**Client, Server and Database**

|  |
| --- |
| **Web** **Server**  ASP  JSP  PHP  Python  C#  Java  **Database**  MS SQL  MySQL  Oracle  MongoDB  SQLite  **Client**  HTML  CSS  JavaScript |

**Client Side Script**

The script which runs on the browser is called as Client Side Script.

**Example**

JavaScript, VBScript, jQuery, Angular JS, React JS

**Server Side Script**

The script which runs on the web server is called as server side script.

**Example**

ASP – IIS (Internet Information System)

JSP – Tomcat/Glassfish

PHP – Apache

Python - Apache

**What is JavaScript?**

JavaScript is a cross-platform, object-oriented scripting language used to make webpages interactive (e.g., having complex animations, clickable buttons, popup menus, etc.).  There are also more advanced server side versions of JavaScript such as Node.js, which allow you to add more functionality to a website than downloading files (such as real time collaboration between multiple computers).

JavaScript contains a standard library of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects; for example:

* *Client-side JavaScript* extends the core language by supplying objects to control a browser and its *Document Object Model* (DOM). For example, client-side extensions allow an place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
* *Server-side JavaScript* extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

This means that in the browser, JavaScript can change the way the webpage (DOM) looks. And, likewise, Node.js JavaScript on the server can respond to custom requests from code written in the browser.

**History of JavaScript**

In **September 1995**, a Netscape programmer named **Brandan Eich** developed a new scripting language in just 10 days. It was originally named **Mocha**, but quickly became known as **LiveScript** and, later, **JavaScript**.

**JavaScript** often abbreviated as **JS**, is a programming language that confirms to the ECMAScript specification.JavaScript is high-level, often just-in-time compiled and multi-paradigm. It has dynamic typing, prototype-based object-orientation and first-class functions.

JavaScript engines were originally used only in web browsers, but they are now core components of some servers and a variety of applications. The most popular runtime system for this usage is Node.js.

**Features of JavaScript**

1. Light weight scripting language
2. Dynamic Typing
3. Functional Style
4. Event Handling
5. Fast Execution
6. Cross Browser Support
7. Object Oriented Programming Support

**Window**

Window is a primary object of JavaScript which provides some inbuilt properties and methods.

alert()

confirm()

prompt()

open()

location()

setInterval()

clearInterval()

setTimeout()

etc…

**Alert()**

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

**Syntax :**

window.alert("sometext");

The window.alert() method can be written without the window prefix.

**Example :**

alert("I am an alert box!");

**Confirm Box**

Confirm box is used to take confirmation by user. It has two buttons. (OK and Cancel)

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

**Syntax :**

window.confirm("sometext");

The window.confirm() method can be written without the window prefix.

**Example :**

if (confirm("Press a button!")) {  
  txt = "You pressed OK!";  
} else {  
  txt = "You pressed Cancel!";  
}

**Prompt()**

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

**Syntax** :

window.prompt("sometext","defaultText");

The window.prompt() method can be written without the window prefix.

**Example :**

*let* person = prompt("Please enter your name", "Harry Potter");  
*let* text;  
if (person == null || person == "") {  
  text = "User cancelled the prompt.";  
} else {  
  text = "Hello " + person + "! How are you today?";  
}

***var*, *let* and *const* keyword**

The scope of a variable defined with the keyword “var” is limited to the “function” within which it is defined. If it is defined outside any function, the scope of the variable is global.**var is “function scoped”.**

The scope of a variable defined with the keyword “let” or “const” is limited to the “block” defined by curly braces i.e. {} .**“let” and “const” are“block scoped”.**

The scope of a variable defined with the keyword “const” is limited to the block defined by curly braces. However if a variable is defined with keyword const, it cannot be reassigned.**“const” cannot be re-assigned to a new value. However it CAN be mutated.**

**== V/s ===**

The ‘==’ operator tests for abstract equality i.e. it does the necessary type conversions before doing the equality comparison.

But the ‘===’ operator tests the strict equality i.e. it will not do the type conversion hence if the two values are not of the same type, when compared, it will return false.

**Type Conversion or Type Casting :**

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

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

**Converting Strings to Numbers**

* The global method Number() can convert strings to numbers.
* Strings containing numbers (like "3.14") convert to numbers (like 3.14).
* Empty strings convert to 0.
* Anything else converts to NaN (Not a Number).

Number("3.14")    // returns 3.14  
Number(" ")       // returns 0  
Number("")        // returns 0  
Number("99 88")   // returns NaN

## Number Methods

In the chapter Number Methods, you will find more methods that can be used to convert strings to numbers:

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

## The Unary + Operator

The **unary + operator** can be used to convert a variable to a number:

**Example :**

let y = "5";      // y is a string  
let x = + y;      // x is a number

If the variable cannot be converted, it will still become a number, but with the value NaN (Not a Number):

**Example :**

let y = "John";   // y is a string  
let x = + y;      // x is a number (NaN)

## Converting Numbers to Strings

The global method String() can convert numbers to strings.

