Reference links

|  |  |
| --- | --- |
| Clean documentation for all concepts, even for data types | <https://devdocs.io/javascript/> |
| Mozilla docs | <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects> |
| Inbuilt objects | <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects> |
|  |  |

About java script

What is js

Its scripting language means, if any issue came in line 20, all previous lines will be executed happily

All browsers will have java script engine, chrome has v8 js engine, and hence browser can execute js code

Current js version is ES6

Why js

Its for action purpose – onclick(), auto scroll, to make page dynamic then use js

Notes / syntaxes

Element means both tag and content ex:- “<h2 id=”dd” class=”bb” style =”cc”> ok na </h3>”

Here id, class, style are the attributes of a Element

1. You can directly Write js code in a .js file , no need to keep js code in any tag, u can include that file as

    <script src="1.ArraysOperations.JS"></script>

1. You can place the script inside body tag also, just to ensure the script to be executed after loading the full page And note:- u can place js code only inside the script tag

<body>

<script>

// Your script code here—why because if u want to run anything after loading the elements that’s why we will keep ele here

</script>

</body>

1. No need of keeping semicolon at end of each line like in java
2. Using braces to give preference to particular operation, just enclose in braces
3. Inside double quotation we can use only single quotes

document.write(“sum value is ”+ (10+20)) if u don’t enclose in braces then o/p will be “sum value is 1020”

when it sees braces, it will give highest preference to that braces

how js works is, it will see the operator + it will see what is in left and right, it will do that operation in left

1. Always use back ticks ``, so that u can write ${} dollar exp or template literals inside ``

We can use back tick ~ , double quote “ “ or single quote ‘ ‘ or - Best is `` instead of double quotes use `` backtick so that u can write ${} template literals also

If u write anything in dollar then that expression will be evaluated, ex:- ${env} if u write in “ ” it will be printed as it is in browser

Note:- we can write this literals or expressions in double quotes hence prefer to use back tick so that we can use template literal as below

Ex:- u can even create a variable with back tick also

|  |  |
| --- | --- |
| function disableAdBlocker(arg) {  //Here this $ expression will be evaluated    console.log(`${arg} method exec successfully`);  }  //This function will be exec after 5 sec from page loading  window.setTimeout(disableAdBlocker, 5000, "AdBlocker");  //either in console.log() or in document.write() we can use backtick | var s1="orini ayya"  var s2=`abba re ----->${s1} <br>`  document.write(`abba re -->${s1} <br>`) // this value will be printed by replacing the s1 in java we use %d here we are using ${var}  output In browser  abba re -->orini ayya |

6.1) we can use dollar inside backticks ~~double or single quotes~~- ${} means fetch something from

Even inside a string

Let x= `manideep ${lastname}` – here we don’t even need to use + operator

|  |  |  |  |
| --- | --- | --- | --- |
| Proof for ${} wont work inside single / double quotes | |  |  | | --- | --- | | let flagPath =”https://flagcdn.com/w320/us.png”;  log("path", flagPath);  let html = "<img src= ${flagPath} >";  log("html tag", html); | See here step 2 whatever is there in double quotes its considered as a full string, it didn’t replaced the dollar | |

1. Back slash is the escape character for double quotes

console.log("hello \"world\"")

Output:- hello "world"

1. Passing function as an argument to another function

Ex:- in java if any function has “functionalInterface” as a parameter, then for that we can pass method reference

Whereas in here, we can simply pass function name as an argument

1. If u want all html code which is written then, go to browser console and type (window.document) you will get full html code and script
2. Asynchronous programming is js is achieved through few functions –

 **Asynchronous Operations:** Use asynchronous techniques like Promises or async/await to handle long-running tasks without blocking the main thread.

 **Web Workers:** For computationally intensive tasks, consider using Web Workers to offload the work to a separate thread.

#### Java vs JavaScript

|  |  |
| --- | --- |
| Javascript | Java |
| We don’t have classes in JavaScript, we only have objects  javaScript is single threaded, pausing the main thread would block entire app  Parent object is object | We have both classes and objects in java  Parent object is java.lang.Object |
| If function is having 3 parameter,   1. while calling u can pass either 10 arguments / u can pass 1 argument also No compilation/no run-time error   ex:- if u didn’t pass it will consider as undefined value   1. In function signature🡪  * We don’t mention a return type in function signature * while mentioning the params we don’t specify the type of each and every parameter   function callbackF3Reduce(acc, ele, index, arr) {    console.log(`exec callback function.`);    if (acc > ele) {      return acc;    } else {      return ele;    }  }  function callbackF3Reduce(acc, ele, index, arr) {  //Here in above there is no type specified infront of each parameter   1. U can pass a function name as an argument to another function 2. U can define a function inside another function   function createCounter() {  let count = 0;  return function() {  count++;  return count;  };  }   1. U can return a function from another function – using js currying concept | 1. If function is expecting 2 params, u should pass only 2 params else u will get compilation error 2. in java every function have a return type and every parameter has some type 3. Here u can pass the method reference as an argument to a function interface only, So that target method reference method will be executed   U can even write lambda expressions for a method reference   1. U can’t define a function inside another function |
| We can get parent obj ref from child, using \_\_proto\_\_ property | In java we can not get parent obj ref using child |
| Even after object creation at runtime we can add properties to an object  emp1.sal=2000// here a new property called sal will be created and added | u can’t define those properties at runtime  u should define them all in class itself |
| To print all properties of object, by default we can use for-in loop on objects  let u1 = new user("punch", "india", "Mid range");  for (key in u1) {    document.write(key + " : " + u1[key] + "<br>");  } | We can’t use for-each loop on individual object  We can loop only on collections/Arrays |
| Same like java super keyword, in js also we have \_ \_proto\_ \_ property in every object  Which can give parent obj ref  e2.\_\_proto\_\_.age=90 // modifying parent class obj ref using child ref | Here with super keyword we can call super class constructor |

Beauty of JavaScript

(what is there in js & not there In java)

1. Window.setTimeout(function to be exec, after how much time it should be exec, arg to that function)

In js, we can define a function to be exec after certain time, but in java we cant do that, what we can do is we can just submit a pool

1. Easy REST api invocation

|  |  |
| --- | --- |
| Within 2 lines u can happily hit a rest api and get the data – in java also just ½ lines only | async function invokeRestAPI() {          //fetch generally returns a promise, await will wait for promise to resolve          let p = await window.fetch("https://jsonplaceholder.typicode.com/todos/1");          p.json().then((data) => {            console.log(data);          });          console.log(p);        }        invokeRestAPI(); |
| Like in java, to calculate the time gap between 2 lines, in java we cal using  System.currentTimeinMillis(), and then we will minus 2 times and find ms  In js  Console.time()  Console.timeEnd() | Promise is the beauty of js  Once promise execution is success (like REST invocation is success)  Then() method will be called automatically (like to save records in db) |

Similarities

|  |  |
| --- | --- |
| Java script | JavaScript |
| Here also we have classes and objects, (classes are only since ES6)  Ex:- Set is a class (but we don’t know in which package is this)  Here also we have try,catch,finally blocks and throw keyword  throw “x must not be zero”  Our js object can have both properties and methods like in java | Even js classes also can have –global variables, Constructor, super() to call super class constructor ,setter methods, getter methods, normal methods,  But there are small diff is – in js we should define cons with constructor keyword  2) setter method should have set pre fixed keyword, same as getter  Public/ private access modifiers are not allowed for js getters and setters/for var  Here also we have inheritance - one class can extend another class – all the parent methods will be inherited into child  Child class can access parent methods but parent cant access child methods  One class can extend multiple classes |
| Here also we have static and instance methods under concept called  \_\_proto\_\_ && prototype | Same like super keyword, \_\_proto\_\_ keyword property gives us parent object reference, like super keyword in java |
| In js also separate object means it have separate data  Here also we have == operator which can compare hash codes to check if 2 var pointing to same object or not   let e1 = {          name: "mani",          age: 31,        };  let e3=e1 //here we are assigning e1 ref into e3, now e1,e3 pointing to same obj  e3==e1  result: - true | Arrow functions in js == lambda expressions in java  completableFuture… in java == Promise.then().catch() in js almost similar to java |

Terminologies

Callback function – means that function will be called by javascript our responsibility is just to provide that function, JS framework will call that function

3 ways to print

1. Alert()
2. document.write(num or any variable this will be passed to browser + “**write html code in double quotes** so that browser will understand”)

let companyName='Tcs'

document.write(` <br> ${companyName} profits in this year is ${profits} <br>`) –here ${} is template literal, we should write template literal in `` back ticks only

write html code inside backticks like in spring double quotes lo dollar “${} ”

let trains = new Array(`krishna`, `vandeBharath`, `pinakini`);

for (const element of trains) {

  document.write(`${element}  <br> `);

}

See Here we write html code inside backticks

document.write(“and here we should write html code, if u write anything inside string it will print as it is”)

|  |  |
| --- | --- |
| We can write `` backtick inside console.log also  Console.log(`vammov ${star name}`) | document.write(str, "-->", s, " <br>");  we can pass any number of argu to write method- all will be printed   function p(str, s) {          document.write(str, "-->", s, " <br>");        }   p("includes", s.includes("avs"));  result  includes-->false |

1. Innerhtml
2. Console.log(2) this will print some data to console

3 modal windows

|  |  |
| --- | --- |
| Window.Alert()- to show some message to user |  |
| Val x=window.Prompt(“Any Message”, ‘default value’)– to take some input from user  By default this will take as a string | let x = prompt("Bro enter a number", "30");  document.write(`Entered value is ${x}`); |
| Boolean window.confirm(“Message to print)-  to get confirmation from user  Confirm will just take yes or not | let sure = confirm(`Are you sure you have entered ${x}`);  if (sure) {    document.write(`You have finalized the number as ${x} <br>`);  } else {    document.write(`You will get another chance to enter `);  } |

Ways to accept values from browser

1. Using prompt()

var n1= +prompt("enter 1st number", ”default value here”)

Variables

1. ways to create variables and

all js variables are case sensitive

var nam=”n1;

var nam =”n2; both these variables are different

java script variables are loosely typed- means we don’t need to mention the type, we can just say var x=”s”;

1. var – which was invented in old versions , with this u can define multiple times

The variables let, const are like java variables, which cant be used before declaring

1. let - new versions
2. const – new versions- these are like final variables u can’t reassign values to that, & it is mandatory to initialise

|  |  |  |
| --- | --- | --- |
| Var (available since starting) | let (Block scope variable- not present in window obj-only in new versions this is like a java variable) | const ( only in new versions) |
| 1. Duplicate declaration is accepted,   var x=10  var x=10 Here we defined 2 times still it is accepted | 1. Duplicate declarations are not allowed   let num1=10  let num1=20 //Duplicate declaration of same variable is Not allowed - as already a var is defied with same name  This duplicate is wrong and not possible in let variables (possible only in var variables) |  |
| We can re-assign | We can re-assign | This is a final variable, we can’t re-assign |
| We can use this var before declaration | We **cannot use** this var before declaration | We cant use this variable before declaration |
|  | Block scope:- let variables have only block scope  a let variable declared within a script tag does not become a property of the window object.  <script>  let myVariable = "Hello";  console.log(window.myVariable);  // Output: undefined  </script>  **Window Object:** The window object represents the global scope in a browser environment. Variables declared with let are not hoisted to the global scope, so they are not added to the window object. |  |

Data types

Main – Number (here all integer, number, float all comes under number only), string, Boolean

Others – undefined (if u don’t pass data to that parameter then it is called un-defined), null

Js is dynamically typed language, means u don’t need to specify type explicitly means Integer, String, Double no need to mention these specific types,

just mention var, it will automatically detect the type

Primitive – number (all integer, float, double comes under number), string, Boolean

To check the type of incoming object we should use (typeof(x) operator same like instance of x in java) operator

var x= 23.34;

var x=10; console.log(typeof x) // this will tell us the type of variable ex:- number

Special – undefined, null

Composite- Array, object

|  |  |
| --- | --- |
| We can define strings in Double quotes, single quotes, backticks also | var x="2"; console.log(x);  var x='1'; console.log(x);  var x=`2`; console.log(x); |
|  |  |
|  |  |
| Number | let x =Number("456") |
| Big integer – its like either append with n or write using constructor | var q= 123n; console.log(q); |
|  | var q= BigInt(123n); console.log(q); |
|  |  |
| Undefined  This data type is for un assigned values | By default if u don’t assign js will auto initialise with undefined  Var x; // by default its undefined  Var x= undefined – not recommended use null over this |
|  |  |
| Null | Var x=null; |
|  |  |
| Array | To create array we have to use square braces  //Storing simple types In array  var names = ['one','two','three',null,undefined, true]; console.log(names);  the above is a heterogeneous array  // Storing objects in array    var obj1= {'name': "manideep",'company': "tcs", employeecount : 60000}      var obj2 = {"name":"santhoshi", "company":"IQA", employeecount : 200}     console.log(obj1)     console.log(obj2)     var objTypeArray= [obj1, obj2] |
|  |  |
| Declare an object | In curly braces |

Operators

1. Arithmetic operators +, -,\*,/,%(reminder operator)
2. Binary operators – these will operate on 2 operands ex:- digits ex:- a+b
3. Unary operators – these will operate on single operand ex:- digit 🡪 ++a or a++
4. Logical operators – a&&b (to check if both conditions are true or not), a||b (any condition can be true) , !a

var a=2;

  var b=200;

//These are pre increment or pre decrement operator

document.write(++a) // first it will increment and print

  document.write(--a) // first it will decrement and print

// post increment or decrement operator a++ or a--

Double equal to vs triple equal to

|  |  |
| --- | --- |
| == double equal means it will compare only data not data type  var a=2;    var b=200;      var c= "2";      //Here even though a & c data types are diff (content is same) it just compares content      document.write(a +"=="+c)      document.write(a==c ) | === this triple equal to compares both data and data type // This is called strictly equal to operator  document.write("<br> triple equal to demo")      document.write(a +"==="+c)      //This will check both content and data type      document.write(a===c ) // here since data types are diff this will return false |
| Spread operator – Triple dot  To deep copy anything like – copy arrays, copy object  Let x= [1,2,3]  Let y= […x,2] //means y= [1,2,3,2] it will copy old ele & 2 | limitation is it cant copy nested object |

Spread operator is to convert a an array into list of elements

Here Math.min(elements) take elements, it doesn’t take an array,

Let a= [1,2,3,4,5]

Math.min(…a)//

Math.min(1,2,3,4,5)

but when we have array, since we can’t pass array we should use spread operator to convert into list

1. Assignment operators

|  |  |  |
| --- | --- | --- |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x/y |
| %= | x %= y | x = x % y |
| \*\* | x\*\*= y | x = x \*\* y |
|  |  |  |
|  |  |  |
| &= | x &= y | x = x & y |
| ^= | x^= y | x = x^y |
| |= |  | x = x | y |
| Bit wise operator | | |
| >>= | x >>= y | x = x >> y signed right shift operator  >>> zero fill right shift |
| <<= | x <<= y X | x = x << y zero fill left shift operator |
| & |  | 5 &1 bitwise and operator |
| | |  | 5 | 1 bitwise or operator |
| ~ |  | ~5 not operator |
| ^ |  | ^ xor operator |

6) Ternary operator

  var a=2;

  var b=200;

//   Ternary operator demo

var ternary= a>b? a:b;

document.write("big value is ")

document.write(ternary)

