**console.log( “hello world” ); - logs messages**

**alert( “message” ) - alert box**

**prompt(“enter email”, “default value”) - prompts user to enter value**

**//comment**  **- comments**

**/\* multi line - multi line comment**

**Comment \*/**

**Primitive types:**

**number(floating-point), string & boolean**

**Variables:**

**let name; - new versions let is used.**

**(results in an error, if tried to re-declare vars with same name)**

**var name; - older versions var is used.**

**(re-declaration of vars with same name is allowed)**

**Const - to make it final**

**Operators:**

**2\*\*3 - exponentiation (2^3 = 8)**

**(decimals also can be used) (2\*\*0.5)**

**Comparision Operators:**

**=== - for strict verification (verifies datatype and value)­­­**

**== - verifies only value**

let x = 5

let y = '5'

console.log(x == y) //true

console.log(x === y) //false

**Strings:**

**‘hi “gangadhar” ’ - hi “gangadhar”**

**\n - new line**

**\t 1 - tab**

**Template Literals:**

* **It is another way to create strings & work with them more flexibly.**
  + **They use back-ticks ` ` instead of “ ”**
  + **let name = `gangadhar`;**
* **With these we can use both single & double quotes within a string.**
* **Allows multiline strings without using \n**
  + **let text = `hello**

**world`;**

* + **output : hello \n world**
* **Allows us to use variables inside the strings using $ sign.**
  + **let name = “gangadhar”;**
  + **let text = `hi ${name}`;**

**Functions:**

**Define Function:**

function login() {

  console.log("Hi!");

}

**Function with Params:**

function login(user) {

  console.log("Hi, "+user);

}

**Function with Return type:**

console.log(add(5,6))

function add(num1, num2){

    sum=num1+num2 (can also written as **var sum=num1+num2**)

    return sum

}

output : 11

**Function can be assigned to variable:**

function add(num1, num2){

    return num1+num2

}

var sum=add

console.log(sum(6,6))

output : 12

**Anonymous Function:**

var add=function(num1,num2){

    return num1+num2

}

var result=add(4,5)

console.log(result)

output : 9

**Javascript Objects (object literals):**

var person = {

    name: "John", age: 31

};

**Functions inside object:**

var user={

    username:'itsme',

    age:21,

    display:function(){

        console.log(this.username+":"+this.age)

    }

}

user.display()

output : itsme:21

**Accessing properties:**

var x = person.age;

var y = person['age'];

// in this person[‘age’] , we can pass property\_name runtime instead of ‘age’.

var student={

    age:10,

    stud:"sam"

}

student.fees=10000

console.log(student)

output : { age: 10, stud: 'sam', fees: 10000 }

var user=new Object()

user.age=19

user.name='sam'

console.log(user)

output : { age: 19, name: 'sam' }

**Delete property:**

let alien = {

    name: 'gangadhar',

    tech: 'java'

}

delete alien.tech

console.log(alien)

output : { name: 'gangadhar' }

**To write on document:**

document.write("This is some text");

**Object Constructor:**

* used to create a number of objects of a single type.

function person(name, age, color) {

    this.name = name;

    this.age = age;

    this.favColor = color;

}

var p1 = new person("John", 42, "green");

var p2 = new person("Amy", 21, "red");

**Object Methods:**

function person(name, age) {

    this.name = name;

    this.age = age;

    this.changeName = function (name) {

        this.name = name;

    }

}

var p = new person("David", 21);

p.changeName("John");

* You can also define the function outside of the constructor function and associate it with the object.

function person(name, age) {

    this.name= name;

    this.age = age;

    this.yearOfBirth = bornYear;

}

function bornYear() {

    return 2023 - this.age;

}

var p = new person("A", 23);

**Arrays:**

var courses = new Array("HTML", "CSS", "JS");

var courses = new Array(3);

**Array Creation Methods:**

fruits=['apple','mango','banana']

fruits.push('orange')

fruits[4]='grapes'

console.log(fruits[3])

output : orange

fruits=['apple','mango']

fruits[10]='grapes'

console.log(fruits[7])