It can be used on any type of numbers, literals, variables, or expressions:

**Example :**

String(x)         // returns a string from a number variable x  
String(123)       // returns a string from a number literal 123  
String(100 + 23)  // returns a string from a number from an expression

The Number method toString() does the same.

**Example :**

x.toString()  
(123).toString()  
(100 + 23).toString()

## More Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| toExponential() | Returns, a string with a number rounded and written using exponential notation. |
| toFixed() | Returns a string, with a numbers rounded and written with a specified number of decimals. |
| toPrecision() | Returns a string, with a number written with a specified length. |

**Window**

**Location :** Location property is used to redirect location one page to another page.

**Syntax :** window.location = “URL”;

**Example :** window.location = “https://[www.techpile.in](http://www.techpile.in)”;

**Open** **:** Open() is used to open a website on a new tab or new window.

**Syntax :** window.open(“URL”, “\_blank”); // New Tab

window.open(“URL”, “\_blank”, “height=value, width=value”); // New Window

**Example :** window.open(“https”//www.techpile.in”, “\_blank”);

window.open(“https”//www.techpile.in”, “\_blank”, “height=450px, width=100%”);

**Print :** print() is used to print all content of body section. It is located in window object.

**Syntax :** window.print();

**Play :** play() is used to play any audio or video element.

**Syntax :** object.play();

**Pause :** pause() is used to pause any running audio or video element.

**Syntax :** object.pause();

**getElementByID() :** getElementById() is used to select HTML elements on the basis of ID.

Return type of **getElementByID()** function is **Current Object.**

**getElementByID()** is located in **Document Object.**

**Document** is the built-in object of DOM (Document Object Model) providing some property and methods to handle HTML tags.

The document method **getElementById()**returns an Element object representing the element whose id property matches the specified string. Since element IDs are required to be unique if specified, they're a useful way to get access to a specific element quickly.

If you need to get access to an element which doesn't have an ID, you can use **querySelector()** to find the element using any selector.

**Parameters**   
**ID** : The ID of the element to locate. The ID is case-sensitive string which is unique within the document; only one element may have any given ID.

**Return Value :** An Element object describing the DOM element object matching the specified ID, or null if no matching element was found in the document.

**Syntax :** document.getElementById(‘Id’);

**OR**

document.querySelector(‘#Id’);

**Function in JavaScript :** Functions are one of the fundamental building blocks in JavaScript A JavaScript function is executed when we call it.

There are two types of function in JavaScript :

**1).Built-in function**

A function which is coming along with system interpreter is known as **built-in function**.

**2). User-defined function**

A function which is developed by user according to business logic is known as **User-defined function**.

**There are two keywords are present to define a User-defined function :**

**1). Function** (mandatory)

**2). Return** (optional)

**Syntax (Without Parameter)4**

*function* **functionName**(){

// Code to be executed.

}

**Value :** Value property is used to get/set the value of selected form control.

**Syntax :** *var* **val** = document.getElementById(“txt”).value;

**Set :** document.getElementById(‘Id’).**value**=”Any Value”;

**OR**

*var* **val** = document.getElementById(‘Id’);

*val*.**value** = “Any Value”;

**innerText :** innerTextproperty is used to set or get plain text without extra spacing of selected HTML tag.

**Get :**

<div id=”data”></div>

var data = document.getElementById(‘data’).innerText;

**OR**

var data = document.getElementById(‘data’);

data.innerText;

**Set :**

document.getElementById(‘data’).innerText=”Value”;

**OR**

var data = document.getElementById(‘data’);

data.innerText = “Value”;

# Object.values()

The Object.values() method returns an array of a given object's own enumerable property values, in the same order as that provided by a for…in loop. (The only difference is that a for…in loop enumerates properties in the prototype chain as well.)

**Syntax :** Object.values(obj);

*const* **object1** = {

a:’somestring’,

b:45,

c:false

};

console.log(**Object.values**(object1));

**Output :**

Array ["somestring", 42, false]

**Parameters**

**obj** : The object whose enumerable own property values are to be returned.

**Return value :** An array containing the given object's own enumerable property values.

**innerHTML :** innerHTML property is used to set/get content with tag and extra space of selected HTML element.

**SET :**

**Syntax :**

document.getElementById(‘dv’).innerHTML=”<b>TECHPILE TECHNOLOGY</b>”;

**OUTPUT :**

**TECHPILE TECHNOLOGY**

**GET :**

**Syntax :**

**<div id=”dv”>**

**<h1>**JavaScript, HTML & CSS**</h1>**

**</div>**

var a = document.getElementById(‘dv’).innerHTML;

**OUTPUT :**

**<h1>**JavaScript, HTML & CSS**</h1>**

**textContent :** textConent property is used to set/get plain text including extra spaces of select HTML element.

**Syntax :**

**SET :**

**<div id=”dv”>**

**<h1>**JavaScript, HTML & CSS**</h1>**

**</div>**

var a = document.getElementById(‘dv’).textContent=”TECHPILE TECHNOLOGY’;

**OUTPUT :**

TECHPILE TECHNOLOGY

**GET :**

**<div id=”dv”>**

**<h1>**JavaScript, HTML & CSS**</h1>**

**</div>**

var a = document.getElementById(‘dv’).textContent;

**OUTPUT :**

JavaScript, HTML & CSS

**setInterval() :** setInterval function is used to execute a group of statement for every given time period. This function is defined in window object.

