>
  </body>
</html>
```

Function Parameters

In JavaScript, functions are first-class objects, because they can have properties and methods. A function can take multiple parameters separated by comma.

Functions are not the same as procedures. A function always returns a value, whereas a procedure might not. The parameters of a function call are the function's *arguments*. Arguments are passed to functions *by value*. If the function changes the value of an argument, this change is not reflected globally or in the calling function.

```
<html>
  <head>

    <script type="text/javascript">
      function sayHello(name, age)
      {
        document.write (name + " is " + age + " years old.");
      }
    </script>
  </head>
  <body>
   <form>
      <input type="button" onclick="sayHello('Gyan', 7)" value="Say Hello">
    </form>

    <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

The return Statement

Return statement is used to return a value from the function to the calling program. This statement should be the last statement in a function.

```
<script type="text/javascript">
      function concatenate(first, last)
      {
        var full;
        full = first + last;
        return full;
      }
        var result;
        result = concatenate('Saint', 'gits');
        document.write (result );
      }
    </script>
    </head>
    <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type="button" onclick="secondFunction()" value="Call Function">
    </form>
    <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

JavaScript - Dialog Boxes

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. JavaScript popup boxes are:

1. Alert box
2. Confirm box
3. Prompt box

Alert Dialog Box

An alert box is a JavaScript Popup Box which can be used to display a short message to the user in a separate window. It can be used to give a warning message to the users. Alert box gives only one button "OK" to select and proceed.

- method of the window object
- used to popup a simple window containing a message

Syntax:

- alert('*message*')
- window.alert('*message*') // reference to current window
- self.alert('*message*') // reference to current window
- top.alert('*message*') // top-level frameset
- parent.alert('*message*') // parent frame in a frameset

Example:

```
<html>
 <head>
    <script type="text/javascript">
      <!--
        function Warn() {
          alert ("This is a warning message!");
```

```
            document.write ("This is a warning message!");
         }
       //-->
     </script>
    </head>
  <body>
     <p>Click the following button to see the result: </p>
     <form>
       <input type="button" value="Click Me" onclick="Warn();" />
     </form>
     </body>
</html>
```

Confirmation Dialog Box

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: Ok and Cancel. They evaluate to a value based on a decision made by the user. This method returns a Boolean value. We can use confirmation boxes to ask the user a yes-or-no question, or to confirm an action. If the user clicks on the OK button, the window method confirm () will return true. If the user clicks on the Cancel button, then confirm() returns false. It's a method of the window object

Syntax:

var con= confirm(message).

Example 1:

var reply = confirm("OK to continue?")

reply is assigned a true value if the user chooses OK, and false if the user selects Cancel.

Example 2:

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display a confirm box.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
var x;
var r=confirm("Press a button!");
if (r==true)
 {
 x="You pressed OK!";
   }
  else
   {
   x="You pressed Cancel!";
   }
  document.getElementById("demo").innerHTML=x;
  }
  </script>
```

Prompt Dialog Box

The prompt () method displays a JavaScript Popup Box with a message and an input field. It is used to receive input from the user. It is similar to the confirm box, except that it returns the value of the input field, rather than true or false. It is used to receive input from the user. It takes two parameters:

(i) A label which you want to display in the text box and
(ii) A default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.

Syntax:

```
var name = prompt("Enter your name:", "anonymous")
or
var variable=prompt("sometext","defaultvalue");
```

The method returns a value of null if the user chooses Cancel. The value of the field is always a string. parseInt() method is used to convert a string into a number.

Example:

```
var number = parseInt(prompt("Enter a number:", 0))

or

var number = prompt("Enter a number:", 0)

number = parseInt(number)
```

JavaScript print function

Print function is used to display or access the print dialog box. It can be used to print the current webpage when executed.

Example:

```
<form> <input type="button" value="Print This Page" onClick="window.print()" /> </form>
```

Document Object Model

The Document Object Model (DOM) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document.