//values in positions 2-9 are defined

output : undefined

**Dynamic Array:**

var courses = new Array();

**using array literal syntax:**

var courses = ["HTML", "CSS", "JS"];

**Array functions:**

**var courselength = courses.length - returns no of elements in the array**

**var courses = arr1.concat(arr2) - it doesn’t affects the arrays**

**push(ele) - insert element into array**

**pop() - removes last element from array**

**sort() - sorts array**

**reverse() - reverses the array**

**toString() - converts array to string**

**Iterate through array:**

var fruits=['apple','mango','orange']

for(var index in fruits)

console.log(fruits[index])

output :

apple

mango

orange

**foreach:**

var fruits=['apple','mango','orange']

for(var fruit of fruits)

console.log(fruit)

output :

apple

mango

orange

**Associative Array:**

var person = []; //empty array

person["name"] = "John";

person["age"] = 46;

document.write(person["age"]);

* **Standard array methods & properties may produce wrong results, since person is object.**

**Array destructuring:**

let nums = [5, 6, 3, 4]

let [a, b, c, d] = nums

let [i, , k] = nums

console.log(c)

console.log([i, , k]);

output :

3

[ 5, <1 empty item>, 3 ]

**//EXAMPLE2**

let a, b, c = 4, d = 8;

[a, b = 6] = [2]; // a = 2, b = 6

[c, d] = [d, c]; // c = 8, d = 4

**//EXAMPLE3**

let obj = {h:100, s: true};

let {h, s} = obj;

console.log(h); // 100

console.log(s); // true

**. . . C:**

let words = 'my name is gangadhar'.split(' ')

let [a, b, ...c] = words //...c = remaining text must store in ‘c’

console.log(c)

output :

[ 'is', 'gangadhar' ]

**Array Methods:**

* **Push() - adds item to array at last index**
* **Unshift() - adds item to array at first index**
* **Pop() - deletes item from array at last index**
* **Shift() - deletes item from array at first index**
* **Splice(pos) - deletes all item from pos**
* **Splice(pos,count) - deletes count no.of item from pos**
* **Splice(pos,count,n) - deletes count no.of item from pos & adds n in its position**

**Set:**

**Create set:**

let set = new Set("madam")

console.log(set);

output : Set(3) { 'm', 'a', 'd' }

**Add Element:**

let set = new Set()

set.add(1)

set.add(4)

set.add(3)

set.add(3)

console.log(set);

output : Set(3) { 1, 4, 3 }

**Map:**

**Create map:**

let map=new Map()

map.set(1,'gangadhar')

map.set(2,'ganesh')

map.set(3,'mani')

console.log(map);

output : Map(3) { 1 => 'gangadhar', 2 => 'ganesh', 3 => 'mani' }

**iterate map:**

for([k,v] of map)

console.log(k+":"+v);

for (k of map.keys())

    console.log(k + ":" + map.get(k));

map.forEach((v, k) => console.log(k+":"+v))

//above all loops print the same following output

output :

1:gangadhar

2:ganesh

3:mani

**Map functions:**

* **Set(k,v)**
* **get()**
* **has(key)**
* **delete(key)**
* **clear()**
* **keys()**
* **values()**
* **entries() - returns iterator of array[key,value] in map for each element**

**Class, Constructor, Getters & Setters:**

class laptop {

    constructor(id, name) {

        this.\_id = id

        this.\_name = name

    }

    set id(id) {

        this.\_id = id

    }

    get id() {

        return this.\_id

    }

}

var lap=new laptop(1,'apple')

//setter

lap.id=4

//getter

console.log(lap.id);

output : 4

**Inheritance:**

* **JS supports only multi level inheritance.**

class Sum {

    constructor(n1, n2) {

        this.n1 = n1

        this.n2 = n2

    }

    get twoSum() {

        return this.n1 + this.n2

    }

}

class AdvanceSum extends Sum {

    constructor(n1, n2, n3) {

        super(n1, n2)

        this.n3 = n3

    }

    get threeSum() {

        return this.twoSum + this.n3

    }

}

let sum = new AdvanceSum(4, 5, 6)

console.log(sum.threeSum);

output : 15

**Math Object Methods:**

**abs(x) random() pow(x, y)**

**aceil(x) round(x) max(x, y, z, ….)**

**floor(x) sqrt(x) min(x, y, z, ….)**

**Date Object:**

**setInterval( methodName, 3000) - sets time interval**

**clearInterval -**

new Date(milliseconds)

new Date(dateString)

new Date(year,

         month,

         day,

         hours,

         minutes,

         seconds,

         milliseconds)

var d1 = new Date(86400000);