This function have two parameters (callback function, time period);

**Syntax :**

window.setInterval(‘functionName()’,time(ms));

**OR**

window.setInterval(functionName,time(ms));

**Example :**

window.setInterval(‘demo()’,1000);

window.setInterval(demo,1000);

**setTimeout() :** setTimeout function is used to execute once a group of statement after given time period.

**Syntax :**

window.setTimeout(callbackFunction(),timePeriod);

window.setTimeout(“callbackFunction()”,timePeriod);

**Example :**

window.setTimeout(‘demo()’,1000);

window.setTimeout(demo,1000);

**Parameter :** Parameter inputs to the function, if a function contains parameter, then at the time of calling compulsory we have to provide value to that parameter.

**Syntax :**

function functionName(param1,param2,…paramn){

// Write your code here.

}

**Example :**

function add(num1,num2){

console.log(num1+num2);

}

add(10,20); //30

**Return Keyword :** Return keyword is used to return output to the caller function after execution of business logic.

**Syntax :**

function functionName(param1,param2,…paramn){

return value;

}

**Example :**

function add(num1,num2){

return num1+num2;

}

a=add(10,20); //30

console.log(a);

function exp(x,y){

res=x\*\*y;

return res;

}

a=exp(3,2);

alert(a);

**clearInterval() :** clearInterval function is used to stop functionality of setInterval.

**Date Object :**

Date is a pre-defined object providing some method related to current date and time.

**Syntax :**

*var* dt = *new* Date();

getDay(); // 0-6

getMonth(); // 0-11

getFullYear(); // 2021

getSeconds(); // 0-59

getMilliseconds(); // 0-999

getMinutes(); // 0-59

getHours(); // 0-23

getDate(); // 1-31

**Example :**

x.getDate(); // 29

x.getMonth(); // 10

x.getFullYear(); // 2021

**ARRAY in JS**

In JS array is collection of heterogeneous data type elements.

Indexing of array always starts from 0 and last index is (arrayName.length-1). Where length is property.

**Syntax :**

var arrayName=[item1, item2, item3, item4,…….itemN];

**Example :**

var arr=[“Deep”, 21, “GPL”, true];

alert(arr[0]); // **Deep**

alert(arr[1]); // **21**

alert(arr[2]); // **GPL**

alert(arr[3]); // **true;**

**Length**

Length property is used to return length of array or string.

**Example :**

alert (arr.length); // **4**

alert (arr[arr.length-1]); // **true**

**Concat :** concat() is used to concat two or more array and returns a new array.

**Syntax :** vararr1=[item1, item2, item3, itemN,….];

var arr2=[item4, item5, item6, itemN,….];

var arr3=[item7, item8, item9, itemN,….];

arr1.concat (arr2);

arr1.concat (arr2, arr3);

**indexOf** : indexOf() is used to return index of first occurrence element in given array.

If given element is not present then it returns -1.

**Syntax :** arrayName.indexOf(item);

**Example :**

var arr=[“HTML”, “CSS”, “JS”];

alert (arr.indexOf(“JS”); //2

alert (arr.indexOf(“js”); //-1

**Syntax 2 :**

arr.indexOf(item, startIndex);

**Example :**

var arr=[“HTML”, “CSS”, “JS”, “HTML”];

alert (arr.indexOf(“HTML”, 2) //3

alert (arr.indexOf(“HTML”, 2) //-1

**lastIndexOf :** lastIndexOf() is used to return index of last occurrence element in given array.

If element is not present then this function will return -1.

**Syntax :** arr.lastIndexOf(item);

**Example :**

var arr=[“HTML”, “CSS”, “JS”, “HTML”];

document.write(arr.lastIndexOf(“HTML”)); //3

**Syntax 2 :** arr.lastIndexOf(item, startIndex);

**Example :** var arr=[“HTML”, “CSS”, “JS”, “HTML”, “PYTHON”, “PHP”];

document.write(arr.lastIndexOf(“PYTHON”, 2); //-1

document.write(arr.lastIndexOf(“HTML”, 4); //3

**Reverse :** reverse() is used to reverse element of give array.

**Syntax :** arrayName.reverse();

**Example :** var arr=[“HTML”, “CSS”, “JS”, “HTML”, “PYTHON”, “PHP”];

alert (arr.reverse());

**Sort :** sort() is used to sort elements of given array.

**Syntax :** arrayName.sort();

**Example :** var arr=[“Deep”, “Roshan”, “Deepak”, “Sandeep”];

alert(arr.sort());

**Slice :** slice() is used to return a part of array on the basis of start index and end index. We can also use negative index. Slice() doesn’t make changes in original array.

