**Javascript**

No arrays videos

Javascript was a scripting language earlier only the functionalty was happening, after few years they made it has a programming lang, we have oops, functions, datatypes, variables, operators whatever we have in java we have in javascripts

Es1 (first) version

LTS - > Es6(latest) version, will be using this and LTS is long term support

For interactions we need javascript ,

Html is used for structure

Css is used for styling

Javascript is used for interaction

**Variables**

Variable is a name given to the memory location to store the data

In 3 types we can declare the varibales

Var: once u declare var a=10; you can redeclare it again var a=30; it’s a drawback as we can declare it many times, u may think u didn’t declare and again u declare it, u can reassign the value as well

Scope: function scope

Let: once u declare the let a=a10; u cant redeclare it , u can reassign the value in let

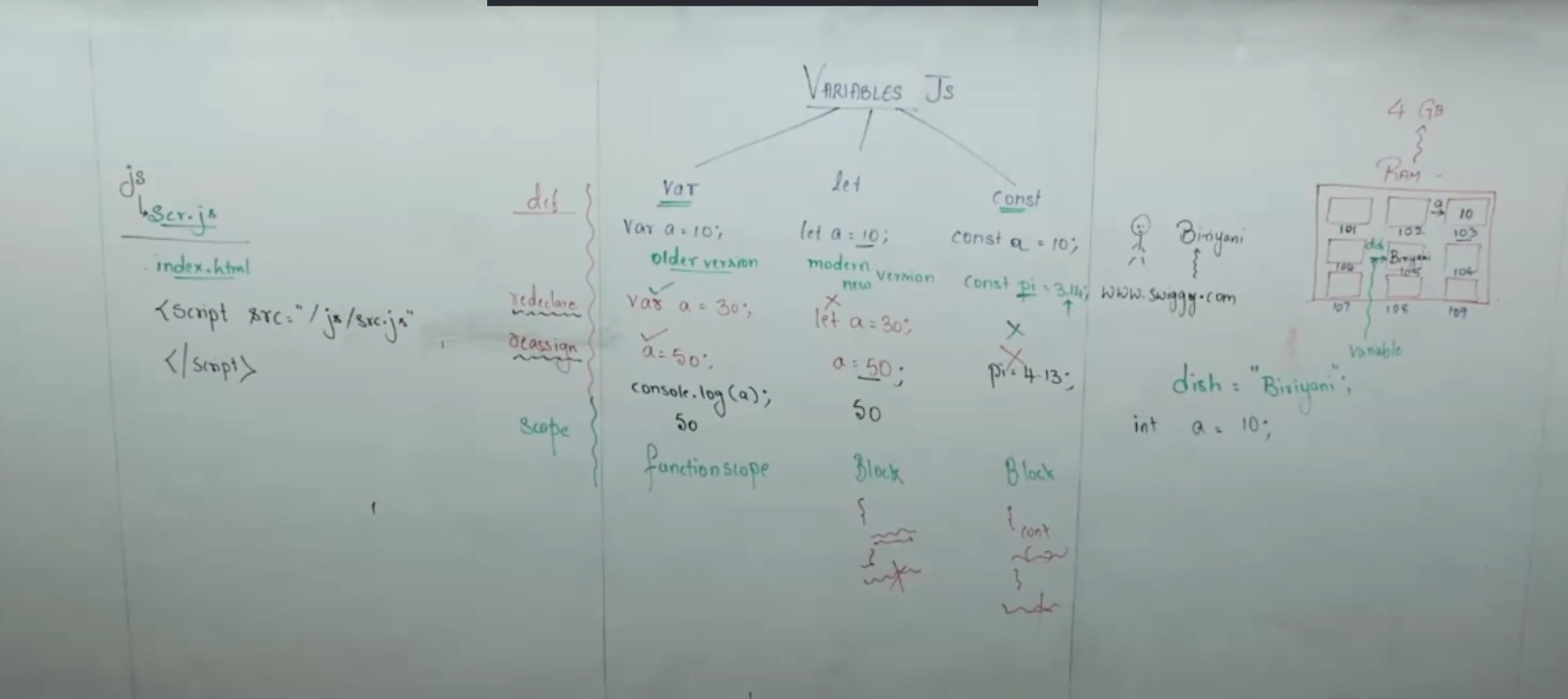
Scope; block scope

Const: its fixed value, it wont change the values that are written in const u cant redeclare the const as well u cant reassign the value .