//Fri Jan 02 2015 10:42:00

var d2 = new Date("January 2, 2015 10:42:00");

//Sat Jun 11 1988 11:42:00

var d3 = new Date(88,5,11,11,42,0,0);

function printTime() {

    var d = new Date();

    var hours = d.getHours();

    var mins = d.getMinutes();

    var secs = d.getSeconds();

    document.body.innerHTML = hours+":"+mins+":"+secs;

}

setInterval(printTime, 1000);

**innerHTML - returns the html content of an element.**

**Date Methods:**

**getFullYear() getHours()**

**getMonth() getMinutes()**

**getDate() getSeconds()**

**getDay() getMilliSeconds()**

**ExceptionHandling:**

function add(n1, n2) {

    if (!n1 || !n2)

        throw new Error("number is undefined")

    return n1 + n2

}

try {

    var result = add(1)

    console.log(result)

} catch (error) {

    console.log("exception caught ", error) //PRINTS ENTIRE STACK TRACE

    console.log("exception caught " + error) //CALLS toString() & prints value only

}

Output :

exception caught Error: number is undefined

**Promise:**

* **“Producing code” is code that can take some time**
* **“consuming code” is code that must wait for the result**
* **Promise is a JS object that links producing & consuming code**
* **Manages a single async value that can be handled in the future**
* **Promise states, (uber example)**
  + **Pending - booked ride (undefined)**
  + **Fulfilled - dropped at location (resolved value)**
  + **Rejected - ride rejected (reason for rejection)**

//executor (callback func) -   function that resolves a value or rejects (error)

//promise starts with pending state

const ride = new Promise((resolve, reject) => {

    if (arrived)

        resolve('driver arrived') //fulfillment

    else

        reject('driver bailed') //rejected

})

ride

    .then(value => { // handles fulfillment

        console.log(value);

//driver arrived

    })

    .catch(error => { // handles rejection

        console.log(error);

//driver bailed

    })

    .finally(() => {

    })

* **Create (promise a future value)**
* **Consume (use a future value)**
* **Simple syntax,**

promise

    .then(...)

    .then(...)

    .then(...)

    .catch(...)

**Promise methods:**

* **Promise.all([promises]) - returns array of promises once all resolved**
* **Promise.allSettled([promises]) - returns array of promises, if all are resolved/rejected**
* **Promise.any([promises]) - returns promise, if any promise is resolved**
* **Promise.race([promises]) - returns promise of 1st resolved/rejected promise**

let v1 = new Promise((resolve, reject) => { resolve('video1 recorded') })

let v2 = new Promise((resolve, reject) => { resolve('video2 recorded') })

Promise.race([v1, v2])

    .then(message => {

        console.log(message)

    })

Promise.all([v1, v2])

    .then(message => {

        console.log(message)

    })

Output :

video1 recorded

[ 'video1 recorded', 'video2 recorded' ]

**Sample Code:**

let confirmation = false

let ride = new Promise((resolve, reject) => {

    if (confirmation)

        resolve('ride confirmed, captain on his way')

    else

        reject('ride is cancelled, look for other captains')

})

ride

    .then(c => {

        console.log('accepted message : ', c);

    })

