To master these front end technologies projects are must essential. Projects like clone projects, games portfolio etc

Comments in Java script:

```
console.log('Hello virat')
// console.log('virat is better than dhoni in batting')
```

For multiple lines you can use that **/* ---------- */**

console.log is used to print a message to the console.

Output:

```
[Running] node "c:\Users\chait\OneDrive\Desktop\programs\Web Development\JS
1\first\comments.js"
Hello virat
```

Variables in Java Script:

Variable are containers of a data.

Variables names are case sensitive is nothing but apple and Apple are different

var: variable can be re-declared & updated. A global scope variable.

let: Variable cannot be re-declared but can be updated. A block scope variable.

const: variable cannot be re-declared or updated. A block scope variable.(we can't change its fixed)

so we can update let variable but cant update constant, but we can update constant object key

```
// declaring variables
// The var keyword was used in all JavaScript code from 1995 to 2015.

// The let and const keywords were added to JavaScript in 2015.

// The var keyword should only be used in code written for older browsers.
// constant we will not able to change it
const age =18
// console.log(age)
console.log("Age =", age)
//var we are able to change values
var my_age =12
my_age = 14
console.log("my_age =:", my_age)
// let
let score = 30
score = 100
console.log("virat scored runs =", score)
```

```
//string
let username = 'Ab Deviliers'
console.log(username)
let greeting ='welcome '+username
console.log(greeting)

// number
let userage = 20, usernumber ='18'//number added as string
console.log(userage+18)
console.log(usernumber+19)
//boolean (True or False) & On or Off
is_qualified = false

//null
let user_bio = null  //no data has been assigned to it

//undefined
// console.log(myusername)

//naming variable
// username
// When to Use var, let, or const?
// 1. Always declare variables

// 2. Always use const if the value should not be changed

// 3. Always use const if the type should not be changed (Arrays and Objects)

// 4. Only use let if you can't use const

// 5. Only use var if you MUST support old browsers.
```

Output:

```
Age = 18
my_age =: 14
virat scored runs = 100
Ab Deviliers
welcome Ab Deviliers
38
1819
```

Operaters in java script:

These are used to perform a operation on data.

Arithmetic Operators,

Modulus operator --- %

Exponentiation → a **b is nothing but a power b

Increment a++, ++a

Decrement a--, --a

Assignment Operators: =,+=, -=, *= , % = , **=

A +=4 => a= a+4

Comparision Operators:

== , = ==, != , !==,>,>=,<,<=

For example a = 5,,number

b = “5” string

Console.log(“a ==b” , a ==b); it will print true because in java script sting will be converted in to number

=== menas it will check equal to & also data type

Logical operators: &&, ||, !

Ternary Operators:

It will work on 2 operands

Condition ? true output : false output

A ? B: C

A condition is true b is false then c will execute

If a condition is true and b is true then b will execute.

```
let age = 17;
let result = age >= 18? "adult":"notadult";
console.log(result)
```

or we can also write like this

```
let age = 17;
age >= 18? Console.log("adult") : console.log("notadult");
console.log(result)
```

refer MDN documents **https://developer.mozilla.org/en-US/docs/Web/JavaScript**

Alert: In java script alert is used to print a message. We will see this message as popup in website.

Prompt: Same like alert tag but it will display a message as well as take input from user.

```
//get user to input a number using promt ("Enter a number":) check if a number
is multiply of 5 or not?
let num = prompt("Enter a Number:");
```

```
console.log("your entered number is",num);
if(num%5 === 0){
    console.log("it will divided by 5");
}
else{
    console.log("Number is not multiple of 5");
}
```

Datatypes in Java Script:

7 primitive data types are there.

Number, String, Boolean, Undefined, Null , BigInt, Symbol

Non-primitive means objects.(Arrays,Functions)

If we use typeof any variable it will represent its data type

Loops in Java script:

Loops are used to execute a piece of code again & again

For (let I = 1; I <=5;i++){

    Console.log(“apna college”);

}

I =1 means initialization, stopping condition, updation i++

```
let sum =0;
for(let i=1;i<=6;i++){
sum = sum +i;
console.log("HI Aravind");
}
console.log("sum is equal to",sum);
```

```
HI Aravind
HI Aravind
HI Aravind
HI Aravind
HI Aravind
HI Aravind
sum is equal to 21
```

Infinite loop: A loop that never ends

Like stopping condition in a loop will always true this means loop never will stop

Like I >=0; your chrome will stuck

While loop in JS:

While(condition){ #initialization

    // do some work updation work doing under loop

}

```
let j =1;
```

```
while(j<=5){
    console.log("Hello naruto");
    j++;
}
```

```
5 Hello naruto
```

Do-while loop in JS:

do{

//do some work

}while(condition);

It will check condition after ending a loop

```
let k = 1;
do{
    console.log("Hello saksuke");
    k++;
}while(k<=5);
```

```
5 Hello saksuke
```

There 2 more loops which as for -of

And for -in

for-of loop:

for(let val of strVar{

//do some work

}

```
let str="Narutoshippuden";
let size =0;
for(let l of str){
    console.log("l=",l);
    size++;
}

console.log("string size=",size);
```

```
l= N
l= a
l= r
l= u
l= t
l= o
l= s
l= h
l= i
l= p
l= u
l= d
l= e
l= n
string size= 15
```

for-in loop:

for(let key in objVar){

//do some work

}

```
let student= {
    name: "Chaitanya",
    age:20,
    cgpa:8.9,
    ispass:true
};

for(let key in student){
    console.log("key=",key,"vale=", student[key]);
}
```

```
key= name vale= Chaitanya
key= age vale= 20
key= cgpa vale= 8.9
key= ispass vale= true
```

From this loop we can access key vale pairs

String methods in Java script:

```
//length
let greeting = "Welcome to my first project of jk"
console.log(greeting.length)
//uppercase
```

```
let upp = greeting.toUpperCase()
console.log("uppercase ="+upp)
//lowercase
let low =greeting.toLowerCase()
console.log("lowercase ="+low)
//index of character

console.log("index of o  is ="+greeting.indexOf('o'))
//or
Console.log(Greeting[0]);
//slicing
console.log("slicing of a greeting is"+greeting.slice(0,9))

//another way to use slicing
email ='viratwinipl2025@gmail.com'
userName =email.slice(0,email.indexOf('@'))
console.log(userName)
//split
console.log(greeting.split(' '))
//for example
fruits = 'Banana,Orange,Apple,mango'
console.log(fruits.split(','))

//number
let num = 45
console.log("Using number string ="+num.toString()+35)

//replace
console.log(greeting.replace('jk','java'))
//by using replaceall we can replace more values
```

Output :

```
33
uppercase =WELCOME TO MY FIRST PROJECT OF JK
lowercase =welcome to my first project of jk
index of o is =4
slicing of a greeting isWelcome t
viratwinipl2025
[ 'Welcome', 'to', 'my', 'first', 'project', 'of', 'jk']
[ 'Banana', 'Orange', 'Apple', 'mango']
Using number string =4535
Welcome to my first project of java
```

Template Literals in strings :

It will write in ` `

Let string =` This is a template literal`;

We can write like this

Let obj = {

Item: “gun”,

Price: 15,

};

Console.log(“the cost of “ , obj.item, “is”, obj.price , “rupees”);

Let output =`the cost of ${obj.iem}  is ${obj.price} rupees`;

Console.log(output);

String Interpolation :

To create strings by doing substitution of place holders.