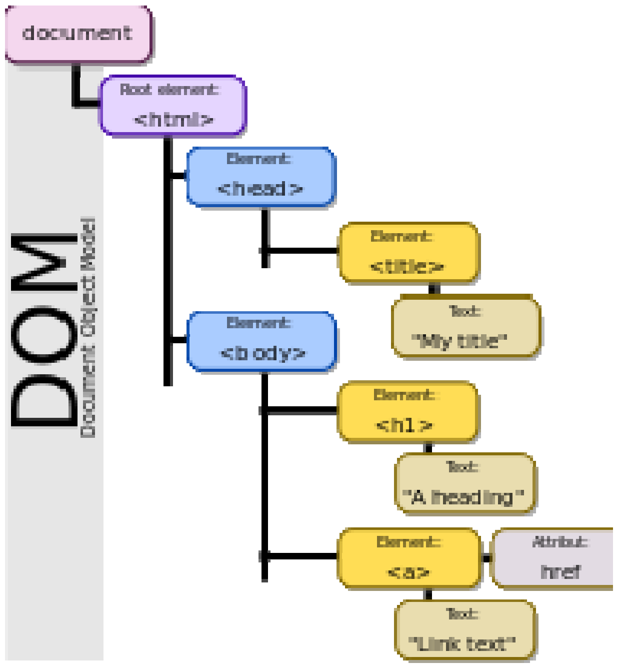
The DOM is a W3C (World Wide Web Consortium) standard, which defines a standard for accessing documents and allows programs and scripts to dynamically access and update the content, structure, and style of a document. The W3C DOM standard is separated into 3 different parts:

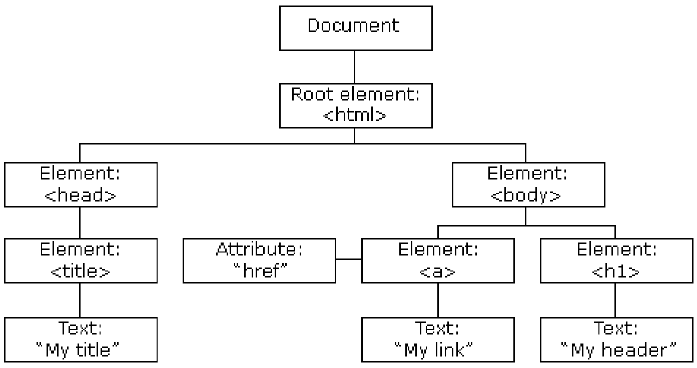
1. Core DOM - standard model for all document types
2. XML DOM - standard model for XML documents
3. HTML DOM - standard model for HTML documents

The HTML DOM is a standard object model and programming interface for HTML. It defines:

- The document itself as a document node
- The HTML elements as objects (All HTML elements are element nodes)
- The properties of all HTML elements (All HTML attributes are attribute nodes)
- The methods to access all HTML elements
- The events for all HTML elements
- Text inside HTML elements as text nodes
- Comments as comment nodes

