**JavaScript (ES : 6)**

* JavaScript is the world's most popular programming language.
* JavaScript is the programming language of the Web.
* JavaScript is easy to learn.
* **JavaScript** to program the **behaviour** of web pages
* **JavaScript is single Thread.**
* **javascript was originally developed as LiveScript by Netscape in 1990’s.**
* **JavaScript was created by Bredan Eich in 1995 during his time at Netscape Communications.**
* **JavaScript is Client-side scripting language used for web applications at that time.**

**Link Java Script To Html**

* Internal script
* External script file

**Internal Script**

**<script>** //java script code **</script>**

**(ex) <html>**

**<head>** **<title>** Document **</title>** **</head>**

**<body>**

**<h1>** Cadd Cae Computers **</h1>**

**<script>**

let a = 10;

let b = 10;

console.log(a+b);

**</script>**

**</body>**

**</html>**

**External Script**

**<script src = “**<jsFilePath>**” >** **</script>**

**(ex) <html>**

**<head>** **<title>** Document **</title>** **</head>**

**<body>**

**<h1>** Cadd Cae Computers **</h1>**

**<script src = “**index.js**” ></script>**

**</body>**

**</html>**

**Output Statement**

# console: log() method

The **console.log()** method outputs a message to the web console.

**(ex)**

console.log(“hello world”);

**output :** hello world

# console: table() method

The **console.table()** method displays tabular data as a table.

**(ex)**

console.table( ["apples", "oranges", "bananas"] ) ;

**output :**

|  |  |
| --- | --- |
| **Index** | **values** |
| 0 | |  |  | | --- | --- | |  | 'apples' | |
| 1 | 'oranges' |
| 2 | 'bananas' |

# console: time() method

The console.time() method starts a timer you can use to track how long an operation takes.

* The **time()** method starts a timer in the console view.
* The **time()** method allows you to time code for testing purposes.

**(ex)**

console.time();  
for (let i = 0; i < 100000; i++) { // some code }  
console.timeEnd();

**Using the labelparameter:**

console.time("test1");

for (let i = 0; i < 100000; i++) { // some code }

console.timeEnd("test1");

**console: warn() method**

The console.warn() method outputs a warning message to the Web console.

**(ex)** console.warn(“This is warning Message “);

**ouput : This is warning Message**

# console: error() method

The console.error() method outputs an error message to the Web console.

**(ex)** console.error(“This is error Message” );

**ouput: This is error Message**

[**Variables**](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics#variables)

**Variables** are used to store information to be referenced and manipulated in a computer program.

JavaScript Variables can be declared in 4 ways:

* Automatically
* Using **var**
* Using **let**
* Using **const**

**Automatically**

**(ex)** x = 5;  
y = 6;  
z = x + y;

console.log(z);

**Using let**

Variables declared by let are only available **inside the block** where they’re defined.(2015 E6)

**(ex) let** x = 5;  
**let** y = 6;  
**let** z = x + y;

console.log(z);

**Using var**

Variables declared by var are available throughout the function in which they’re declared.(1997)

**(ex) var** x = 5;  
**var** y = 6;  
**var** z = x + y;

console.log(z);

**Using const**

Const is another keyword to declare a variable when you do not want to change the value of that variable for the whole program.(2015 E6)

**ex) const x = 10;**  
x = 20; // const variable can’t change

**output : error**

**JavaScript has 8 Data types**

1. String  
2. Number  
3. Bigint  
4. Boolean  
5. Undefined  
6. Null  
7. Symbol  
8. Object

**The Object Data types**

The object data type can contain:

1. An object  
2. An array  
3. A date

**The typeof Operator**

* You can use the JavaScript **typeof** operator to find the type of a JavaScript variable.
* The **typeof** operator returns the type of a variable or an expression:

**(ex)** let name = “cadd cae computers”;

console.log(typeof(name));

**ouput:** string

let num = 10;

console.log(typeof(num));

**output:** Number

console.log(typeof(true));

**output:** Boolean

**JavaScript Comments**

* JavaScript comments can be used to explain JavaScript code, and to make it more readable.
* JavaScript comments can also be used to prevent execution, when testing alternative code.

**Single Line Comments**

Single line comments start with //.

**(ex) var** name = “cadd cae computers”; // this is variable

**Multi-line Comments**

* + Multi-line comments start with /\* and end with \*/.
  + Any text between /\* and \*/ will be ignored by JavaScript.

**(ex) /\*** This

Is

Multiline comments **\*/**

**Type cast in Js**

1. String(<value>) 🡺 convert given value to String
2. Number(<value>) 🡺 convert given value to Number
3. Boolean(<value>) 🡺 convert given value to Boolean
4. parseInt(<value>) 🡺 convert given value to Integer or Number
5. parseFloat(<value>) 🡺 convert given value to Float
6. + <value> 🡺 convert given value to Integer or Number
7. - <value > 🡺 convert given value to Integer or Number
8. ! <value > 🡺 convert given value to Boolean value

**Using String(<value>) Function**

**(ex) let** number = 123;

**let** stringNum = String(number); // convert number to string

console.log(typeof(number));

console.log(typeof(stringNum));

**Using Number(<value>) Function**

**(ex) let** stringNum = “123”;

**let** number = Number(stringNum); // convert string to number

console.log(typeof(stringNum));

console.log(typeof(number));

**Using Boolean(<value>) Function**

**(ex) let** string = “true”;

**let** bool = Boolean(string); // convert string to boolean

console.log(typeof(string));

console.log(typeof(bool));

//another example ( 1 == true : 0 == false )

**let** number = 1;

**let** boolValue = Boolean(number); // convert num to boolean

console.log(typeof(number));

console.log(typeof(boolValue));

**Using parseInt(<value>) Function**

**(ex) let** stringNum = “123”;

**let** number = Number(stringNum); // convert string to integer

console.log(typeof(stringNum));

console.log(typeof(number));

**(you can convert float value to number value)**

**Using parseFloat(<value>) Function**

**(ex) let** stringNum = “123”;

**let** fNum = parseFloat(stringNum); // convert string to float

console.log(typeof(stringNum));

console.log(typeof(fNum));

**(you can convert number value to Float value )**

**Using +(<value>) Function**

**(ex) let** stringNum = “123”;

**let** number = +(stringNum); // convert string to number

console.log(typeof(stringNum));

console.log(typeof(number));

**Using -(<value>) Function**

**(ex) let** stringNum = “123”;

**let** number = -(stringNum); // convert string to number

console.log(typeof(stringNum));

console.log(typeof(number));

**Using !(<value>) Function ( 1 == true : 0 == false)**

**(ex) let** number = 1;

**let** bool = !(number); // convert number to boolean

console.log(typeof(number));

console.log(typeof(bool));

//another example

**let** string = “true”;

**let** boolValue = !(string); // convert string to boolean

console.log(typeof(string));

console.log(typeof(boolValue));

**Types of JavaScript Operators**

There are different types of JavaScript operators:

* Arithmetic Operators
* Assignment Operators
* Comparison Operators
* Logical Operators

**Arithmetic Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Addition | 10 + 20 = 30 |
| - | Subtraction | 20 -1 0 = 10 |
| \* | Multiplication | 10 \* 20 = 200 |
| / | Division | 20 / 10 = 2 |
| % | Modulus (Remainder) | 20 % 10 = 0 |
| ++ | Increment | **var** a=10; a++; Now a = 11 |
| -- | Decrement | **var** a=10; a--; Now a = 9 |

**Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Assign | 10 + 10 = 20 |
| += | Add and assign | var a=10; a + = 20; Now a = 30 |
| -= | Subtract and assign | var a=20; a - = 10; Now a = 10 |
| \*= | Multiply and assign | var a=10; a \* = 20; Now a = 200 |
| /= | Divide and assign | var a=10; a / = 2; Now a = 5 |
| %= | Modulus and assign | var a=10; a % = 2; Now a = 0 |