`string text &{expression} string text`

Let string =` This is a template literal  ${5+6+9}`; here all num will be calculated into value it will be noted into string

String concatenation:

Let str1 =”virat”;

Let str2 = “kohli”;

Let res = str2.concat(str1);

Console.log(res);

str.charAt(idx)

Practice questn: Promt the user to enter their full name. Generate a user name for them based on input. Start username with @, followed by their full name and ending with fullname length.

```
let username = prompt("enter a name");
console.log("Your usename is @"+ username+ username.length);
```

Functions in JS:

Block of code that performs a specific task, can be invoked whenever needed

We can write arrow functions

const sum = (a,b) =>{

console.log(a+b);

}

This is an arrow function.

Here only right side part is only arrow function
remaining part is stored as variable like ‘sum’

Arrow function uses for small functions only

```
//creating functions
function greetings(){
    let username = "virat kohli"

    console.log("Welcome"+" "+username)
}


//calling function
greetings();

//Arrow function
let greeting = () => {
    let user= "Ab Devilliers"
    console.log(user)
}

greeting()
//Arguements, returning a value

let calculate = (x,y) =>{
    console.log("x is ",x)
    console.log("y is ",y)
    return x*y
}
// we can write above function short cut as
// let calculate = (x,y) => x * y
let total = calculate(20,5)
console.log("x*y is "+total)

let welcome = (userName) => "welcome"+" "+ userName
let userName ="Maxwell"
let welcomeuser = welcome(userName)
console.log(welcomeuser)
```

More on the Arrow Function Syntax

When working with **Arrow Functions**, you have a couple of *"syntax shortcuts"* available.

Most importantly, you should know about the following alternatives:

1) Omitting parameter list parentheses

If your arrow functions takes **exactly one parameter**, you may **omit** the wrapping parentheses.

Instead of

```
(userName) => { ... }
```

you could write

```
userName => { ... }
```

Please note:

- If your function takes **no parameters**, parentheses **must not be omitted** - `() => { ... }` is the **only correct form** in that case.
- If your function takes **more than one parameter**, you also **must not omit** parentheses - `userName, userAge => { ... }` would be invalid (`(userName, userAge) => { ... }` is correct)!

2) Omitting function body curly braces

If your arrow function contains **no other logic but a `return` statement**, you may **omit the curly braces** and the `return` keyword.

Instead of

Instead of

```
number => {
  return number * 3;
}
```

you could write

```
number => number * 3;
```

The following code would be invalid:

```
number => return number * 3; // invalid because return keyword must also be omitted!
number => if (number === 2) { return 5 }; // invalid because if statements can't be returned
```

3) Special case: Just returning an object

If you go for the shorter alternative explained in 2) and you're trying to return a **JavaScript object**, you may end up with the following, **invalid** code:

```
number => { age: number }; // trying to return an object
```

This code would be invalid because JavaScript treats the curly braces as **function body wrappers** (not as code that creates a JS object).

To *"tell"* JavaScript that an object should be created (and returned) instead, the code would need to be adjusted like this:

```
number => ({ age: number }); // wrapping the object in extra parentheses
```

By wrapping the object and its curly braces with an **extra pair of parentheses**, JavaScript understands that the curly braces are not there to define a function body but instead to create an object. Hence that object then gets returned.

```
function mul(a,b) {
    return a*b;
}

let mularr = (a,b) => {
    console.log(a*b);
}

// we can also return values in arrow functions

const arrmul = (a,b) =>{
    return a*b;
}
```

We can aloo declare any arrow functions like a variable as

arrmul =5

Output:

```
Welcome virat kohli
Ab Devilliers
x is  20
y is  5
```

```
x*y is 100
welcome Maxwell
```

Write a program to check number of vowels in a given string

```
function vowelcnt(str){
    let count=0;
    for(const char of str){
        if( char === "a"||
            char ==="i" ||
            char ==="e" ||
            char ==="o" ||
            char ==="u"){
            count++;
        }

    }
    console.log("no of vowels in a given string is ="+count)
}
```

forEach Loop in Arrays: its main purpose is to create a loop

arr.foreach(callBackfunction)

CallbackFunction: Here, it is a function to execute for each element in a array.

A call back is a function passed as an argument to another function.

If any function is attached with object then it will link as method

arr.forEach((val) => {

console.log(val);

})

We will for each when we want to access every element in array since we used it for array right

In java script we can pass a function as a parameter and we can also return these functions in java script

Eg:

function abc() {

console.log(“Hello”);

}

Function myFun(abc){

console.log(abc);

}

arr.forEach((val) =>{

console.log(val);

}

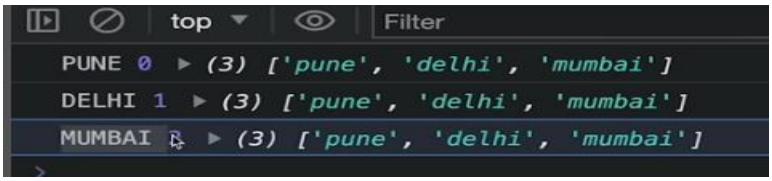
We can use 3 parameters. Like value/item

Index,

Array

Like wise

```
arr.forEach((val, idx, arr) => {
  console.log(val.toUpperCase(), idx, arr);
});
```

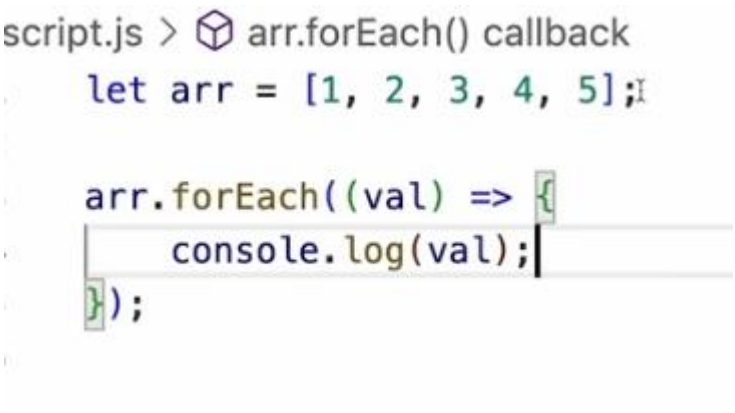
Output: 

We will get questions in interview like higher order functions/methods like what

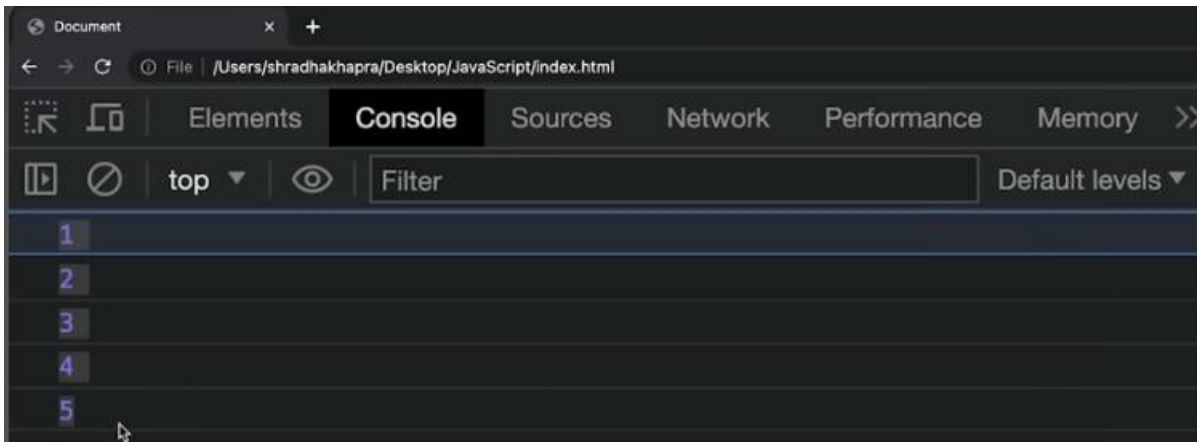
Higher order functions are nothing but they can use other functions as a either parameter or return function as value

```
function myfunc(num) {
    return num;
}
// it is common we can use function to use as a parameters and we can return
that parameter as a function

// in same if we create another function
function abc(){
    console.log("hello");
}

// we can use that function as a parameter
function myFunc(abc){
    return abc;
}

// A call back is a function passed as an argument to another function.

let arr = [1,2,3,4,5];
arr.forEach(function printval(val) {// use of value parameter is value at each
index
    console.log(val);
})
```