Destructuring

|  |  |
| --- | --- |
| Destructuring array | Destructuring object |
|  | let emp = {          name: "mani santu",          age: 30,          city: {            cityName: "hyd",            stateCode: "TN",          },  };  // Destructuring object now object keys will be mapped to individual variables   * + To create object flower braces will be right side , to destruct flower braces will be left side   + But here the rule is obj prop name and left side var name must be same         // Destructuring object now object keys will be mapped to individual variables        //while destructuring obj prop name and left side var name must be same        let { name, age,city: { cityName, stateCode } } = emp;        console.log(name1, age, cityName, stateCode); |

Type conversion

To check type we should use “typeof” keyword

document.write("type of var "+n1 +"is" +(typeof n1))

document.write("type of var "+n1 +"is" +(typeof(n1)))

|  |  |
| --- | --- |
| String to number – just we need to keep + infront of it | var n1= +prompt("enter 1st number")    var n2=  prompt("enter 2nd number") //Here we didn’t use + so it will be string    var operation = prompt("enter the operator among + or - / or %")    document.write("type of var "+n1 +"is" +(typeof n1))    document.write("type of var "+n2 +"is" +(typeof n2))  output will be  “type of var 12isnumber type of var 2isstring” |
| Any data type 🡪 Number  Number(“any string”) | let x =Number("456")  document.write(x +" is "+ (typeof x) +" <br>") // 456 is number |
| String to number data type (to int value)  parseInt(“any String”)  it takes only the integer part not points and all | let y= parseInt("45F") // This is self intelligent and it will extract 45 num  document.write(y +" is "+ (typeof y)) //45  let z= parseInt("45.02")  document.write(z +" is "+ (typeof z)) //45 |
| String to number (int)  parseFloat(“any float value”)  output will contain points but still data type is float only | let a= parseFloat("11.02")  document.write(a +" is "+ (typeof a) +"<br> ") //11.02 |
| Any data type 🡪 String | //Converting Number to string  b= String(b)  document.write(b +" is "+ (typeof b) +"<br> ")  //Converting boolean to string  b= String(true)  document.write(b +" is "+ (typeof b) +"<br> ") |
| Any data type 🡪 Boolean  Any thing other than zero is true  For zero its is false | let c =0  b= Boolean(c)  document.write(c +" is "+ (typeof b) +" -->"+b+ "<br> ") //false  c= -120  b= Boolean(c)  document.write(c +" is "+ (typeof b) +" -->"+b+ "<br> ")  //true  c= 240  b= Boolean(c)  document.write(c +" is "+ (typeof b) +" -->"+b+ "<br> ") //true  c=1.0  b= Boolean(c)  document.write(c +" is "+ (typeof b) +" -->"+b+ "<br> ") //true  c=`23F`  b= Boolean(c)  document.write(c +" is "+ (typeof b) +" -->"+b+ "<br> ") //true |
|  |  |

Switch For If- else conditional Statements

|  |  |
| --- | --- |
| Basic **if else** | var num =prompt("enter a positive number")   if (num<0) {      alert("Bro kindly enter positive values only...")   } else {      document.write(" <h1> <br> Thanks u have enterered a positive number -->"+ num +"</h1>")   } |
| Basic **while** – until a condition is false  If u want to execute every time based on a condition | Until u give a positive number this will give alerts keep on  while(num<0) {      alert("Bro kindly enter positive values only...")      num =prompt("enter a positive number")   } |
| If u want to perform a task minimum 1 time max n number of times- then use **do-while**  do {  }  While(condition) | do {          var num=Number(prompt("enter a num to sum"))          sum =sum +num          if(x==0){              x=5;          }          --x;          document.write("<br> entered number is "+ num )      } while (x>0); |
| If  Else if  Else if  …  …  Else  Here also this is possible | let marks=prompt("enter ur marks i will tell d grade");    if(marks>90){     marks="A";    }    else if(marks>80){     marks="B";    }    else if(marks>70){     marks="c";    }    else {     marks="D";    }    document.write(" <h1> <br>your grade is  -->"+ marks +"</h1>") |
| **Switch case**  Break statement is mandatory, else all cases will be executed eventhough all cases have a condition, so keep break statement  Switch(expression) {  Case choice1:  run some code  break;  Case choice2:  run some code:  break;  } | var n1= +prompt("enter 1st number")    var n2=  prompt("enter 2nd number")    var operation = prompt("enter the operator among + or - / or %")    document.write("type of var "+n1 +"is" +(typeof n1))    document.write("type of var "+n2 +"is" +(typeof n2))    switch(operation){     case '+':        alert("The sum value is "+ (n1+n2))        break;     case '-':        alert("The diff is "+ (n1-n2))        break;     case '\*':        alert("the multipled val is "+(n1\*n2))        break ;     default :        alert("The reminder value is "+(n1%n2));    }   document.write(" <br> type of var " + n1 + "is" + typeof n1);  /output  type of var 1isnumber type of var 2isstring type of var 1isnumber |
| **For loop** | //Basic for loop  for(let i=0;i<companies.length;i++){      document.write("<br>"+companies[i])  } |
| **For in** loop – **in** means **index** – means when u use this loop u will get index only  Fact:- if I want index then I can directly use normal for loop na, why should I come and use this?   * U can use for-in loop on arrays – here u will get index 0,1,2.. * U can use for-in loop on object – then u will get keys/property/fields names | //for in loop -- in means index  document.write("<br><br> printing using for in loop <br>")  // this keyword index can be any name  for (index in companies) {      document.write("<br>"+companies[index])  }  //using For-in loop on object  let u1 = new user("punch", "india", "Mid range");  for (key in u1) {    document.write(key + " : " + u1[key] + "<br>");  }  carName : punch ManufacturerCountry : india Range : Mid range |
| **For of** loop – this is like **for each** loop In java = in each iteration we will get object  When u use this loop u will directly get object  Both these **for in & for of** loop will operate only on **iterable** objects like collection and arrays not on objects, because object is not an iterable it just contain properties & methods  Ex:- in js set class is iterable, so directly u can use for-of loop on it | //for of ---- for each loop like in java  document.write("<br> printing using for of loop <br>")  // this keyword element can be any name  for (const element of companies) {      document.write("<br>"+element)  } |
| **Printing object properties using for in loop** | var employee = {"name":"manideep","company":"tata", "vehicle":"bike"}  document.write("printing obj properties using for in loop")  for (key in employee) {      document.write("<br>"+key+"-->"+employee[key])  } |
| **Printing string characters using for in loop** | // printing ele in string using for in loop  let s="wells fargo"  document.write("printing string elements using for in loop")  for(index in s){  document.write("<br> "+s[index])  } |

Template literal

Template literals means using $

$ - means fetch something from – this will be evaluated only inside backticks `` {this wont work inside double or single quotes}

If u want to use template literal then we have to enclose that $ string in back tick `

1. No need using \n 🡪 In backticks instead of \n, we can directly click enter so that it will print into new line

Ex:-

|  |  |
| --- | --- |
| //Demo for template literal  let profits = '110000cr'  let companyName='Tcs'  //Here this ${val} is called a template literal  //if u are using this literals we have to enclose this string in back tick  document.write(` <br> ${companyName} profits in this year is ${profits} <br>`)  output: Tcs profits in this year is 110000cr | var s1="orini ayya"  var s2=`abba re ----->${s1} <br>`  document.write(s2)  output:  abba re ----->orini ayya |
|  |  |

Suggestions

1. Best to use `` backticks instead of double quotes, because inside backticks we can write expressions
2. It is always best to associate an event to a element

|  |  |
| --- | --- |
| Don’t use        var fnEle = document.getElementById("fn");        fnEle.addEventListener("mouseenter", autofiller); | Attaching event listener  Here onclick is the event listener      <button id="submit" onclick="autofiller()">Submit</button>  This is easily readable- bec attaching some function somewhere else may not be readable |

Arrays

Array is an object in js

Most of the methods takes a callback function as argument like - find( callback function),

Callback function is a function that will be called back by the JavaScript engine

Advantages: - has best framework support like it has many built in methods

//Well the callback function signature is fixed – always 1st param is element, 2nd is index, 3rd is entire array

Ex:- array.map(f1) – this map function takes a 3 parameterized callback function as an argument and that function will be called by js engine

for every element present in Array

array destructuring let [x] =arry; //If arry contains 1 element then that 1 ele will come into x var

|  |  |
| --- | --- |
| Methods in JavaScript | |
| push(‘ele’)- to add ele at last  unshift(‘ele’) – add at 1st – means it wont do shift operations |  |
| 1. Delete at last - Pop()   Delete at first – shift()  Splice () – is same for both addition , deletion at any point  Splice (index number to add/delete, how many ele to delete , new ele) |  |
| Methods which takes callback function  Array map(callback function f1)  Array filter(callback function f)  = reduce (function f, accumulator default value)  value find( 3-param callback function which returns Boolean )  Boolean Some( 3-param callback function which returns boolean)  Boolean every(3-param callback function which returns Boolean) |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Declaring array | var trains=['vande bharath', 'vande sadaran','garibrath','jan shatabdi']   let buses= new Array('airavath','ambari','amaravathi','morning star') |
| Printing array using for in & for of loops | for(index in trains){      document.write(`${trains[index]} <br>`)  }  document.write(`<br>buses list is  <br> <br>`)  for (bus of buses) {      document.write(`${bus} <br>`)  } |
| Adding an ele to array at last index | //Adding element at last index,  //we have size of 4 if we store in 100th index instead of 5th index, all values from 5-100th position will be stored as null          buses[100]=`vakveri travels`          for (bus of buses) {              document.write(`${bus} <br>`)          }  Output  Bus1, bus2, bus3,undefined\*95 times, vkaveri |
| Fetching ele from nth index | //storing in 100th index  buses[100]=`vakveri travels`  //fetching from 100th index  document.write(` <br> element in 99 th index is ${buses[100]}`) |
| storing heterogeneous ele |  |
| Array methods like  Adding at last - push(‘ele’)- to add ele at last  Adding at first - unshift(‘ele’) – add at 1st – means it wont do shift operations  Adding at specific position – splice (add at which index, how many el ,ele )  This push(ele) is like list.add() in our java  Method name instead of unshift() – they should have renamed as pushAtFirst()  Since add at 1st is frequent operation, if shift happens for every frequent operation it is costly so they would have designed in such a way of re-creating by copying all ele into new array instead of shifting  And since deleting at not so frequent- they designed that delete operation as shift  Main point to note is  ==================  Splice () – is same for both addition , deletion at any point  Splice (index number to add/delete, how many ele to delete , new ele)  trains.splice(2, 0, "Dasara special ");   * Here bec of middle 0 – no ele will be deleted – and a ele will be added   trains.splice(2, 1, "Dasara special ");   * Bec of this 1 ele will be deleted and given ele will be added   trains.splice(1, 2)   * Here because of middle val 2 – then 2 ele will be deleted from 1st index | var trains=['vande bharath', 'vande sadaran','garibrath','jan shatabdi']  let buses= new Array('airavath','ambari','amaravathi','morning star')  //push(ele) method to add element at last  trains.push('pinakini')  trains.push('krishna express')  document.write(` <br>Trains after adding at last <br>`)  for(index in trains){      document.write(` --  ${trains[index]}  --     `)  }  // unshift() to add element at 1st  trains.unshift('Grand trunk')  for(index in trains){      document.write(` --  ${trains[index]}  --     `)  }  // Adding at specific position  //Adding at specific position  document.write(`----Adding at specific position- <br>`);  trains.splice(2, 0, "Dasara special ");  document.write(`${trains} <br> `);  Early morning train ,krishna,intercity,pinakini,sunday special train ----Adding at specific position- Early morning train ,krishna,Dasara special ,intercity,pinakini,sunday special train |
| Replacing an ele – bec of this 1st ele is replaced with given element | //replacing at any position  let trains = new Array(`krishna`, `vandeBharath`, `pinakini`);  document.write(`----Replacing at any position- <br>`);  trains[1] = "intercity";  document.write(`${trains}  <br> `);  output  ----Replacing at any position- krishna,intercity,pinakini |
| Deleting and adding at 1st- use shift() & pop operations   1. Delete at last - Pop() 2. Delete at first – shift() – may be shift operations might happen after deleting at 1st 3. Delete at specific position- splice(index to delete, how many elements to delete) 4. Element Pop- to delete ele at last index, it will delete and it will gives us the deleted element –   Pop()- delete last ele, shift()- delete first element | Delete at last - Pop()  ===================  let deletedTrain = trains.pop()// this will delete element at last & print  document.write(` <br> Deleted train is --> ${deletedTrain}  <br>`)  document.write(` <br>Trains after deleting at last <br>`)  //pop() to delete ele at last  for(index in trains){      document.write(` --  ${trains[index]}  --     `)  }  document.write(` <br>Trains after adding element at 1st <br>`)  Delete at first – shift()  =================================  //shift() method to delete ele at 1st  deletedTrain =trains.shift()  document.write(`<br> deleted train is --> ${deletedTrain}  <br>Trains after deleting element at 1st is <br>`)  for(index in trains){      document.write(` --  ${trains[index]}  --     `)  }  Delete at specific position  ============================  // Here from index 1 we want to delete 2 elements  document.write(trains.splice(1, 2) + `<br>`);  document.write(`${trains}`); |
| Looping over array ele using forEach() method  Syntax:- <your array object>.forEach(<pass the function name here>)  Here u are passing function name as an argument to other function  ForEach() expects a 3 parameterized function means we should pass a function which accepts 3 parameters | // ---- arrays .forEach() method iteration  document.write(` <br> arrays .forEach() method iteration`);  var ar1 = new Array(40, 50, 60, 70, 80);  //forEach(fun) this forEach method can accept a function with 3 parameters  // This function can accept 3 parameters ele,index,full array these are fixed  function myIterator(ele, ind, arry) {    let hike = ele / 10;    //here we are    ar1[ind] = ele + hike;  }  document.write(`<br> Array content before modifying ${ar1}`);  //HERE forEach is a function which accepts another function as an argument  ar1.forEach(myIterator);  document.write(`<br> Array content AFTER modifying ${ar1}`); |
| Spread operator :- this will spread the elements  The spread operator (...) can be used to create a new array with the elements of an existing array and additional elements.   * This spread operator is even applicable for strings to convert into array * Used to convert an array into individual elements      * This we can use to merge 2 arrays – means each array will be converted to individual elements and those ele will be placed In array | const myArray = [1, 2, 3];  const newArray = [...myArray, 4, 5];// if u don’t give spread operator, in 1st index array will be there, in 2nd,3rd indexes individual elements will be there  console.log(newArray); // Output: [1, 2, 3, 4, 5]  --with spread operator (), without spread operator()  //Merging 2 arrays with and without spread operator  var a1= [1,2,5,4] var a2= [1,22,5,4]  var a3= […a1,any number of elements, …aa2]  var a4=[a1,a2] |
| let index= Movies.indexOf(ele)- it will give index of that element  let arr= Movies.slice(1,4)- means it will print ele in index 1,2,3//This will give another array | let thirdElement = trains[3]  document.write(`<br> 3rd ele is --> ${thirdElement}`)  let splittedArr= trains.slice(0,2);  document.write(`splitted array is --> ${splittedArr}`) |
| Array map(function f1)  Map() – will just take return ele as is   1. We can pass a 3 parameterized callback function to a map & filter   Sample function to pass  function f(ele,index,arr){  }   1. This map and filter methods itself will create a new array and keep content in that array and returns us   This map method will be called for each and every element in the array | const arr = [1, 2, 3, 4, 5];  function callbackf1(ele, index, arr) {    console.log(typeof ele);    return ele \* ele;  }  //Here this map is talking this callback function as an  //argument that function should take and return a value  let arrPower = arr.map(callbackf1);  console.log("map function output is ", arrPower);  result  (5) [1, 4, 9, 16, 25] |
| Array filter(function f)  // This filter method callback function will be called for each and every element present in array  if the return condition value is true then only that element will be collected  for filter method   1. we should pass a function that returns Boolean value 2. That function should take 1 or 3 arguments   Boolean function f(ele ,index ,arr){  } | //filter(callback) method demo- which takes callback function as argument  // that callback function takes 3 params and it should return boolean  function callbackf2(ele, index, arr) {    return ele % 2 == 0;  }  //Here map is talking the function as arg  console.log(`filter method op`, arr.filter(callbackf2)); |
| accumulator final value = reduce (function f, accumulator default value)  When to use reduce- if u want to reduce a big array to single /few element then use reduce() – any operation like – find(), min(), max(),   * This reduce method will take a callback function and a default value as argument * That callback function should take 4 parameters as arguments along with acc * That function should return some value that it is called accumulator value- this value only will be returned finally as reduce method output * Main diff between map () vs reduce() is 🡪 map takes only array ele ,but reduce() * takes both array ele, previous operation accumulator output | //reduce method demo - which takes callback function & accumulator initial value  //callback function takes 4 params  function callbackF3Reduce(acc, ele, index, arr) {    console.log(`exec callback function.`);    if (acc > ele) {      return acc;    } else {      return ele;    }  }  console.log(`reduce method o/p`, arr.reduce(callbackF3Reduce, arr[0]));  output:  reduce method o/p 5 |
| value find( 3-param callback function which returns Boolean )   * This callback function will be called for every element in array * n that callback function returns true, iteration will be stopped even if there are multiple ele in the array * This method will return an ele which satisfies the condition | //find()- takes a 3 param callback function that function returns boolean value  let train = trains.find(callbackf4find);  function callbackf4find(ele, index, arr) {    console.log(`ele type is --> ${typeof ele}, excec callback function `);    var eleStr = new String(ele);    return eleStr.startsWith("V");  } |
| Boolean Some( 3-param callback function which returns boolean)  This method tells whether some of the ele in array are satisfying the condition or not?  The difference between some and find methods is – find() returns the value  Some() returns Boolean value | //some(3 param callback function) - that function should return true  function callbackf5Some(a, b, c) {    console.log(a);    if (a > 0) {      return true;    }  }  console.log("some() result-->", arr.find(callbackf5Some)); |
| Boolean every(3-param callback function which returns Boolean)  If every element in the array is satisfying the condition, then only that every() method returns true,  Ex:- if every sub is passed then only issue certificate | function callback6Every(ele, index, arr) {    console.log(`executing Array.every() callback method`);    if (ele > 0) {      return true;    }  }  //every(3 param callback function)  //that 3 param callback function will return true when all ele satisfies the condition  console.log(`array.every() result is -->`, arr.every(callback6Every)); |
|  |  |