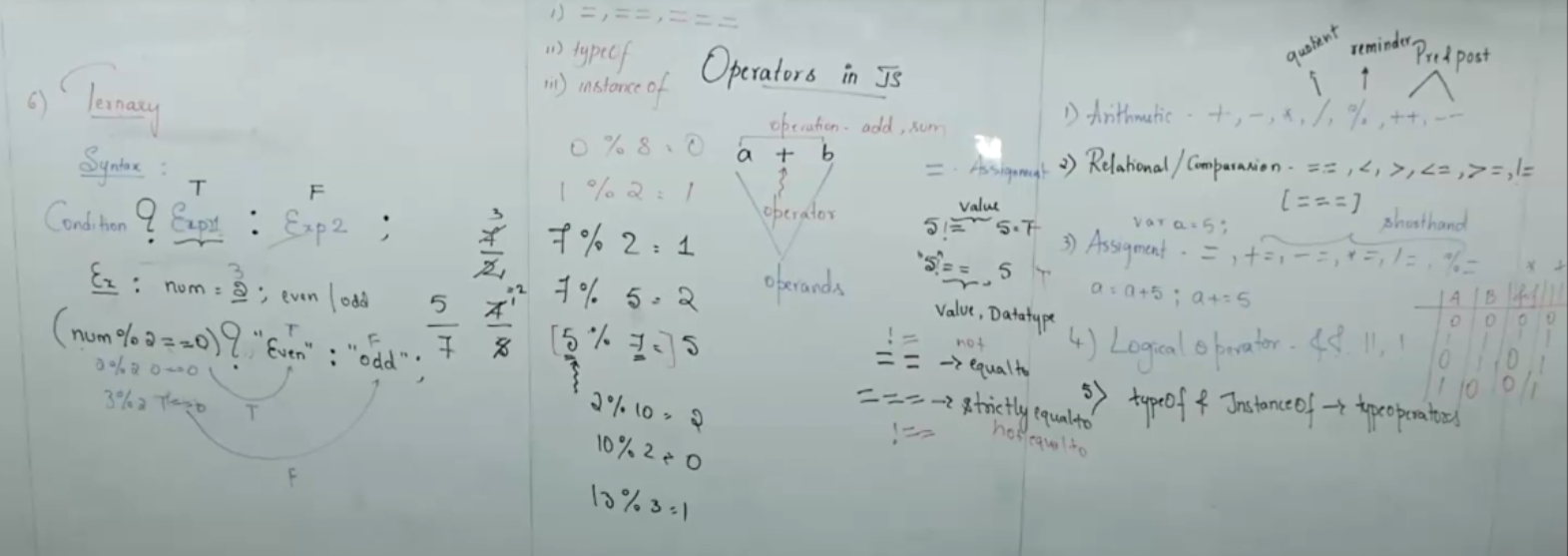
Scope: block scope

**For variables : variable and methods we will give like this Stu\_name ,this is snake case**

**For arrays and filename we can give like this StuName, HelloMan**



**Operators**

Operator is a special symbols that are used to perform some operations on the operands

**Functions**

**Function is a sequence of steps written inside a block of code to perform specific task is called as function**

**Syntax:**

**Function (parameters)**

**{**

**// body [coe]**

**}**

**Ex: write a function to add nos 10 & 20 without i/p and o/p**

**Function add() { // function declaration or function creation**

**Let a =10;**

**Let b =20;**

**Let c = a+b;**

**Document.log(c) ;**

**}**

**Add(); // calling the function**

**Square of a number**

**Var a=10;**

**Function square() {**

**Let nu =4;**

**Let res = num\*num;**

**Console.log(res);**

**}**

**Square();**

**In java jvm will execute the program but**

**In js engine will control for the program so here before the execution only the memory will be allocated for the global execution context it contains of 2 phases**

**1st only memory gets allocated for all the functions and variables**

1. **Memeory space: memory gets created for all the variables and functions**

**Memeory is allocated in the form of key value pair in javascript**

**a: undefined // for let and const its different**

**square: {…….} entire code will sit here which is this (Let num =4;**

**Let res = num\*num;**

**Console.log(res);**

**)**

**1st it will allocate the memory for functions and varibales later it will assign the values**

1. **Code space: code execution will take place code will execute line by line as its an interpritted lang**

**Here from 1st it goes to execute the code**

**So here we will initialize the values here**

**A :1 0 and removed undefined**

**Next line is function squrea that is already executed in memeory ryt so next it will go**

**Next it c let num=4;**

**So again it have to create a memeory for this also like in thid code phase only it will execute ok naa**

**Like again memory and code phase it will do indie the memeory phase now so for each and every new variables it will create a new call stack , it will check is there any functions and variables**

**Let num = undefined**

**Let res = undefined**

**Now code execution phase ok naa**

**Let a =4; it removes undefined**

**Let res = a\*a; // 4\*4 and res = 16**

**Console.log(res); it will print this and the loop ends and it removes the function that is deactivated and outside the function u cant access that o/p the function as the memory get deleted .**

**Next it see console.log(a); ok now it is still available in the memory so it get executed and after it comes out and the control will go to js engine when it goes to js engine the memeory all will get deleted**

4 Types of functions

No i/p no o/p

With i/p no o/p

No i/p with o/p

With i/p with i/p

*// no parameters and no return type*

function *add*() {

    let a=10;

    let b =20;

    let c= a+b;

    console.*log*(c);

}

*add*();

*//with parameter and no return type*

function *sub*(a,b) {

console.*log*(a-b);

}

*sub*(5,10);

*// no parameter with return type*

function *pro*(){

     let a=10;

    let b =20;

    let c= a\*b;

    return c;

}

*pro*();

*// with parameter, with return type*

function *pro*(){

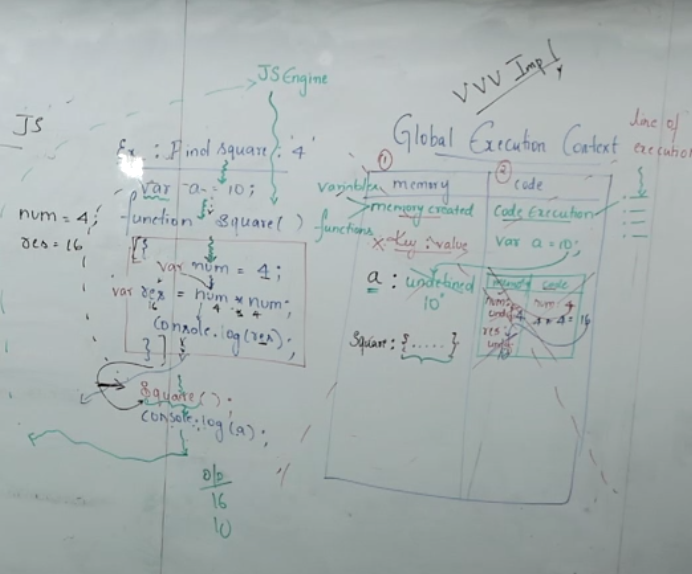
    let c= a/b;

    return c;

}

*div*(20,10);