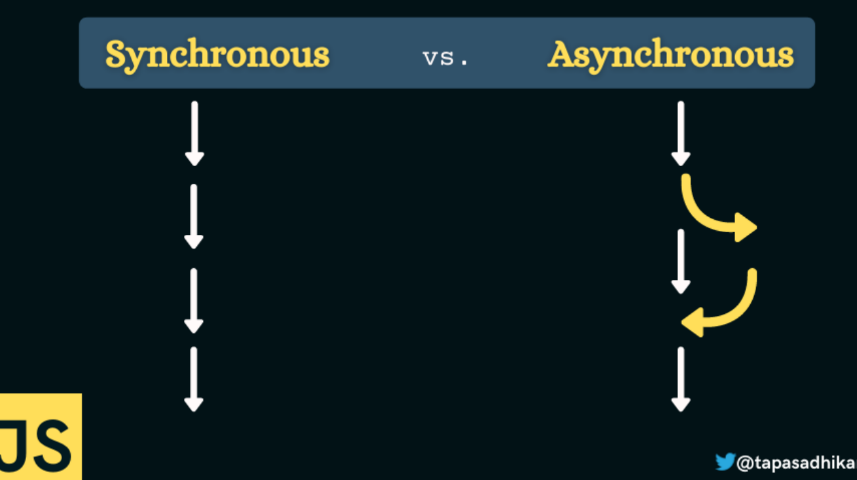
    .catch(c => {

        console.log('rejected message : ', c);

    })

Output : rejected message : ride is cancelled, look for other captains

**Synchronous & Asynchronous:**

****

**Synchronous:**

//code executes in order (i.e; 2nd line waits for 1st line to complete)

console.log('gangadhar')

console.log('ganesh')

console.log('mani')

output :

gangadhar

ganesh

mani

**Asynchronous:**

//code won’t execute in order

(i.e; 2nd line won’t wait for 1st line to complete)

console.log('gangadhar')

setTimeout(\_ => console.log('ganesh'))

console.log('mani')

output :

gangadhar

mani

ganesh

**Async & Await:**

* **Async - makes a function return a promise**
* **Await - makes a function wait for a promise & converts into result**

function bookRide(location) {

  return new Promise((resolve, reject) => {

    if (location == 'bangalore')

      resolve('captain is on your way to pickup')

    else

      reject('captains are too busy in your area')

  })

}

async function uber() {

  try {

    const response = await bookRide('bangalore')

    console.log(response);

  } catch (err) {

    console.log(err);

  }

}

uber()

output :

captain is on your way to pickup

**Callbacks:**

* **It is a function passed as an argument to another function, it can run after another function finished**

setTimeout(\_ => console.log('ganesh'), 1000)

**example:**

function step1(value, callback) {

  callback(value + 10)

}

step1(5, function (r) {

  console.log('result = ', r);

})

Output :

result = 15

**Fake rest api/json server at local:**

**Install Json server globally:**

npm install -g json-server

**Run json file:**

json-server --watch db.json

* **Finally, test the apis from postman (localhost:3000/<end-point>)**

**ES6:**

**Var is not block scoped:**

function method() {

    if (true) {

        var i = 10

        console.log("inside condition: " + i)

    }

    console.log("outside condition: " + i)

}

method()

output:

inside condition: 10

outside condition: 10

**let is block scoped:**

function method() {

    if (true) {

        let i = 10

        console.log("inside condition: " + i)

    }

    console.log("outside condition: " + i)

}

method()

output :

i is not defined

**const is block scoped:**

function method() {

    if (true) {

        const i = 10

        console.log("inside condition: " + i)

    }

    console.log("outside condition: " + i)

}

method()

output:

i is not defined

**stupid stuff in js (but imp):**

let product1={model:"nexon", price:98765}

let product2={...product1,model:""}

product2.model="nexon pro"

console.log(product1)

console.log(product2)

output:

{model: 'nexon', price: 98765}

{model: 'nexon pro', price: 98765}

**Aggregate 2/more arrays:**

let fruits=['apple','mango','orange']

let cars=['supra','mustang']

let aggregate=[...fruits,...cars,"onepiece"]

console.log(aggregate)

output:

[ 'apple', 'mango', 'orange', 'supra', 'mustang', 'onepiece' ]

**Lamda expressions:**

let add = (n1, n2) => n1+n2

//without using expression

let add = function (n1, n2) {

    return n1 + n2

}

**Sort array using comparator:**

const array = [1, 4, 3, 2]

array.sort((e1, e2) => e1 - e2)

console.log(array)

output:

[ 1, 2, 3, 4 ]

**Map in array: (similar to map in stream api)**

let array1 = [1, 2, 3, 4]

const array2 = array1.map(e => e + 1)

console.log(array2)

output:

[ 1, 2, 3, 4, 5 ]

**Reduce in array: (similar to reduce in stream api)**

let array = [1,2,3,4]

const sum=array.reduce((a,e)=>{

    return a + e

})

console.log(sum)

output:

10

**DOM (Document Object Model):**

* **The DOM represents the structure of a web page as a hierarchial tree of objects, where each object represents a part of document.**
* **Using javascript, we can manipulate the content, structure & style of web page dynamically**
* **Document is the root of all objects on webpage**

document.body.innerHTML = "Some text";

body - is element of DOM

innerHMTL - is used on almost all html elements to change its content

**Selecting Elements:**

* **document.getElementById(id)**

var elem = document.getElementById("demo");

elem.innerHTML = "Hello World!";

//it assumes html has <div id=”demo”></div>

* **document.getElementsByClassName(name)** 
  + **returns a collection of all elements in document having specified class name**
  + **if class = ”demo”, there are 3 elements with same class name, then it returns all**

var arr = document.getElementsByClassName("demo");

//accessing the second element

arr[1].innerHTML = "Hi";

* **document.getElementsByTagName(name)**
  + **returns all elements having same tag name as an array**

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

for (var x = 0; x < arr.length; x++) {

  arr[x].innerHTML = "Hi there";

}

output :

    <p>Hi there</p>

    <p>Hi there</p>

    <p>Hi there</p>

**Working with DOM:**

* **Each element in the DOM has set of properties & methods**
  + **element.childNodes - return array of element’s child nodes**
  + **element.firstChild - return first child node of element**
  + **element.lastChild - return last child node of element**
  + **element.hasChildNodes - return true if element has child nodes else false**
  + **element.nextSibling - return next node at the same tree level**
  + **element.previousSibling - return previous node at the same tree level**
  + **element.parentNode - return parent node of an element**

**Changing Attributes:**

* **we can change the content of an element using innerHTML property.**
* **Similarly, we can change attributes of elemets**
  + **We can change src attribute of an image.**

<img id="myimg" src="orange.png" alt="" />

<script>

var el = document.getElementById("myimg")

el.src = "apple.png"

</script>

//calling the function in window.onload to make sure the HTML is loaded

window.onload = function() {

    var el = document.getElementsByTagName("a")

    el[0].href= "http://www.youtube.com"

}

* **Style of html elements can also be changed using JS.**
* **All style attributes can be accessed using style object of element**
  + **Background-color is referred as backgroundColor**

//calling the function in window.onload to make sure the HTML is loaded

window.onload = function() {

    var x = document.getElementById("demo")

    x.style.color = '#6600FF'

    x.style.width = '100px'

}

**Adding or Removing [[[[[[[……….Elements:**

**Creating Elements:**

* **element.cloneNode() - clones an element & returns resulting node**
* **document.createElement(element) - creates a new element node**
* **document.createTextNode(text) - creates a new text node**
  + var node = document.createTextNode("Some new text")
  + **above code,creates a new text node, but it won’t appear in document until we append it to an existing element.**
  + **It can be done using the following methods**
    - **element.appendChild(newNode) - adds a new child node to an element as the last child node**
    - **element.insertuBefore(node1, node2) - inserts node1 as a child before node2**

//calling the function in window.onload to make sure the HTML is loaded

window.onload = function() {

    //creating a new paragraph

    var p = document.createElement("p");

    var node = document.createTextNode("Some new text");

    //adding the text to the paragraph

    p.appendChild(node);

    var div = document.getElementById("demo");

    //adding the paragraph to the div

    div.appendChild(p);

};

// this creates a new paragraph & adds it to the exising div element on the page

**Removing Elements:**

* **select the parent of element & use removeChild(node) method**

<div id="demo">

    <p id="p1">This is a paragraph.</p>

    <p id="p2">This is another paragraph.</p>

  </div>

  <script>

  var parent = document.getElementById("demo");

  var child = document.getElementById("p1");

  parent.removeChild(child);