**Comparison Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Is equal to | 10==20 = false |
| === | Identical (equal and of same type) | 10==20 = false |
| != | Not equal to | 10!=20 = true |
| !== | Not Identical | 20!==20 = false |
| > | Greater than | 20>10 = true |
| >= | Greater than or equal to | 20>=10 = true |
| < | Less than | 20<10 = false |
| <= | Less than or equal to | 20<=10 = false |

**Logical Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Logical AND | (10==20 && 20==33) = false |
| || | Logical OR | (10==20 || 20==33) = false |
| ! | Logical Not | !(10==20) = true |

**Conditional Statement**

* + Ternary Statement
  + If Statement
  + Else If Statement
  + Else If ladder Statement
  + Switch Statement

**Ternary Statement**

**Syntax :**

condition ? exprIfTrue : exprIfFalse;

**(ex)** **let** num = 10;

**let** temp = (num == 10) ? true : false ;

console.log(temp); //true

console.log( (10 == 10) ? true : false)

**If Statement**

**Syntax :**

**if**(expression){

//statement

}

**(ex)** **let** number = 10 //create variable

**if** (number == 10){

console.log(true);

}

**Another method**

**if** (number == 10) console.log(true);

**Else if Statement**

**Syntax :**

**if**(expression){

//statement

}

**else**{

//statement

}

**(ex)** **let** number = 10 //create variable

**if** (number == 10){

console.log(true);

}

**else**{

console.log(false);

}

**Another method**

**if** (number == 10) console.log(true);

**else** console.log(false);

**Else if leader Statement**

**Syntax :**

**if**(expression){

//statement

}

**else if** {

//statement

}

**else if** {

//statement

}

**(ex)** **let** number = 20 //create variable

**if** (number == 10){

console.log(true);

}

**else if**(number == 20){

console.log(“from else if “);

}

**else**{

console.log(false);

}

**Another method**

if (number == 10) console.log(true);

else if(number == 20) console.log(“from else if”);

else console.log(false);

**Switch Statement**

**Syntax :**

**switch**(expression){

**case** <expression>: { //statement **break**; }

**case** <expression>: { //statement **break**; }

**case** <expression>: { //statement **break**; }

**default** : { //statement **break**; }

. . . .

}

**(ex) let** num = 1;

**switch**(num){

**case** 1 :{

console.log(“one”);

break;

}

**case** 2 :{

console.log(“two”);

break;

}

**case** 3 :{

console.log(“three”);

break;

}

**default** : {

console.log(“value is greater than 3”);

}

}

**JavaScript Loops**

Loop is used to execute the block of code several times according to the condition given in the loop.

* **for** - loops through a block of code a number of times
* **for/in** - loops through the properties of an object
* **for/of** - loops through the values of an iterable object
* **while** - loops through a block of code while a specified condition is true
* **do/while** - also loops through a block of code while a specified condition is true

**For Loop**

**Syntax :**

**for** (<initial>; <condition>; <increment of decrement>) {  
  // statement  
 }

**(ex) for**(let i = 1; i<=10; i++){

console.log(i);

}

**Another method**

**for**(let i = 1; i<=10; i++) console.log(i);

**while Loop**

**Syntax :**

**while**(<condition>){

//statement

}

**(ex) let** i = 1;

**while**( i <= 10){

console.log(i);

}

**Another method**

**while**(i<=10) console.log(i);

**Do while**

**Syntax :**

**do**{

**//**statement

}

**while**(<condition>);

**(ex) let** num = 1;

**do{**

console.log(i);

**i++;**

**}**

**while**(i<=10);

**For of Loop**

* The for...of statement executes a loop that operates on a sequence of values sourced from an [**iterable** object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols#the_iterable_protocol).
* Iterable objects include instances of built-ins such as
* **Array, String ,TypedArray ,Map, Set, NodeList, Dom Collection, Generoters**

**Syntax** : **for** ( <variable> of <iterable>){

//statement

}

**(ex) let** array = [10,20,30,40,50,60,70,80,90,100];

**for(let** element of array){

console.log(element);

}

**For in Loop**

The for...in statement iterates over all [enumerable string properties](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Enumerability_and_ownership_of_properties) of an object (ignoring properties keyed by [symbols](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol)), including inherited enumerable properties.

**Syntax : for**(<variable> in <object>){

//statement

}

**(ex) let** user = {“name” : “Surya”, “Age” : 20, “salary” : 1000 };

**for**(let property in user){

console.log(property);

}

**Break and Continue**

* The **break** statement "jumps out" of a loop.
* The **continue** statement "jumps over" one iteration in the loop.

**Math Methods**

1. **Math.round(x)** 🡺 Returns x rounded to its nearest integer
2. **Math.abs(x)** 🡺 returns the absolute (positive) value of x
3. **Math.min(x,y)** 🡺 return the smallest number of given tow numbers
4. **Math.max(x,y)**  🡺 return the biggest number of given two numbers
5. **Math.random()**  🡺 returns a random number

**String Methods**

* **length** 🡺 return the length of given string.
* **toUpperCase()** 🡺 convert given string to uppercase.
* **toLowerCase()** 🡺 convert given string to lowercase.
* **indexOf(<string>)** 🡺 return the index of given string.
* **lastIndexOf(<string>)** 🡺 return the index of the last occurrence

of string.

* **substring(<start>,<end>)** 🡺 return a part of the string between the

start and end index.

* **split(<separator>)** 🡺 spilt a string into a array of substring

based on specified separator

* **concat(<anotherString>) 🡺** concat two string
* **trim()** 🡺 remove unwanted white space in string
* **search(<searchString>) 🡺** return start index of given string
* **includes(<string>) 🡺** return Boolean if given string is find
* **repeat(<count>) 🡺** return string repeated count times of str
* **slice(<start>,<end>)** 🡺 Extract a section of string between given

Start and end

* **trimLeft() 🡺** Trim whitespace from left side of string
* **trimRight()** 🡺 Trim whitespace form right side of string
* **replace(<oldstr>,<newStr>)** 🡺 replace old string to new String
* **replaceAll(<oldstr>,<newStr>) 🡺** replace all occurrence of old string
* **matches(<regex>)**  🡺 Match a regular expression to find string
* **charAt(<index>)** 🡺 return given index character from string
* **startsWith(<string>)** 🡺 return true if the string start with given
* **endsWith(<string>)** 🡺 return true if the string end with given

string

**Template Literals**

Use back-ticks (``) rather than the quotes ("") to define a string:

**(ex) let** text = `cadd cae computers`;

**Interpolation**

Template literals provide an easy way to interpolate variables and expressions into strings.

**Syntax :** ${<variable>}

**(ex)** let firstName = "John";  
let lastName = "Doe";  
  
let text = `Welcome ${firstName}, ${lastName}!`;

**Output :** Welcome John,Doe

**Array**

The Array object, as with arrays in other programming languages, enables [storing a collection of multiple items under a single variable name](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays), and has members for [performing common array operations](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#examples).

**Syntax :** <variableName> = [];

**(ex) let** arr = [10,20,30,40,50,60,70,80,90,100];

console.log(arr[0]); // output : 10

//another method

**for**( **let** element of arr) console.log(element);

**Tow Dimensional Array**

**(ex) let** arrays = [ [10,20,30], [40,50,60], [70,80,90] ];

console.log(arr[0][0]); //access single data in array

//access hole data in array

**for**( **let** arr of arrays){

**for**( **let** element in arr) console.log(element);

}

**output :** 10 20 30 40 50 60 70 80 90

**Array of Objects**

**(ex) let** users = [

{“fname” : “person1”, “age” : 10, “salary” : 100},

{“fname” : “person2”, “age” : 10, “salary” : 100},

{“fname” : “person3”, “age” : 10, “salary” : 100}

];

**for**( **let** user of users) console.log(user);**Array Methods**

**length**  **🡺** return the count of array elements.

**concat(<anotherArray,..) 🡺** the concat() method is used to merge two

or more arrays.