**Syntax :** arrayName.slice(startIndex, endIndex); //endIndex is excluding index.

**Example :** var arr=[“Deep”, “Roshan”, “Deepak”, “Sandeep”];

arr.slice(1,3); //Roshan, Deepak

arr.slice(-2) // Deepak, Sandeep

**Splice :** splice() is used to add/remove elements of given array. Splice method make changes in original array.

**Syntax :** arrayName.splice(startIndex, noOfItemsToDelete, itemsToAdd);

**Example :** var arr=[“Deep”, “Roshan”, “Deepak”, “Sandeep”];

alert(arr.splice(1, 2, “Rahul”)); // Deep, Rahul, Sandeep

alert(arr.splice(0, 3, “Riya”,”Kajal”,”Pooja”)); // Riya, Kajal, Pooja, Sandeep

**Pop :** pop() is used to remove last element of any array. It returns removed element.

**Syntax :** arrayName.pop();

**Example :** arr=[10,20,30,40,50];

alert(arr.pop()); //50

**Push :** push() is used to add a new element in the last position in array and returns new length of array.

**Syntax :** arrayName.push(element);

**Example :** arr=[10,20,30,40,50];

alert(arr.push(60)); //6

**Shift :** shift() is used to remove first element of any array. It returns removed element.

**Syntax :** arrayName.shift();

**Example :** var arr=[10,20,30,40,50];

alert(arr.shift()); //10

**Unshift :** unshift() is used to add a new element in the first position in array and returns new length of array.

**Syntax :** arrayName.unshift(element);

**Example :** arr=[10,20,30,40,50];

alert(arr.unshift(60)); //6

**Includes :** includes() is used to check any specific element is present or not in the existing array. It returns true if the given element is present otherwise it returns false.

**Syntax 1 :** arrayName.includes(element);

**Example :**

arr=[10,20,30,40,50];

alert(arr.includes(20)); // true

**Syntax 2 :** arrayName.includes(item,startIndex);

alert(arr.includes(20,2); // false

**Join :** join() is used to join element of array on the basis of separator.

**Syntax :** arrName.join(separator);

**Example :** arr=[10,20,30,40,50];

alert(arr.join(“^”)); // 10^20^30^40^50

**forEach :** forEach() is used to call function for every element present in array.

**Syntax :** arrayName.forEach(function(currElem, index, arrayName){});

Index and arrayName are optional.

**Example :** var arr=[“Deep”,”Roshan”,”Deepak”,”Sandeep”];

arr.forEach(function(currElem,index){

alert(index+” ”+currElem);

});

**Entries :** entries() is used to return array iterator object with key/value pair.

**Syntax :** var iterator **=** arrayName.entries()

for (var a of iterator){ iterator // array iterator object

alert(a);

}

**Example :** var arr=[“HTML”, “CSS”, “JS”, “JQUERY”, “BOOTSTRAP”];

var iterator =arr.entries();

for(var a of iterator){

alert(iterator);

}

0, HTML

1, CSS

2, JS

3, JQUERY

4, BOOTSTRAP

**Values :** values() is used to return array iterator object with array items.

**Syntax :** var iterator **=** arrayName.values()

for (var a of iterator ){ iterator // array iterator object

alert(a);

}

**Example :** var arr=[“HTML”, “CSS”, “JS”, “JQUERY”, “BOOTSTRAP”];

var iterator =arr.values();

for(var a of iterator){

alert(iterator);

}

HTML

CSS

JS

JQUERY

BOOTSTRAP

**String functions**

**indexOf() :** indexOf() is used to return index of first occurrence character in given string.

**Syntax :** stringName.indexOf(‘character’);

**Example :**

var str=”Techpile Technology”;

str.indexOf(‘T’); //0

We also can pass one more parameter for define starting index.

**lastIndexOf() :** lastIndexOf() is used return last index of any character.

**Syntax :** stingName.lastIndexOf(‘character’);

**Example :** str.lastIndexOf(‘0’); // 16

str.lastIndexOf(“l”,14); // 6

**slice() :** slice() is used to return a part of string in given string on the basis of startIndex and endIndex. Negative index is acceptable here.