Similarlly we can write it as a arrow function



Val output:



Program for a given array of numbers print the square of each value using the forEach loop.

```
array.forEach((val) => {
    let num =val*val;
    console.log(num);
});
```

Output: 324,289,361

We can also write like like this as call back function

let nums= [12,13,15];

let calsquare = (num) => {

    console.log(num*num);

};

nums.forEach(calsquare);

Some More Array Methods:

Map

It creates a new array with results of same operation. The value its callback returns are used to form new array.

Arr.map(callbackFnx(value,idx,array))

```
let nums = [43,54,66];

nums.map((val) => {
       console.log(val);
});
```

From above if we return any function we can create a array with it.

```
let newArray = nums.map((val) => {
    return val;
});
console.log(newArray);
```

it means it will return any index value and its stored as array.

Filter Method:

Creates a new array of elements that give true for a condition/filter.

Eg: even numbers

let newArr = arr.filter((val) =>{

    return val% 2===0;

})

```
// WAP to return students who scored greater than 90 + score using filter
method
let marks =[87,94,93,76,99,100,89];

let output = marks.filter((val)=>{
    return val>90;
});
console.log(output);
```

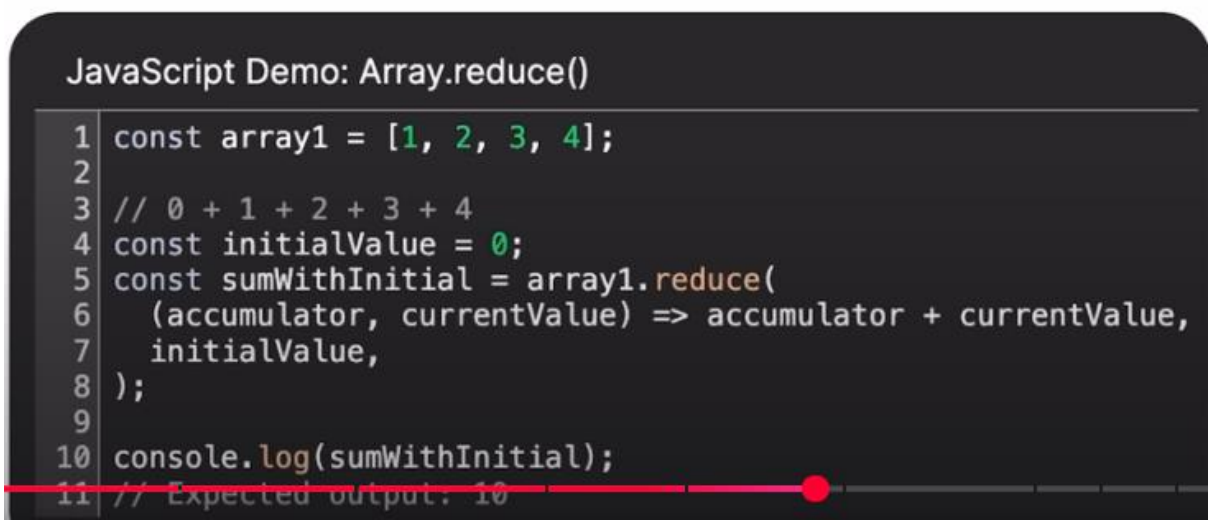
Reduce Method:

Performs some operations and reduces the array to a single value. It returns that single value.

```
const a1 = [1,3,4,5,6];

const opt= a1.reduce((res,cur)=>{
    return res+cur;
})
console.log(opt);
```

output: 21

here the logic is res is starting from 1 current is 3 then result becomes to 4 and current moves next item in an array.



Similarly if we want to find out the largest element in an array elements

We write like

```
const a1 = [1,3,4,5,6];

const opt= a1.reduce((prev,cur)=>{
    return prev>cur? prev:cur; //it means true means prev will execute
otherwise cur
})
console.log(opt);
```

```
let n = prompt("enter a number=");

let arrn =[];

for(let i=1;i<=n;i++){
    arrn[i-1]=i; //1(0), 2(1),3(2)  it means 1  is stored in 0 index
viceversa.
}
console.log(arrn);

let sumn = arrn.reduce((res,cur)=>{
    return res+cur;
});

let factorial = arrn.reduce((res,cur)=>{
    return res*cur;
})
console.log(sumn);
console.log(factorial);
```

So if we 3

Conditional statements in Java script:

```
// comparision operatures
// equal to ==,  not equal to as !=, > ,<, >= ,<=


// if and else statements
let num = 20
if(num == 20){
    console.log("number is matched with 20")
}
else if(num == 13){
    console.log("number is equal to 13")
}
else if(num > 20){
    console.log("number is greater than 20")
}
else{
    console.log("number is not matching")
}

// we can also check for string in if else conditions
user = "Virat kohli"

 if (user !="Chaitanya"){
    console.log("Hello "+user)
}
//switch
day = 'Monday'
switch (day){
    case 'Monday':
        console.log("This is monday")
        break;
    case 'Tuesday':
        console.log("this is friday")
        break
    default:
        console.log('week consists 7 days')
}
```

Output:

```
number is matched with 20
Hello Virat kohli
This is monday
```

Practice question: create a game where you start with any random given number. Ask the user to keep guessing the game number until the user enters correct value

```
let num = prompt("Enter a number from 1 to 100:");

let etrnum = 18;
while(num != etrnum){
    // console.log("enter a number again");
    num = prompt("enter number again to match your number because your
previous numbr didn't matched");
}

console.log("Congratulations you have matched yor number");
```

Arrays in java script:

```
//creating arrays

users =[
    'Chaitanya Bejjanki',
    'Virat kohli',
    'Ab Deviliers'
]
console.log(users)
console.log("Length:", users.length)

//accessing elements
console.log(users[0])
// for printing last element in a array in js
console.log(users[users.length-1])

//adding and removing elements
users.push('Glenn Maxwell') //adding element uses push
console.log(users)
users.pop('Glenn Maxwell')
console.log(users)

//accessing all elements
//mapping
users.map(doc =>{
    console.log(doc)
})
```

Output:

```
[ 'Chaitanya Bejjanki', 'Virat kohli', 'Ab Deviliers' ]
Length: 3
Chaitanya Bejjanki
Ab Deviliers
```

```
[
  'Chaitanya Bejjanki',
  'Virat kohli',
  'Ab Deviliers',
  'Glenn Maxwell'
]
[ 'Chaitanya Bejjanki', 'Virat kohli', 'Ab Deviliers' ]
Chaitanya Bejjanki
Virat kohli
Ab Deviliers
```