**push(<element>) 🡺** add one or more elements to the end of array

**pop()** **🡺** remove the last element of array

**shift() 🡺** remove the first element of array

**unshift(<element>) 🡺** add one or more elements to the beginning of

array.

**slice(<start>,<end>) 🡺** return the sallow copy of a portion of array

**reverse()** 🡺 this method used to reverse the array(modified)

**sort() 🡺** sorts the elements as strings in alphabetical

and ascending order.

**join() 🡺** method returns an array as a string.

**join(<separators>) 🡺** concatenating all of the elements in an array

separated by specified separator string.

**includes(<element>,<index>)** 🡺 return Boolean value if element is find

**splice(<startIndex>,<endIndex>) 🡺** add or remove element from array (modified)

**indexOf(<findElement>) 🡺** return the given element index from array.

**forEach(<callBackFunction>) 🡺** The forEach() method executes a provided

function once for each array element.

**map(<callBackFunction>) 🡺** The map() method **creates a new array** populated

with the results of calling a provided function

on every element in the calling array.

**fill(<element>,<start>?,<end>?) 🡺** The fill() method changes all elements in an

array to a static value

**Destructuring Assignment**

The **Destructuring assignment** syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

**(ex) let** array = [10,20,30,40,50];

**let** [a,b,c] = array; //unpacking

console.log(a,b,c); // a = 10, b = 20, c = 30

//another example (...) 🡺 is called spread operator

**(ex)** **let** [a,b,...c] = array; // a = 10, b = 20, c = [30,40,50]

console.log(a,b,c)

//another example

**(ex)** **let** array = [ [10,20], [30,40], [50,60] ]; // nested array

**let** [ [a,b], [c,d], [e,f] ] = array; //unpacking nested array

console.log(a,b,c,d,e,f);

//unpacking objects

**(ex) let** user = {

“name” : “Surya” ,

“age” : 20 ,

“salary” : 1000

};

**let** {name,age,salary} = user; //unpacking object

console.log(name,age,salary);

**JavaScript Objects**

* JSON object literals are surrounded by curly braces **{}**.
* JSON object literals contains **key/value** pairs.
* Keys and values are separated by a **colon**.
* Keys must be strings, and values must be a valid **JSON data type**:

**Using object literals**

**(ex) let** user = { “fname” : “surya”, “sname” : “kumar”, “age” : 20 };

console.log(user.fname ,user.sname, user.age);

**Using object Constructor**

**(ex) let** user = **new Object();**

user.fname = “surya”;

user.sname = “kumar”;

user.age = 20;

console.log(user.fname, user.sname, user.age);

**Using Object.create() method**

**Syntax : Object.create(<prototype>);**

**(ex) let** name = { “fname” : “surya”, “sname” : “kumar” };

**let** user = Object.create(name);

user.job = “pro”;

user.salary = 1000;

console.log(user.fname);

console.log(user.sname);

console.log(user.job);

console.log(user.salary);

**Multiple Data Type Object**

**(ex) let** user = {

“firstName” : “Deepak”,

“secondName” : “surya”,

“age” : 20,

“job” : { “job1” : “programmer”, “job2” : “Hacker” },

“salary” : 10000,

“hobbies” : [ “football”, “volleyball”, “Book reading” ]

};

console.log(user.firstName,user.secondName,user.age,user.salary);

console.log(user.job); // return object

console.log(user.job.job1); //now return value from object

console.log(user.hobbies); //return array

console.log(user.hobbies[0]); //return given index value

**Using Class**

**(ex) class** User{

firstName = “Deepak”;

secondName = “surya;

message(){

console.log(“FullName : ” + firstName + secondName);

}

}

user = **new** User();

console.log(user.firstName);

console.log(user.secondName);

user.message();

**Iterating Object**

**for of loop, Object.values(), Object.keys(), Object.entries()**

**(ex) let** user = { “fname” : “Deepak”, “sname” : “surya”, “age” : 20 };

**for**( **let** key of user) console.log(key);

**Using Object.keys()**

**(ex) let** keys = Object.keys(user); //return all keys of object

**for**( **let** key of keys) console.log(key);

**(ex)** //another method

**for**( **let** key in Object.keys(user) ) console.log(key);

**Using Object.values()**

**(ex) let** values = Object.values(user); //return all values of object

**for**( **let** value of values) console.log(value);

**(ex)** //another method

**for**( **let** value of Object.values(user)) console.log(value);

**Using Object.entries()**

**(ex) let** pairs = Object.entries(user); //return key value pair of object

**for**( **let** pair of pairs ) console.log(pair[0] + “ ” + pair[1]);

**(ex)** //another method

**for**( **let** pair of Object.entries(user))

console.log(pair[0] + “ ” + pair[1]);

**Function in Java Script**

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

**Syntax : function** <functionName> () {

//code

}

**(ex)** //function without argument and no return type

**function** message(){ console.log(“welcome to Js”); }

**(ex)** //function with argument and no return type

**function** message(name){ console.log(“welcome” + name); }

**(ex)** //function with argument and return type

**function** getName(name){ **return** name; }

**(ex)** //function with return type and with arguments

**function** add(num1, num2){ **return** num1 + num2 };

console.log(add(100,100));

**Function with Arbitrary Arguments**

In java script a function can accept an arbitrary number of arguments using the **arguments** keyword.

**(ex) function** add(){

let sum = 0;

**for**( **let** i=0; i<arguments.length; i++){

sum += arguments[i];

}

**return** sum;

}

console.log(sum(100,100,100,100,100)); //output : 500

**Function as Expression**

* Js a Function can also be define as an expression.
* This means that a function can be assigned to a variable or passed as an argument to another function.

**(ex) const** add = **function**(num1, num2){ return num1 + num2; }

console.log(add(100,100); //output : 200

**Pass function as argument (or) Call Back Function**

* A callback is a function passed as an argument to another function

This technique allows a function to call another function

* A callback function can run after another function has finished

**(ex) function** inner(){ console.log(“inner Function called ”); }

**function** outer(inner){ inner(); } //calling given function

outer(inner); //pass inner function address to outer function

**output :** inner Function called

# Anonymous Functions or Arrow Function

# The meaning of the word 'anonymous' defines something that is unknown or has no identity.

* Passing an anonymous function to other function as its argument.
* We can also use an anonymous function as an argument for another function. To understand better, let's implement a code under which we will pass the anonymous function as an argument value for another function:

# (ex) const add = (num1,num2) => { return num1 + num2; }

# console.log(add(100,100);

# (ex) //another example

# setInterval(() => { console.log(“running”); },1000);

# JS Bom (Browser Object Model)

# The Browser Object Model (BOM) is used to Interact with the Browser.

# 

# Window Object

# The window object represents a window in browser. An object of window is created automatically by the browser.

# Properties

# [document](https://www.w3schools.com/jsref/prop_win_document.asp) 🡺 Returns the Document object for the window.

# history 🡺 Returns the History object for the window.

# location 🡺 Returns the Location object for the window.

# navigator 🡺 Returns the Navigator object for the window.

# screen 🡺 Returns the Screen object for the window.

# Methods

# alert(<message>) 🡺 display the alert box containing given

# message with ok button.

# confirm(<message>) 🡺 display the alert box containing given

# message with ok and cancel button.

# print() 🡺 Print the content of the current window.

# stop() 🡺 method stops this document from loading.

# close() 🡺 close the created window.

# prompt(<message>,<defaultValue>?) 🡺 Displays a dialog box that prompts the visitor for input Return Type : string

# Settimeout() Method

# The setTimeout() method calls a function after a number of milliseconds.

# syntax :

# setTimeout(<callBackFunction>,<milliSeconds>?,<param1>?,<parm2>?,...)

# (ex) setTimeout(message,2000,”Deepak”,”surya”);

# function message(firstName, secondName){

# console.log( “ Hi ” + firstName + “ ” + secondName );

# }

# SetInterval() method

# The setInterval() method calls a function at specified intervals

# (in milliseconds).

# syntax :

# setTimeout(<callBackFunction>,<milliSeconds>?,<param1>?,<parm2>?,...)

# (ex) setInterval(errMessage,1000);

# function errMessage(){ console.log(“running . . . . .”); }

# ClearInterval() method

# The clearTimeout() method clears a timer set with the setTimeout()and setInterval() method.

# syntax : clearTimeout(<object\_of\_setInterval\_or\_setTimeout>);

# (ex) let runningCode = setInterval(message,1000);

# var i = 1;

# function message(){

# console.error(`count : ${i}`);

# if(i==10) clearTimeout(runningCode);

# i++;

# }

# Window Location

* The location object contains information about the current URL.
* The location object is a property of the window object.
* The location object is accessed with:
* **window.location** or just **location**