**Syntax :** stringName.slice(startIndex, endIndex);

**Example :**

var str=”Techpile Technology”;

str.slice(3,7); // hpil

str.slice(9,19); // Technology

str.slice(-4,-2); // lo

**substring() :** substring() is used to find substring from any string.

**Syntax :** stringName.substring(startIndex,endIndex);

**Example :** var str=”Techpile Technology Pvt. Ltd.”;

alert(str.substring(9,12)); // Tec

alert(str.substring(-3,-2)); // Error

**substr() :** substr() is used to return a part of string in given string on the basis of startIndex and length.

**Syntax :** stringName.substr(startIndex,charactersCount);

**Example :** var str=”TECHPILE”;

alert(str.substr(5,2)); // IL

alert(str.substr(2,4)); // CHPI

**trim() :** trim() is used to remove spaces from any string.

**Syntax :** stringName.trim();

**Example :** var str=” TECHPILE “;

alert(str.length); // 16

str=str.trim();

alert(str.length); // 8

**rtrim() :** rtrim() is used to remove spaces from right side in any string.

**Syntax :** stringName.rtrim();

**Example :** var str=”TECHPILE “;

alert(str.length); // 11

str=str.rtrim();

alert(str.length); // 8

**ltrim() :** ltrim() is used to remove spaces from left side in any string.

**Syntax :** stringName.ltrim();

**Example :** var str=” TECHPILE“;

alert(str.length); // 12

str=str.ltrim();

alert(str.length); // 8

**split() :** split() is used to split a string into array of substring based on separator.

**Syntax :** stringName.split(separator);

**Example :** var str=”TECHPILE TECHNOLOGY PVT. LTD.”;

alert(str.split(“ “)); // “TECHPILE”,”TECHNOLOGY”,”PVT.”,”LTD.”;

**join() :** join() is used to join array elements based on separator.

**Syntax :** stringName.join(separator);

**Example :** var str=”TECHPILE TECHNOLOGY PVT. LTD.”;

alert(str.join(“,“)); // “TECHPILE”,”TECHNOLOGY”,”PVT.”,”LTD.”;

**charAt() :** charAt() is used to return character at specified index in a given string. If character doesn’t exist then it will return ‘’;

**Syntax :** stringName.charAt(index);

**Example :** str=”Techpile Technology”;

str.charAt(5); // i

**charCodeAt() :** charCodeAt() is used to return ASCII value of any character in a given string through index.

**Syntax :** stringName.charCodeAt(index);

**Example :** str=”Techpile Technology”;

str.charCodeAt(5); // 73

**includes() :** includes() function is used to check whether given string present in sentence or not. It returns true if given string is present in sentence otherwise it return false.

**Syntax 1 :** stringName.includes(“string”);

**Example :** var str=”Techpile Technology”;

alert(str.includes(“Tech”); // true

alert(str.includes(“TECH”); // false

**Syntax 2 :**

stringName.includes(“string”,startIndex);

**Example :** var str=”Techpile Technology”;

alert(str.includes(“Tech”,10); // false

alert(str.includes(“Tech”,0); // true

**replace() :** replace() function is used to replace old sting with new string. By default replace function replaced first matches string.

**Syntax 1 :** stringName.replace(“Initial String”, “New String”);

**Syntax 2 :** stringName.replace(/tech/g,”New String”); // **Global Replacement**

**Example :** var str=”Techpile Technology”;

str.replace(“Tech”,”TECH”); // TECHpile Technology

**Example :** var str=”Techpile Technology”;

str.replace(/Tech/g,”TECH”); // TECHpile TECHnology

**startsWith() :**

**endsWith() :**

**toString() :** toString() function is used to convert other datatype value into string.

**Syntax :** variableName.toString();

**typeof() :** typeof() function is used to check type of data of any variable.

**Syntax :** typeof(variableName);

**getElementsByTagName() :** getElementsByTagName() function is used to return collection of all element with specified tag name as collectionHTMLObject.

**Syntax :** document.getElementsByTagName(‘tagName’);

**Example :**

<h2>HTML</h2>

<h2>CSS</h2>

<h2>JS</h2>

var a1=document.getElementsByTagName(‘h2’)[0].innerText;

var a2=document.getElementsByTagName(‘h2’)[1].innerText;

var a3=document.getElementsByTagName(‘h2’)[2].innerText;

**getElementsByClassName():** getElementsByClassName() function is used to return collection of elements with specified class name as ObjectHTMLCollection.

**Syntax :** document.getElementsByClassName(‘className’);

**Example :** <b class=”bolder”>Summer Training</b>

<b class=”bolder”>Winter Training</b>

<b class=”bolder”>Apprenticeship</b>

var a=document.getElementsByClassName(‘bolder’)[0].textContent; // Summer Training

var b=document.getElementsByClassName(‘bolder’)[1].textContent; // Winter Training

var c=document.getElementsByClassName(‘bolder’)[2].textContent; // Apprenticeship

**querySelector() :** querySelector() is used to select any first matching HTML elements on the basis of CSS selectors like ID selector, class selector, tag selector etc.