Arrays in java script: it is nothing but collection of items

They behave like objects

Like value pair as index value

```
In an array we can store different data types also
```

Array Indices:

arr[0], arr[1], arr[2]

Strings in java script is inmutable

In arrays in js is mutable

Looping over in Array:

Print all elements in array

Loops --> iterable (Strings, objects, arrays)

For loop using through length property

For(idx=0; idx < arr.length; idx++){

For printing elements in a array we can use for-in or for-of loop

```
// for example if we want to store a names of 5 students in a class and there
marks also
let classes =["Virat","Ab Devilliers", "Ms Dhoni", "Sachin", "Rohit"];
// In object key was mainly matters where as in arrays position/index.
console.log(classes);
console.log(classes.length);

for(let idx =0;idx<classes.length;idx++){
    console.log(classes[idx]);
}

//for-of loop
for (let cricketer of classes){
```

```
    console.log(cricketer);
    //we can print with properties also
    console.log(cricketer.toUpperCase());
}

// For a given array with marks of students --> [85,97,44,37,76,60]
// find average of entire class.

let students = [85,97,44,37,76,60];
let sum=0;
for(let i=0;i<students.length;i++){

    sum = sum + students[i];
}
let avg = sum/students.length;
console.log("average of students marks are ",avg);
```

Output:

```
▼ (5) ['Virat', 'Ab Devilliers', 'Ms Dhoni', 'Sachin', 'Rohit']
    0: "Virat"
    1: "Ab Devilliers"
    2: "Ms Dhoni"
    3: "Sachin"
    4: "Rohit"
    length: 5
  ▶ [[Prototype]]: Array(0)
5
Virat
VIRAT
Ab Devilliers
AB DEVILLIERS
Ms Dhoni
MS DHONI
Sachin
SACHIN
Rohit
ROHIT
average of students marks are  66.5
```

ARRAY METHODS:

Push(): add to end

Push has two methods one changes with array, other one not changes array return new array

Pop() : delete from end & return

toString() : converts array to string

```
let fooditems = ["potato", "apples","tomato"];
// we can add at console as
// fooditems.push("chips"); like wise
// we can multiple items also like fooditems.push("chips","chocos","sweets");

fooditems.pop();
// then it will remove last elemnt from js

//  we can return deleted value also as

let delvalue = fooditems.pop();
console.log("deleted",delvalue);

// toString used to convert an array into string
console.log(fooditems.toString());
```

Concat(): Joins multiple arrays & returns results

```
let rcb = ["virat","siraj","abd","patidar"];
let lsg =["rahul","pooran","paddikal"];

let ipl = rcb.concat(lsg);
console.log(ipl);
```

output:

```
[
  'virat',    'siraj',
  'abd',      'patidar',
  'rahul',    'pooran',
  'paddikal'
]
```

We can concatenate another array also like , let mi is also array

```
let ipl = rcb.concat(lsg,mi);
```

like wise

Unshift: it was push method as add from the start

Shift: it was like pop method delete from start

Slice() : we can write like

Console.log(ipl.slice());

Slice(startidx, endidx)

Splice() :

Change original array (add,remove,replace)

Splice(startIdx, delCount, newEl1)

For example if we have an array consist

A= [1,2,3,4,5,6,7]

Splice[2,3

After index if we write 3 3 elements will delete after index 2.

If we wont to delete any element we can represent like

Splice[2,0]

If we write splice(2,2,101,102)

A= [1,2,101,102,5,6,7] replaced with 101,102

We can use splice add a element in index 2 with deleting that element we can use as

Splice(2,0,101)

A= [1,2,101,3,4,5,6,7]

findIndex method it will used to find our index value at which index.

```
let classes = ["Virat","Ab Devilliers", "Ms Dhoni", "Sachin", "Rohit"];
const index = classes.findIndex((item)=>{
    return item === ”Sachin”;
});

console.log(index);
```

o/p: 3

DOM: Document Object Model

Html connects css & js by using 2 tags style & script’

Style => html with css

Script => html with js

When a page is loaded browser creates a DOM of page.

DOM Manipulation:

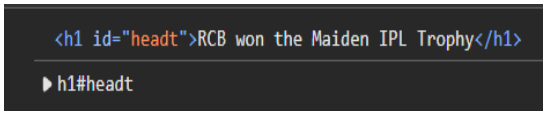
Selecting by ID

It has to be unique

document.getElementById(“myId”);

```
let opener = document.getElementById("headt");
console.log(opener);
console.dir(opener);
```

o/p:

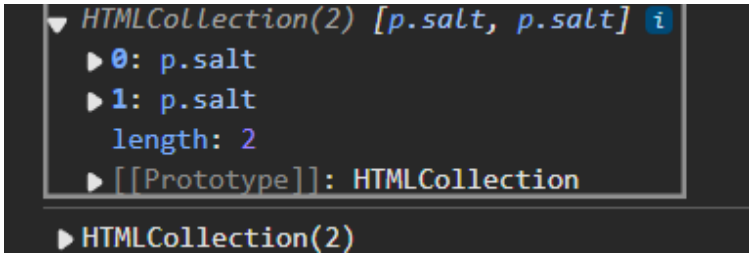


Selecting with class:

document.getElementsByClassName(“myclass”);

it will return html collections which is similar to an array.

```
let opener2 = document.getElementsByClassName("salt");
console.log(opener2);
console.dir(opener2);
```



Selecting with tag:

document.getElementsByTagName(“p”);

To level up these selectors we need to move on to Query Selectors

Query Selector:

In query selector we can pass any element like id,class or tag

document.querySelector(“id / class/ tag”);

It will return the first element.

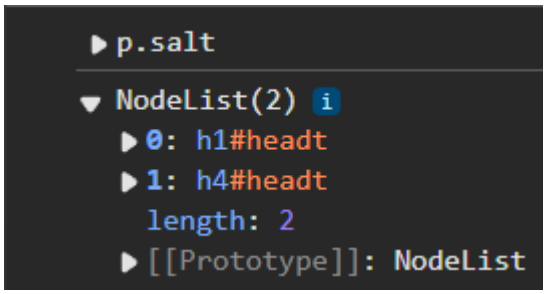
If want to use all elements we can use query selector all.

document.querySelectorAll((“id / class/ tag);

we need to declare class name in query selector as **.class** name.

we need to declare id name in query selector as **#id** name.

```
let firsel = document.querySelector("p");
console.dir(firsel);

let allel = document.querySelectorAll("#headt");
console.dir(allel);
```



to print an object we will use **console.dir**

we can change temporarily entire whole background of a page using dom.

document.body.style.background =”green”;

or we can write like this

document.body.childNodes[2].innerText =”abcd”;

There is also a **properties**,

tagName : returns tag for element nodes

like

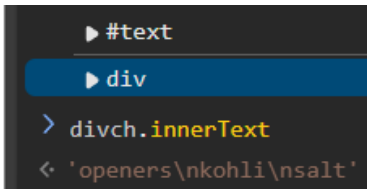
```
firsel.tagName
```

innerText : returns the text content of the element and all its children

In DOM we have three nodes 1st one Text node 2nd comment nodes 3rd element nodes.