# Properties

**host 🡺** returns the hostname and port number of a url

**hostname 🡺** return the hostname of the url

**origin 🡺** return the protocol, hostname and port number of url

**href 🡺** return the entire url

**port 🡺** return port number of url

**protocol 🡺**  return the protocol of url

**search 🡺** return the query string of url

**pathname 🡺** return the pathname of url

# Methods

# reload() 🡺 reload the current document.

# Window document (DOM)

## The HTML DOM (Document 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

**DOM TREE**

# A DOM tree for an example web page

# DOM Properties

# activeElement 🡺 returns the currently focused element in the docs.

# URL 🡺 returns the full url of the HTML document.

# baseURL 🡺 returns the absolute url of the HTML document.

# head 🡺 returns the <head> element in the HTML document.

# title 🡺 sets or return the title of the document.

# body 🡺 return the document’s body (<body> element).

# cookie 🡺 returns the all name/value pairs of cookies in the

# document.

# domain 🡺 returns the domain name of the server that loaded

# the document

# forms 🡺 return a collection of all <form> element in the

# document.

# images 🡺 returns a collection of all <img> element in the

# document.

# links 🡺 returns a collection of all <a> element in the

# document.

# doctype 🡺 returns the document type declaration associated

# with the document.

# readyState 🡺 returns the loading status of document.

# referrer 🡺 returns the url of the document that loaded the

# current document.

# scripts 🡺 returns a collection of all <script> element in docs.

# children 🡺 Returns the number of child elements of the current

# document.

# childElementCount 🡺 Returns the number of child elements of the current

# document.

# currentScript 🡺 Returns the [<script>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script) element whose script is currently

# being processed

# DOM Methods

# write(<string>) 🡺 The document.write() method writes a string of text to

# a document stream opened by [document.open()](https://developer.mozilla.org/en-US/docs/Web/API/Document/open).

# Deprecated

# Note: Because document.write() writes to the document stream, calling document.write() on a closed (loaded) document automatically calls document.open()

# writeln(<string>) 🡺 document.writeln is the same as [document.write](https://developer.mozilla.org/en-US/docs/Web/API/Document/write) but

# adds a newline. Note : Deprecated.

# Finding HTML Elements

* Finding HTML elements by **id**
* Finding HTML elements by **tag name**
* Finding HTML elements by **class name**
* Finding HTML elements by **CSS selectors**

**Finding HTML Element By Id**

* The easiest way to find an HTML element in the DOM, is by using the element id use **document.getElementById(). Note : return** HTML element of document

**Syntax : document.getElementById( <id\_name> );**

**(ex) <html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1 id = “**heading” **>** Welcome to Java Script **</h1>**

**<script>**

**let** element = **document.getElementById(**“heading”**);**

console.log(element);

**</script>**

**</body>**

**</html>**

# output: <h1 id = “heading”>

## Finding HTML Elements by Class Name

* If you want to find all HTML elements with the same class name, use **getElementsByClassName().**
* The [Element](https://developer.mozilla.org/en-US/docs/Web/API/Element) method **getElementsByClassName()** **returns** a live [HTMLCollection](https://developer.mozilla.org/en-US/docs/Web/API/HTMLCollection) which contains every **descendant** element which has the specified class name or names.

**Syntax : document.getElementsByClassName(<class\_name>);**

**(ex) <html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1 class = “**heading” **>** Welcome to Java Script **</h1>**

**<p class = “**heading” **>**

**<script>**

**let** collection = **document.getElementsByClassName(**“heading”**);**

console.log(collection);

**</script>**

**</body>**

**</html>**

**output :** **HTMLCollection** { 0: **h1.heading**, 1: **p.heading**, length: **2** }

* 0: <h1 class="heading">​
* 1: <p class="heading">

length: 2

​

## Finding HTML Elements by Tag Name

# The getElementsByTagName method of [Document](https://developer.mozilla.org/en-US/docs/Web/API/Document) interface returns an [HTMLCollection](https://developer.mozilla.org/en-US/docs/Web/API/HTMLCollection) of elements with the given tag name.

* **Return :** A live [**HTMLCollection**](https://developer.mozilla.org/en-US/docs/Web/API/HTMLCollection) of found elements in the order they appear in the tree.

**Syntax : document.getElementsByTagName(<html\_tag\_name>);**

**(ex) <html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1>** Heading 1 **</h1>**

**<h1>** Heading 2 **</h1>**

**<script>**

**let** collection = **document.getElementsByTagName(**“h1”**);**

console.log(collection);

**</script>**

**</body>**

**</html>**

**output :** **HTMLCollection** { 0: **h1**, 1: **h1**, length: **2** }

* 0: <h1>​
* 1: <h1>

length: 2

## Finding HTML elements by CSS selectors

* The [Document](https://developer.mozilla.org/en-US/docs/Web/API/Document) method **querySelector()** returns the first [Element](https://developer.mozilla.org/en-US/docs/Web/API/Element) within the document that matches the specified selector, or group of **selectors**. If no matches are found, null is returned.

**Method 1 :** **querySelector()**

**Syntax : document.querySelector(<css\_selectors>);**

**(ex) <html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1 id = “**heading1” **>** Heading 1 **</h1>**

**<h1 class = “**heading2” **>** Heading 2 **</h1>**

**<script>**

**let** usingTagName = **document.querySelector** (“h1”**);**

**let** usingId = **document.querySelector**(**“**#heading1”);

**let** usingClass = **document.querySelector**(**“.**heading2”);

console.log(“Using TagName : ”,usingTagName);

console.log(“Using Id : ”,usingId);

console.log(“Using Class : ”,usingClass);

**</script>**

**</body>**

**</html>**

**output :** Using Tag Name : **<h1 id="heading1">**

Using Id : **<h1 id="heading1">**

Using Class :  **<h1 class="heading2">**

**Method 2 :** **querySelectorAll()**

# The [Document](https://developer.mozilla.org/en-US/docs/Web/API/Document) method querySelectorAll() returns a static (not live) [NodeList](https://developer.mozilla.org/en-US/docs/Web/API/NodeList) representing a list of the document's elements that match the specified group of selectors.