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

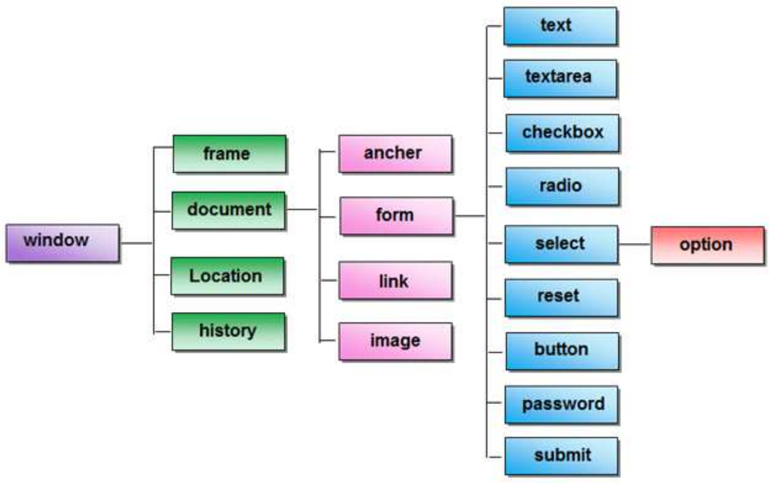




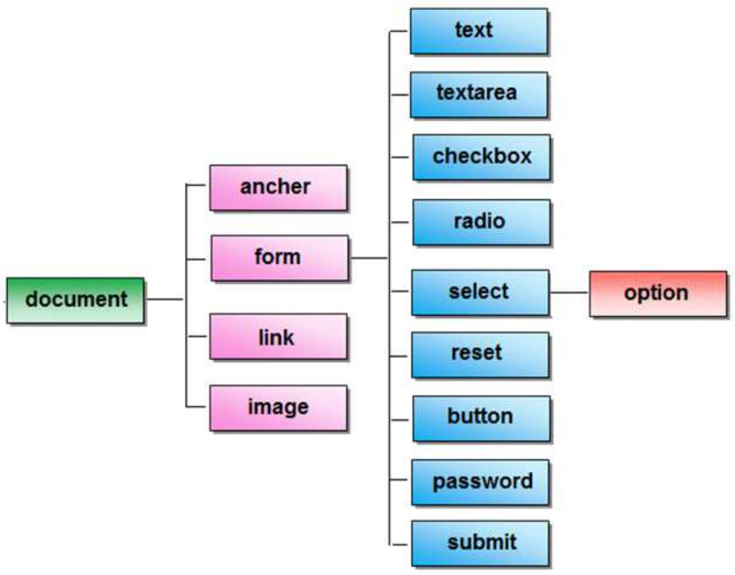
Window Object represents a window in browser. An object of window is created automatically by the browser. It is not the object of JavaScript. The JavaScript objects are string, array, date etc. It is used to display the popup dialog box such as alert dialog box, confirm dialog box, input dialog box etc. The window methods are mainly for opening and closing new windows.

Methods of window object are

Method	Description
alert()	Displays the alert box containing message with ok button.
confirm()	Displays the confirm dialog box containing message with ok and cancel button.
prompt()	Displays a dialog box to get input from the user.
open()	Opens the new window.
close()	Closes the current window.
setTimeout()	Performs action after specified time like calling function, evaluating expressions etc.



Document Object : It represents the whole html document. When html document is loaded in the browser, it becomes a document object. It is the root element that represents the html document.



The HTML DOM can be accessed with JavaScript (and with other programming languages). In the DOM, all HTML elements are defined as **objects**. The programming interface is the properties and methods of each object. A **property** is a value that a user can get or set (like changing the content of an HTML element). A **method** is an action that a user can do (like add or deleting an HTML element).

Methods of document object

Method	Description
write("string")	writes the given string on the document.
writeln("string")	Same as write(), but adds a newline character after each statement.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

Events in JavaScript

All objects have properties and methods. Events are things that happen, usually user actions, that are associated with an object The "event handler" is a command that is used to specify actions in response to an event

Event	Description
onLoad	Occurs when a page loads in a browser
onUnload	occurs just before the user exits a page
onMouseOver	occurs when you point to an object
onMouseOut	occurs when you point away from an object
onSubmit	occurs when you submit a form
onClick	occurs when an object is clicked

Events and Objects

Event	**Object**
onLoad	Body
onUnload	Body
onMouseOver	Link, Button
onMouseOut	Link, Button

onSubmit	Form
onClick	Button, Checkbox, Submit, Reset, Link

Finding HTML Elements

Method	**Description**
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

Changing HTML Elements

Method	**Description**
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

Adding and Deleting Elements

Method	**Description**
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*element*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method	**Description**
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

Examples:

document.getElementById("demo").innerHTML = "Hello World!";

var myElement = document.getElementById("intro");

var x = document.getElementsByTagName("p");

Finds the element with id="main", and then finds all <p> elements inside "main"

var x = document.getElementById("main");
var y = x.getElementsByTagName("p");

Finding HTML Elements by Class Name

var x = document.getElementsByClassName("intro");

Example:

```
<html>
<body>
 <head>
<script type="text/javascript">
 function squre() {
 var num=document.getElementById("number").value;
alert(num*num);
 }
</script> </head> <body><form>
 Enter No:<input type="text" id="number" name="number"/>
<br/>
<input type="button" value="squre" onclick="squre()"/>
 </form> </body> </html>
```

JavaScript Forms Validation

Forms validation provides the facility to validate the form on the client side so processing will be fast than server-side validation. User can validate the data such as date, name, email etc.

```
 <html> <head> <script>
 function form_validation(){
 var name=document.myform.name.value;
 if (name==null || name==""){
 alert("Name can't be blank");
 return false;
 }
}
</script> </head> <body>
<form name="myform" method="post" action="register.php"
onsubmit="return form_validation()" >
 Name: <input type="text" name="name">
<input type="submit" value="submit">
 </form> </body> </html>
```