Most imp one are elemt nodes

```
let divch = document.querySelector("div");
console.dir(divch);
```



innerHTML : returns the plain text or HTML contents in the element

```
▶ #text
▶ div
> divch.innerText
'openers\nkohli\nsalt'
> divch.innerHTML
'\n        <h4>openers</h4>\n        <ul>\n
<li>salt</li>\n        </ul>\n    '
> divch.innerText= "Virat Kohli";
'Virat Kohli'
> divch.innerHTML = "<div> virat </div>"
'<div> virat </div>'
>
```

textContent : returns textual content even for hidden elements

```
    <h1 id="headt" style="visibility: hidden;">RCB won the Maiden IPL
Trophy</h1>
```

```
let opener = document.getElementById("headt");
console.log(opener);
```

```
> opener.innerText
''
> opener.textContent
'RCB won the Maiden IPL Trophy'
```

By using these properties we can get values (check) or else set (change/update) values.

Practice create a H2 heading element with text “Hello JavaScript”. Append “from Apna College students” to this text using in js.

```
<body>
    <h2 id ="headt">Hello JavaScript</h2>
    <div class = "box">first div</div>
    <div class = "box">second div</div>
    <div class = "box">third div</div>
    <style>
        .box{
            height: 100px;
            width: 100px;
            margin: 5px;
            border: 1px solid black;
            text-align: center;
            background-color: aquamarine;
        }
    </style>
    <script src="newpract.js"></script>
</body>
```

```
let newh = document.querySelector("#headt");
console.dir(newh.innerText);
newh.innerText = newh.innerText+" from apna college students";

let divs = document.querySelectorAll(".box");
let idx=1;
for(div of divs){
    div.innerText = `unique value of ${idx}`;
    idx++;
}
console.log(divs[0].innerText);
```

Attributes:

These are nothing but any characters which are created under a node

for ex:

```
<div class = "box">first div</div>
```

Here class is an attribute

getAttribute(attr) //to get the attribute value

```
    <div class = "box">first div</div>
```

```
let newd = document.querySelector("div");
console.log(newd.getAttribute("class"));
```

o/p: box

setAttribute(attr, value) //to set the attribute value

```
let newd = document.querySelector("div");
console.log(newd.setAttribute("class",”Newclass”));
```

node.style: it means we can change style of any node.

```
let newd = document.querySelector("div");
console.log(newd.style.backgroundColor = "green");
```

o/p:

But here in js css style properties will be converted like this

background-color → backgroundColor

font-size → fontSize

we can change styles of any elements using a js properties as well

Insert Elements let el = document.createElement("div")

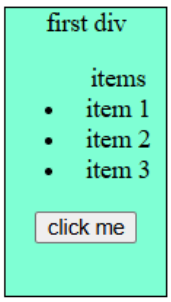
node.append(el) //adds at the end of node (inside)

```
<div class = "box">first div
        <ul>items
            <li>item 1</li>
            <li>item 2</li>
            <li>item 3</li>

        </ul>
    </div>
```

```
let newBtn = document.createElement("button");
newBtn.innerText = "click me";
console.log(newBtn);

let newd = document.querySelector("div");
newd.append(newBtn);
```

so we added button in div through append

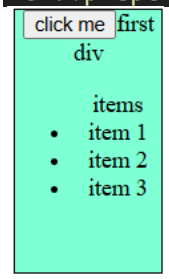
o/p:



node.prepend(el) //adds at the start of node (inside)

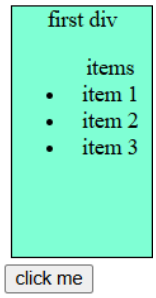
to add newly created button at starting of an element tag

```
newd.prepend(newBtn);
```



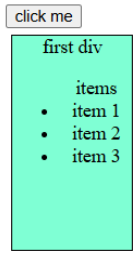
node.before(el) //adds before the node (outside)

```
newd.after(newBtn);
```



node.after(el) //adds after the node (outside)

```
newd.before(newBtn);
```



Delete Element

node.remove( ) //removes the node

Practice questions:

Create a new button element. Give it a text “click me”, background color of red & text color of white. Insert the button as the first element inside the body tag.

```
let newBtn = document.createElement("button");
newBtn.innerText = "click me";
newBtn.style.color="white";
newBtn.style.backgroundColor = "red";

let newb = document.querySelector("body");
newb.prepend(newBtn);
```

Qs. Create a tag in html, give it a class & some styling. Now create a new class in CSS and try to append this class to the

element. Did you notice, how you overwrite the class name when you add a new one? Solve this problem using classList.

```
<p class="own">Lorem ipsum dolor sit amet consectetur, adipisicing elit.
Fugiat reiciendis eligendi debitis. Reprehenderit labore et pariatur quas
eos!</p>
```

```
let ncl = document.createElement("class");
ncl.innerText ="newclass";

let p = document.querySelector("p");
p.append(ncl);
```

```
.own{
            color: aqua;
            background-color: aliceblue;
            font-size: x-large;
        }
        .newclass{
            background-color: yellow;

        }
```

```
> p
< ▶<p class="own" own="value">…</p>
> p.getAttribute("class")
< 'own'
> p.setAttribute("class","newclass")
< undefined
```

classList Property: It is a collection of class attributes of an element.

Events in JS:

The change in the state of an object is known as an Event

```
    <button onclick="console.log('button was clicked');alert('Hello!')">click
me!</button>
```

In this case, we need to write the string using single quotation marks inside the double quotations. After triggering any event, if we are able to perform some work, we can say that **event handling** has occurred.

```
<button ondblclick="console.log('button was clicked 2x
times');alert('Hello')">click me 2 times!</button>
```

```
    <div onmouseover="console.log('you are on inside div')">This is a
box</div>
```

These properties we can consider them as **Inline event handling.**

Events are fired to notify code of "interesting changes" that may affect code execution.

- Mouse events (click, double click etc.)
- Keyboard events (keypress, keyup, keydown)
- Form events (submit etc.)
- Print event & many more

Instead of making html code complex for using events we can use java script

node.event = () =>{

}

```
let btn1 = document.querySelector("#bt1");

let b1 = btn1.onclick =() =>{
    console.log("button was clicked");
    alert("Hello!");
}
// console.log(b1);

let di = document.querySelector("div");

di.onmouseover =() =>{
    console.log("you are on inside div");
}
```

This is **Java script handling**

Here priority will be given to Java script handling only than inline handling

If we handled one property then we can't do same handling because it will **overwrite** existing handling.

Event Object:

It is a special object that has details about the event.