**Syntax :** var variableName = document.querySelector(‘#id’);

var variableName = document.querySelector(‘.className’);

var variableName = document.querySelector(‘selector’);

**Example :** <div>HTML</div>

<div>CSS</div>

<div id=”js”>JS</div>

<div id=”jQuery”>jQuery</div>

<script>

var dv1=document.querySelector(‘div’).innerText; // HTML

var dv2=document.querySelector(‘#js’).innerText; // JS

var dv3=document.querySelector(‘#jquery’).innerText; // jQuery

var dv4=document.querySelector(‘#js, #jquery).innerText; // JS jQuery

</script>

**querySelectorAll :** querySelectorAll() is used to return collection of all matches element with specified selector name as NodeList Object.

**Syntax :** var variableName=document.querySelectorAll(‘selector’);

**Example :** <div>HTML</div>

<div>CSS</div>

<div id=”js”>JS</div>

<div id=”jQuery”>jQuery</div>

<script>

var dv=document.querySelectorAll(‘div’).innerText;

var dv=document.querySelectorAll(‘div’)[1].innerText; // JS

</script>

**classList :** classList is an object providing some methods related to CSS class.

add()

remove()

toggle()

contains()

**add() :** add() is used to add CSS class to the selected HTML element.

**Syntax :** document.querySelector(‘selector’).classList.add(‘className’);

**Example :**

.a{

background:red;

color:white;

font-size:40px;

height:100px;

text-align:center;

width:100%;

}

var box=document.getElementById(‘box’).classList.add(‘a’);

**remove() :** remove() is used to remove applied CSS class to the selected HTML element.

**Syntax :** document.querySelector(‘selector’).classList.remove(‘className’);

**Example :**

.a{

background:red;

color:white;

font-size:40px;

height:100px;

text-align:center;

width:100%;

}

var box=document.getElementById(‘box’).classList.remove(‘a’);

**toggle :** toggle() is used to provide toggle between add and remove function.

**Syntax :** document.querySelector(‘selector’).classList.toggle(‘className’);

**Example :**

.a{

background:red;

color:white;

font-size:40px;

height:100px;

text-align:center;

width:100%;

}

var box=document.getElementById(‘box’).classList.toggle(‘a’);

**Contains :** contains() is used to check whether given CSS class is applied or not applied.

It returns true if given CSS class is applied otherwise returns false.

**getAttribute :** getAttribute() is used to get value of attribute of any selected HTML element.

**Syntax :** objectName.getAttribute(‘attributeName’);

**Example :** <img src=”abc.jpg” id=”im1” class=”im1”>

<script>

var im=document.querySelector(‘#im1);

alert(im.getAttribute(‘src’)); // abc.jpg

alert(im.getAttribute(‘id)); // im1

alert(im.getAttribute(‘class)); // im1

</script>

**setAttribute :** setAttribute() is used to set attribute value to the selected HTML element.

**Syntax :** objectName.setAttribute(‘attributeName’,’value’);

**Example :** <img src=”abc.jpg” id=”im1”>

<script>

var im=document.querySelector(‘#im1’);

alert(im.setAttribute(‘class’,’img1’);

</script>

**MATH OBJECT**

Math is an object which provides some predefined functions related to mathematical operations.

random, ceil, floor, round, min, max, count, pow, sqrt, cbrt, etc.

**random :** random() is used to generate random numbers between 0 (inclusive) and 1(exclusive).

0 **to** 0. 99999999999999999

**Syntax :** Math.random(); // 0.143

**ceil :** ceil() is used to return largest integer value of given number.

**Syntax :** Math.ceil(5.5); // 6

**floor :** floor() is used to return lowest integer value of given number.

**Syntax :** Math.floor(5.5); // 5

**round :** round() is used to return closest integer value of given number.

**Syntax :** Math.round(5.7); // 6

**sqrt :** sqrt() is used to return square root value of given number. It doesn’t accepts negative numbers. If we try to provide negative values then it will return Nan.

**Syntax :** Math.sqrt(4); // 2

**cbrt :** cbrt() is used to return cube root value of given number. It can accept negative numbers.

**Syntax :** Math.cbrt(27); // 3

**pow :** pow() is used to calculate power value of any number.

**Syntax :** Math.pow(4,2); // 16 Where 4 is base and 2 is power.

**min :** min() is used to find out minimum value in any collection.

**Syntax :** Math.min(…arrayName);

**max :** max() is used to find out maximum value in any collection.

**Syntax :** Math.max(…arrayName);

**log :** log() is used to return logarithm value of given number with respect to base e.

**Syntax :** Math.log(number);

**Example :** document.write**(**Math.log(10)); // 1

**OBJECT**

Object is the collection (variables) and methods (functions).

How to access property and method from object.

**Syntax :**

var objectName = {

propertName1:value1,

properyName2:value2,

propertyName3:value3,

…………………………………..