  </script>

//this removes the paragraph with id=”p1” from page

* **Alternative way is,**

<div id="demo">

    <p id="p1">This is a paragraph.</p>

    <p id="p2">This is another paragraph.</p>

</div>

<script>

    var child = document.getElementById("p1");

    child.parentNode.removeChild(child);

</script>

//parentNode property is used to get the parent of element

**Removing Elements:**

* **Element.replaceChild(newNode, oldNode) method is used**

<div id="demo">

    <p id="p1">This is a paragraph.</p>

    <p id="p2">This is another paragraph.</p>

  </div>

  <script>

  var p = document.createElement("p");

  var node = document.createTextNode("This is new");

  p.appendChild(node);

  var parent = document.getElementById("demo");

  var child = document.getElementById("p1");

  parent.replaceChild(p, child);

  </script>

**Creating Animations:**

* **To create an animation, we need to change the properties of an element at small intervals of time. It can be achieved using setInterval(), which allows us to create a timer & call a function to change properties repeatedly at defined intervals (in ms)**

var t = setInterval(move, 500)

var pos = 0

//our box element

var box = document.getElementById("box")

function move() {

  pos += 1

  box.style.left = pos+"px"

}

* **To stop the animation, when the box reaches the end of container, use clearInterval()**

function move() {

    if(pos >= 150)

     clearInterval(t)

    else {

      pos += 1;

      box.style.left = pos+"px";

    }

  }

**Handling Events:**

* **We can write js code that executes when an event occurs such as click on html element, moves the mouse, submits form**
* **When an event occurs on target element, a handler function is executed.**
* **Common HTML events**
  + **onclick - when user clicks on element**
  + **onload - when object is loaded**
  + **onunload - occurs once, when page is unloaded (for <body>)**
  + **onmouseover - when pointer is moved onto an element/its children**
  + **onmouseout - when pointer is moved out of element/its children**
  + **onmouseup - when user releases a mouse button over an element**
  + **onblur - when element loses focus**
  + **onfocus - when element gets focus**
  + **onchange - when content/selection/,checked state changes**

**(<input>, <keygen>, <select>, <textarea>)**

Corresponding events can be added to HTML elements as attributes.

<p onclick="someFunc()">some text</p>

**Onclick:**

<button id="demo">Click Me</button>

//example1

//calling the function in window.onload to make sure the HTML is loaded

window.onload = function() {

    var x = document.getElementById('demo');

    x.onclick = function () {

        document.body.innerHTML = Date();

    }

};

//example2

<button onclick="show()">Click Me</button>

function show() {

    alert("Hi there");

}

**Onload:**

<body onload="doSomething()"></body>

window.onload= function() {

    //some code

}

**Onchange:**

* **It is mostly used on textboxes. It gets called when text inside textbox changes & focus is lost from the element**

<input type="text" id="name" onchange="change()">

function change() {

    var x = document.getElementById('name');

    x.value = x.value.toUpperCase();

}

**Event Listeners:**

* **addEventListener() attaches an event handler to an element without overwriting existing event handlers. We can add multiple or same type of event handlers to same element (i.e., two ‘click'** **events)**
* **removeEventListener() removes the event listener**
* element.addEventListener(event, function, useCapture);
* element.removeEventListener(event, function)
* event - eventType (mousedown, click) (neglect ‘on’ before event)
* function - method to call when event occurs
* useCapture(optnal) - boolean value, we should specify event bubbling/
* event capturing

//example

<button id="demo">Start</button>

//calling the function in window.onload to make sure the HTML is loaded

window.onload = function() {

    var btn = document.getElementById("demo");

    btn.addEventListener("click", myFunction);

    function myFunction() {

        alert(Math.random());

        btn.removeEventListener("click", myFunction);

    }

};