Functions

Note

Callback function is a function that will be called back by java script engine after sometime

Prototype is related to function name not instance

\_\_proto\_\_ (underscore means its assoc with object)

|  |  |
| --- | --- |
| function functionName(var1 , n1,num2…){  return ..;  } | Here function is the keyword  Infront of variable name no need to give the type of variable  It can have return type , / it can be void method also  function sum(a,b){  }  If u want to exec function sum() – here flower braces are mandatory  If u keep flower braces- then func will be executed immediately |

In JS,

1. We can call a function
2. And we can even create objects for a function

In JavaScript, functions are first-class objects, meaning they can be assigned to variables, passed as arguments to other functions, and returned 1 from functions. This flexibility 2 allows you to create objects that contain functions as properties

|  |  |
| --- | --- |
| function emp() {          console.log("called a function ");   } | 1. Generally we call a function as “” emp(); 2. In JS we can even create objects for a function   let x = new emp(); |
| **Constructor function**  In js we can 1) write a function inside another function (called instance methods)   1. write global variables   //The below is called a constructor function – this is used before ES6 classes introduction to js, In modern JavaScript, classes are generally preferred over constructor functions for several reasons:    function movie(name, director) {          console.log(this);          //declaring global variable even if moview doesn’t have any variable called name- since we are using this.name a name variable will be created and populated          this.name = name;          this.director = director;          //declaring instance method – this is assigning anonym fun to global var          this.getDetails = function () {            console.log("details are ", name, director);          };          console.log("fun executed");        }        //Declaring a static method associated with class movie  //prototype property is assoc with class        movie.prototype.collectionDetails = function () {          console.log("static method fun  collectionDetails executed");        };        let m = new movie("RRR", "rajamouli");  Note:- this constructor functions are used before ES6, in ES6 they introduced classes in JS |  |
| function Person(name) {  this.name = name;  }  //similar to static method  Person.prototype.greet = function() {  console.log("Hello, my name is " + this.name);  };  const john = new Person("John Doe");  console.log(john.greet()); // Output: Hello, my name is John Doe | In this example:   1. Person.prototype.greet defines a greet method on the Person prototype. 2. john is an instance of the Person object. 3. When john.greet() is called, JavaScript first checks if john has a greet property. Since it doesn't, it checks john.\_\_proto\_\_ (which points to the Person prototype). 4. The greet method is found on the prototype, and it's executed with john as the this context. |

|  |  |
| --- | --- |
| function sum(num1,num2){  } | To call function  Sum(10,20) |
| **Calling a named function**  Inside java script file  U can directly call other functions like  Console.log()  Document.write() | //All the below code is from a javascript file  function sum(v1, v2) {    return v1 + v2;  }  console.log(sum(10, 20)); //Here even console.log is also a named function  so here we defined that function and we need to call the function then only it will be executed  let n1 = 100;  let n2 = 20;  var result = sum(n1, n2);  document.write(`sum of ${n1} + ${n2} numbers is ${result} `); |
| **Type of function is – function** | function callback6Every(ele, index, arr) {    console.log(`executing Array.every() callback method`);    if (ele > 0) {      return true;    }  }  //every(3 param callback function)  //that 3 param callback function will return true when all ele satisfies the condition  console.log(`array.every() result is -->`, arr.every(callback6Every));  console.log(typeof callback6Every);  //Result is function |
| **Passing function as argument to other fun/**  **How to execute /call a function** | console.log(`array.every() result is -->`, arr.every(callback6Every));  here for a method named “every()” we are passing the function – while passing we should not give braces  because if u give braces- function will be executed then and there |
| One function can happily **return another function**  **Inner function** can access outer function values | function p1(v1) {          console.log(`executed p1 fun received value ${v1}`);          return function p2(v2) {            console.log(`executed p2 fun  received value ${v2}`);            //Here this inner function can access outer fun variables v2            return v1 + v2;          };        }        console.log(`sum using currying fun  p1(5)(6)--> ${p1(5)(6)}`);   * + Here p1() function is returning another function – this process is also called currying |

Anonymous functions

Anonymous function is a nameless function, but u will assign that function to a variable, u can call that function with that variable name

I don’t know what is the advantage, instead of calling using function name, we called using variable name

Here we defined a function without name, but we have called that function using variable name

|  |  |
| --- | --- |
| //Anonymous function – with this we  var f1 = function (v1, v2) {    return v2 - v1;  };  var result2 = f1(n1, n2);  document.write(`<br> diff of ${n1} , ${n2} numbers is ${result2} `); | Result:  diff of 100 , 20 numbers is -80 |

If u want to use function only for single them the use this anonymous function

IIFE- immediately invoked function expressions

No-reusability functions, If u want to use function only once, then only prefer this, bec these functions are not-reusable

If u want to execute that function immediately then prefer IIFE

Syntax :- (Function definition) ();

(Function definition) (arg1, arg2);

Here in 1st parenthesis we will write function, next parenthesis is to call that function

|  |  |
| --- | --- |
| //Immediately invoked functions  var r3 = (function multiply(v1, v2) {    return v1 \* v2;  })(20, 40);  document.write(`<br> IIFE function result is ${r3}`); | Result  IIFE function result is 800 |

Arrow functions

These functions are since ES6

//arrow functions

//Here as this is a single line either keep both {} and return or remove both statements

|  |  |
| --- | --- |
| var f4 = (v1, v2) => v1 \* v2;  r4 = f4(5, 42);  document.write(`<br> Lambda function output is ${r4} `);  output  Lambda function output is 210  When no arguments are there either use () parenthesis or use underscore  ()=>  If single param is there then no need of ()parenthesis , we can directly use var name  Let square = a => a\*a  If multiple statements are present then only we need to use flower braces  Let square = a => {  console.log(a\*a) ;  return a\*a;  }  If single line is there- then no need of return statement also  If u want to keep, then keep both  flower braces  return statement | General function  function multiply(v1, v2) {    return v1 \* v2;  } |

These are exactly same like java lambda expressions

Here we

1) NO need to mention the function keyword

2) NO need to mention the function name, instead assign it to variable and use that variable to call that function

Fact:- I still don’t know instead of giving name to function and using why are we assigning to a variable and using variable, what’s the problem with using the function name directly

Syntax:

When u are using this no need of using function keyword, and no need of using function name

(param1, param2 ) => {write lines of code here}

Same like java if there is only single line then no need of return statement and no need of flower braces

If u have multiple statement then write all of them in flower braces

Either keep both {}, return statement or remove both {} and return statement

For single line, {} is not at all required ,{} is required only when multiple statements are there

\_

|  |  |
| --- | --- |
| Using parenthesis  When arguments are no there u can just use () | Using underscore  When arguments are not there u can just use \_ also instead of () |
| //Arrow functions for no arg functions  f11 = () => {    document.write(" <br> NO arg function with  () paranthesis");    return 1891;  };  result = f11();  document.write(`arrow functions with no arg using paran ${result}`); | //Here we are writing arrow function with doesnt take any arguments, so we kept underscore  var y = (\_) => {    return 711;  };  let y1 = y();  document.write(` <br> result is ${y1} `);  result  result is 711 |

Variable scopes

1. Variables defined inside a function are local to that function- u can’t use those variables outside of that function

Same like in java, here also local variables of a function are visible only inside that function

, if global and local variable are of same name, then Inside method local variable values= will get high priority

1. Same like java instance blocks, in js also we have blocks, block level variables are visible only inside the block

|  |  |
| --- | --- |
| Global scope | Variables defined outside of a function or outside of block have global scope accessible from any where within java script code |
| Function scope | Variables defined inside function has only function scope |
| Block scope |  |
| Note:-  Avoid using global variables , instead use function scope and block scope variables in js | |

|  |  |
| --- | --- |
| Accessing var type variable | Accessing let type variable |
| Defining “var” type variables in a block as global variables  {    //If u define a var type variable inside a block then those are    // considered as global variables    var x = 900;  }  //Immediately invoked function  (function abcd() {    console.log("Bro");    document.write(`<br> IIFE function result is .. ${x}`);  })();  Result  IIFE function result is .. 900  Function  If u define a var type variable in blocks, they are considered as global variable ,u can happily use them in other functions  Defining in a function  --------------  Inside any function if you declare a variable of below types all those are considered as a local variables, you can’t use outside of that function | Block scope :- ES6+   |  |  | | --- | --- | | Variables declared using “let”, ”const” in a block ( {}, if, else, while)–  Have a block scope means they are visible only inside that block  {    //If u define a let/const type variable inside a block then those are    // NOT considered as global variables, those are block level var only    //those are treated as local variables only    let y = 900;    const z = 100;  }  //Immediately invoked function  (function abcdef() {    //These will not print bec these 2 were considered as local var    document.write(`<br> IIFE function result is .. ${y} ${z}`);  })(); |  | |
|  |  |

Default parameters

If u don’t pass any value, default value will get stored

// Default parameters demo

//with this syntax u can assign the default values for the parameters

function multiply(n1, n2 = 10) {

  document.write(`<br> values received are ${n1} , ${n2}`);

}

//see here even though we passed 1 parameter we could see the second value

multiply(2);

output:

values received are 2 , 10

Rest parameters

This is like java var arg variable, this parameter can store many number of arguments

Syntax : … <any variable name>

We should just use triple dot in front of any variable name

//Demo for rest parameters == var args in java

function varArgsDemo(n1, ...n2) {

  document.write(` <br> values received in var args method is `);

  //to print all values we are using for of loop

  for (ele of n2) {

    document.write(`${ele}  ,`);

  }

}

varArgsDemo(2, 3, 4, 5, 9, 5, 6, 1, 2);

Arguments object

By default, inside every function, we have an object called “arguments”, this variable will have all the arguments which we passed to that method

//------Demo for arguments object inside a method

document.write(`<br> ------Demo for arguments object inside a method <br>`);

function argObjDemo() {

  document.write(`<br> all ele inside argument object are  <br>`);

  for (const element of arguments) {

    document.write(` ${element},  `);

  }

  document.write(`<br> total args length is -->`, arguments.length);

  document.write(` <br> second argument value is --> ${arguments[1]}`);

}

argObjDemo(10, 20, 30);

Here we are passing 3 values but the function doesn’t have any parameter to catch ,

|  |  |
| --- | --- |
| but still **all the argument** are stored in a separate variable called **arguments**, | In java if u do like this we will get compilation error immediately  U should pass same number of arguments for those params  No separate variable that holds all args in java |

Passing function name as argument

We can pass function name as an argument to anther function, by just passing function name alone without braces

If u want to exec a function then “braces ()” is mandatory,

|  |  |
| --- | --- |
| function helloTeller(name) {    document.write(`hello ${name}`);  }  function GoodMrngPrinter(name) {    document.write(`GoodMrng ${name}`);  }  //This function for 1st para u should pass function name  function callFunction(funName, Msg) {    //we are just calling another function    funName(Msg);  }  //Here we are passing function as an argument  callFunction(helloTeller, "bro");  callFunction(GoodMrngPrinter, "bro"); | //callback function demo  function add(e1, e2) {    console.log("add() function called");    return e1 + e2;  }  function multiply(e1, e2) {    console.log("multiply() function called");    return e1 \* e2;  }  //calling multiply function, for calculator function we are passing multiply as argument  calculator(5, 6, multiply);  function calculator(e1, e2, operation) {  //here we are calling that function with name stored in operation variable      operation(e1, e2);  } |
| <script>        //this is an arrow function- as this is a single line - no return keyword and no braces        let incrementer = (a) => a + 1;        //Here we are passing the incrementer fun as arg to another function        invoker(incrementer);        //Here this invoker fun takes any function as arg it will just invoke the passed function        function invoker(anyF) {          let x = anyF(5);          console.log("invoking a function and got ret as ", x);        }      </script> |  |

Taking function name from Prompt () & exec that function

--

//callback function demo

function add(e1, e2) {

  console.log("add() function called");

  return e1 + e2;

}

function multiply(e1, e2) {

  console.log("multiply() function called");

  return e1 \* e2;

}

//calling multiply function, for calculator function we are passing multiply as argument

// calculator(5, 6, multiply);

**//here if we are talking function name from prompt if we pass add, then add method will be called** ..with help of eval function

var fname = prompt("enter function name");

//here if u pass add, then add method will be called ..