All event handlers have access to the Event Object's properties and methods.

node.event = (e) => {

    //handle here

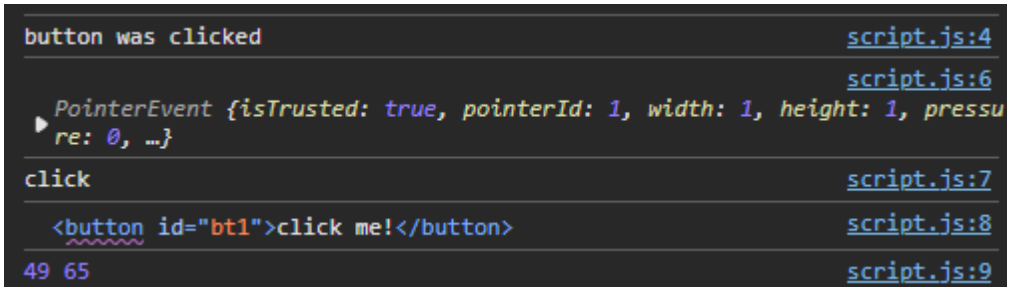
}

e.target, e.type, e.clientX, e.clientY

```
btn1.onclick =(event) =>{
```

```
    console.log("button was clicked");
    alert("Hello!");
    console.log(event);
    console.log(event.type);
    console.log(event.target);
    console.log(event.clientX,event.clientY);
}
```

O/p:



Event Listeners:

These are better than those two methods (inland & JS handling)

node.addEventListener(event, callback)

Here callback is nothing but it as a function which as passed as a argument in other function.

node.removeEventListener(event, callback)

*Note : the callback reference should be same to remove

```
btn1.addEventListener("click",()=>{
    console.log("button was clicked");
});

btn1.addEventListener("click",()=>{
    console.log("button was clicked --handler2");
});

btn1.removeEventListener("click", ()=>{
    console.log("button was clicked --handler2");
});
//to remove above arrow function we neet declare above aroow function with
const
const handler3 = ()=>{
    console.log("button was clicked --handler3");
}

btn1.addEventListener("click", handler3);
btn1.removeEventListener("click", handler3);
```

O/p:

button was clicked

button was clicked --handler2

Create a toggle button that changes the screen to dark-mode when clicked & light-mode when clicked again?

Classes & Objects:

Prototypes in JS: A javaScript object is an entity having state and behavior (properties and method).

JS objects have a special property called prototype.

We can set prototype using _ _ proto _ _

```
// using of prototype
const employee = {
    calavg(){
        console.log("the avg is 51.16")
    }
};

const karannair = {
    runs:4000
};

const virat = {
    runs:37000
};
const sachin = {
    runs:50000
};
const MSD = {
    runs:27000
};

karannair.__proto__ = employee;
virat.__proto__ = employee;
sachin.__proto__ = employee;
MSD.__proto__ = employee;
```

O/P:

```
> virat.calavg()
the avg is 51.16
```

```
// using of prototype
const employee = {
    calavg(){
        console.log("the avg is 51.16")
    }
};

const karannair = {
    runs:4000,
    calavg(){
        console.log("the avg is 75.8%");
    }
};
```

```
> karannair.calavg()
the avg is 75.8%
```

We will get output of only in karannair object method only.

By using prototype we used as a object

*If object & prototype have same method, object's method will be used.

Objects in Java script:

Objects are nothing but a key value pairs

We can retrieve through either obj.key or obj["key"]

```
//creating objects
let user = {
    'firstName':'Chaitu',
    'lastName':'Bejjanki',
    'email':'chaitanyabejjanki@gmail.com',
    'fullname': function(){
        return this.firstName+" "+this.lastName
    }
}
console.log(user)

//Accessing elements
let email,firstName,lastName,fullname
email = user.email
firstName =user.firstName
// we can also write like it
firstName = user['firstName']
lastName = user.lastName
console.log("first name is :"+firstName)
```

```
// you can write like this
// console.log("first name :", firstName)
fullName = user.fullname()
console.log(fullName)

//adding and modifying elements
user.address ='3440 S Cottege grove'
console.log(user)

user.firstName="Chaitanya"
console.log(user)
```

Output:

```
{
  firstName: 'Chaitu',
  lastName: 'Bejjanki',
  email: 'chaitanyabejjanki@gmail.com',
  fullname: [Function: fullname]
}
first name is :Chaitu
Chaitu Bejjanki
{
  firstName: 'Chaitu',
  lastName: 'Bejjanki',
  email: 'chaitanyabejjanki@gmail.com',
  fullname: [Function: fullname],
  address: '3440 S Cottage grove'
}
{
  firstName: 'Chaitanya',
  lastName: 'Bejjanki',
  email: 'chaitanyabejjanki@gmail.com',
  fullname: [Function: fullname],
  address: '3440 S Cottege grove'
}
```

Classes:

Class is a program-code template for creating objects.

Those objects will have some state (variables) & some behaviour (functions) inside it.

```
class MyClass {
   constructor( ) { ... }
   myMethod( ) { ... }
}
```

let myObj = new MyClass( ) ;

```
class Toyotacar{
    start(){
        console.log("start");
    }
    stop(){
        console.log("stop");
    }
}

let rav4 = new Toyotacar();
let fortuner = new Toyotacar();
```

Constructor( ) method is :

automatically invoked by new

this means we doesn't need to create it, it automatically create a constructor

initializes object

class MyClass {

  constructor( ) { ... }

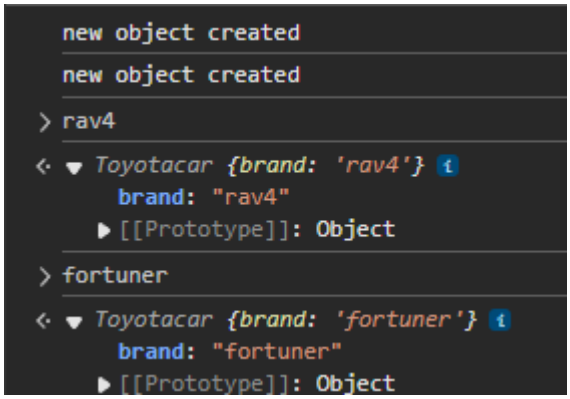
  myMethod( ) { ... }

}

```
class Toyotacar{
    constructor(brand){
        console.log("new object created");
        this.brand = brand;
    }
    start(){
        console.log("start");
    }
    stop(){
        console.log("stop");
    }
    // setBrand(brand){
```

```
    //      this.brandName = brand;
    // }
}

let rav4 = new Toyotacar("rav4");
// rav4.setBrand("rav4");
let fortuner = new Toyotacar("fortuner");
```

O/P:



Inheritance:

Inheritance in JS inheritance is passing down properties & methods from parent class to child class.

class Parent {

}

class Child extends Parent {

}

```
class Person{
    eat(){
        console.log("Eat ")
    }
    sleep(){
        console.log("It's time for your sleep");
    }
}

class Engineer extends Person{
    work(){
        console.log("start working, build some code for client");
    }
}

let aravindobj = new Engineer();
```

O/p:

```
> aravindobj.work()
  start working, build some code for client
< undefined
> aravindobj.eat()
  Eat
```

If we have same methods in parent & child class then java script will return output of child class method. (**Child class method will invoke**)

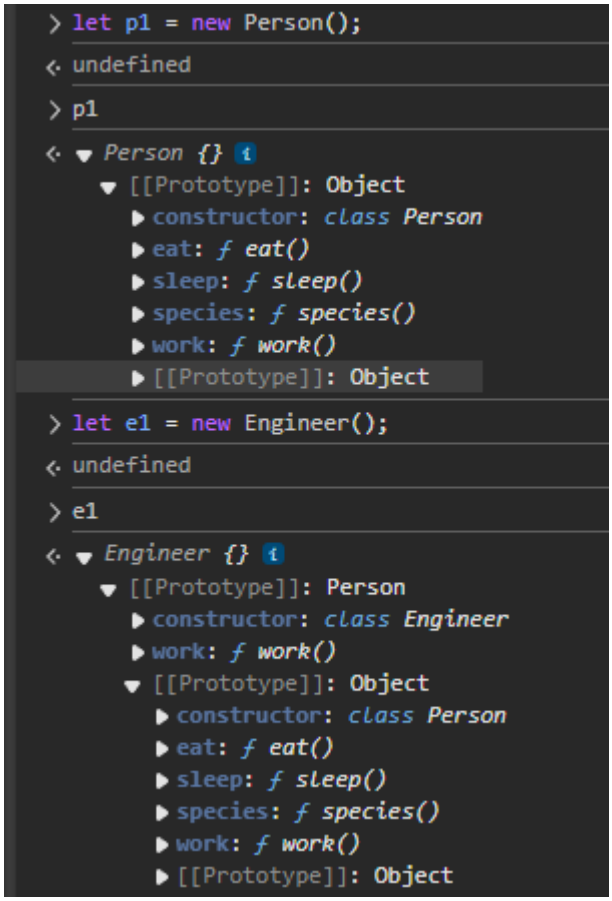
```
class Person{
    eat(){
        console.log("Eat ")
    }
    sleep(){
        console.log("It's time for your sleep");
    }
    work(){
        console.log("do nothing");
    }
}
```

```
> aravindobj.work()
  start working, build some code for client
```

***If Child & Parent have same method, child's method will be used. [Method Overriding]**

Let if we add a constructor in Person class

We will get all properties from child class.