**Event Propagation:**

* **It allows for the definition of the element order when an event occurs.**
* **If <p> is inside <div> & the user clicks on <p>, which element’s ‘click’ event should be handled first is defined by event propagation**
* **2 types,**
  + **Bubbling - innermost element event handled first then outermost**
  + **Capturing - outermost element event handled first then innermost**
* **Capturing goes down the DOM**
* **Bubbling goes up the DOM**
* **It is helpful when we have same event handled for multiple elements in DOM hierarchy**

//Capturing propagation

elem1.addEventListener("click", myFunction, true)

//Bubbling propagation

elem2.addEventListener("click", myFunction, false)

**Image Slider:**

* **Changing images using next & prev**

<div>

<button> Prev </button>

<img src=" 1.jpg" width="200px" height="100px"/>

<button> Next </button>

</div>

var images = [

    'http://www.sololearn.com/uploads/slider/1.jpg',

    'http://www.sololearn.com/uploads/slider/2.jpg',

    'http://www.sololearn.com/uploads/slider/3.jpg'

];

var num = 0;

function next() {

    var slider = document.getElementById('slider');

    num++;

    if(num >= images.length) {

        num = 0;

    }

    slider.src = images[num];

}

function prev() {

    var slider = document.getElementById('slider');

    num--;

    if(num < 0) {

        num = images.length-1;

    }

    slider.src = images[num];

}

**Event Propagation:**

* **Form element has onsubmit event that can be handled to perform validation**

<form onsubmit="return validate()" method="post">

    Number: <input type="text" name="num1" id="num1" /><br />

    Repeat: <input type="text" name="num2" id="num2" /><br />

    <input type="submit" value="Submit" />

</form>

function validate() {

    var n1 = document.getElementById('num1')

    var n2 = document.getElementById('num2')

    if(n1.value != '' && n2.value != '') {

        if(n1.value == n2.value) {

            return true;

        }

    }

    alert("The values should be equal and not blank")

    return false;

}

**ECMAScript6:**

* **Supports classes, modules, iterators, for/of loops, generators, arrow functions, binary data, typed arrays, collections(maps, sets & weak maps), promises, number & math enhancements, reflection & proxies**
* **Let - block scope**
* **Var - global/local scope**
* **Const - block scope**
* **Use for...of loop for array iteration**
* **Use for...in loop for key:value pair of object**
* //default parameters
* const test = (a, b = 3, c = 42) => {
* return a + b + c;
* }
* console.log(test(5)); //50
* **Object.assign({},person,student) - combines two objects into one/copy object**

let basic = {

    ex1: 'PushUps: 20 times',

    ex2: 'Jumps: 20 times'

};

let advanced = {

    ex3: 'Squats: 30 times',

    ex4: 'Run: 2km'

};

let total=Object.assign({},basic,advanced)

for(let ex in total) {

    console.log(total[ex])

};

* **Rest,**

function magic(...nums) {}

* **Spread,**

let nums = [3, 4, 5];

let all = [1, 2, ...nums, 6];

console.log(all[3]);

* **Classes,**

//example1

var Square = class {

    constructor(height, width) {

     this.height = height;

     this.width = width;

   }

 };

//example2

 var Square = class Rectangle {

    constructor(height, width) {

      this.height = height;

      this.width = width;

    }

  };

**//example3**

class Rectangle {

    constructor(height, width) {

      this.height = height;

      this.width = width;

    }

    //getter

    get area() {}

    //setter

    set area(height){}

    //method

    calcArea() {}

    static fun(){}

  }

  const square = new Rectangle(5, 5);

  console.log(square.area); // 25

**Built-in-Methods:**

* **Array element finding**
  + - **find( x=>x>3 )**
    - **findIndex( x=>x>3 )**
  + **Repeating strings**
    - **join(“foo”)**
    - **repeat(3)**
  + **searching strings**
    - **indexOf(“solo”)**
    - **indexOf(“solo”,2)**
    - **startsWith(“solo”,0)**
    - **endsWith(“solo”,0)**
    - **includes(“solo”,2)**

**NOTES:**

* **<script> should be at botttom of the page, because, before running the script, we should load the page**
* **let is not subject to variable hosting, which means that let declarations do not move to the top of current execution context**