Function overloading is not allowed in javascript



Diff ways of declaring functions in javascript

Function declaration

Function expression

Arrow function

Functions declaration is hoisted but expression and arrow are not hoisted

1. **Function declaration:**

Is a normal way is used to create a function

Function square (num) {

Let res = num\* num;

Retrun res;

}

Console.log(Square(4));

1. Function Expression

Let square = Function (num) {

Let res = num\* num;

Retrun res;

}

Console.log(Square(4));

This is function declaration with expression

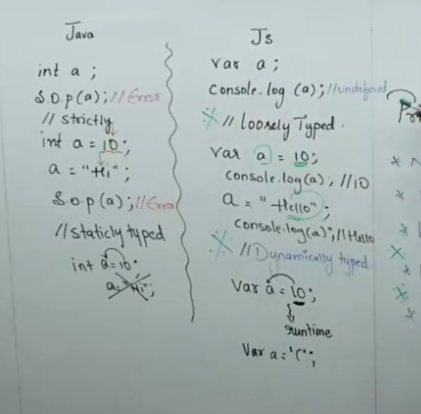
1. Arrow function: to simplify the traditional way we can use the arrow function in 1 single line

We don’t need to write everything , it works when u have 1 single line block of code

Data types in javascript

Primitive complex

* Number
* String
* Boolean
* Undefined – new we have only in js not java
* Null – only in js



**Arrays**

Array is a collection of elements and we can store heterogenous means we can store different type of data as well.

Declaration: we have 2 ways

**1st way using square bracket**

Let color = [“pink”,’black”,”red”];

Console.log(color); //inside array we will get all the colors

**2nd way using constructor**

Let odd = new Array (1,3,5,7,9);

Console.log(odd); //inside array we will get all nos

Console.log(odd[2]); // 2nd index value is 5 so/p is 5

When we call the index value which is not present we will get undefined

*//array operations,adding,removing and accessing*

*// adding*

*//push : used to add the values to the end*

var numbers =[1,2,3];

numbers.*push*(4); *// adds 4 to the end*

console.*log*(numbers);

*// unshift: used to add the values in the beginning*

var numbers =[1,2,3];

numbers.*unshift*(0); *// adds 0 to the start*

console.*log*(numbers);

*//removing*

*//pop: used to remove last element*

var fruits = ["apple","banana","cherry"];

fruits.*pop*(); *//removes last element*

console.*log*(fruits);

var fruits = ["apple","banana","cherry"];

fruits.*shift*(); *//removes first element*

console.*log*(fruits);

*// modifying and accessing*

*// access by index*

var colors = ["red", "green","blue"];

console.*log*(colors[1]); *//green*

*//splice() :is used to addng,removing or replacing elements in an array*

*//(strat,deletecount,items to be added multiple)*

var colors = ["red", "green","yellow","pink"];

colors.*splice*(1,2,"blue");

console.*log*(colors);

*//slice():is used to return the copy of the portion of an array without modifying the original part*

var numbers =[1,2,3,4,5];

let part = numbers.*slice*(1,3); *// extracts from index 1 to index 3 (exclusive)*

console.*log*(part);

*// array searching  and sorting*

*//indexOf(): return the 1st index of a specified element or -1 if not found*

var fruits = ["apple","banana","cherry"];

console.*log*(fruits.*indexOf*("cherry")); *//2*

*//includes(): chceks if the elment is present or not in the array*

var fruits = ["apple","banana","cherry"];

console.*log*(fruits.*includes*("apple")); *//true*

*//sort():sorts elements in place as string by defalut*

var letters =["b","a","c"];

console.*log*(letters.*sort*());

*//reverse():revers a set of elemts in an array*

var numbers =[1,2,3,4,5];

console.*log*(numbers.*reverse*());

*//array iteration methods*

*//forEach():*

*//combining*

*//concat():*

let arr1 =[1,2];

let arr2 = [3,4];

console.*log*(arr1.*concat*(arr2));

*// join():*

let words = ["hello","world"];

console.*log*(words.*join*(" "));

*// array destructing*

**controlconstructors**

Controlconstructors.js

// if statement – A student is eligible for voting if age>=18

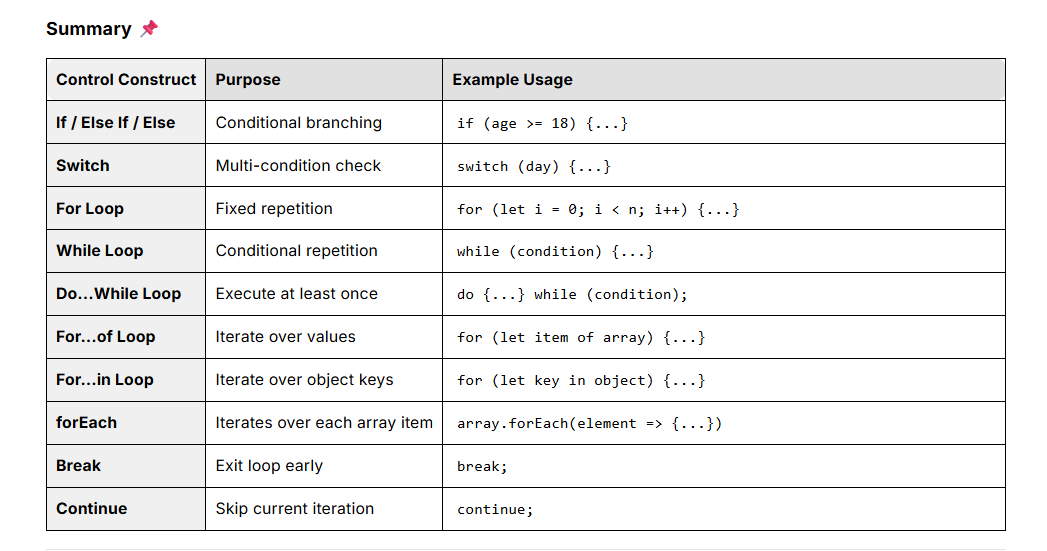
/if-else statement: write a program to check if a number is + or –