…………………………………..

properyNameN:value

};

**OR**

var objectName={

propertName1:value1,

methodName : methodDefintion,

propertyName2:value2

};

**Example :**

<html>

<head></head>

<body>

<script>

var obj={

name:”Deeproshan Kumar”,

college : “GP Lucknow”,

age : 21,

contact : 7271053944

};

var student={

name:”Sandeep Kumar”,

college:”Modern Inter College”,

age:16,

address:”Ghazipur”

}

document.write(“Name : “+obj.name+”<br>”);

document.write(“College Name : “+obj.college+”<br>”);

document.write(“Age : “+obj.age+”<br>”);

document.write(“Contact : “+obj.contact+”<br>”);

</script>

</body>

</html>

**ARROW FUNCTION**

Also known as arrow / fat arrow function.

**Syntax :**

()=>{

};

**Example :**

// Definition

const greet =(name)=>{

document.write(“Hello”+name);

}

greet(“Deeproshan”); // Calling

**2.**

const sum =(n1,n2)=>{

alert(`Result : ${n1+n2}`);

}

sum(10,20); // Calling

**ARRAY OF OBJECTS**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Array of Objects</title>

</head>

<body>

    <p></p>

    <button onclick="func();">CLICK ME</button>

    <script>

        const func=()=>{

        const studentInfo=[

            {

                name : 'Deeproshan Kumar',

                collegeName : 'GPL',

                branch : 'IT',

                age : 21

            },

            {

                name : 'Sandeep Kumar',

                collegeName : 'MIC',

                branch : 'CSE',

                age : 17

            },

            {

                name : 'Ajit Yadav',

                collegeName : 'CCA',

                branch : 'CSE',

                age : 20

            },

            {

                name : 'Ayushi Yadav',

                collegeName : 'GGPG',

                branch : 'CS',

                age : 21

            },

            {

                name : 'Ishika Jaiswal',

                collegeName : 'GGPL',

                branch : 'IT',

                age : 21

            }

        ]

        let tableInfo="<table border='1' cellpadding='10px'><tr><th>Name</th><th>College Name</th><th>Branch</th><th>Age</th></tr>";

        let dropDown="<select><option>---Select---</option>";

        let orderList="<ol>";

        let unorderList="<ul>";

        for(let x of studentInfo){

            tableInfo+="<tr><td>"+x.name+"</td><td>"+x.collegeName+"</td><td>"+x.branch+"</td><td>"+x.age+"</td></tr>";

            dropDown+="<option"+x.name+"</option>";

            orderList+="<li>"+x.collegeName+"</li>";

            unorderList+="<li>"+x.branch+"</li>";

        }

        tableInfo+="</table>";

        dropDown+="</select>";

        orderList+="</ol>";

        unorderList+="</ul>";

        document.querySelector('p').innerHTML=tableInfo+dropDown+orderList+unorderList;

        }

    </script>

</body>

</html>

**Onsubmit :** Form Validation

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Onsubmit</title>

    <style>

        body{

            margin: 0;

            padding: 0;

            min-height: 100vh;

            width: 100%;

            display: flex;

            justify-content: center;

            align-items: center;

            background: #552d8a;

            color: #fff;

        }

        \*{

            font-family: sans-serif;

        }

        form{

            background: rgba(255,255,255,0.08);

            backdrop-filter: blur(10px);

            padding: 50px;

            width: 250px;

            box-shadow:  20px 20px 60px #4c2199,

             -20px -20px 60px #662dcf;

        }

        input{

            padding: 7px 5px;

            border: none;

            border-radius: 4px;

            background: rgba(255,255,255,0.08);

            outline: none;

            border: 1px solid rgba(255,255,255,0.09);

            box-shadow: 0 0 10px rgba(0,0,0,0.2);

            width: 100%;

        }

        ::placeholder{

            color: #333;

        }

        input[type="submit"]{

            background: #1072d3;

            color: #fff;

        }

    </style>

</head>

<body>

    <form action="" method="" onsubmit="return validation()" name="frm">

    <h2>Password Validation</h2>

    <input type="password" name="pass" id="pass" placeholder="Enter Your Password">

     <br><br>

    <input type="password" name="cpass" id="cpass" placeholder="Re-Type Your Password">

    <br><br>

    <span id="msg"></span>

    <br><br>

    <input type="submit" value="CHECK">

    </form>

    <script>

        const validation=()=>{

            let pass=document.forms['frm']['pass'];

            password=pass.value;

            let cpass=document.forms['frm']['cpass'];

            confirmPassword=cpass.value;

            let msg=document.querySelector('#msg');

            if(password!=confirmPassword){

                pass.style.border="1px solid red";

                cpass.style.border="1px solid red";

                msg.innerHTML="Password and Confirm Password Didn't Matched...!";

                msg.style.color="red";

                return false;