*eval*(fname + "()");

Higher order function

Higher order function is a function

1. Which takes function as an argument
2. Or which returns function as an argument
3. Or Both

Array.map(function f) – in this map function takes another function as an argument, so here map is the higher order function

Closures-Obj state

Ex

In JavaScript we can happily write a function inside another function

A Closure is an inner function which has access **to outer function variable** even after that outer function execution is completed

* Closures are often used to create private variables and functions – alternate for global variables
* Actually if we want private variables we can write inside function, for global variables we can write outside function
* But if u want global private variables then write inside an outer function- this outer function should be exec once that will return inner function

Tip:- instead of working with closures why cant we use the global variables concept,

My view- just to avoid declaring global variables, we are declaring inside an outer method and executing inner method

* And outer method will be executed only once – so that variables declared inside that function also created once for invocation
* And since we are returning only inner method ref, every time we are calling inner method directly

|  |  |
| --- | --- |
| //Closures in java script - a closure is an inner function which has access to outer variable        //even after the outer function exec is completed      <button id="b1" onclick="innerFunc()">Clicked count is :0</button>        <script>        //Closures in java script - a closure is an inner function which has access to outer variable        //even after the outer function exec is completed        function outer() {          let count = 0;          let dummyCount = 0;          ++dummyCount;          document.write(`<br> outer function got executed ${dummyCount}`);          let f1 = function inner() {            //See here this closure or inner function is able to access outer vars even after outer exec is over            ++count;            console.log(`inner function has been executed`);            document.getElementById("b1").innerHTML = `Clicked count is ${count}`;          };          return f1;        }        let innerFunc = outer();        innerFunc.dummyCount = 100;        let innerFunc2 = outer();        innerFunc2.dummyCount = 500;        //Here outer() got exec 2 times so 2 objs are created each obj global var is diff        // and change in content of 1 object doesnt impact other obj global var data        document.write(`<br> obj-1 global var data is ${innerFunc.dummyCount}`);        document.write(`<br> obj-1 global var data is ${innerFunc2.dummyCount}`);      </script> | result  Clicked count is 2 outer function got executed 1 outer function got executed 1 obj-1 global var data is 100 obj-1 global var data is 500 |

Imp notes on function

Same like java, writing a function is not enough, to execute that function we should call that function which we wrote

If u want to execute a function then we need to place () after that name

We can assign a function to a variable

1. Named function (a function who has name), Anonymous functions (function who doesn’t have name)
2. Immediately invoked functions – here functions are not –reused – if u want no re-usability then prefer this – (function definition) ();
3. Higher order functions – here we are passing one function to another function like we pass a method reference to a lambda expression

Like in java, here also we can call a function from another function, like how we call console.log()

|  |  |
| --- | --- |
|  |  |

1. We can write/define a function inside another function like a class inside an another class [because in some cases outer funciton is treated as object & obj can have methods right in same way- same way function can have methods inside it]
2. Calling a function is similar to executing an object, like when u call a function all variables defined in that object will be created and stored in heap
3. In js, function is different and method is different –

|  |  |  |
| --- | --- | --- |
| Feature | Function | Method |
| Association | Independent –not tied to any object  Will be tied to js window object  function greet(name) {  console.log("Hello, " + name + "!");  }  // Calling the function  greet("Bob"); | All methods Belongs to an object  Functions defined in a object is called an method   |  |  | | --- | --- | | const person = {  firstName: "Alice",  fullName: function() {  return this.firstName + " " + this.lastName;  }  }; | let emp = {          name: "mani",          age: 40,          salary: 40000,  //here getDetails is a method bec this fun is defined in object          getDetails: function (){            }        }; |   // Calling the method  console.log(person.fullName()); |
| Declaration | function keyword or expression | Defined within an object |
| Invocation | Direct call | Called on an object |
| Purpose | General-purpose tasks | Object-specific actions |
|  |  |  |

|  |  |
| --- | --- |
| function outer() {          let count = 1;          let outerCount = 0;          ++outerCount;          document.write(`outer function got executed ${outerCount}`);          let f1 = function inner() {            console.log(`inner function has been executed`);            document.getElementById("b1").innerHTML = `Clicked count is ${count}`;            ++count;          };          return f1;        }        let innerFunc = outer();        let innerFunc2 = outer(); | Here when u called outer function 2 times, its like creating object 2 times  Each object state can be different (both can have diff state/diff global var values)  Changes in global variables content of 1 object doesn’t impact the global variables content of other object  Because 2 objects are different |
| Declaring and using variables  //the below is called immediately invocable functions        (function outer() {          console.log("x val is", x);          function inner() {            let x = 10;          }          inner();        })();  Here u will get error, bec u declared a variable inside a fun, but trying to access it outside a fun, inner fun can use outer variables, but reverse is not possible  Same like child can use parents asset, but parent can’t use child assets |  |

1. There are many ways to call a function

Using braces

Using call(), bind(),apply()

Syntax:- functionName.call(thisArg,arguments to function)

1. We can pass function as an argument to another function,

* We can pass arrow fun as an arg to another func
* We can pass anonymous function as arg to another fucnction

Ex:-

|  |  |
| --- | --- |
| then(f){  f();  } | pr.then(function (data) {            log("Exec pr.then() as promise is success ", data);  });  Here for then method we are passing an anonymous function, internally if we see the left side code, that passed function will be executed by () parenthesis  So in JS we can pass a function as argument to other function |
| A Fun can Return another function – called currying | A function can happily return another function  function a(){  //see here fun a returns function b as return type  return function b(){  }  }  A()() // let r=a(), r1= r()  Here 2 braces because a() exec returns a function, again we need to invoke that function right hence 2 braces are required |

|  |  |
| --- | --- |
| Pure function | IMPURE function |
| A function is said to be pure only   1. It should not have any side effects 🡪If that fun is not modifying any global var 2. If that is not modifying any args passed to that function   & It should not change the browser state also ex:- console.log() inside fun   1. Fun should be Predictable 🡪 - means how many times we call that function should return same value for same arguments | let n1 = 10;        function impure(a, b) {          //This is called impure fun bec its modifying the outside variables and modifying args passed to that func          n1 = 4;          ++a;          //Here for same args this function will return diff values - so this is not predictable          return a \* b \* Math.random();        }        console.log(impure(5, 6));        console.log(impure(5, 6));        console.log(impure(5, 6)); |

Currying

Currying means process of converting one function to multiple functions and **each function should return another function as return type**

|  |  |
| --- | --- |
| function sum(a, b) {          return a + b;        } | //Here this is called currying, as one function returns another function        function p1(v1) {          console.log(`executed p1 fun received value ${v1}`);          return function p2(v2) {            console.log(`executed p2 fun  received value ${v2}`);            //Here this inner function can access outer fun variables            return v1 + v2;          };        }    console.log(`sum using currying fun  p1(5)(6)--> ${p1(5)(6)}`);  p1(5)(6)  result  executed p1 fun received value 5  executed p2 fun received value 6 |
|  |  |

Pass by value pass by ref

If u pass primitive var to a function then it called pass by value

When u pass array or any other object, then it is pass by reference, means address/reference will be passed , not original object

|  |  |
| --- | --- |
| Passing primitive values – it will be pass by value only  let x = 10;        function f1(val) {          val = val + 1;          console.log(`value inside function ${val}`);        }        console.log(`value before calling to function ${x}`);        f1(x);        console.log(`value after calling to function ${x}`); | let emp = {          name: "mani",          age: 40,          salary: 40000,        };        function increaseSal(e) {          e.salary = 50000;          console.log(`sal after increasing ${e.salary}`);        }        console.log(`value before calling to function ${emp.salary}`);        increaseSal(emp);        console.log(`value after calling a function ${emp.salary}`);  Result  value before calling to function 40000  sal after increasing 50000  value after calling a function 50000  //Here when we passed object, it will internally pass the obj reference |
|  |  |

Memory management

Javascript also have stack, heap memory area

Heap stores – objects, functions, arrays, String

All left side variables are just pointers

|  |  |
| --- | --- |
|  | Here both rabbit, anotherRabbit reference variables are pointing to same object    If any object is un-reachable from root, that object will be garbage collected |
| Same like java, here also we have young and old generation    Gargbage collector will perform on young generation if any object survived then it will be moved to old generation  Mark—sweep-compact algorithm  Mark the un-reacheable objects from root sweep the object from memory and  Compact the space – all the gaps will be compacted to form a continous memory location |  |

Java script code will be executed by java script engine

Execution context is nothing but some memory is allocated to executed the code

A new memory component is created – when we load the file and – when we call a function (separate memory block will be allocated during function execution- so that local var declared inside function will be allocated in that memory )

Always Whenever execution context is created it has 2 components

– Memory component (all variables will be stored here)

- Code component

And it has 2 phases –

|  |  |
| --- | --- |
| Creation phase- – it will initialise all the variables with “undefined” as value | Execution phase - here it will execute all lines – like variable declaration |
|  |  |
| Function execution context – separate memory allocation for function execution |  |

Even js also has stack area- all method calls will be stored here

What js engine will do?

|  |  |
| --- | --- |
| It will create call stack and execution context memory area  Like how hyd city needs all dept. – electricity, municipal water dept., roads...  Js runtime environment has many components   1. engine, 2. event loop 3. web api – DOM api, timer api, fetch api 4. callback queue 5. micro task queue 6. memory heap |  |

Event loop

Event loop is a program which runs continuously until a page is closed /application is terminated to transfer the tasks from queue to callstack()

This is responsible for async programming and its one of the component of JS runtime env

ex:- all window.setTimeout(), fetch() - these tasks will be submitted to queue

|  |  |
| --- | --- |
|  | Micro task queue – high priority tasks like fetch() function will be stored in this q  Task/ Macro task queue – all low priority tasks will be stored in this queue, like window.setTimeout(fun())  Here event loop is responsible to check call stack, if call stack is empty then all the tasks present in the queue will be put in call stack , so that engine will pick those tasks & execute  So finally, js is single threaded all the async tasks also will be executed by callstack only |

Keywords

|  |  |
| --- | --- |
| What is strict mode | Strict mode detects whether u are following syntaxes correctly or not   * Declaring a variable without var /let/const keyword – if u are using variable without declaring it will warn |
| <script>        "use strict" | This String “use strict” can be placed either globally or inside a block or function   * If u keep globally, then strict mode is applicable to full code * If u keep inside method /function , then it is applicable inside that method  |  |  | | --- | --- | | <script>        "use strict";        x = 10;        console.log(`x value is ${x} `, this);      </script> | See here as we didn’t defined the type js engine throwing error | |

The behaviour of this keyword differs on “how the function is called“ & its “context”

* outside a function
* inside a regular function /global function
* inside a method
* inside event handler
* Inside a function in strict mode

1. Value of this keyword – outside a function

|  |  |
| --- | --- |
| <script>   var aaaa = 10;        function abcd() {          console.log(`abcd`);          console.log(this);        }        abcd();        console.log(window.aaaa);      </script> | Here both the variable “aaaa” and method “abcd” since both are inside script tag so they will be associated with window object |

1. Value of this keyword – inside a regular function /global function

Inside normal function also this – represents window object

|  |  |
| --- | --- |
| Global function – without strict mode  --Here in below code strict mode is commented   <script>        //   "use strict";        let x = 10;        console.log(`Generally outside this means `, this);        function abcd() {          console.log(` value of this inside a global function -->`, this);        }        abcd();  This represents window object as strict mode is commented  result  value of this inside a global function --> Window {window: Window, self: Window, document: document, name: '', location: Location, …} | Global function – with strict mode   <script>        "use strict";        let x = 10;        console.log(`Generally outside this means `, this);        function abcd() {          console.log(` value of this inside a global function -->`, this);        }        abcd();  Undefined  Result  value of this inside a global function --> undefined |
|  |  |

Ex:- console.log(this) – will represents window object

|  |  |
| --- | --- |
| * + Value of this keyword – inside a method of an object   Then at that time console.log(this) will give that object , not window object | var obj1 = {          y: 10,          getName: function f1() {            console.log(`executing lines inside object methods`);            console.log(              `value of this key word inside a object's function is ${this}`,              this            );          },        };        obj1.getName();      </script>  Output  executing lines inside object methods  2.1.This keyWord demo.HTML:23 value of this key word inside a object's function is [object Object] {y: 10, getName: ƒ} |

1. Value of this keyword – inside event handler – here it represents that particular element

But when I practiced I am getting window object

|  |  |
| --- | --- |
| Value of this keyword inside a external event handler : it will be that particular element  last name: <input type="text" id="tb2" value="." />   document.getElementById("tb2").addEventListener("click", function () {          console.log("clicked........");          console.log(this);        });      </script>  Result  clicked........  <input type=​"text" id=​"tb2" value=​"." fdprocessedid=​"2vwdue">​ | Value of this keyword inside an inline event handler : it will be the window object      first name: <input type="text" id="tb1" value="..." onclick="f2()" /> <br />    function f2() {          console.log(`executing a function trigged by event listener`);          var ele = document.getElementById("tb1");          ele.setAttribute("value", "Manideep");          console.log(            `value of this key word inside a function triggered by event listener is  is ${this}`          );          console.log(this);        }  Result  value of this key word inside a function triggered by event listener is is [object Window]  Window {window: Window, self: Window, document: document, name: '', location: Location, …} |
|  |  |

\_ \_proto\_ \_ is a inbuilt property related to every object (this property is added to all obj by js)

new String().\_\_proto\_\_==String.prototype //true

here prototype function is also an object

Object prototype()

Methods

Call () apply () bind()

call (), apply() , bind() are crucial methods to manipulate this keyword are versatile tools for function invocation

With these 3 methods we can change “this” keyword pointing to our passed object instead of default pointing to “window” object

Both call() & apply() methods will execute the target function by replacing the this keyword with passed object

|  |  |
| --- | --- |
| TargetFunctionName.call(thisArg, arg1, arg2, ...) | TargetFunctionName.apply(thisArg,[ arg1, arg2, ...]) |
| Both call(), apply() are same they are different only in signatures | For apply method we will pass the arguments in array |

the call() method in JavaScript is used to invoke a particular function with specified this value and arguments

that function instead of considering this as window / some other object, it will consider this as passed arguments

Note:- QQ – instead of passing this argument,

why cant we pass that main object, inside the function instead of using “this” object why can’t we use the passed parameter name ?

any ways we are sending one argument which to be considered as this argument, then why can’t the receiving method use that passed variable instead of using this ?