//if-else ladder: check if a number is +,negative or 0

// nested if: a student passes if makrs >=40,if marks >=80,show”distinction”

//switch case1:print the grade based on the letter (A=exc , b=good,c=avg,d=poor)

2:Print the season based on months in the yr (3,4,5 – summer)(11,12,1,2 – winter) (6-7-8,9-rainy)



**200% important**

**Hoisting**

**Variable function**

Var function declaration

Let function expression

Const arrow function

Hoisting is memory allocation

Variable hoisting

Variable hoisting is the process during the run time the declaration of the variable is moved to the top of the function call, making this particular variable available to the console.log

var

In simple

Calling anything before the declaration and getting the o/p without error then it’s called hoisted

And if u call anything and u get error then it’s not an hoisting

Ex:

Console.log(A)//undefined as the memory allocation will be done before the execution only

Var a=10;

Console.log(a); // 10

// so her variable a is hosited

**Let**

**The memory is allocated for let and const inside the TDZ which is an temporal dead zone, once you iniatilize it will come out of the TDZ**

**1st the memoey allocation will be donw for all the variables which is let in the tdz and when we call z it will check in the memory and yes its inside the memoey inside the tdz because the js engine will search only in the memory not inside the tdz so it will give an error**

**Next it will see the let z=10; initailze the z , that is called in the code executeion and once its initialzed it will come out of the tdz and store it in memory and it can fetch it from memory automatically**

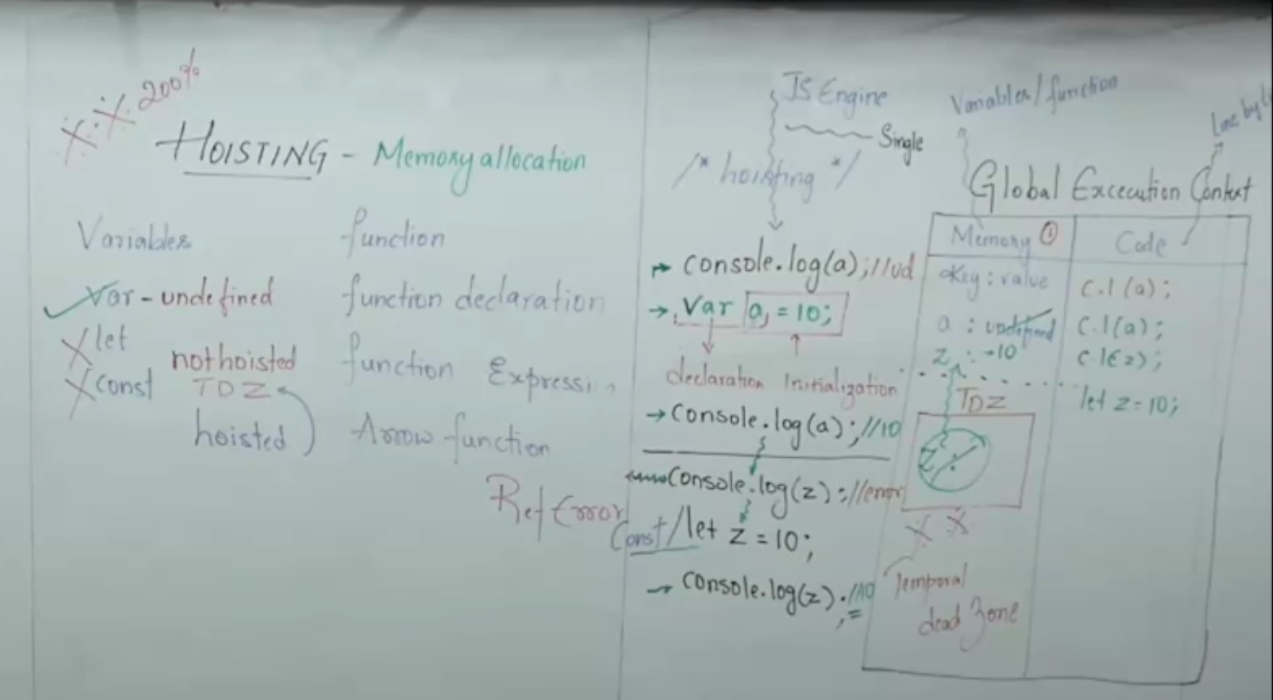
**And hence variable let and const are not hoisted as we call anything before the declaration and get the error**

**But its hoised as its memeory is allocated in TDZ.**

Console.log(z); // error reference error we will get

Let z=10;

Console.log(z);



f unction

function declartion

function add (){

var a=10;

var b =20;

console.log(a+b);

}

Add();

All the function declared with function declaration is hoisted

Because all the function is moved to the top before the call of the function and it’s called as hoisting and for let, const and var also its hoisted in function declaration

Function expression

Add();

Var add = function () {

----

---

}

Add();