**Syntax : document.querySelectorAll(<css\_selectors>);**

**(ex) <html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1 id = “**heading1” **>** Heading 1 **</h1>**

**<h1 id = “**heading1” **>** Heading 2 **</h1>**

**<h2 class = “**heading2” **>** Heading 3 **</h2>**

**<h2 class = “**heading2” **>** Heading 4 **</h2>**

**<script>**

**let** usingTagName = **document.querySelectorAll**(“h1”**);**

**let** usingId = **document.querySelectorAll**(**“**#heading1”);

**let** usingClass = **document.querySelector**(**“.**heading2”);

console.log(“Using TagName : ”,usingTagName);

console.log(“Using Id : ”,usingId);

console.log(“Using Class : ”,usingClass);

**</script>**

**</body>**

**</html>**

**output :** Using Tag Name : **NodeList [ h1#heading1, h1#heading1 ]**

Using Id : **NodeList [ h1#heading1, h1#heading1 ]**

Using Class :  **NodeList [ h2.heading2, h2.heading2 ]**

# Changing HTML Elements Content

# The easiest way to modify the *content* of an *HTML element* is by using the Properties.

# Property

# innerHTML 🡺 The [Element](https://developer.mozilla.org/en-US/docs/Web/API/Element) property innerHTML gets or sets the HTML or XML markup contained

# within the element.

# innerText 🡺 The innerText property of the [HTMLElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement) interface represents the rendered text content of a node and its descendants.

# warning : Setting innerText on a node removes *all* of the node's children and replaces them with a single textnode with the given string value.

# textContent 🡺 The textContent property of the [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node) interface represents the text content of

# the node and its descendants.

# warning : Setting textContent on a node removes *all* of the node's children and replaces them with a single textnode with the given string value.

# Using innerHTML Property (get innerHTML)

# (EX : 1) <html>

**<head> <title>** Document **</title> </head>**

**<body>**

**<div id = “**container**”>**

Sample Text 1

**<h1>** Sample Text 2 **</h2>**

**<h2>** Sample Text 3 **</h2>**

**</div>**

**<script>**

**let** container = **document.getElementById (**“container”**);**

console.log(element.**innerHTML**); //get innerHTML

**</script>**

**</body>**

**</html>**

**output :** Sample Text 1 //output in browser console window.

<h1> Sample Text 2 </h1>

<h2> Sample Text 3 </h2>

# (EX : 2) Set content

**<html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<div id = “**container**”>** Sample Text 1 **</div>**

**<h1 id = “**heading**”>** Sample Text 2 **</h1>**

**<script>**

**let** container = **document.getElementById (**“container”**);**

**let** headingTag = **document.getElementById(**“heading”**);**

container.**innerHTML** = “<h1> computer </h1>”; //set value

headingTag.**innerHTML** = “welcome to js”;

console.log(“Container Text : ”,container.**innerHTML)**;

console.log(“HeadingTag Text : ”,headingTag.**innerHTML**);

**</script>**

**</body>**

**</html>**

**output :** Container Text : <h1> computer </h1> //output in browser console

HeadingTag Text : welcome to js

# Using innerText Property (get innerHTML)

**(ex)** //write this code inside of the html element

**<div id = “**container**”>**

Sample Text 1

**<h1>** Sample Text 2 **</h2>**

**<h2>** Sample Text 3 **</h2>**

**</div>**

//write inside of the body element

**<script>**

**let** container = **document.getElementById (**“container”**);**

console.log(container.**innerText**);

**<script>**

**output :** Sample Text 1 //output in browser console

Sample Text 2

Sample Text 3

# (Ex : 2) Set content

//write this code inside of the html element

**<div id = “**container**”>**

Sample Text 1

**<h1>** Sample Text 2 **</h2>**

**<h2>** Sample Text 3 **</h2>**

**</div>**

//write inside of the body element

**<script>**

**let** container = **document.getElementById (**“container”**);**

container.**innerText = “**<h1> computer </h1>**”**d; **//replace all child elements**

//assign string

//container.**innerText** = “welcome to js”;

**<script>**

**output :** <h1> computer </h1>

# Using textContent Property

**(Ex)** //write this code inside of the html element

**<div id = “**container**”>**

Sample Text 1

**<h1>** Sample Text 2 **</h2>**

**<h2>** Sample Text 3 **</h2>**

**</div>**

//write inside of the body element

**<script>**

**let** container = **document.getElementById (**“container”**);**

console.log(container.**textContent**);

**<script>**

**output :** Sample Text 1 //output in browser console

Sample Text 2

Sample Text 3

**(Ex) Set content**

//write inside of the body element

**<script>**

**let** container = **document.getElementById (**“container”**);**

container.**textContent = “**<h1> computer </h1>**”**d; **//replace all child elements**

//assign string

//container.**textContent** = “welcome to js”;

**<script>**

**output :** <h1> computer </h1>

# Dom Traversal

# Access Element Parent

# <element>.parentNode 🡺 returns the parent node of an element.

# <element>.parentElement 🡺 returns the parent Element of an element.

# (ex) //Access Element Parent

# <html>

# <head> <title> Document </title> </head>

# <body>

# <div> <h1 id = “child”> This is Child Element </h1> </div>

# 

# <script>

# 

# let element = document.getElementById(“child”);

# let parentNode = element.parentNode;

# let parentElement = element.parentElement;

# console.log(“Using ParentNode : ”,parentNode);

# console.log(“Using ParentElement : ”,parentElement);

# </script>

# </body>

# </html>

# (ex) // Access Element Parent of Parent

# 

# // write inside of script tag

# 

# let element = document.getElementById(“child”);

# let parentElement = element.parentElement;

# let grandParent = parentElement.parentElement;

# console.log(“Parent : ”,parentNode);

# console.log(“Grand Parent: ”,grandParent);

# Access Child Element

# <element>.children 🡺 returns a collection of all child elements

# of an element Note : excluding text and comment nodes.

# <element>.childNodes 🡺 returns a collection of all child nodes of an

# element Note : including text and comment nodes.

# (ex) //Access Child Element

# <html>

# <head> <title> Document </title> </head>

# <body>

# <div id = “container” >

# <h1> Child 1 </h1>

# <h2> Child 2 </h2>

# <h3> Child 3 </h3>

# </div>

# 

# <script>

# 

# let parent = document.getElementById(“container”);

# let childrens = parent.chidren;

# let childNodes = parent.childNodes;

# console.log(“Using Children : ”,childrens);

# cosole.log(“Using ChildNodes : ”,childNodes);

# </script>

# </body>

# </html>

**output :** Using Children : **HTMLCollection { 0: h1, 1: h2, length: 2 }**

* 0: <h1>​
* 1: <h2>​
* 2: <h3>

length: 2

Using ChildNodes : **NodeList(7) [#text, h1, #text, h2, #text, h3,#text]**

​

* 0: #text " \n\n\t\t"​
* 1: <h1>​
* 2: #text "\n\t\t"​
* 3: <h2>​
* 4: #text "\n\t\t"​
* 5: <h3>​
* 6: #text "\n\t\n\t"