**Replacing the this keyword value in global functions**

Function bind(thisArg,arg1,arg2)

bind will not call the function, instead it will return the function by binding the “this” keyword with the passed object

and we should execute the returned function

|  |  |
| --- | --- |
| // Replacing this keyword value in global functions        var name = "\*\*\*\*\* Global var Name";        function print(band) {          console.log("Received is --->", this);          console.log(`Hello ${this.name}, you got Band -->  ${band}`);        }        //calling function normally        print("A"); | Result    See here, for 1st time we got window object for this keyword value  Next time onwards, when we are printing this 🡪 we are getting the passing value |
| //calling function with method by replacing this keyword value with another obj        var empSathish = {          name: "sathish",        };        print.call(empSathish, "B"); |
| //Calling print function with apply method by replacing the value for this object  //The only diff between call() vs apply() is in apply , we will pass values inside array        var empAnush = {          name: "Anush babu",        };        print.apply(empAnush, ["A"]);   //Fetching the function by Replacing the value for this keyword        //This bind() will returns function by just replacing this keyword with passed obj        //we have to execute the returned function        var empMani = {          name: "Manideep",        };        var f1 = print.bind(empMani, "A+");        f1(); |

Calling methods inside objects

objectName.methodName.call(this argument, remaining arguments)

Operators

|  |  |  |  |
| --- | --- | --- | --- |
| == vs ===  == is content comparison operator (if types are diff it will bring both into same type and then it will compare)  === will check both type and content (if type is diff then it wont even check for content)  'aaa'=='aaa' if both are strings it will do string comparison  And this == will also performs hash code /address comparison when we are comparing objects | 5=='5'  Now both will be converted to number data type and then it do comparison  true=='true' – will give false because  LEFT side number conv will give 1, right side number conv will give NAN   |  |  | | --- | --- | | Number(true) // will give 1 | Number(‘true’) // will give nan |   Address comparison  [8,9] ==[8,9] -- here even though content is same since both are here objects  It will perform address comparison which will give false  let a =[1,2] let b=a //here this will copy the address  b==a //true –bec this will compare adress |
| In operator for objects |  |

Keywords

Async and defer these 2 are attributes of script tag

|  |  |
| --- | --- |
| Without async / defer keywords |  |
| <head>      <meta charset="UTF-8" />      <meta name="viewport" content="width=device-width, initial-scale=1.0" />      <title>Currying functions demo</title>      <script src="jsfiles/demo.JS"></script>    </head>    <body>      <h3>Currying functions demo</h3>      <h5>vammov</h5>      <h5>vammov</h5>   <h5 id="one">one</h5>  Js file content  let ele = document.getElementById("one");  window.alert(ele);  Generally, Engine will execute the code line by line, Here js file will be loaded/executed 1st as script tag is placed In top,  Now error comes when “document.getElementById("one");” this is exec  Bec the moment this line Is exec, that corresponding html tag is not loaded by browser into memory,  Solution 1:- 1st browser should create DOM / 1st html coded should be exec by browser fully, then we should run js code | So  <script defer src="jsfiles/demo.JS"></script>  Defer means- to delay something /to postpone  Here due to defer keyword, js file will be loaded when html code Is fully loaded/executed   * + In real time also, as js files are big, its better to load js file after html code loading, so this defer keyword will be most used in real time   <script async src="jsfiles/demo.JS"></script>  Here async means, js file will be loaded parallelly,  Some times js file could also be loaded 1st |
| “debugger” this will acts a breakpoint, if u place this keyword in script tag, |  |

Object & Class

|  |  |
| --- | --- |
| Inbuilt global objects  Array- in JS even array is an object – index will be the key  \_ \_proto\_ \_ is a property/variable present in every object – it refers to its parent | <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects> |

We don’t have classes in JavaScript Like in java, we can create object in JavaScript using constructor function

A constructor function is also a regular function used to create objects in java script

Ways to create object

1. Object literal (flower braces)
2. Using new keyword with object constructor let x= new Object();
3. Using new keyword with parameterized constructor
4. Object.create() function – this is used for inheritance – means creating child obj from parent obj
5. Classes

Note:- We can create objects for function & we can create objects for classes since ES6

|  |  |
| --- | --- |
| function emp() {          console.log("called a function ");   } | 1. Generally we call a function as “” emp(); 2. In JS we can even create objects for a function   let x = new emp(); |

Notes on object

\_\_proto\_\_ is a property of an object that points the prototype of an object /parent == \_\_proto\_\_ property will give parent obj reference

Prototype:- in js, objects are created from prototypes, a prototype is a blueprint / template that defines the properties and methods that obj can inherit

How to access object values

1. Dot notation – same like java :- obj.key
2. Array notation:- Obj[“key”] – Here quotes are mandatory

How to add properties to an object

1. Dot notation – same like java :- obj.key= value
2. Array notation:- Obj[“key”] =value –-> Here quotes are mandatory

|  |  |  |
| --- | --- | --- |
| Creating object using literal   * Object is nothing but key collection of value pairs * Our js object can have both properties and methods like in java | //create a dummy object  let e1 = {};  document.write(e1);  // create a object with value {key1: value1, key2: value2}  let e2 = {  name: "mani",  //we can write double quotes in both sides  “state”:”ap”  sal: 20000 ,  //one object can have another object  address: {  ‘city’ ;’hyd’  }  //Here this getName is a method belong to object  getName: function() {  console.log(this)  }  }; | note:- Even after creating obj also we can create new/add new properties to the existing object unlike java  e1.name=”allu arjun”  let e1 = {          name: "mani",          age: 31,        };        console.log(e1);        e1.companyName = "tcs";        console.log(e1); |
| New operator | let x= new Object(); x.name=’mani’ ,x.sal=20000, x.address=’hyd’  then in that we have a property called “\_\_proto\_\_” that value is object  \_\_proto\_\_ is a property that references the prototype of an object | X:{  Name: ‘mani’,  Salary : 20000  \_\_proto\_\_: {} //This key is assoc with obj (**undersc** prop assoc with **obj**)  prototype: {} // -- this is assoc with class  ---Here\_\_proto\_\_ && prototype both are same  m.\_\_proto\_\_==movie.prototype //true  } |
| Using function | function Movie(){  }  Let x=new Movie()/ here this object also have a property called “\_\_proto\_\_” |  |
| Creating obj using inheritance  **\_\_proto\_\_** property gives us to parent obj ref like super keyword  Object.create(e1);  Will create separate object in heap by copying all properties from old object | let e1 = {          name: "mani",          age: 31,   };        console.log(e1);        //here e2 obj will inherit all props from e1- if u modify e2 prop e1 will not be modified- u can’t see those var in console as u inherited but u can use var       //Bec of this e2 doesnt have any prop but after creating e2 obj we can add prop to it        let e2 = Object.create(e1);        e2.name = "santu";        //as u created obj using create method , it really created another obj in heap        console.log("is e1==e2", e1 == e2); //since both are 2 diff obj this is false        //Here after creating e1 obj we are adding a new property called companyName        e1.companyName = "tcs";        console.log(e1);  //below gives true, bec e2 parent is e1        console.log("e2.\_\_proto\_\_==e1 -->", e2.\_\_proto\_\_ == e1); | **Access**  While accessing variables 1st it will check in current object  If those var are not present in current obj it will fetch those var data from parent obj  **Modify**  Even though you created object using inheritance  Child can read/access parent properties i.e., with child var even if u try to modify it will modify current class var  But child can’t modify parent properties ex:- it can’t modify parent variable data  Like super keyword in java, with \_\_proto\_\_ we can Access parent obj prop from current obj  Change parent class data with child variable  (with \_\_proto\_\_ we will get parent obj ref like super keyword)   |  |  | | --- | --- | | Normally if u        let e2 = Object.create(e1);  as e2 is created using e1,  after doing e2.age=40  since e2 doesn’t have any var  it will create age var in e2  with above code u cant directly change parent var | e2.\_\_proto\_\_.age=90  console.log(e1.age) //- 90  now e2.\_\_proto\_\_ means we will get parent e1 ref  e2.\_\_proto\_\_ ==e1  means e2 parent ==e1 | |

|  |  |  |
| --- | --- | --- |
|  | Using dot operator | Using array operator |
| Access | document.write(e2.name); | document.write(e2["obj property name"]);  document.write(e2["name"]);  This is especially useful when u want to pass the  Property name as string   |  |  | | --- | --- | | //IIFE- immediately invocable function  (function print() {    //creating an object    let u1 = new user("nexon", "india", "high range");    printer(u1);  })(); | //This function will accept an object and it will  // print the properties of an object  function printer(obj) {    //This will give the keys as a list of objects    const keys = Object.keys(obj);    for (k of keys) {      document.write("<br>" + k + ":  " + obj[k]);    } |   Output  carName: nexon manufacturerCountry: india range: high range |
| Adding a property | e2.address = "kavali";  document.write(e2.address); | e2["zipcode"] = 524201;  document.write(e2["zipcode"]); |

Printing all properties & values of an object

|  |  |
| --- | --- |
| Using for in loop | Using object.keys () |
| let u1 = new user("punch", "india", "Mid range");  for (key in u1) {    document.write(key + " : " + u1[key] + "<br>");  }  Output:  carName : punch ManufacturerCountry : india Range : Mid range | (function print() {    //creating an object    let u1 = new user("nexon", "india", "high range");    printer(u1);  })();  //This function will accept an object and it will print the properties of an object  function printer(obj) {    //This will give the keys as a list of objects    const keys = Object.keys(obj);    for (k of keys) {      document.write("<br>" + k + ":  " + obj[k]);    }  }  // we have a method called object.values(obj) which will give only values |
| Using Object.entries  [[key1,value1], [key2,value2], [key3,value3]]  This entries will return an array of entries, whereas each entry is again an array  //Printing using Object.entries() method  document.write(" <br> print using Object.entries() method <br> <br>");  let u2 = new user("Civic", "India", "Premium");  const entries = Object.entries(u2);  for (entry of entries) {    //Here each entry is an array , 1st ele will be the key and 2nd ele is value    document.write(`${entry[0]} : ${entry[1]} <br>`);  } |  |

Class

We will store data in objects, class is a template for which we can create object with well-defined structure and behaviour

Classes provide a structured way to organize code, making it easier to understand, maintain, and debug

Advantage

Classes have the advantage of inheriting the properties from parent into child, provides clear segregation – avoids duplicate var declaring in multiple classes

If we create 10 obj for same class – all 10 objects will look uniformly -

|  |  |
| --- | --- |
| Classes in java script without inheritance   class Employee {          companyName;          //this defines a constructor          constructor(name, address) {            this.name = name;            this.address = address;            this.salary = 10000;          }          //setter method- note this setter method name must be different than variable name          //Note if u are declaring setter method, you must declare a global variable          set companyName(value) {            console.log("setter method invoked");            this.companyName = value;          }          // Corrected getter method          get companyName() {            return `Bro company name is ${this.companyName}`;          }          //this is general public method          printAllDetails() {            console.log(this.name, this.address, this.salary);          }        }        let emp = new Employee("mani", "hyd");        //Here both getter & setter methods are treated as variables instead of functions        //Here companyName is a setter method but we treated as a variable        emp.companyName = "TATA consultancy services";        console.log(emp.companyName);  Note: here setter & getter methods are treated as variables instead of methods  So while calling then we should not use () braces, we should treat like variables        emp.companyName = "TATA consultancy services";  without declaring setter methods also happily we can create new property  emp.grade ='A'  it will create a new property called grade in emp object  Setter methods is to provide abstraction, abstraction means hiding variables,  Here if u can’t make variable’s private, then what is the use of setter methods again?  Instead of setter method we can have normal method, which can also do the same thing (only adv is setter method forces to have same variable name declared globally )  **Why setter methods are treated as property instead of methods?**  Why are we using setter methods – to create a new variable and value in js object right?  To create a new variable (p.height =5.1/ obj.varname=value)  we can do like this also, we don’t need any setter methods at all, since users are already habituated to this style, if we design like a method invocation nobody will use those methods  hence, js people designed like this like just “ref.var=value” style | With inheritance      <script>        class Person {          nationality;          //to declare cons , constructor keyword is mandatory          constructor(n, m) {            console.log("parent class cons invoked");            this.name = n;            this.mobileNum = m;          }          //This is a normal method          getPersonDetails() {            console.log("person details are -->", this.name, this.mobileNum);          }          //while writing setter methods make sure u declare those variable globally          set nationality(n) {            this.nationality = n;            console.log("invoked setter method");          }          //this is a getter method - but normal method also can do same          get getPersonInfo() {            return this.name + "-->" + this.mobileNum;          }        }  //a class can happily extend another class        class Employee extends Person {          constructor(n, m, sal, cn) {            //calling super class cons with super()            console.log("calling child class con");            super(n, m);            this.salary = sal;            this.companyName = cn;          }        }        let e = new Employee("Radhika", 8500, 120, "broad");      </script> |

Set, Map

Set is a class in java script, same like java here also set won’t allow duplicates, it has many constructor

Array, string, map,set are iterable objects in javascript

The Set class provides methods that you can use to interact with its data, such as add, delete, has, clear, size, forEach, and values,entries

1. Default constructor – let s1= new Set()
2. Array argument cons –

let ar1= new Array(‘a’,’b’)

let s2= new Set(ar1) // this is talking array as an argument

1. String parameterized constructor

let sr= “ania”

let s3=new Set(sr); // String arg constructor – this will take string and returns array of ele, those ele are stored in set collection

|  |  |
| --- | --- |
| * [Set](https://devdocs.io/javascript/global_objects/set) * [set.add](https://devdocs.io/javascript/global_objects/set/add) * [set.clear](https://devdocs.io/javascript/global_objects/set/clear) * [set.delete](https://devdocs.io/javascript/global_objects/set/delete) * [set.difference](https://devdocs.io/javascript/global_objects/set/difference) * [set.entries](https://devdocs.io/javascript/global_objects/set/entries) * [Set.forEach](https://devdocs.io/javascript/global_objects/set/foreach) * [set.has](https://devdocs.io/javascript/global_objects/set/has) * [set.intersection](https://devdocs.io/javascript/global_objects/set/intersection) * [Set.isDisjointFrom](https://devdocs.io/javascript/global_objects/set/isdisjointfrom) | * [Set.isSubsetOf](https://devdocs.io/javascript/global_objects/set/issubsetof) * [Set.isSupersetOf](https://devdocs.io/javascript/global_objects/set/issupersetof) * [set.keys](https://devdocs.io/javascript/global_objects/set/keys) * [Set.set](https://devdocs.io/javascript/global_objects/set/set) * [set.size](https://devdocs.io/javascript/global_objects/set/size) * [Set.symbol.iterator](https://devdocs.io/javascript/global_objects/set/symbol.iterator) * [Set.symbol.species](https://devdocs.io/javascript/global_objects/set/symbol.species) * [Set.symmetricDifference](https://devdocs.io/javascript/global_objects/set/symmetricdifference) * [set.union](https://devdocs.io/javascript/global_objects/set/union) * [set.values](https://devdocs.io/javascript/global_objects/set/values) |

|  |  |
| --- | --- |
| Methods in Set class |  |
| Set add(ele)  // Here the method return type is set, to perform method chaining they designed like this | let s = new Set("Manideep");  // this add method will return set as return type hence we are able to perform method chaining  s.add("Kumar").add("HTC").add("-->Virtusa").add("-->TCS");  s.forEach(print);  function print(ele, index) {    document.write(`${ele}`);  }  result  ManidepKumarHTC-->Virtusa-->TCS |
| boolean delete(ele)  document.write("before delete <br>");  s.forEach(print);  s.delete("e");  document.write("<br> after delete <br>");  s.forEach(print); | before delete M ,a ,n ,i ,d ,e ,p ,Kumar ,HTC ,-->Virtusa ,-->TCS , after delete M ,a ,n ,i ,d ,p ,Kumar ,HTC ,-->Virtusa ,-->TCS , |
| Boolean has (ele)  This is like contains() method | document.write(` <br> Set.has() method demo --> ${s.has("HTC")} `); |
|  |  |
| void clear() | s.clear();  s.forEach(print); |
| number size  This size is not a method, it is just a property  It returns the number data type | let ss = s.size;  document.write(`size is ${ss} and type is ${typeof ss} `);  Result  size is 10 and type is number |

#### 4 ways to iterate set

|  |  |
| --- | --- |
| Set is by default iterable,  Same like in java as all collections are iterable ,here also set is iterable | //Set is by default iterable -- u dont need iterator also  document.write(    `<br> Set is by default iterable -- u dont need iterator also <br>`  );  for (ele of s) {    document.write(`${ele} ,`);  } |
| Iterator entries()  This method same like java hasmap entries, this also will return Iterator,  Iterator has a method called next() – which can fetch the element | //1 Entry will have 2 components- 1) value 2) also value (in java entry will give both key and value,but here its different)  //This entries() method will return an iterator  const itr = s.entries();  document.write(`<br>entries return tye is  ${itr} <br>`);  //This itr.next() will give the next entry  document.write(`<br> ${itr.next().value}`);  document.write(`<br> ${itr.next().value}`);  document.write(`<br> ${itr.next().value}`);  document.write(` <br> Iterator demo <br> `);  for (const element of itr) {    document.write(`${element}`);  } |
| void forEach(takes 3 param function )  //This forEach will return void as return type  //Here 1 param function is mandatory atleast that function should accept 1 param which is actual element | let s = new Set("Manideep");  // here for each method expects a 3-parameterized/minimum 1 param function to be passed  //This is like passing a method reference to a functional interface  s.forEach(print);  function print(ele, index, sett) {    document.write(`${ele},`);  } |
|  |  |

Map

|  |  |
| --- | --- |
| const myMap = new Map();  myMap.set("name", "Alice");  myMap.set(123, "number");  myMap.set(true, "boolean");  console.log(myMap.get("name")); // Output: "Alice" |  |

Predefined objects in js

Console is the predefined object in JavaScript, in this object we have many methods

**Core Objects**

* **Array:** Represents an ordered collection of elements.
* **Boolean:** Represents a logical value (true or false).
* **Date:** Represents a specific point in time.
* **Error:** Represents an error that occurred during program execution.
* **Function:** Represents a JavaScript function.
* **Math:** Provides mathematical constants and functions.
* **Number:** Represents a numerical value.
* **Object:** The base object from which all other objects inherit.
* **RegExp:** Represents a regular expression for matching text patterns.
* **String:** Represents a sequence of characters.