Var add = ()=>

{

----

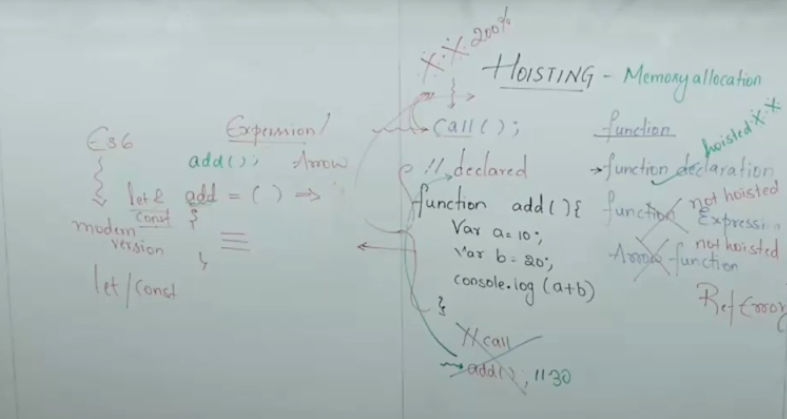
---

}

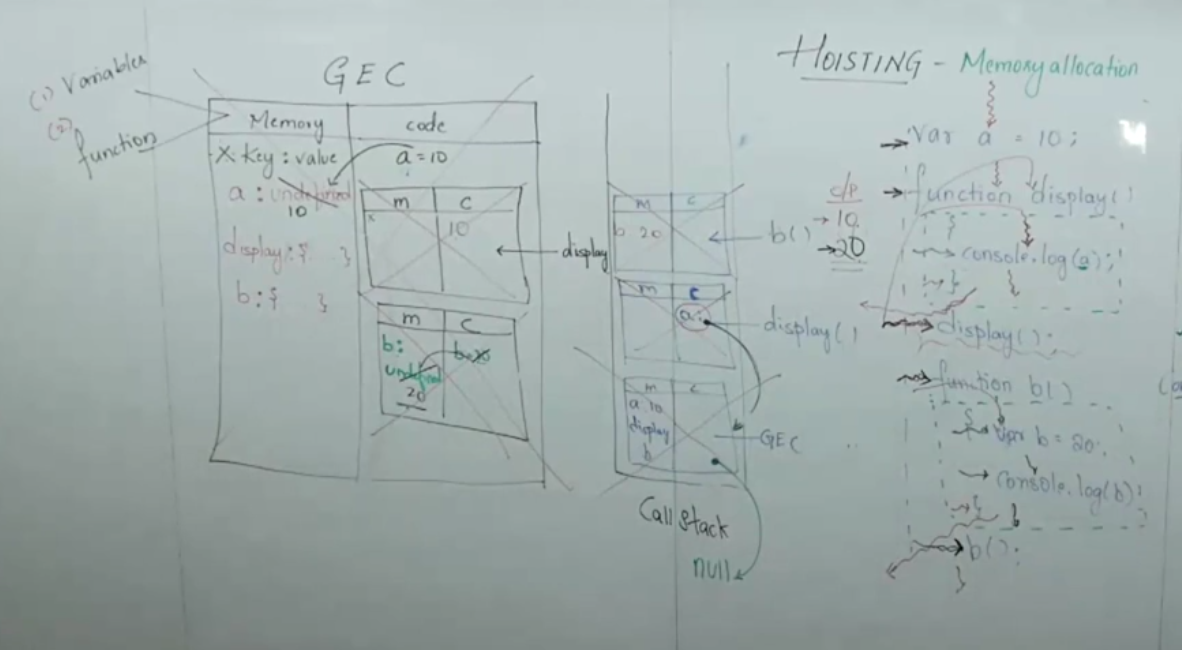
Function expression is also not hoisted for var,let and const

And here function expression and arrow function is from the modern verion which is ES6 and it has only let and const and let and const is not hoisted hence all the expression and arrow functions are not hoisted

And in function expression and arrow function only the var is declared so its partally hoisted



Memory allocation for the function



**Function overloading:**

Function overloading is not possible in javascript , because every function in js treated as objectsand we can make use of overloading in different ways but directly we cant use

We can make use of over loading in 2 ways:

1. Arguments
2. Spread operators / rest operators (varargs) very imp represented with ( . . .)

Spread operators is a new feature in ES6

Features in ES6 we have seen

Arrow function

Let and const

Spread opeartors

***using arguments***

function *add*() {

    if (arguments.length === 2) {

        return arguments[0] + arguments[1];

    } else if (arguments.length === 3) {

        return arguments[0] + arguments[1] + arguments[2];

    } else if (arguments.length === 4) {

        return arguments[0] + arguments[1] + arguments[2] + arguments[3];

    } else {

        return "Invalid number of arguments";

    }

}

console.*log*(*add*(2, 1));        *// 3*

console.*log*(*add*(1, 2, 3));     *// 6*

console.*log*(*add*(1, 2, 3, 4));  *// 10*

console.*log*(*add*());             *// "Invalid number of arguments"*

*// so her we need to use arguments only as ita an built in and are not availabe for arrow-function*