length: 7

# Note : you can access childElements or childNodes using index value

# (ex) //write inside of html tag

# 

# <div id = “container” >

# <h1> Child 1 </h1>

# <h2> Child 2 </h2>

# <h3> Child 3 </h3>

# </div>

# 

# //write inside of body tag

# <script>

# 

# let parent = document.getElementById(“container”);

# let childrens = parent.chidren;

# let childNodes = parent.childNodes;

# console.log(childrens[0]);

# cosole.log(childNodes[0]);

# </script>

**output :** Using ChildNodes : **<h1>**

Using Children : **#text " \n\n\t\t"**

**<element>.childElementCount 🡺** return child element count of element

**Note : excluding text and comment nodes.**

# (ex) //write inside of html tag

# 

# <div id = “container” >

# <h1> Child 1 </h1>

# <h2> Child 2 </h2>

# <h3> Child 3 </h3>

# </div>

# 

# //write inside of body tag

# <script>

# 

# let parent = document.getElementById(“container”);

# console.log(parent.childElementCount);

# </script>

**output :** 3

**<element>.firstChild 🡺** return first Child Node of element.

**<element>.lastChild 🡺** return last child Node of element.

**<element>.firstElementChild 🡺** return the first child element of element.

**<element>.lastElementChild 🡺** return the last child element of element.

# (ex) //write inside of html tag

# 

# <div id = “container” >

# <h1> Child 1 </h1>

# <h2> Child 2 </h2>

# <h2> Child 3 </h2>

# </div>

# 

# //write inside of body tag

# <script>

# 

# let parent = document.getElementById(“container”);

# console.log(“FirstNodeChild : ”,parent.firstChild);

# console.log(“LastNodeChild : ”,parent.lastChild);

# console.log(“FirstElementChild : ”,parent.firstElementChild);

# console.log(“LastElementChild : ”,parent.lastElementChild);

# </script>

**output :** FirstNodeChild : **#text " \n\n\t\t"**

LastNodeChild **: #text " \n\n\t\t"**

# FirstElementChild : <h1>

# LastElementChild : <h3>

**<element>.previousSibling 🡺** return previous sibling Node

**<element>.nextSibling 🡺** return next sibling Node

**<element>.previousElementSibling 🡺** return previous sibling Element

**<element>.nextElementSibling 🡺** return next sibling Element

# (ex) //write inside of html tag

# 

# <div>

# <h1> Child 1 </h1>

# <h2 id = “child2” > Child 2 </h2>

# <h3> Child 3 </h3>

# </div>

# 

# //write inside of body tag

# <script>

# 

# let element = document.getElementById(“child2”);

# console.log(“PreviousNode: ”,element.previousSibling);

# console.log(“NextNode : ”,element.nextSibling);

# console.log(“PreviousElement : ”,element.previousElementSibling);

# console.log(“NextElement : ”,element.nextElementSibling);

# </script

**output :** PreviousNode : **#text "\n\t\t**

NextNode : **#text "\n\t\t**

PreviousElement : **<h1>**

NextElement : **<h3>**

**<element>.hasChildNodes() 🡺**  returns **(Boolean value )** true if

specified node has any child nodes.

**<element>.nodeName 🡺** returns the name of the Node.

# (ex) //write inside of html tag

# 

# <div id = “container” >

# <!— This is Text comments -->

# <h1> Child 1 </h1>

# <h2 id = “child2” > Child 2 </h2>

# <h3> Child 3 </h3>

# </div>

# 

# //write inside of body tag

# <script>

# 

# let parent = document.getElementById(“container”);

# console.log(“Parent Has ChildNodes : ”, parent.hasChildNodes()); console.log(“Node Type : ”,parent.nodeType); console.log(“Node Type : ”,parent.childNodes[0].nodeType);

# </script

**output :** Parent Has ChildNodes : **true**

Node Name : **DIV**

# Creating and Modifying Elements

**Methods**

**document.createElement(<tagName>) 🡺** method creates the HTML element

specified by *tagName*, or an

[**HTMLUnknownElement**](https://developer.mozilla.org/en-US/docs/Web/API/HTMLUnknownElement) if *tagName*

isn't recognized.

# (ex) <html>

# <head> <title> Document </title> </head>

# <body>

# 

# <div id = “container”> </div>

# 

# 

# <script>

# let parent = document.getElementById(“container”);

# 

# let newElement = document.createElement(“h1”);

# parent.appendChild(“newElement”);

# 

# </script>

# </body>

# </html>

**document.createTextNode(<stringText>) 🡺** createTextNode() creates a new

Text Node and returns it.

# (ex) <script>

# let parent = document.getElementById(“container”);

# 

# let comment = document.creatTextNode(“This is comment”);

# parent.append(comment);

# 

# </script>

**document.createAttribute(<attributeName>) 🡺** create new Attribute

# (ex) <html>

# <head> <title> Document </title> </head>

# <body>

# 

# <div id = “container”> </div>

# 

# 

# <script>

# let parent = document.getElementById(“container”);

# 

# let att = document.createAttribute(“id”);

# parent.setAttributeNode(att);

# 

# </script>

# </body>

# </html>

**document.append(<element>,...) 🡺** The **Document.append()** method inserts a

set of [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node) objects or string objects

after the **last child of the document**.

String objects are inserted as equivalent

[**Text**](https://developer.mozilla.org/en-US/docs/Web/API/Text) **nodes.**

# (ex) //create new HTML Element and add the element at end of body tag

# <html>

# <head> <title> Document </title> </head>

# <body>

# 

# <script>

# 

# let newElement = document.createElement(“h1”);

# newElement.innerHTML = “ Heading ”;

# document.body.append(newElement);

# </script>

# </body>

# </html>

**document.prepend(<element>,...) 🡺** The **Document.prepend()** method inserts a set of [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node) objects or string objects before the first child of the document. String objects are inserted as equivalent [Text](https://developer.mozilla.org/en-US/docs/Web/API/Text) nodes.

# (ex) //create new HTML Element and add the element at being of body tag

// write inside of the body tag

# <script>

# let newElement = document.createElement(“h1”);

# newElement.innerHTML = “ Heading ”;

# document.body.prepend(newElement);

# </script>

**before(<element>,...) 🡺** The **Element.before()** method inserts a set of [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node) or string objects in the children list of this Element's parent, just before this Element. String objects are inserted as equivalent [Text](https://developer.mozilla.org/en-US/docs/Web/API/Text) nodes.

# (ex) <html>

# <head> <title> Document </title> </head>

# <body>

# 

# <div>

# 

# <h1> child 1 </h1>

# <h2 id = “child2” > child 2 </h2>

# <h3> child 3 </h3>

# </div>

# 

# 

# <script>

# let element = document.getElementById(“child2”);

# 

# let newElement = document.createElement(“h4”);

# newElement.innerHTML = “ child 4 ”;

# element.before(newElement);

# </script>

# </body>

# </html>

**after(<element>,...) 🡺** The **Element.after()** method inserts a set of [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node) or string objects in the children list of the Element's parent, just after the Element. String objects are inserted as equivalent [Text](https://developer.mozilla.org/en-US/docs/Web/API/Text) nodes.

# (ex) <script>

# let element = document.getElementById(“child2”);

# 

# let newElement = document.createElement(“h4”);

# newElement.innerHTML = “ child 4 ”;

# element.after(newElement);

# </script>

**appendChild(<childElement>) 🡺** The **appendChild()** method of the [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node) interface adds a node to the end of the list of children of a specified parent node.

# (ex) <html>

# <head> <title> Document </title> </head>

# <body>

# 

# <div id = “parent”> </div>

# <script>

# let parent = document.getElementById(“parent”);

# 

# let child1 = document.createElement(“h1”);

# let child2 = document.createElement(“h2”);

# let child3 = document.createElement(“h3”);

# child1.innerHTML = “ Heading 1 ”;

# child2.innerHTML = “ Heading 2 ”;

# child3.innerHTML = “ Heading 3 ”;

# parent.appendChild(child1);

# 

# parent.appendChild(child2);

# parent.appendChild(child3);

# 

# </script>

# </body>

# </html>

# Remove Element From DOM

**<element>.remove() 🡺** The **Element.remove()** method removes the element

fromthe DOM.

**removeChild(<element>) 🡺** The **removeChild()** method of the [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node) interface

removes a child node from the DOM and returns the

removed node.

**Element : Remove() Method**

**(ex)** //write inside of html

**<div id = “**container**”>**

**<h1>** Child 1 **</h1>**

**<h2>** Child 2 **</h2>**

**</div>**

//write inside of script

**<script>**

**let** element = **document.getElementById(“**container**”);**

element.**remove();**

**</script>**

**RemoveChild() Method**

**(ex)** //write inside of html

**<div id = “**container**”>**

**<h1>** Child 1 **</h1>**

**<h2>** Child 2 **</h2>**

**</div>**

//write inside of script

**<script>**

**let** parent = **document.getElementById(“**container**”);**

**let** child1 = parent**.children[0];**

parent**.removeChild(**child1**);**

**</script>**

# Replace Element From DOM

**<element>.replaceWith(<element>) 🡺** method replaces this Element in the

children list of its parent with a set

of [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node) or string objects. String

objects are inserted as equivalent

[Text](https://developer.mozilla.org/en-US/docs/Web/API/Text) nodes.