**Document Object Model (DOM) Objects**

* **Document:** Represents the entire HTML document.(methods- createElement(), getElementById(), )
* **Element:** Represents an HTML element.
* **Node:** Represents a node in the DOM tree.
* **Text:** Represents a text node.
* **Window:** Represents the browser window.

**In window object we have method called settimeout(function, timeafterthis funtio to be exec, arg to that function)**

**Browser Objects**

* **History:** Represents the browser's history.
* **Location:** Represents the URL of the current page.
* **Navigator:** Represents the user agent (browser) information.
* **Screen:** Represents the screen on which the browser is running.

**Other Objects**

* **XMLHttpRequest:** Used for making asynchronous HTTP requests.
* **JSON:** Used for working with JSON data.

1. Document, Element, Node,Text,Window
2. Console

*Console object*

|  |  |
| --- | --- |
| Console.log(a,b)- for console obj we can pass any num of arguments  console.log("Hello, world!");  console.info("This is an informational message.");  //2 arg console logging    console.log("innerText is -->", document.getElementById("one").innerText);  console.warn("Be careful!");  console.error("An error occurred!");  console.dir(person);  if u want to print in the form of table in console then use below  //for this table we should pass an array ex:- an array of objects/ array of numbers  console.table([aa,b,c,d]);  console.table([{ name: "Alice", age: 25 }, { name: "Bob", age: 30 }]);  console.group("Group 1");  console.log("Message 1");  console.log("Message 2");  console.groupEnd();  console.assert(2 + 2 === 4, "2 + 2 should equal 4");  console.time("My Timer");  // Do something that takes time  console.timeEnd("My Timer");  console.count("Counter");  console.count("Counter");  console.trace(); | console.table([    { name: "Alice", age: 25, city: "hyd" },    { name: "Bob", age: 30 },  ]);    Console.time()  //some lines here  Console.timeEnd()  Now js will minus those diff between 2 times and it will log that time in console |
|  |  |
|  |  |

DOM- Document object Model

Every object in java / JS will have some properties and methods

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Here everything is an object  Ex:- here html tag, head, body , h1 everything is an object  console.info(typeof document.getElementsByClassName("c1"));  // object  So above state each element / list of elements is nothing but an object | Using this document object methods and properties  We can 1) select elements, modify elements, remove, create elements   1. We can change style, create style, add, remove   Ex:- document object has below   |  |  | | --- | --- | | Methods of **Document** object | Properties (called global variables) | | 1. Element getElementById(“unique id name”)   return ele with specific id   1. <List<Elements>> getElementsByClassName(“cls name”)   Returns list of all ele belong to spec class   1. <List> getElementsByTagName(“h2”)   Instead of above 3 methods we can use below  ex:- document.getElementsByTagName(“body”) ; -- u will get the entire body   1. document.querySelector(“.class/#id/tag name”)   The problem with query selector is it will give only 1 ele even in case of query selector it will give 1st matched element   1. <NodeList> document.querySelectorAll(“.class/#id/tag”)   the over advantage with this is even there are 2 id’s it will give both those 2, so u should make sure id should be unique   1. Element createElement(“h2”)   //This will create <h2> an element with name h2  //1) Creating an element        var h4ele = document.createElement("h4");        h4ele.innerText = "Bro i am a h4 tag"; | innerHtml, outerHtml  innerText,  outerText  body  head  title  links  forms  images  scripts | |  |  | |

Element object

With Element object property and methods we can do following (Here an element is tag + content ex:- <img id=’a’ class=’b’ width=’100px’ height=’20px’>)

Here id, width, height are the attribute of img Element

* 1. We can add any attribute at runtime ex:- height, width, id , class attributes
  2. We can add a class to the existing class at runtime
  3. We can add an event listener ex:- like on clicking of this element which function should execute
  4. If this is a parent element, We can add, remove child elements to the parent elements

Changing attributes

|  |  |
| --- | --- |
| Element object  If an element with the given ID is found, the method returns that element as an Element object. If no element with that ID exists, the method returns null.  The Element object represents an HTML element in the DOM. It provides various properties and methods to interact with the element, such as:   * **innerHTML**: Gets or sets the HTML content of the element(gets tag+content). * **inerText**: it will get only the content between the tag, it won’t get any tag * **textContent**: Gets or sets the text content of the element. * **style**: Accesses the element's style properties. * **className**: Gets or sets the element's class attribute. * **classList**: returns a list of classes for that attribute   **e.**classList.add(‘new class name’), **e.**classList.remove(‘old class name’) – this is especially useful when u want to add or remove a class to the existing class – to change the styles   * **getAttribute(“anuAttributename”):-** <h2 id=”a” class=”b” style=”border”> content</h2>   This Elements h2 tag has many attribute like id, class, style.. to get that attribute name we can use this method   * **setAttribute(“any attribute name”, “value”) ex:- setAttr(‘class’,’value’)** * **addEventListener(“some action”,”some function to exec”)**: Adds an event listener to the element.   Ex:-myButtonElement.addEventListener("click",myFunction);   * **remove()**:- remove that complete element * **removeChild(Element eleToBeRemoved)**: Removes a child element using the parent element. * **appendChild(Element e)**: Appends a child element at last. * insertBefore(Element eleToBeInserted, Element beforeThisElement) //add a child before any ele   parentElement.insertBefore(newChild, referenceNode);   * **replaceChildren(Element eleToBeReplaced)** | <body>      <div id="one">        <h2>First div h2 tag</h2>      </div>      <div class="c1" id="d2">        <h3 id="h3d2">h3 tag of 1st div class-c1</h3>      </div>      <div class="c1" id="d3">        <h3 id="c1h3">hh3 tag of 2nd div class-c1</h3>      </div>      <div id="s1">StyleDemo</div>      <div id="remParent">Direct remove parent element</div>      <img src="images/chirala house.PNG" alt="" id="i1" /> <br />      <style>        .green {          background-color: lemonchiffon;        }        .g1 {          width: 100px;        }        .g2 {          width: 300px;        }      </style>  //2) Adding a child element after last element        var d3 = document.getElementById("d3");        d3.appendChild(h4ele);        console.log("d3 after adding ele -->", d3);        //3) Adding an child before first element        var d2parent = document.getElementById("d2");        var h2ele = document.createElement("h2");        h2ele.innerText = "Bro i am h2 element inserted using insertBefore()";        //Here 1st arg is what ele to insert, 2nd arg is before which ele 1st arg must be placed        d2parent.insertBefore(h2ele, document.getElementById("h3d2"));        //4) Remove the element directly -This will remove the entire element and all child elements        var ele1 = document.getElementById("remParent");        ele1.remove();        //5) Remove the child present in that ele        var h3c2 = document.getElementById("c1h3");        var d3 = document.getElementById("d3");        d3.removeChild(h3c2);        console.log("after removing child", d3); |
|  |  |

Examples

|  |  |  |  |
| --- | --- | --- | --- |
| <script>        //This will get the entire element along with tag        console.log(document.getElementById("one"));        //This will give all the elements on same class        console.info(document.getElementsByClassName("c1"));        console.log(document.getElementsByTagName("h3"));        //for query selector we can pass id/class/tag        //But it will give only 1st matching ele        console.log(document.querySelector("#one"));        console.log(document.querySelector(".c1"));        console.log(document.querySelector("h3"));        //To return all matching ele use this        console.log(document.querySelectorAll("h3"));      </script> | |  |  | | --- | --- | | <div id="one">        <h2>id tag h22</h2>      </div>      <div class="c1">        <h3>h3 tag of class-1</h3>      </div>      <div class="c1">        <h3>h3 tag of class-2</h3>      </div> |  | |
| Inserting an element before another element |  |
|  |  |

Changing the styles -like changing image height, width, adding the border – using style property of Element class

  //6) change styles of an ele using style property of Element object

      var s1Ele = document.getElementById("s1");

      s1Ele.style.border = "2px black dashed";

color =’#23AA34’

//    s1Ele.style.backgroundColor = "#aa3344"; //red

//7) applying a class at runtime

      var clsAttr = s1Ele.getAttribute("class");

      var clsList = s1Ele.classList;

      document.write(

        `Fetching Existing classes using attr --> ${clsAttr} <br>`

      );

      document.write(`Fetching Existing classes using property --> ${clsList}`);

Set the values for text boxes – using setAttribute(‘attribute name’, “value”)

    //Auto fill a first name and last name on mouseenter

      var fnEle = document.getElementById("fn");

      var lnEle = document.getElementById("ln");

      function autofiller() {

        fnEle.setAttribute("value", "Manideep");

        lnEle.setAttribute("value", "Kumar");

      }

      function autoRemover() {

        fnEle.setAttribute("value", "");

        lnEle.setAttribute("value", "");

      }

      //registering a function on mouse enter

      fnEle.addEventListener("mouseenter", autofiller);

      lnEle.addEventListener("mouseenter", autofiller);

      fnEle.addEventListener("mouseleave", autoRemover);

      lnEle.addEventListener("mouseleave", autoRemover);

Adding a new class, Adding another class to existing class, – to add more styles – using SetAttribute() Method

      //Adding a new class using  setAttribute

      s1Ele.setAttribute("class", "green");

      document.write(`existing classes --> ${clsList}`);

      //Adding a class using classList property

      s1Ele.classList.add("g1");

#### Events and Event handling

Event is nothing but an action in JavaScript,

Sample:- user clicked something or scrolled down …

Who generates events

* 1. User – keypress, scroll, focus
  2. System – load, error, abort

Types of user generated events

1. Browser specific events – scroll up/down, resize browser
2. DOM/Web page specific events – click, hover, focus on some element’s

|  |  |  |  |
| --- | --- | --- | --- |
| Mouse specific events   * 1. Click   2. Doubleclick   3. Mouseover   4. Mouseout   5. Mouse move | Keyboard events   * 1. Keyup (after pressing and when we are releasing the key)   2. Keydown (means when it is pressed)   3. Keypress | Focus events   1. Focus 2. Blur 3. Focusin (when we clicked then its called focusin) 4. Focusout | Form events   1. Submit 2. Reset 3. change |

1. Attaching inline event handler, :- means here we will attach the listener to that tag itself

Here onclick is the event listener

    <button id="submit" onclick="autofiller()">Submit</button>

 function autofiller() {

        fnEle.setAttribute("value", "Manideep");

        lnEle.setAttribute("value", "Kumar");

      }

I personally feel inline event handler is best, because instead of writing script to get the Element and attaching event listener and then passing the function

Here in single step we can attach the listener in the html element itself

But it in general we should **avoid using inline event handlers**

**Cross-Site Scripting (XSS):** Inline event handlers can introduce security vulnerabilities, especially if user-generated content is included in the HTML. This can lead to XSS attacks, where malicious code is injected into the page.

1. Using event listener

Attaching an event listener to that element: - like on click some function must be executed … / on-hover some function should be executed

JavaScript can do any manipulation

 //Adding a event listener on click some function will exe

//That function can be o or 1 parameterized function, JS will pass the event as an object

//      s1Ele.addEventListener("anyAction", 0 or 1 param function);

      s1Ele.addEventListener("dblclick", () => {

        s1Ele.classList.add("g2");

      });

      //Adding a event listenere on click add a attribute called image with anonymous function

      //  <img src="images/chirala house.PNG" width="" height="" alt="" id="i1" />

      //initially the img ele doesnt have attributes we are adding the attr based on event

      var iele = document.getElementById("i1");

      iele.addEventListener("mouseenter", () => {

        iele.setAttribute("height", "100px");

      });

      iele.addEventListener("mouseleave", () => {

        iele.setAttribute("height", "300px");

      });

//Referring a named single parameterized function

|  |  |
| --- | --- |
| First name: <input type="text" id="fn" /> <br />      Last name: <input type="text" id="ln" /> <br />      Sur name: <input type="text" id="sn" onclick="colorChanger()" />  //so when u click on this surname text box, fn text box background color will be changed | The argument will be passed here is event, we don’t need to pass it will be passed by browser  function colorChanger(e) {          console.log(e);          fnEle.style.backgroundColor = "#aa3344";          fnEle.setAttribute("value", "Manideep");          lnEle.setAttribute("value", "Manideep");        }   var fnEle = document.getElementById("fn");        var lnEle = document.getElementById("ln"); |

Window object

In java script window object represent the browser window that contains a DOM element. It provides a vast array of properties and methods to interact with the browser environment.

1 browser =1 window = 1 tab

Every browser window will have a name

With window object what u can get

|  |  |
| --- | --- |
| U can get/set a name to window and u can get full document/html web page of that wind  U can get inner, outer width & height of window and window url address | * U can open a new window and get ref and close the window and check if it is close or not * Recently opened window position can be moved horizontally & vertically, and that window is resizable (width and height wise adjustable) |

**Key Properties and Methods:**

### Properties:

* **document**: Represents the HTML document loaded in the window.
* **location**: Provides information about the current URL and allows navigation. Like href (href= protocol + path)

window.location.protocol =http, window.location.pathname = '/codes/objects/2.1.2.Window%20object.HTML'

* **history**: Manages the browser's history.
* **navigator**: Contains information about the user's browser and system.
* **screen**: Provides information about the user's screen.
* **innerWidth and innerHeight**: The width and height of the browser window's content area.
* **outerWidth and outerHeight**: The total width and height of the browser window, including toolbars and borders.
* **Closed** : this returns Boolean , to check whether the window is closed or not
* **Name**: every window can have a name we can set and fetch using window.name proerty