Super:

The super keyword is used to call the constructor of its parent class to access the parent's properties and methods.

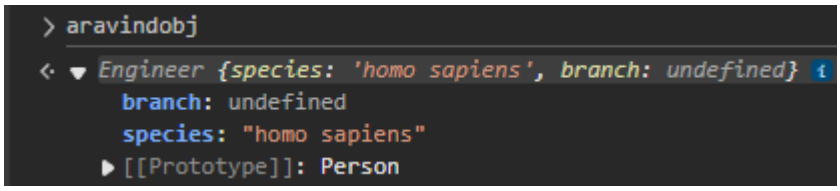
super(args) // calls Parent's constructor

super.parentMethod(args)

```
class Person{
    constructor(){
        // console.log("homo sapiens");
        this.species = "homo sapiens";
    }
    eat(){
        console.log("Eat ")
    }
    sleep(){
        console.log("It's time for your sleep");
    }
}

class Engineer extends Person{
```

```
    constructor (branch){
        super(); // to invoke parent class constructor
        this.branch=branch;
    }
    work(){
        console.log("start working, build some code for client");
    }
}

let aravindobj = new Engineer("computer science");
```

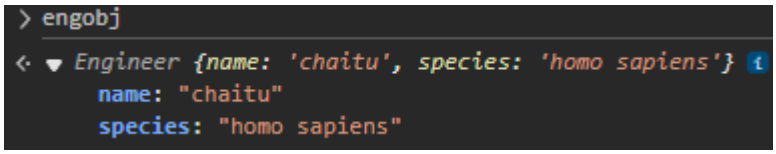
O/P:



If we want to pass another argument from our child class constructor to the parent class constructor using **super(name)**

```
class Person {
    constructor(name){
        this.name = name;
        // console.log("homo sapiens");
        this.species = "homo sapiens";
    }
    eat(){
        console.log("Eat ")
    }
    sleep(){
        console.log("It's time for your sleep");
    }
}

class Engineer extends Person{
    constructor (name){
        super(name); // to invoke parent class constructor
        this.name=name;
    }
    work(){
        console.log("start working, build some code for client");
    }
}
let engobj = new Engineer("chaitu");
```

O/p:



We can access any parent class method in child class using super.method();

Eg:

In child class

Work(){

Super.eat();

```
console.log("start working, build some code for client");
```

}

Super.eat(); }

Practice:

You are creating a website for your college. Create a class User with 2 properties, name & email. It also has a method called viewData( ) that allows user to view website data.

```
let DATA = "secret Information";
class user{
    constructor(name,email){
        this.name=name;
        this.email=email
    }
    viewdata(){
        console.log("website data is ")
    }
}

let obj = new user("chaitanya","chaitanya@gmail.com");

class admin extends user{
    constructor(name,email){
        super(name,email);
    }
    editdata(){
        DATA = "some new value";
    }
}

let admin1 = new admin("admin1","admin1@gmail.com");
```

What this chapter is about?

async await >> promise chains >> callback hell

Sync in JS

Synchronous: Synchronous means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

Asynchronous Due to synchronous programming, sometimes imp instructions get blocked due to some previous instructions, which causes a delay in the UI.

Asynchronous code execution allows to execute next instructions immediately and doesn't block the flow.

We can use setTimeout() to decide the function to exeute in how long

Callbacks

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

```
function sum(a,b) {
    console.log(a+b);
}

function calc(a,b,sumCallback){
    sumCallback(a,b);
}

calc(18,17,sum);
```

```
//callback example

const hello = ()=>{
    console.log("Hello! Hi How Are You?")
}

setTimeout(hello,3000);
//this is example of callback hello function output will print after 3 sec
```

Callback Hell

Callback Hell : Nested callbacks stacked below one another forming a pyramid structure. (Pyramid of Doom)

This style of programming becomes difficult to understand & manage.

```
function getdata(data, getNextdata){
    setTimeout(()=>{
        console.log("your data is",data);
        if(getNextdata){
            getNextdata();
        }
```

```
    },2000);
}

// we can declare a callback function in function like wise
getdata(1,()=>{
    getdata(2)
});
```

This is how we need to declare a call back function.

```
getdata(1,()=>{
    getdata(2,()=>{
        getdata(3);
    });
});
```

For this we will get output as

Your data is 1

Your Data is 2

Your Data is 3

For every 2 sec we will get output

Promises :

Promise is for “eventual” completion of task.

It is an object in JS. It is a solution to callback hell.

let promise = new Promise((resolve, reject) => { .... })

Function with 2 handlers

*resolve & reject are callbacks provided by JS

Promises A JavaScript Promise object can be:

- Pending : the result is undefined
- Resolved : the result is a value (fulfilled) resolve(result)
- Rejected : the result is an error object

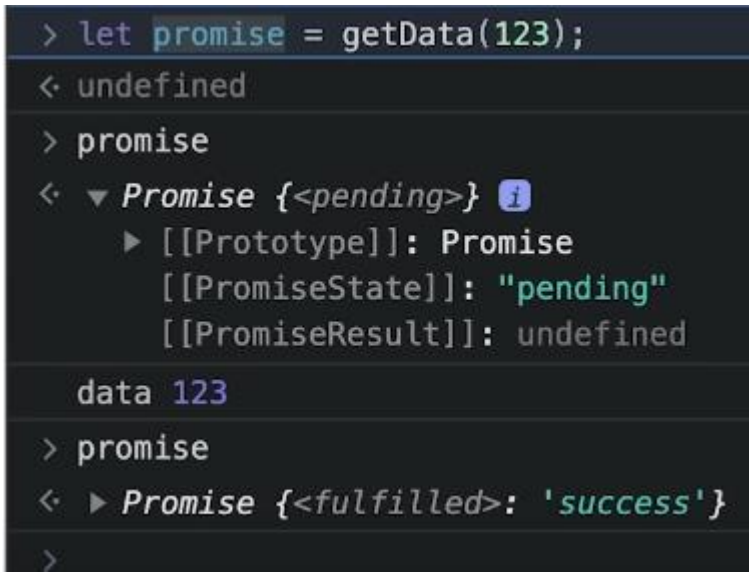
*Promise has state (pending, fulfilled) & some result (result for resolve & error for reject)

Promise can be in 3 states as Pending, Fullfilled, Rejected

Resolve:

```
function getdata(data, getNextdata){
    return new promice ((resolve,reject)=>{
        setTimeout(()=>{
            console.log("data",data);
            resolve("success");
            if(getNextdata){
                getNextdata();
            }

        },2000);
    });
}
```