            }

            else if(password=='' && confirmPassword==''){

                msg.innerHTML="Both Fields are Required...!";

                msg.style.color="yellow";

                return false;

            }

            else if((!isNaN(password)) && (!isNaN(confirmPassword))){

                msg.innerHTML="Only Numbers Can't Be Consider as Password...!";

                msg.style.color="yellow";

                return false;

            }

            else if(password.length<=2 && confirmPassword.length<=2){

                msg.innerHTML="Password Length Must Be 3 or Greater than 3...!";

                msg.style.color="yellow";

                return false;

            }

            else{

                pass.style.border="1px solid skyblue";

                cpass.style.border="1px solid skyblue";

                msg.innerHTML="Correct Password...!";

                msg.style.color="skyblue";

                return false;

            }

        }

    </script>

</body>

</html>

**addEnventListener :** This function is used to add events to the selected HTML elements.

We can add more than one events at a time on selected HTML element.

**Syntax :**

objectOfHTMLElement.addEventListener(‘eventName’,’functionName());

**Example :**

Button.addEventListener(‘click’,sum());

button.addEventListener(‘click’,function(){

alert(“iAmEventListener”);

})

button.addEventListener(‘dblclick,function(){

alert(“iAmEventListener2”);

})

button.addEventListener(‘dblclick,()=>{

alert(“iAmEventListener2”);

})

**Radio Button Validation**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Form Validation - Radio Button</title>

</head>

<body>

    <form action="" method="GET" onsubmit="return validate()" name="frm">

        <p>Kindly Select Your Traning -</p>

        Summer Training : <input type="radio" name="r" id="">

        Winter Training : <input type="radio" name="r" id="">

        Vocational Training : <input type="radio" name="r" id="">

        Apprenticeship : <input type="radio" name="r" id="">

        <input type="submit" value="Save">

        <br>

        <b></b>

    </form>

    <script>

        "use strict"

        const validate=()=>{

            var radio = document.querySelectorAll('input[type="radio"]');

            var count=0;

            for(var x of radio)

            {

                if(x.checked==true){

                    count++;

                    return true;

                }

            }

            if(count==0){

                document.querySelector('b').innerHTML="Please select at least one option.";

                document.querySelector('b').style.color="red";

                return false;

            }

        }

    </script>

</body>

</html>

**‘use strict’;**

JavaScript's strict mode, introduced in ECMAScript 5, is a way to opt in to a restricted variant of JavaScript, thereby implicitly opting-out of "sloppy mode". Strict mode isn't just a subset: it intentionally has different semantics from normal code. Browsers not supporting strict mode will run strict mode code with different behavior from browsers that do, so don't rely on strict mode without feature-testing for support for the relevant aspects of strict mode. Strict mode code and non-strict mode code can coexist, so scripts can opt into strict mode incrementally.

Strict mode makes several changes to normal JavaScript semantics:

Eliminates some JavaScript silent errors by changing them to throw errors.

Fixes mistakes that make it difficult for JavaScript engines to perform optimizations: strict mode code can sometimes be made to run faster than identical code that's not strict mode.

Prohibits some syntax likely to be defined in future versions of ECMAScript.

**Rest Parameter :** The rest parameter syntax allows a function to accept an indefinite number of arguments as an array, providing a way to represent variadic functions in JavaScript.

**Example**

function sum(...theArgs) {

return theArgs.reduce((previous, current) => {

return previous + current;

});

}

console.log(sum(1, 2, 3));

// expected output: 6

console.log(sum(1, 2, 3, 4));

// expected output: 10

**Spread Operator : Spread syntax** (...) allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected.

**Example**

function sum(x, y, z) {

return x + y + z;

}

const numbers = [1, 2, 3];

console.log(sum(...numbers));

// expected output: 6

console.log(sum.apply(null, numbers));

// expected output: 6

**VERSIONS OF ECMA (European Computer Manufacturing Association) SCRIPT**

|  |  |  |
| --- | --- | --- |
| ES1 | ECMAScript 1(1997) | First Edition |
| ES2 | ECMAScript (1998) | Editorial Changes |
| ES3 | ECMAScript 3 (1999) | Added Switch and do-while |
| ES5 | ESCMAScript (2009) | String.trim(),  Array.isArray(), Array.forEach(),  Multiline Strings,  Objects,  Array of Objects,  Reserved word as property, Trailing commas. |
| ES6 | ESCMAScript (2015) | Let and const,  Arrow function,  Default parameters, Destructuring array,  Iterable objects.entries, Iterable Object.values, String.includes(), String.startsWith(), String.endsWith(),  for(of and in). |
|  | ECMAScript (2016) | Javascript Exponentiation (\*\*),  Exponentiation |
|  | ECMAScript (2017) | Javascript String padding, Javascript Object.entries,  Javascript Object.values  Etc. |
|  | ECMAScript (2018) | String interpolation or template string.  Spread operator…,  Rest parameter… |