*// argument.legth gives the number of arguments passed*

*// u can acess each aruguments by index (arguments[0],arguments[1]) etc.*

***Spread operator/ rest operator***

*It accepts numbers from 0 to n.*

*function add(...numbers) {*

*let sum=0;*

*for(num of numbers){*

*sum +=num;*

*}*

*return sum;*

*}*

*console.log(add(1,2));*

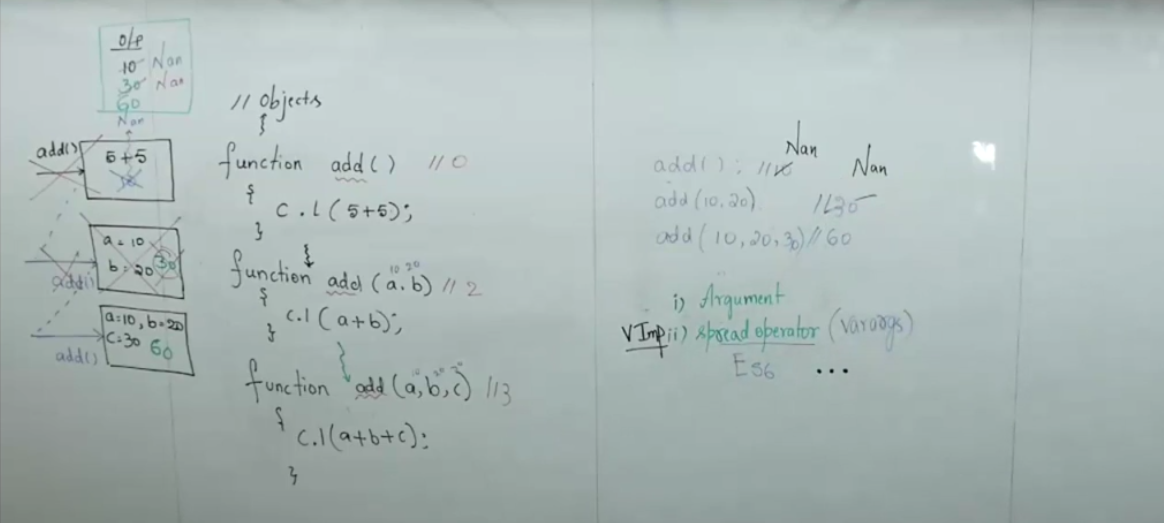
*console.log(add(2,3,4));*

*console.log(add(3,4,5,6));*

*console.log(add());*

*// so here we are making use of spread opeartor and use for loop to iterate from 1st num to last num*

*// so here we will make use of (...numbers) so will know that we are making use of spread or rest opeartor*

**

***Lexical environment – very imp for interview***

*Ascending [hirearcy] hirearcy means one above the other and is in the ascending format from parent to child and child to upper child something like*

*Local scope + lexical scope*

*(parent scope)*

*Scope is an boundary, within the limited area that*

***In scopes we have 3 types***

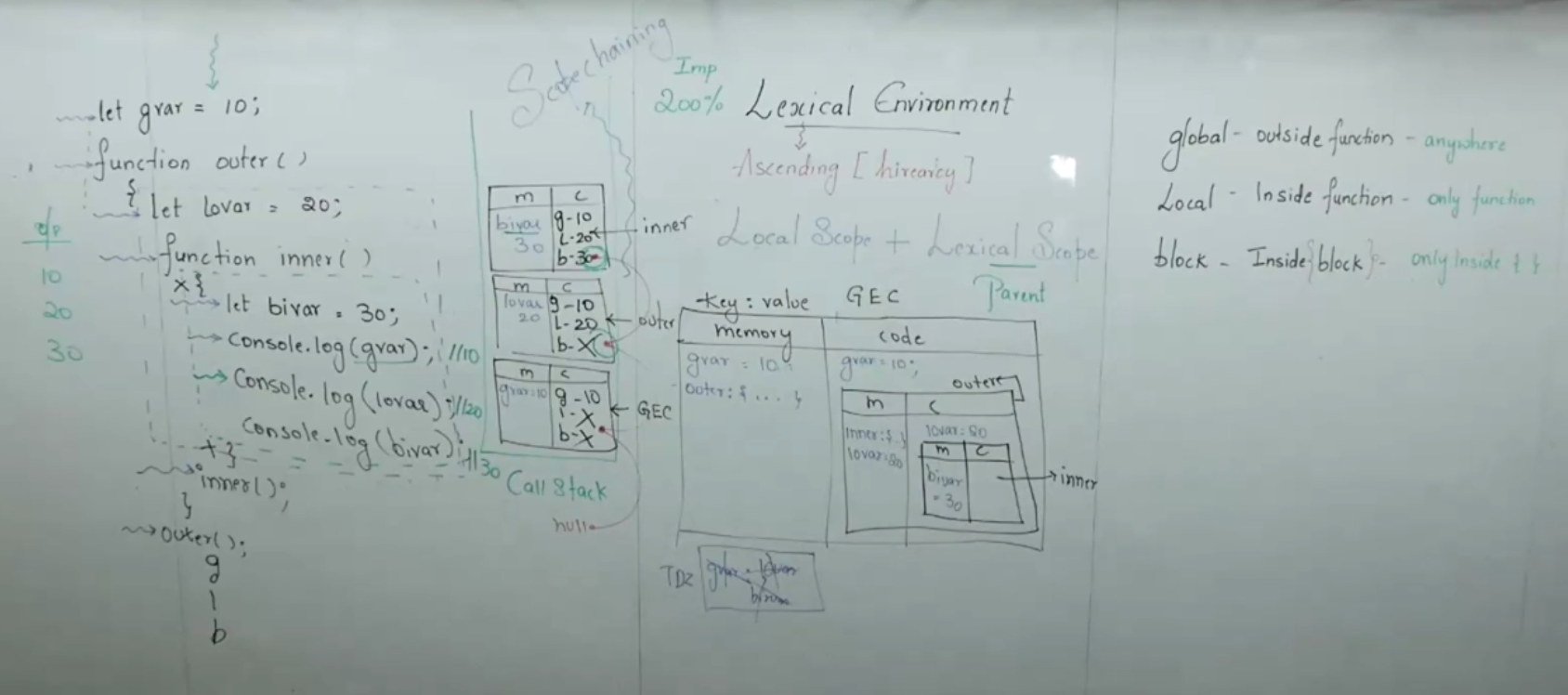
*Global: outside the function – anywhere can access (extrovert just remember like that)*

*Local: inside the function – only inside the function where it is declared (improvert)*

*Block: inside the block – only inside { … } (introvert)*

***Scope chaining:***

***Scope chaining refers to the child element collected to its lexical scope, like one child is connected to its parents element and repeat the process when we have scope chaining only we have lexical environment***

******

***Here’s where the magic begins. The Lexical Environment is a structure that JavaScript uses to keep track of all variables and functions available at any given point in your code.***

***Breaking It Down:***

1. ***Local Memory (Variable Environment): This stores variables and functions for a specific block or function.***
2. ***Reference to Parent Environment: If a variable isn’t found locally, JavaScript looks for it in the parent environment (like asking your elder sibling when you can’t find something at home).***