### Methods:

* **alert()**: Displays an alert dialog box.
* **confirm()**: Displays a confirmation dialog box.
* **prompt()**: Displays a prompt dialog box to get user input.
* **setTimeout()**: Sets a timer to execute a function after a specified delay.
* **setInterval()**: Sets a timer to repeatedly execute a function at a specified interval.
* **Window open()**: Opens a new window or tab and returns us the reference of that opened window
* **close()**: Closes the current window.
* **resizeTo()**: Resizes the current window. resizeTo(400,200)- final window size will be width 400 and height 200
* **resizeBy()**:Resizes the current window. resizeTo(400,200)- final window size will be width current width+400 and current height + 200
* **moveTo()**: Moves the current window to a specific position / to a specific x-axis and y-axis position. (u can give minus values also)
* **moveBy()**: Moves the current window with these values ex:- moveBy(400,200)- current win will be moved 400 x-axis,200 y axis from current place
* **scroll()**: Scrolls the window to a specific position
* **print()**:
* **Promise fetch(resource) :** this is to make ajax call – same like XMLHttpRequest

|  |  |
| --- | --- |
| Window is a global object in JavaScript   <script>   var aaaa = 10;        function abcd() {          console.log(`abcd`);          console.log(this);        }        abcd();        console.log(window.aaaa);      </script> | Here if u see the variable aaaa and the function abcd are present in window object    Note:  a let variable declared within a script tag does not become a property of the window object.  Only “var” type variables will be visible under window object |
| Long Window.setTimeout(func to be exec, time, arguments to that function );  //means this function will be executed once after mentioned time  --Here this setTimeout is a callback function,  - this function will return a timer id ,in case if u want to stop that function we should call Window.clearInterval(‘Timer id’) to stop this function execution  Ex:- here this passed function will be **executed once after 3s**  function greet() {  console.log("Hello, world!");  }  Window.setTimeout(greet, 3000); // Greet after 3 seconds  A function – which should show pop-up after 10sec after loading the page  Ex:- subscribe to our news letter popup after 5 sec | Ex:-2  function disableAdBlocker(arg) {    window.alert(`${arg} method exec successfully`);  }  //This function will be exec only once that to after 5 sec from page loading  let timerId = **window.setTimeout(disableAdBlocker, 5000, "AdBlocker");**  //if u wish to prevent that function to start then clear with timerId  window.clearTimeout(timerId);  console.log("timer id -->", timerId);  how it works internally?  This particular function will be pushed to task queue,  Event loop is a really a for-ever loop which will check the call stack() if this is empty then this function() will be put into call stack memory by removing from queue,  Then that function will be executed |
| Timer id=Window.setInterval(func name, time, arguments to that function)  Here the same function will be called continuously until we clear the interval  //Interval is nothing but gap between 2 executions  Use case- a function to check the session is valid or not – this function should exec every 5 minutes | var times = 0;  //means this notifier method will be called for every 2 sec with same arguments  **var timerid = window.setInterval(notifier, 2000, "Leadership");**  //This method will return a timer id it is just a number data type    console.log(typeof timerid);  function notifier(name) {    console.log(`received times as ${times}`);    times++;    document.write(      `Reminder ${times} ->Complete your mandatory training name --> ${name} <br>`    );    if (times > 10) {      document.write(        ` 10 reminders already over, no more left, Now escalation only`      );      //Here we are clearing the recursion program with help of timer id  **window.clearInterval(timerid);**    }  } |
| Window.clearInterval(‘Timer id’)  This will cancel the execution or abort the execution |
| Window Open(url,windowName/target, window features)  -it says – what page to open, where open(current/new tab), how window should look(height, width, location,position)  Window name – any custom name as every tab/window can have separate name  Target allowed values 🡪   * \_self – means open in current tab * \_blank – means opens in new window either new tab / new window like popup, this is the default behaviour if u don’t mention anything (if u give any spec then new window will open else new tab only will be opened – bec tab doesn’t have any spec na – hence if u don’t give any spec it means new tab only) * \_parent * \_top   **windowFeatures (string, optional):** A comma-separated list of features to enable or disable in the new window. Each feature is specified as featureName=setting. Here are some common features:   * 'toolbar=yes|no': Enables or disables the toolbar. * 'location=yes|no': Enables or disables the address bar. * 'menubar=yes|no': Enables or disables the menu bar. * 'status=yes|no': Enables or disables the status bar. * 'scrollbars=yes|no': Enables or disables scrollbars. * 'resizable=yes|no': Allows or prevents resizing of the window. * 'width=pixels': Sets the width of the content area of the new window (excluding scrollbars). Minimum value is 100. * 'height=pixels': Sets the height of the content area of the new window (excluding scrollbars). Minimum value is 100. * 'left=pixels': Sets the horizontal position of the new window relative to the left edge of the screen. * 'top=pixels'   Ex:-  Window.open(“child.html”,”\_blank”, “left=100px,top=200px,left=400px”)    Features:  Ex:- "width=600,height=400,scrollbars=yes,resizable=yes"  Closing – only with that prev opened window ref we can call close() method   <button onclick="openNewWin()">Click here to open new window</button> <br />      <button onclick="closePrevOpened()">        Click here to close recently opend window      </button>   <script>   let w1;        function openNewWin() {          //Note we can only close our page- we cant close google/instagram other pages          //to close we need some plugins          w1 = window.open("6.Date Object.HTML", "\_blank","height=300px,width=800px");          console.log("window opened");        }   function closePrevOpened() {          w1.close();          console.log("popup closed");        } | //MoveTo  //  Everytime u click resize it should increase a bit using resizeBy & reduce a bit everytime  // moveBy- everytime move sidewards using moveBy() |
|  |  |

String object

* Same like java, here also 2 ways to create string, here also string is immutable, to concat() method we can use either + operator or
* In string 0th index means the 1st letter,
* For slice method 🡪 last letter is considered as -1 index

**Let x=’s1’;**

**Let x1=new String(html“)**

**JavaScript String Object Methods**

JavaScript provides a rich set of methods to manipulate strings. Here are some of the most commonly used methods:

Properties

length

Accessing Characters

* **charAt(index):** Returns the character at the specified index.
* **charCodeAt(index):** Returns the Unicode character /ascii code at the specified index.

Searching for a Substring

* **indexOf(searchValue[, fromIndex]):** Returns the index of the first occurrence of a specified value.
* **lastIndexOf(searchValue[, fromIndex]):** Returns the index of the last occurrence of a specified value.
* **search(regexp):** Searches a string for a specified value and returns the position of the match.

Modifying Strings

* **concat(...strings array ):** Combines two or more strings.
* **slice(startIndex, endIndex):** Extracts a section of a string and returns a new string this is preferred over substring().
* **substring(startIndex, endIndex):** Similar to slice(), but wont work for negative indices differently.
* **substr(startIndex, length):** Extracts a specified number of characters from a string.
* **replace(searchValue, newValue):** Searches a string for a specified value and replaces it with another value.
* **toUpperCase():** Converts a string to uppercase.
* **toLowerCase():** Converts a string to lowercase.
* **trim():** Removes whitespace from both ends of a string.
* **repeat()**

Other Useful Methods

* **split(separator, limit):** Splits a string into an array of substrings.
* **match(regexp):** Searches a string for a match against a regular expression.
* **includes(searchString[, position]):** Determines whether a string contains a specified substring.
* **startsWith(searchString[, position]):** Determines whether a string begins with a specified substring.
* **endsWith(searchString):** Determines whether a string ends with a specified substring.
* **padStart(targetLength[, padString]):** Pads a string from the start to a specified length.
* **padEnd(targetLength[, padString]):** Pads a string from the end to a specified length.

**“x”.**concat(“y”,”z”)

Let x=’manideep’ let s=’java script’

|  |  |
| --- | --- |
| Str.split(‘’) without any value – then entire aray will be split char wise  //Where ever small v is there that will be cutted splitted        //Here it will remove the char v and split the rest        p("split", s3.split("v")); //split is also case sensitive        //since we didnt gave any char all char will be seperated        p("split without any arg", s2.split("")); | split-->A,s Vaidya,raVtn,am split without any arg-->A,v,s, ,V,a,i,d,y,a,r,a,t,n,a,m |
| x.startWit(‘m’) – its case sensitive  s.endsWith(‘html’)  //Both starts with & ends with performs case sensitive comparision        p("starts With", s3.startsWith("avs"));        p("Ends With", s3.endsWith("Avs")); | starts With-->false En With-->true |
| s.slice(1,4)- result AVA  For slice method 🡪 last letter is considered as -1 index  slice means piece- that small cutted piece  so here after slicing that piece will be returned   let s2 = "Avs Vaidyaratnam ";   p("slice", s2.slice(1));  slice-->vs Vaidyaratnam //Here since we didn’t mentioned end , it took till end        let s2 = "Avs Vaidyaratnam";    p("slice 1 till end", s2.slice(1)); //from 1st index till end        p("slice 0", s2.slice(0)); //Here zero means from 0th index till end        //For slice method last letter is considered as -1th index        p("slice -1", s2.slice(-1)); //Here zero means from last letter till end        p("slice -1", s2.slice(-2)); //Here zero means from last 2nd letter till end        p("slice -5,-2", s2.slice(-5, -2)); //Here zero means from last 2nd letter till end        p("slice 2,5 ", s2.slice(2, 5)); //Here zero means from last 2nd letter till end | s.slice(1,3) – AV – here only two char will be printed 1 and 2 – from index 1 till 3 , but 3rd index will not be included  s.slice(4) - single arg means it will be considered as starting integer till end  s.slice(-2) –  last index is -1, last before index will be -2  s.slice (-4,-1)  note:- we should always give numbers from left side to right side  ex:- index 2 to 7 or from index -7 to -3  slice 1 till end-->vs Vaidyaratnam slice 0-->Avs Vaidyaratnam slice -1-->m slice -1-->am slice -5,-2-->atn slice 2,5 -->s V charAt->0-->A |
| **substring(startIndex, endIndex):**        let s2 = "Avs Vaidyaratnam";  p("substring 1", s2.substring(1)); //from 1st index till end  p("substring 1,3", s2.substring(1, 3)); //from 1st index till 3rd , 3rd will not be incl  p("substring -3 -1", s2.substring(-3, -1)); //substring wont work for negative indexes | It wont allow negative indices at all  Substring(4) from 4th index till last  substring 1-->vs Vaidyaratnam substring 1,3-->vs substring -3 -1--> |
| Substr(startindex,no of char to print) | //From 1st index it will print 5 char totally        p("substr - from 1 it will print 5 char", s2.substr(1, 5));  substr - from 1 it will print 5 char-->vs Va |
| Ex1:- “Html”.includes(‘html’) false- bec it performs **contains** operation with case sensitive comparison  Ex2:- Str.includes(‘content’, index from where to check) | This includes in nothing but a **contains operation**   let s = "Avs Vaidyaratnam”  p("includes", s.includes("Avs", 1));  result includes-->false  means after 1st index avs is not there, it is there only from 0th index |
| Str.repeat(4) means repeat 4 times that string  In java also we have this | let s2 = "Avs ";        p("repeat 2 ", s2.repeat(2));  repeat 2 -->Avs Avs |
| Str.replace(what str to replace, desired str ) only 1st occurrence will be replaced, not all occurances  Str.replace(/VA/g,’python’) - replace all/global occurrences with this target  Since we are passing regex, we should not pass as string | //single replace without regex  //since below is not a regex this will replace only in single place in case sensitive manner  p("replace", s3.replace("Avs", "BVS"));  replace-->BVS VaidyavraVtnvam avs Avs |

Practice

Date Object

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date#instance_methods>

Date is a pre-defined object in java script it has following constructors and methods

In java/ JS standard time is 1st January 1970 00:00:00 GMT or (1st January 1970 05:30:00 AM IST) as our IST is 5:30 ahead of London GMT

In js month starts with 0, for January month/ month index is 0

When you are passing month number as argument

Unlike java, this Date object in js is mutable and it have even setter methods

Static methods – all these static methods will return Millis as the output and constructor will take that millis

|  |  |
| --- | --- |
| Long Date.now();  This will give the lapsed Millis since epoch or since standard time | let millis = Date.now();   p("Date.now() elapsed millis since standard date-->", millis);  Date.now() elapsed millis since standard date--> 1732860834815  //This single arg takes only millis        p("cons takes millis ", new Date(millis));  cons takes millis Fri Nov 29 2024 12:08:30 GMT+0530 (India Standard Time) |
| Long Date.parse(‘date’)  This also will give millis since provided time |  |
| Long Date.UTC(yyyy,MM,dd,hh,mm,ss)  This will take UTC time give the lapsed Millis since the provided time | //This takes UTC time and converts accordingly  let ms = Date.UTC(2024, 11, 20, 1, 2, 3);  p("Date.UTC()", new Date(ms));  Date.UTC() Fri Dec 20 2024 06:32:03 GMT+0530 (India Standard Time) |

Constructors

|  |  |
| --- | --- |
| new Date()  It will give current system date and time in String | let x = new Date();  Fri Nov 29 2024 10:44:25 GMT+0530 (India Standard Time) |
| new Date(millis)  //This single arg takes only millis  //1000 mills ==1sec will be added to standard date jan 1 1970  p("millis param cons-->", new Date(1000));  millis param cons--> Thu Jan 01 1970 05:30:01 GMT+0530 | It will add this many millis to above standard time        //add 1 day to standard date jan 1 1970        p("millis param cons-->", new Date(24 \* 60 \* 60 \* 1000));  millis param cons--> Fri Jan 02 1970 05:30:00 GMT+0530 (India Standard Time)        //Minus 1 day to standard date jan 1 1970        p("millis param cons-->", new Date(-24 \* 60 \* 60 \* 1000));  millis param cons--> Wed Dec 31 1969 05:30:00 GMT+0530 (India Standard Time) |
| Single year cons is not available bec, if u give single year alone then  It will take year as Millis param cons  new Date(year, monthIndex)  new Date(year, monthIndex, day, hours)  new Date(year, monthIndex, day, hours, minutes)  new Date(year, monthIndex, day, hours, minutes, seconds)  new Date(year, monthIndex, day, hours, minutes, seconds, milliseconds)  p("year,monthIndex -->", new Date(2024, 1));  year,monthIndex --> Thu Feb 01 2024 00:00:00 GMT+0530 (India Standard Time)  p("year,monthIndex,date -->", new Date(2024, 0, 2));  year,monthIndex,date --> Tue Jan 02 2024 00:00:00 GMT+0530 (India Standard Time) | // param cons  p("year,monthInd,d,hours", new Date(2021, 11, 1, 23));  year,monthInd,d,hours Wed Dec 01 2021 23:00:00 GMT+0530 (India Standard Time)  p("year,monthInd,d,hours,min", new Date(2021, 11, 1, 23, 59));  year,monthInd,d,hours,min Wed Dec 01 2021 23:59:00 GMT+0530 (India Standard Time)  p("year,monthInd,d,hours,min,sec", new Date(2021, 11, 1, 23, 59, 6));  year,monthInd,d,hours,min,sec Wed Dec 01 2021 23:59:06 GMT+0530 (India Standard Time)  p("year,mon,d,hours,min,sec,ms", new Date(2021, 11, 1, 22, 59, 6, 990));  result  year,mon,d,hours,min,sec,ms Wed Dec 01 2021 22:59:06 GMT+0530 (India Standard Time) |

|  |  |
| --- | --- |
| String parameterised constructor  --------------  Year must be either in beginning or at ending, but month always at start before date  new Date(‘yyyy-mm-dd’)  new Date(‘mm-dd-yyyy’)  //--String param cons yyyy-MM-dd  p("String param cons", new Date("2022-04-28"));  String param cons Thu Apr 28 2022 05:30:00 GMT+0530 (India Standard Time)  //String param cons MM-dd-YYYY  p("String cons MM-dd-YYYY -->", new Date("11-28-2083"));  String cons MM-dd-YYYY --> Sun Nov 28 2083 00:00:00 GMT+0530 (India Standard Time) | const date1 = new Date('December 17, 1995 03:24:00');  // Sun Dec 17 1995 03:24:00 GMT...  const date2 = new Date('1995-12-17T03:24:00');  // Sun Dec 17 1995 03:24:00 GMT...  console.log(date1 === date2);  // Expected output: false  console.log(date1 - date2);  // Expected output: 0 |

Methods

|  |  |
| --- | --- |
| Long getTime() | This getTime() method will give the lapsed mills between standard time and current object time  Let d= new Date(2019,06) –it will provide the lapsed millis from jan 1 1970 till june 2019  Get lapsed millis from provided time  Long Date.now(); //This will always give lapsed millis between standard time and current time  This also will give the lapsed Millis since epoch or since standard time   let d2 = new Date(1970, 0, 1, 5, 30, 1);  p("provided date is", d2);  provided date is Thu Jan 01 1970 05:30:01 GMT+0530 (India Standard Time) p("lapsed millis b/n standard time and give date", d2.getTime());  lapsed millis b/n standard time and give date 1000 |
|  | getFullYear()- 2022  getMonth() – It will give month index 0- January  getDate()  Int getDay() - 0 means Sunday, 6-Saturday |
|  | setFullYear() – its mutable object |
|  |  |
|  |  |
|  |  |