**(ex)** //write inside of html

**<div id = “**container**”>**

**<h1>** Child 1 **</h1>**

**<h2>** Child 2 **</h2>**

**</div>**

//write inside of script

**<script>**

**let** parent = **document.getElementById(“**container**”);**

**let** newElement = **document.createElement(“**h6**”);**

newElement.**innerHTML = “**New Element**”;**

parent**.removeChild(**newElement**);**

**</script>**

**<parent>.replaceChild(<new\_node>,<old\_node>) 🡺** method of the [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node)

interface replaces a child

node within the given

**(parent)** node.

**(ex)** //write inside of script

**<script>**

**let** parent = **document.getElementById(“**container**”);**

**let** newElement = **document.createElement(“**h6**”);**

newElement.**innerHTML = “**New Element**”;**

parent**.removeChild(**newElement**);**

**</script>**

**<parent>.replaceChildren(<newElement>,...) 🡺** The method replaces the

existing children of a [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node)

with a **specified new set**

**of children**. These can be

string or [Node](https://developer.mozilla.org/en-US/docs/Web/API/Node) objects.

**(ex)** //write inside of html

**<div id = “**container**”>**

**<h1>** Child 1 **</h1>**

**<h2>** Child 2 **</h2>**

**</div>**

//write inside of script

**<script>**

**let** parent = **document.getElementById(“**container**”);**

**let** newElement = **document.createElement(“**h6**”);**

newElement.**innerHTML = “**New Element**”;**

parent**.replaceChildren(**newElement**);**

**</script>**

# Attributes and Properties

**<element>.getAttribute(<attributeName>) 🡺** returns the attribute value.

**(ex) <h1 id = “**heading” **> Heading 1 </h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

console.log(element**.getAttribute(“**id**”)**);

**</script>**

**<element>.setAttribute(<attribute>,<value>) 🡺** set the new attribute & value.

**(ex) <h1> Heading 1 </h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

element**.setAttribute(**“id”,”heading”**);**

console.log(element**.getAttribute(“**id**”)**);

**</script>**

**<element>.removeAttribute(<attributeName>) 🡺** remove attribute.

**(ex) <h1 id = “**heading**” > Heading 1 </h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

element**.removeAttribute(**“id”**);**

console.log(element**.getAttribute(“**id**”)**);

**</script>**

**<element>.hasAttribute(<attributeName>) 🡺** return Boolean value.

**(ex) <h1 id = “**heading**” > Heading 1 </h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

console.log(element**.hasAttribute(**“id”**)**)**;**

**</script>**

**<element>.attributes 🡺** return all attributes of element.

**(ex) <h1 id = “**idName**” class = “**className” **> Heading 1 </h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

console.log(element**.attributes**)**;**

//access attributes with index value

console.log(element.**attributes[0]**);

**</script>**

**Access Class and Id Attributes**

**(ex) <h1 id = “**idName**” class = “**className” **> Heading 1 </h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

console.log(element.**className**); // returns className

console.log(element.**classList**); // returns class List

console.log(element.**id**); // return id Name

//set id and class name using property

//element**.id** = “newIdName”;

//element**.className** = “newClassName”;

//element**.id** += “anotherId”;

//element**.className** += “anotherClassName”;

**</script>**

**Class List Methods**

**<element>.classList.add(<className>) 🡺** Add new class Name end

of class List.

**<element>.classList.remove(<className>) 🡺** Remove specific class

in classList.

**<element>.classList.contains(<className>) 🡺** return Boolean value.

**<element>.classList.toggle(<className>) 🡺** if name is present in class

list the toggle method

**remove** that className.

if name is not present in class list the toggle

method **add** given class name

end of the class list.

**<element>.classList.replace(<oldName>,<newName>) 🡺** replace className.

**Examples**

**(ex) <h1 id = “**idName**” class = “**className”**> Heading 1 </h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

//element**.classList.add(“**newClassName**”);**

//element**.classList.remove(“**newClassName”**);**

//element**.classList.contains(**“newClassName”**);**

//element**.classList.replace(**“newClassName”,”class”**);**

//element**.classList.toggle(“**class**”);**

**</script>**

# Styling Elements

**Syntax : <element>.style.<property> = <value> ;**

**(ex) <h1 id = “**heading**”>** Heading **</h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

element**.style.border** = “2px solid red”;

**</script>**

**(ex) <html>**

**<head> <title>** Document **</title> </head>**

**<style>**

.newStyle{

height : 100px;

width : 100px;

background-color : yellow;

}

**</style>**

**<body>**

**<div id = “**container**”> </div>**

**<script>**

**let** element = **document.getElementById(“**container**”);**

element**.classList.add(“**newStyle**”);**

**</script>**

**<body>**

**</html>**

# JavaScript Events

* The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser.
* When [javascript](https://www.javatpoint.com/javascript-tutorial) code is included in [HTML](https://www.javatpoint.com/html-tutorial), js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**.
* Thus, js handles the HTML events via **Event Handlers**.

**Methods**

**Syntax : <element>.addEventListener(<event>,<callBackFunction>)**

* The **addEventListener()** method of the [**EventTarget**](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget) interface sets up a function that will be called whenever the specified event is delivered to the target.
* **Note :** This Method gives **current executed event.** you can receive that event as a parameter in your call Back Function.

**(ex)** //write inside of HTML

**<h1 id = “”>** Heading **</h1>**

**<script>**

**let** element = **document.getElementById(**“heading”**);**

element**.addEventListener(“**click”,myFunction**);**

**function** myFunction(event){

**alert(“** You Clicked This H1 Tag . . . .**”);**

console.log(“Event : ”,event);

console.log(“Clicked Element : ”,event.target);

console.log(“Clicked Element Id : ”,event.target.id);

}

**</script>**

**Syntax : <element>.removeEventListener(<event>,<function>);**

## Mouse events

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| click | onclick | When mouse click on an element |
| mouseover | onmouseover | When the cursor of the mouse comes over the element |
| mouseout | onmouseout | When the cursor of the mouse leaves an element |
| mousedown | onmousedown | When the mouse button is pressed over the element |
| mouseup | onmouseup | When the mouse button is released over the element |
| mousemove | onmousemove | When the mouse movement takes place. |

**Example : onclick & click**

**(ex) <html>**

**<head> <title>** Document **</title> </head>**

**<body>**

**<h1 onclick = “alert()”>** click me one time **</h1>**

**<h1 id = “**heading**” >** click me one time **</h1>**

**<body>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

element**.addEventListener(“**click**”,**myFun**);**

**function** myFun(){ **alert();** }

**</script>**

**</html>**

**Example : ondblclick & dblclick**

**(ex)** //write inside of HTML

**<h1 ondblclick = “alert()”>** click me double times **</h1>**

**<h1 id = “**heading**” >** click me double times **</h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

element**.addEventListener(“**dblclick**”,**myFun**);**

**function** myFun(){ **alert();** }

**</script>**

**Example : onmouseover & mouseover**

**(ex)** //write inside of HTML

**<h1 onmouseover = “alert()”>** Hover me **</h1>**

**<h1 id = “**heading**” >** Hover me **</h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

element**.addEventListener(“**onmouseover**”,**myFun**);**

**function** myFun(){ **alert();** }

**</script>**

**Example : onmousedown & mousedown**

**(ex)** //write inside of HTML

**<h1 onmousedown = “alert()”>** Click Me **</h1>**

**<h1 id = “**heading**” >** Click Me **</h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

element**.addEventListener(“**mousedown**”,**myFun**);**

**function** myFun(){ **alert();** }

**</script>**

**Example : onmouseup & mouseup**

**(ex)** //write inside of HTML

**<h1 onmouseup = “alert()”>** Click Me **</h1>**

**<h1 id = “**heading**” >** Click Me **</h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

element**.addEventListener(“**mouseup**”,**myFun**);**

**function** myFun(){ **alert();** }

**</script>**

**Example : onmousemove & mousemove**

**(ex)** //write inside of HTML

**<h1 onmousemove = “**myFun()**”>** Click Me **</h1>**

**<h1 id = “**heading**” >** Click Me **</h1>**

**<script>**

**let** element = **document.getElementById(“**heading**”);**

element**.addEventListener(“**mousemove**”,**myFun**);**

**function** myFun(){console.log(“Hello”); }

**</script>**

## Keyboard events:

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| Keydown | onkeydown | when user press key the event is work. |
| Keyup | onkeyup | When user press key then release that key the event is work. |
| Keypress | onkeypress | When user press key the event is work.(like keydown) |