***It’s like a family tree where every child (function or block) has a parent. When you need something, you ask your family members in a chain-like manner. This chain is called the Scope Chain.***

***A lexical environment is a mechanism for how variable scope is determined in JavaScript. It refers to the structure of variable scopes as they are defined in the source code.***

* ***Definition: A lexical environment consists of:***
  1. ***The environment record: Where variables and function declarations are stored.***
  2. ***A reference to its outer lexical environment (the parent scope).***

***Oops object orientation***

* + - ***Object is an real world entity , same we call it in an programming world as an object***
    - ***Object is a instance of an class***

***Example: A car is an object — it has data (color, model) and behavior (start(), stop()).***

* + - ***A class is a blue print of an object because without class we cant write an object***

***And every object has 2 things properties and behaviours***

***We can create in 2 ways the object***

1. ***object: take key value pair***
2. ***class***
3. ***object: using object we can create only 1 object***

*const student = {*

*id = 101,*

*name = “Anu”,*

*grade=7.8,*

*study: function (){*

*console.log(“student study”);*

1. ***class: where as using class we can make create multiple objects***

*// creatinf with class*

class student {

*//properties*

    id;

    name;

    grade;

    institue="kodnest";

*//behaviours : so in class it will already knows that we will be giving function ok na so we dont need to create a class*

*study*(){

        console.*log*(this.name,"student studies");*//this is used to point ot current object*

    }

*eat*(){

        console.*log*(this.name,"student eats");

    }

*sleep*(){

        console.*log*(this.name,"student sleeps");

    }

}

*// now we need to create an object*

*// object creation in js*

const s1 = new *student*();*// when we tell new it will creates and object as an s1*

s1.id =101;

s1.name="Anusha";

s1.grade=7.8;

console.*log*(`this is:${s1.id} !`);

console.*log*(`this is:${s1.name}!`);

console.*log*(`this is:${s1.grade}!`);

console.*log*(`this is:${s1.institue}!`);

s1.*study*();

s1.*eat*();

s1.*sleep*();

*// 2nd object creating*

const s2 = new *student*();

s2.id =102;

s2.name="Rukmini";

s2.grade=8.5;

console.*log*(`this is:${s2.id} !`);

console.*log*(`this is:${s2.name}!`);

console.*log*(`this is:${s2.grade}!`);

console.*log*(`this is:${s2.institue}!`);

s2.*study*();

s2.*eat*();

s2.*sleep*();

*// we have draw back in this as we are giving the vlaues here so we make use of constructor*

1. **constructor**:

constructor is also a function which will be executed when an object is created , how do we create an constructor we create an constructor using constructor{} keyword

constructor is a special function which accepts the initial values and when the object is created

*// creating with class*

class student {

*//constructor*

   constructor(name,age){

    this.name=name; *// this points to the current object*

    this.age=age;

   }

*//behaviours : so in class it will already knows that we will be giving function ok na so we dont need to create a class*

*study*(){

        console.*log*(this.name,"student studies");*//this is used to point ot current object*

    }

*eat*(){

        console.*log*(this.name,"student eats");

    }

*sleep*(){

        console.*log*(this.name,"student sleeps");

    }

}

*// now we need to create an object*

*// object creation in js*

const s1 = new *student*("Anu",20); *// calling the constructor and we pass the values here as the constructor aslo has the paramters*

*//execution starts from here like when it sees an new key word it creats an memeory in ram for student object and it loads all the properties and behaviorus to the ram*

*// next it will give the refernce of the object as s1, as soon as object is created the constructor call happens with the parametrs it matches*

*//  and we pass name and age that will be assigned to the this which is the current objcet memory and the constructor come back to the object creating line and next it sees console.log(name); and it fetches all the detials from the ram memory*

console.*log*(`this is:${s1.name}!`);

console.*log*(`this is:${s1.age}!`);

s1.*study*();

s1.*eat*();

s1.*sleep*();

*// 2nd object creating*

const s2 = new *student*("Ruk",22); *// calling the constructor and we pass the values here as the constructor aslo has the paramters*

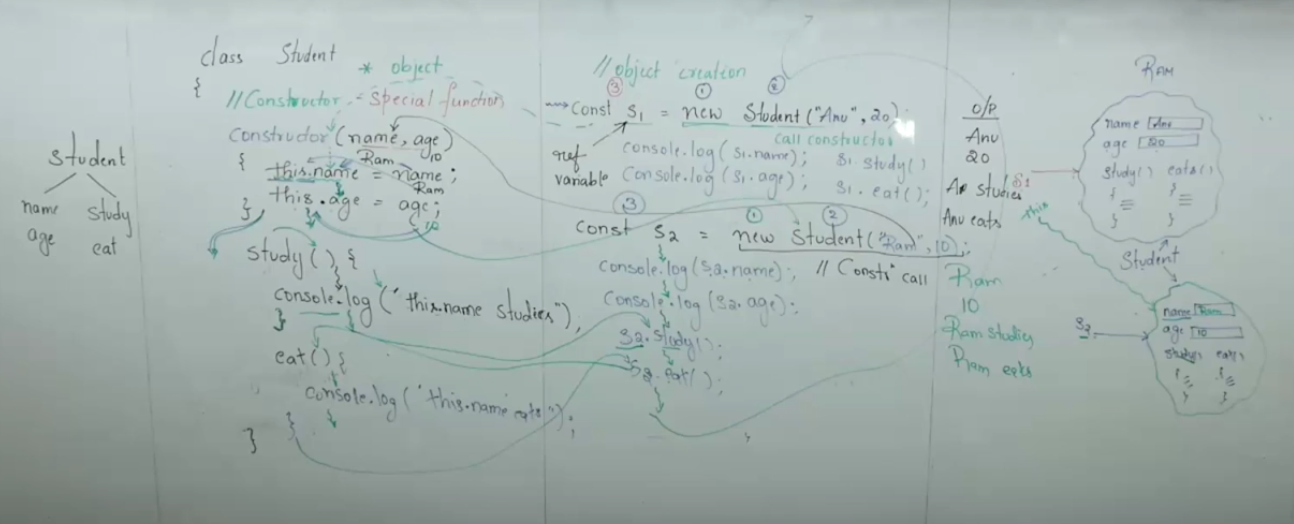
console.*log*(`this is:${s2.name}!`);

console.*log*(`this is:${s2.age}!`);

s2.*study*();

s2.*eat*();

s2.*sleep*();