Math,Number

static methods

 function p(text, d) {

        document.write(`${text} ${d}<br> `);

}

|  |  |
| --- | --- |
| Sign() | Returns -1 or 1 0r 0 |
| Abs() – absolute value- means it will give only positive value even if u pass negative value |  |
| Sqrt()ex:- sqrt(16) =4 |  |
| Pow(5,2) = power(base,power)=52= 25 | p("power", Math.pow(10, 16));//16 zero after 10  power 10000000000000000        p("power", Math.pow(100, 8)); //16 zero after 10 |
| Max(), min()  let a = [1, 2, 7, 1, -20];        //below will give error as min wont take an array so suse spread operator        p("min", Math.min(a));        p("min", Math.min(...a));  min NaN min -20 | Math.min(4,2,5,7,8) =2  Math.max(4,2,5,7,8) =8    p("max", Math.max(...a));  max 7 |
| Round(), - round will round to nearest integer  ceil() – to upper always  floor() – to lower always  Trunc(),- it will remove the fraction part completely | Round(6.1) =6  Round(6.8)=7  Trunc(2.999) =2 |
| random() –random will give a random number between 0 and 1 – if u want multiply and trunc() to get only the integer par  p("random", Math.random() \* Math.pow(10,16));  random 5454311955520015 | Math.trunc(Math.random()\*1000)  p("random", Math.random());  random 0.4200155764849334 |
| exp() - exponential | Number(‘4’) – to convert string to number  Number(string) /  Number(boolean) = Number(true) = 1  Number( ‘true’) = NAN = we cant convert this string to number as this is treated as string  Number(‘g’) = NaN = bec this cant be parsed as number  Number(‘ ‘ ) ==0 Number(‘’ ) ==0 //any space/tab/new line will give zero |

JSON vs. JavaScript objects

We can write array of objects

|  |  |
| --- | --- |
| {  //left side also we can enclose in “”  “name”:”mani”,  “age”: 30  }  In .json files u can write single value like [boolean , number] | [ {}, {}, {}] |

|  |  |
| --- | --- |
| JSON Parse(String) |  |
| Stringify() | To convert java script object to json **string**  Generally if u want to send data to server, we should convert to string  I don’t know why server won’t accept json obj directly |

|  |  |
| --- | --- |
| Java script objects | Json object |
| whereas JavaScript obj doesn’t need left side quotes | In original .json file, left side double quotes are mandatory like postman |
| In javascript object we can even define function  let emp = {          name: "mani",          age: 40,          salary: 40000,  //functions defined in any object is called a method          getDetails: function (){            }        }; | Json objects doesn’t allow us to define any method unlike JavaScript object |

AJAX

Note:- Modern web development often leverages the **Fetch API**, which provides a more concise and flexible approach to making HTTP requests than this

Asynchronous java script and XML (it can be either sync/async)

Ajax helps in **fetching the data asynchronously** from webserver and data will be embedded /**replaced asynchronously** into current webpage **without page refresh**

Initial day’s server used to send xml data hence they named like that, now this Ajax can fetch the json data as well

In JS we have to use XMLHttpRequest predefined object to make AJAX call to exchange data from webserver

Steps

Create object of XMLHttpRequest

Call open method open (method, url, async, username, password)

Send the request using send method

methods

|  |  |
| --- | --- |
| open(method, url, async, username, password) | This is like postman, 1st we can see protocol then URL.. |
| Send() | Used to send GET requests |
| send(data) | **data (optional):** The data to send with the request, typically used for POST requests. If omitted, a GET request is sent |
| setRequestHeader(headerName, value) |  |
| getAllResponseHeaders() | Returns a string containing all response headers. |
| getResponseHeader(header) | Returns the value of the specified response header. |
| abort() | Cancel the current requests |
|  |  |

|  |  |
| --- | --- |
| Property | Description |
| onload | When the request is completed and when we response is fully downloaded  (This event will be auto triggered when ajax framework fully received the response) |
| Onreadystatechange | Defines a function to be called when the ready state property changes |
| Onprogress | Triggers periodically while the response is being downloaded reports how much have been downloaded |

The above 3 are properties cum events and below are only properties

|  |  |
| --- | --- |
| readyState  .(they should have renamed as current status) | This holds the status of the request  0: request not initialised , 1: server conn established, 2: request received  3:processing request, 4:req finished and response is ready  Funny- even if url is wrong conn status shows as established I don’t know why it shows wrong |
| status | This holds the HTTP status code/number of the response. Ex:- 200 /400/403… |
| statusText | The HTTP status message. Ex:- OK, forbidden, Not Found, |
| responseText | The response data as text. (By default we will receive as string not as object)  So we need to convert using json.parse |
| responseXML | The response data as an XML Document Object Model (DOM). |

Example

|  |  |
| --- | --- |
| const xhr = new XMLHttpRequest();  xhr.open('GET', 'https://api.example.com/data');  xhr.onload = function() {  if (xhr.status === 200) {  console.log(xhr.responseText);  } else {  console.error('Request failed. Returned status of ' + xhr.status);  }  };  xhr.send(); |  |

When u hit until we receive we should get a message called loading – using onprogress event

Error

It’s also an object in java script –

Promises

To avoid call back hell in java script they introduced promise Object – callback hell looks ugly code

Callback hell- means Window.setTimeout(fun,2000) – that function again uses another window.setTimeout()

|  |  |
| --- | --- |
| Promise object cons take a 2 param function– those 2 params are functions which needs to be invoked by us -   * if success comes **we need to call 1st function arg**, it means we are declaring as success flow- in that case “then(function)” method will be executed automatically with success arg to that function * if failure comes we need to call 2nd arg function - it means we are declaring as failure flow- in that case “catch(function)” method will be executed automatically | const promise1 = new Promise((resolve, reject) => {  setTimeout(() => {  //Here as we felt as success we are calling 1st arg function  resolve('foo');  }, 300);  });  promise1.then((value) => {  console.log(value);  // Expected output: "foo"  });  console.log(promise1);  // Expected output: [object Promise] |

My note:- I doubt this object is useless- because

* + 1. why I should call 1st cons arg, then based on that framework should considered as success and
    2. framework need to call then() method—why this much of indirect calling

Instead When success comes myself can directly call those 2 functions na,

It has 2 fields – promiseResult, promiseState

**States of a Promise:**

* **Pending:** Initial state, neither fulfilled nor rejected. (means promise hasn’t been executed yet)
* **Fulfilled:** Operation completed successfully.
* **Rejected:** Operation failed

Instance methods – same like java – try catch finally – here also then,catch,finally

|  |  |
| --- | --- |
| Promise then(function) | Then() method takes function as arg and this is **a callback function**, and this will be auto exec by framework when promise is success   * + Here u are calling then() with function as arg internally it might have coded like then   then(f){  f(); //here we are just invoking the function , bec function has been passed we need to invoke with args  } |
| Promise catch(function) | catch() method takes function as arg and this is a **callback function**, and this will be auto exec by framework when promise is failed |
| Promise then(fun1, function2) | Then() method takes 2 fun, 1st will be executed on success, 2nd fun will be executed on failure |
| Promise finally() |  |

Static methods

|  |  |
| --- | --- |
| Promise.all(p1,p2,p3)  Same like Thread.join() in java | Waits for multiple promises to resolve and returns an array of their results:  Promise.all([promise1, promise2, promise3])  .then(results => {  // Handle all results  })  .catch(error => {  // Handle error from any promise  }); |
| Promise.race() | Promise.race([promise1, promise2, promise3])  .then(result => {  // Handle the result from the first settled promise  })  .catch(error => {  // Handle the error from the first rejected promise  }); |
| Promise.any(p1,p2) | It takes multiple promises and returns single promise – this returned promise fulfils when any of the promise fulfils with its 1st fulfilment value |
| Promise.resolve(123); | Always success (here always promise state will be fulfilled) |
| Promise.reject | Always promise state will be rejected  function resolved(result) {  console.log('Resolved');  }  function rejected(result) {  console.error(result);  }  Promise.reject(new Error('fail')).then(resolved, rejected); |

**Why Use Promises?**

* **Cleaner code:** Avoids callback hell and makes asynchronous code more readable.
* **Error handling:** Provides a structured way to handle errors.
* **Asynchronous flow control:** Enables sequential and parallel execution of asynchronous operations.

Example

|  |  |
| --- | --- |
| <button onclick="p1()">Click here to exec promise</button>      <script>        function p1() {          //Here for below promise cons we are passing 2 parm anonymous fun          let p = new Promise(function (func1, func2) {            let r = false;            if (r) {     //Here we are calling 1st arg function to declaring promise as success              func1("Bro success Bro");            } else {   //Here we are calling 2nd arg function to declaring promise as failed              func2("Bro failed bro");            }          });          //for then method we are passing anonymous function          //This then will be executed only when promise is success          p.then(function (datah) {            console.log(datah, " from then method");          });          //This then will be executed only when promise is failed          p.catch(function (data1) {            console.log(data1, " from catch method ");          });        } | <button onclick="p1()">Click here to exec promise with XHR demo</button>      <script>        function p1() {          console.log("executing function as button click");          let pr = new Promise(function (arg1, arg2) {            let xhr = new XMLHttpRequest();            xhr.open("GET", "https://jsonplaceholder.typicode.com/users", true);            log("xhr status before send", xhr.readyState);            xhr.send();            xhr.onload = function () {              if (xhr.status == 200) {                log("xhr status after receiving data", xhr.readyState);                console.log("HTTP status after receiving data", xhr.status);                //by default we will get data as String - hence we need to conver to string                // log("response", xhr.responseText);                log("response", JSON.parse(xhr.responseText));   //If we got response as 200 then we can declare promise as success by invoking 1st arg fun                arg1(JSON.parse(xhr.responseText)); //after then promise.then() will be exec by js framework              } else {                log("http req failed , declaring promise as failed");                arg2("Bro exec failed");              }            };          });          //this then method will be exec by fw when promise is success          pr.then(function (data) {            log("Exec pr.then() as promise is success ", data);          });          //this catch method will be exec by fw when promise is failed          pr.catch((error) => {            log("in pr.catch() as promise exec is failed  ", error);          });          //this finally method will be exec by fw when promise is either success or failed          pr.finally(() => {            log("in pr.finally block");          });        }        function log(arg1, arg2) {          console.log(arg1, "-->", arg2);        }      </script> |

Fetch

**This is an alternative method to invoke REST apis using javascript promise objects**

**Promise<Response> Window.fetch(resource/URL) :** this is to make ajax call for GET method– same like XMLHttpRequest

**Promise< Response > Window.fetch(resource/URL ,options) -**  this is to make ajax call- for options we can call post method

Here options is also an object

//Here we will get Response object after invoking REST api

**Response**

**Instance methods**

|  |  |
| --- | --- |
| **Promise json()** | **here this json method is returning a Promise again** |
| **Promise text()** | **here this text() method is returning a Promise again** |

window

.fetch(myRequest)

.then((response) => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.blob();

})

|  |  |
| --- | --- |
| async function invokeRestAPI() {          //fetch generally returns a promise, await will wait for promise to resolve          let p = await window.fetch("https://jsonplaceholder.typicode.com/todos/1");          p.json().then((data) => {            console.log(data);          });          console.log(p);        }        invokeRestAPI(); | Sample response object   1. *Response {type: 'cors', url: 'https://jsonplaceholder.typicode.com/todos/1', redirected: false, status: 200, ok: true, …}*    1. body: (...)    2. bodyUsed: false    3. headers: Headers {}    4. ok: true    5. redirected: false    6. status: 200    7. statusText: ""    8. type: "cors"    9. url: "https://jsonplaceholder.typicode.com/todos/1" |
|  |  |

Fetch method return type is promise, so when promise is executed successfully, in that case “then()” method will be executed automatically

|  |  |  |
| --- | --- | --- |
| GET method demo (for get method we don’t need to pass options) | POST method demo |  |
| <button onclick="p1()">        Click here to use window.fetch() <br />        to hit REST api like XMLHttpRequest      </button>      <script>        function p1() {   //Here fetch method will return a promise, if succes then() will be called else catch()          //Here we will get Response object after invoking REST api          window            .fetch("https://jsonplaceholder.typicode.com/users/1")  //here we are passing func to then method, they will take this // arg and execute that function            .then(function s1(data) {              //Here we got response object              log("Got response object", data);   //Here the json() method also gives us promise hence we called then()              data.json().then(function (data) {                log("data is ", data);              });            })            .catch(function f1(data) {              log("get method");            });        }        function log(arg1, arg2) {          console.log(arg1, "-->", arg2);        }      </script> | fetch('https://api.example.com/users', {  method: 'POST',  headers: {  'Content-Type': 'application/json'  },  body: JSON.stringify({  name: 'John Doe',  email: 'johndoe@example.com'  })  })  .then(response => {  if (!response.ok) {  throw new Error('Network response was not ok');  }  return response.json();  })  .then(data => {  console.log(data);  })  .catch(error => {  console.error('Error:', error);  }); | Post method another demo  <button onclick="p1()">        Click here to use window.fetch() <br />        to hit REST api like XMLHttpRequest      </button>      <script>        function p1() {          //this fetch() method will return promise          window   .fetch("https://jsonplaceholder.typicode.com/posts", {              method: "POST",              body: JSON.stringify({                name: "kalki 2898 AD",                email: "prabas@28998gmail.com",              }),              headers: {                "Content-Type": "application/json",              },            })            .then((response) => response.json().then((r) => log(r)));   //in above the json method also returns promise        }        function log(arg1, arg2) {          console.log(arg1, "-->", arg2);        }      </script> |
|  |  |  |

Note:- Like java file api this fetch() method not only invoking api but also it can fetch data from file present in same server

Keywords - async, await

async function fetchData() {

const response = **await** fetch('https://api.example.com/data'); // Await – is nothing but blocking behaviour

const data = **await** response.json();

console.log(data);

}

Using async keywords we can change a function to asynchronous

(it may not run asynchronously like java but when you declare a function with async keyword **it will return the promise object instead of original obj**)

 **Makes asynchronous code look synchronous:** It allows you to write asynchronous code in a more synchronous style, making it easier to read and understand

async function s1() {} 🡪 now this function will be exec asynchronously and it will return a Promise not direct value

Await- in general await means wait for something to arrive- here in js world **await** keyword will wait for a promise to execute completely,

* It will pause the execution of function until the promise obj completes it execution
* Once promise is resolved, await will returns promise original value
* If promise is rejected an error is thrown
* await can be used on inside only in async functions

|  |  |
| --- | --- |
| Without await | With await |
|  | This await method will make that method to wait until promise get executed completely – to make it sync  If u use await keyword, then instead of promise u will get the direct response  If u don’t want to get return type as promise then use await keyword in front of promise return type |
| <script>        //an async keyword will make a function to return the promise obj instead of original obj        async function p1() {          log("1.exec js function as onclick fired");          let p = fetch("https://jsonplaceholder.typicode.com/users");          log("2.after fetch() REST invocation", p);          p.then((response) =>            //here json() method returns promise again hence then() is invoked again on json()            response.json().then((data) => log("3.no await data--", data))          );          log("4.promise defined");        }        let x = p1();        console.log(x);        console.log("5.fun p1 executed"); | console.log("----------------------- await demo");        async function f2await() {          log("1.exec f2await() function");          //Bec of await this promise will be executed after that only next statements will be exec          await fetch("https://jsonplaceholder.typicode.com/users").then((res) =>            res.json().then((data) => log("2.resp frmo await", data))          );          log("3.after fetch() REST invocation");        }        f2await();        function log(arg1, arg2) {          console.log(arg1, "-->", arg2);        }      </script> |