**Example : onkeydown & keydown & keypress**

**(ex)** //write inside of HTML

**<input type = “**text**” id = “**input” **onkeydown = “**myFun()**”>**

**<script>**

**//let** input = **document.getElementById(“**input**”);**

**//**element**.addEventListener(“**keydown**”,**myFun**);**

**//function** myFun(){

//console.log(input.value);

//}

**</script>**

**Example : onkeyup & keyup**

**(ex)** //write inside of HTML

**<input type = “**text**” id = “**input” **onkeyup = “**myFun()**”>**

**<script>**

**//let** input = **document.getElementById(“**input**”);**

**//**element**.addEventListener(“**keyup**”,**myFun**);**

**//function** myFun(){

//console.log(input.value);

//}

**</script>**

## Form events:

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| focus | onfocus | When the user focuses on an element. |
| submit | onsubmit | When the user submits the form. |
| Blur | onblur | When the focus is away from a form element. |
| change | onchange | When the user modifies or changes the value of a form element. |
| input | oninput | Script to be run when an element is invalid. |
| reset | onreset | Fires when the Reset button in a form is clicked |
| select | onselect | Fires after some text has been selected in an element |

**Example : onfocus & focus**

**(ex)** //write inside of HTML

**<label>** Enter your Name : **</label>**

**<input type = “**text**” id = “**input” **onfocus = “**myFun()**”>**

**<script>**

**//let** input = **document.getElementById(“**input**”);**

**//**element**.addEventListener(“**focus**”,**myFun**);**

**//function** myFun(){

//console.log(input.value);

//}

**</script>**

**Example : onblur & blur**

**(ex)** //write inside of HTML

**<label>** Enter your Name : **</label>**

**<input type = “**text**” id = “**input” **onblur = “**myFun()**”>**

**<script>**

**//let** input = **document.getElementById(“**input**”);**

**//**element**.addEventListener(“**blur**”,**myFun**);**

**//function** myFun(){

//console.log(input.value);

//}

**</script>**

**Example : onsubmit & submit**

**(ex)** //write inside of HTML

**<form onsubmit = “**myFun()” **id = “**form**” >**

**<input type = “**submit” **value = “**submit” **>**

**</form>**

**<script>**

**//let** form = **document.getElementById(“**form**”);**

**//**element**.addEventListener(“**submit**”,**myFun**);**

**//function** myFun(){

**//alert();**

//}

**</script>**

**Example : onreset & reset**

**(ex)** //write inside of HTML

**<form onreset = “**myFun()” **id = “**form**” >**

**<input type = “**reset” **value = “**reset” **>**

**</form>**

**<script>**

**//let** form = **document.getElementById(“**form**”);**

**//**element**.addEventListener(“**reset**”,**myFun**);**

**//function** myFun(){

**//alert();**

//}

**</script>**

**Example : onchange & change**

**(ex)** //write inside of HTML

**<select id = “**select” **onchange = “**myFun()**”**>

**<option value = “**one” **>** one **</option>**

**<option value = “**two**”>** Two **</option>**

**</select>**

**<script>**

**//let** selectElement = **document.getElementById(“**select**”);**

**//**selectElement**.addEventListener(“**change”,myFun**);**

**//function** myFun(){

//console.log(selectElement.value);

//}

**</script>**

**Example : onchange & change**

**(ex)** //write inside of HTML

**<input type = “**text**” id = “**input**” oninput = “**myFun()**” >**

**<script>**

**//let** input = **document.getElementById(“**input**”);**

**//**input**.addEventListener(“**input”,myFun**);**

**//function** myFun(){ console.log(input.value); }

**</script>**

**Example : onselect & select**

**(ex)** //write inside of HTML

**<input type = “**text**” id = “**input**” onselect = “**myFun()**” >**

**<script>**

**//let** input = **document.getElementById(“**input**”);**

**//**input**.addEventListener(“**select”,myFun**);**

**//function** myFun(){ **alert();** }

**</script>**

## Window/Document events

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| load | onload | When the browser finishes the loading of the page |
| unload | onunload | When the visitor leaves the current webpage, the browser unloads it |
| resize | onresize | When the visitor resizes the window of the browser |

**Example : onload & load**

**(ex)** //write inside of HTML

**<body onload = “**myFun()**” id = “**body**”>**

**<script>**

**//let** bodyElement = **document.getElementById(“**body**”);**

**//**input**.addEventListener(“**load”,myFun**);**

**//function** myFun(){ **alert();** }

**</script>**

**</body>**

**Example : onresize & resize**

**(ex)** //write inside of HTML

**<body onresize = “**myFun()**” id = “**body**”>**

**<script>**

**//let** bodyElement = **document.getElementById(“**body**”);**

**//**input**.addEventListener(“**resize”,myFun**);**

**//function** myFun(){ **alert();** }

**</script>**

**</body>**

# Event Methods

**<event>.preventDefault() 🡺** stop default element action.

**<event>.stopPropagation() 🡺** restart the default element action.

**(ex)**

//write inside of the html tag

**<form>**

**<label>** Enter your Name : **</label>**

**<input type = “**text” **id = “**userName” **>**

**<input type = “**submit**” value = “**Submit” **id = “**myBtn”**>**

**</form>**

**<script>**

**let** userName = **document.getElementById(“**userName**”);**

**let** myBtn = **document.getElementById(“**myBtn**”);**

myBtn**.addEventListener(“**click**”,**isValidName**);**

**function** isValidName(event){

**if(**userName.value == “”**)** event**.preventDefault();**

**else** event**.stopPropagation();**

}

**</script>**

# JSON

# The JavaScript JSON is an acronym of JavaScript Object Notation. It provides a format for storing and transporting data. It is a lightweight human readable collection of data that can be accessed in a logical manner.

## Points to remember

* It generates and stores the data from user input.
* It can transport the data from the server to client, client to server, and server to server.
* It can also build and verifying the data.

**(ex) let** user = {

“firstName” : “Deepak”,

“secondName” : “surya”,

“age” : 20,

“job” : { “job1” : “programmer”, “job2” : “Hacker” },

“salary” : 10000,

“hobbies” : [ “football”, “volleyball”, “Book reading” ]

};

//Access value in object

console.log(user.firstName,user.secondName,user.age,user.salary);

console.log(user.job); // return object

console.log(user.job.job1); //now return value from object

console.log(user.hobbies); //return array

console.log(user.hobbies[0]); //return given index value

**Methods**

**JSON.parse(<json\_string>) 🡺** return js object.

**JSON.stringify(<object>) 🡺** return json string.

**JSON.parse() Method : Return JS Object**

**(ex) //Note : this is string not a object.**

**let** userData = `{ “fname” : “Deepak”, “sname” : “Surya” }`;

**let** userObject = **JSON.parse(**userData**);**

console.log(“Fname : ” , userObject.fname);

console.log(“Sname : ” , userObject.sname);

**JSON.stringify() Method : Return JSON String**

**(ex) //Note : now this is not a string**

**let** userData = { “fname” : “Deepak”, “sname” : “Surya” };

**let** string = **JSON.stringify(**userData**);**

console.log(string);

console.log(typeof(string));