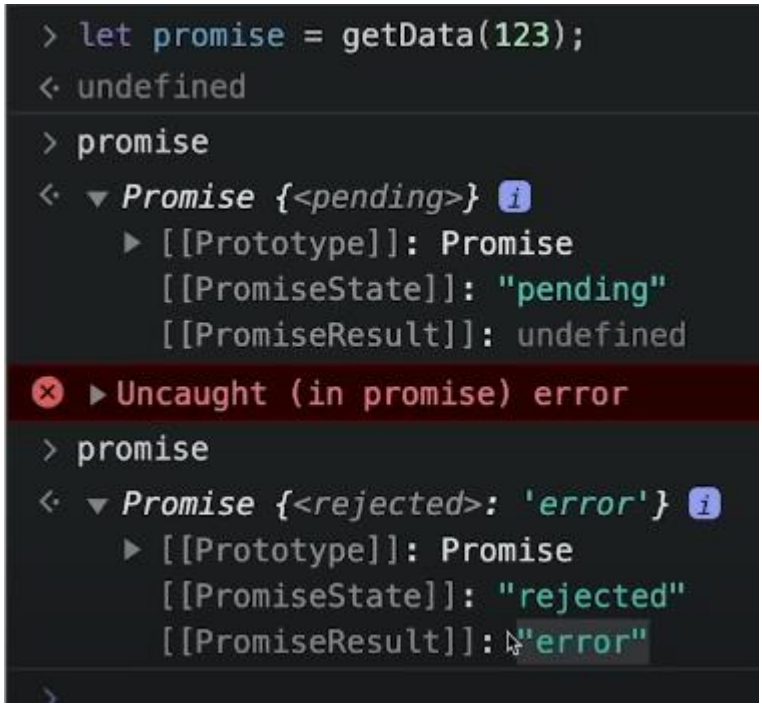
O/p:



Reject:

```
function getdata(data, getNextdata){
    return new promice ((resolve,reject)=>{
        setTimeout(()=>{
            // console.log("data",data);
            resolve("rejected");
            if(getNextdata){
                getNextdata();
            }

        },2000);
    });
}
```

O/p:



If we get any promise we need to know that promise reject or fulfilled

```
const getpromise = () =>{
    return new Promise((resolve,reject)=>{
        console.log("I am a promise");
        resolve("123");
    });

};

let promise = getpromise();
//if promise is resolve then we can use it
promise.then((res)=>{
    console.log("promise fullfilled",res);
});
```

o/p:

```
I am a promise
promise fullfilled 123
```

Similarly for reject

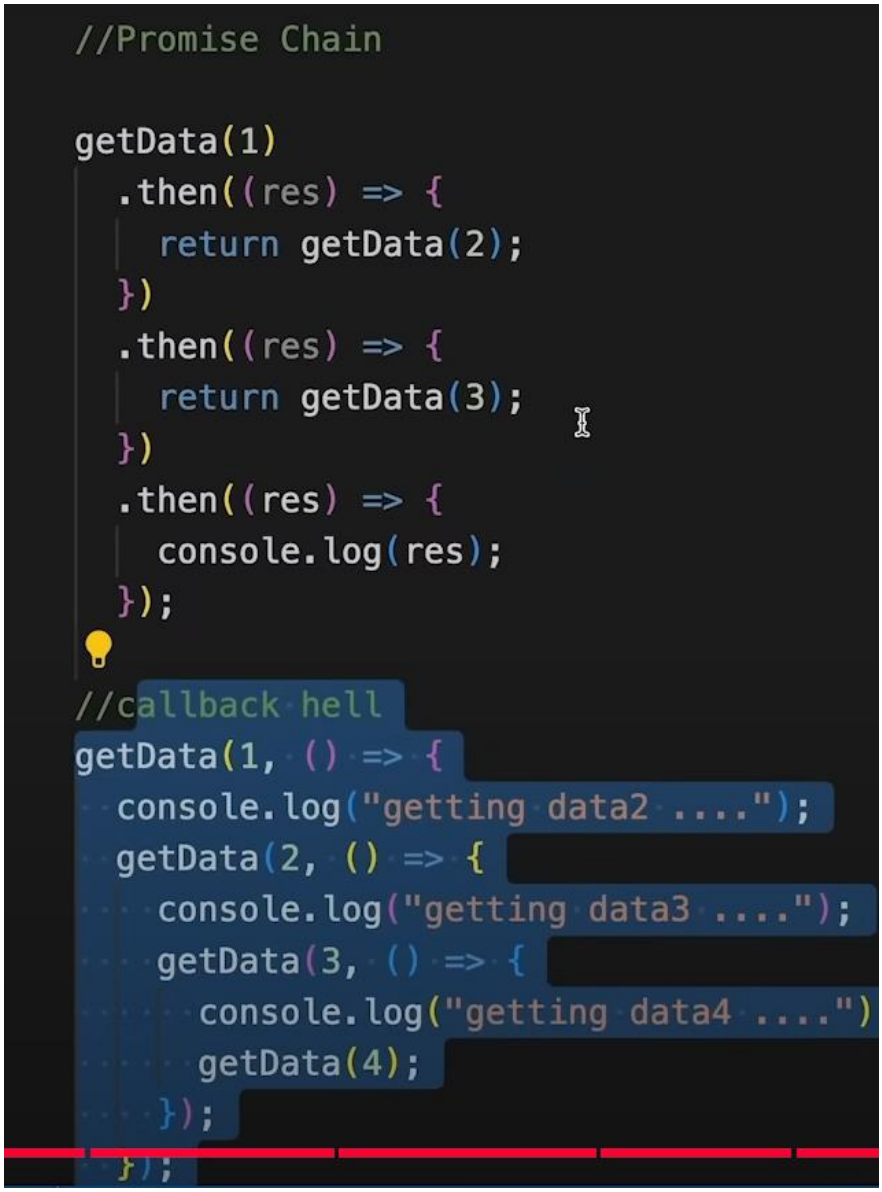
```
promise.catch((err)=>{
    console.log('rejected',err);
});
```

.then( ) & .catch( )

promise.then((res) => { .... })

promise.catch((err)) => { .... })

Promise Chain:



```
//Promise Chain

getData(1)
  .then((res) => {
    return getData(2);
  })
  .then((res) => {
    return getData(3);
  })
  .then((res) => {
    console.log(res);
  });

//callback hell
getData(1, () => {
  console.log("getting data2 ....");
  getData(2, () => {
    console.log("getting data3 ....");
    getData(3, () => {
      console.log("getting data4 ....")
      getData(4);
    });
  });
```

Async-Await :

async function always returns a promise.

async function myFunc( ) { .... }

await pauses the execution of its surrounding async function until the promise is settled.

Await must be used in a async function

We are no need to declare a promise

```
function Api(){
    return new Promise ((resolve,reject)=>{
        setTimeout(()=>{
            console.log("Weather data");
```

```
            resolve(250);
        },3000);
    });
}

async function getweatherdata(){
    await Api();
}
```

IIFE :

Immediately Invoked Function Expression IIFE is a function that is called immediately as soon as it is defined

Syntax is like wise

(funct)();

Fetch API:

The Fetch API provides an interface for fetching (sending/receiving) resources.

It uses request and Response objects.

The fetch() method is used to fetch a resource (data).

Let promise = fetch(url,[options])