**Inheritance**

Is one of the most important feature of oops , it is defined as intheriting the properties of the parents class to the child class

**We have 3 methods**

1. inherited methods: directly inherited from the parent class and used directly in child class
2. overridden methods: the method inherited from the parent will be modified by the child class
3. specialized methods: is a method which is not present in the parent class but only present in the child class

inheritance

employee (parent)

emp\_name, emp\_id – > properties

work(), project() 🡪 behaviours

and we have 2 children

developer tester

and it has all the properties from parent

but we need to just change the project as developer and tester

info() info() is a specialized method

class Employee {

  constructor(name, id) {

    this.name = name;

    this.id = id;

  }

*// inherited*

*work*() {

    console.*log*(`Employee ${this.name} is working`);

  }

*project*() {

    console.*log*(`Employee ${this.name} is working on a project`);

  }

}

class Developer extends Employee {

*// overridden*

*project*() {

    console.*log*(`Employee ${this.name} is working as a developer`);

  }

*// specialized*

*info*() {

    console.*log*(`The developer name is ${this.name}, id is ${this.id}`);

  }

}

class Tester extends Employee {

*// overridden*

*project*() {

    console.*log*(`Employee ${this.name} is testing the project`);

  }

*info*() {

    console.*log*(`The tester name is ${this.name}, id is ${this.id}`);

  }

}

const e = new *Employee*("varsha", 1);

e.*work*();

e.*project*();

const d = new *Developer*("arjun", 2);

d.*work*();

d.*project*();

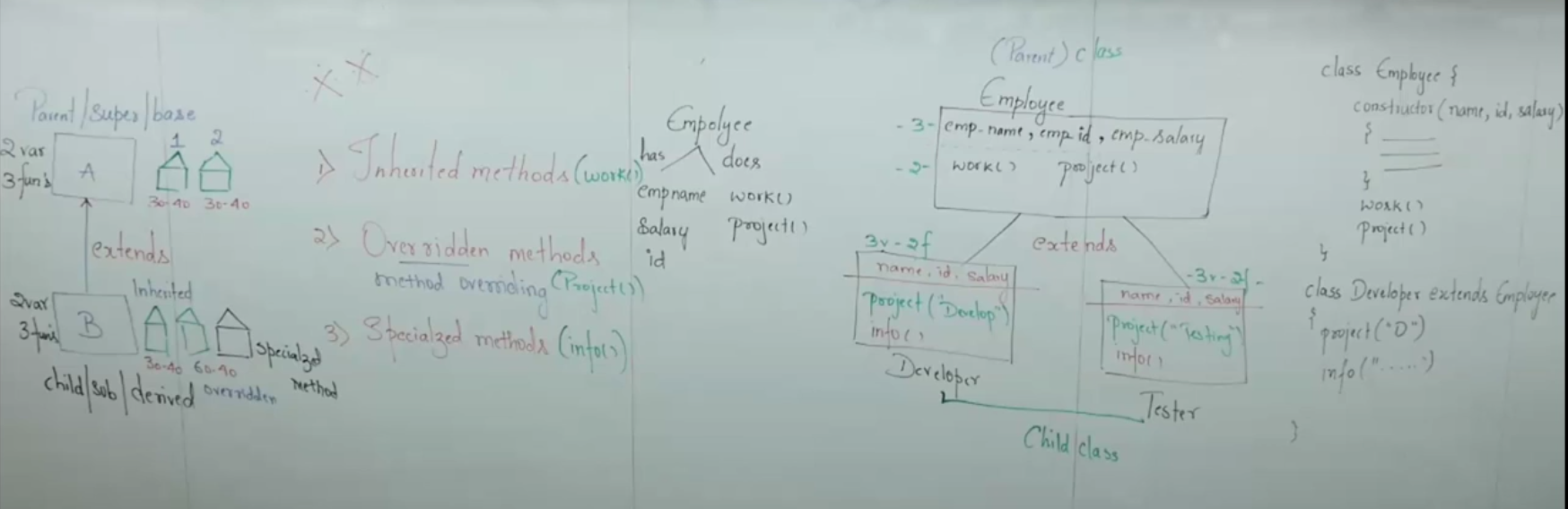
d.*info*();

const t = new *Tester*("rukmini", 3);

t.*work*();

t.*project*();

t.*info*();



**super, this**

super.speak(): this is super keyword it always points to the parent class

super() constructor call: this is super constructor it will always point to parent constructor

this: always refers to the current object

*//parent class*

class animal {

    constructor(name){

        this.name=name; *// this refers to animal*

    }

*speak*() {

          console.*log*(`${this.name}makes a sound`);

        }

    }

*//child class*

    class dog extends animal {

        constructor(name,breed) {

            super(name); *// calls parent constructor(animal)*

            this.breed = breed;*//'this' refers to dog itself*

        }

*speak*() {

            super.*speak*();*// calls animal speak()*

            console.*log*(`${this.name}barks`);

        }

*showdetails*() {

            console.*log*(`name: ${this.name}, breed: ${this.breed}`);

        }

        }

        const dog1 = new *dog*("Buddy","Golden Retriever");

        dog1.*showdetails*();

        dog1.*speak*();

**dom**

**document object model**

why we make use of dom?

In order to make understand the html documents by js we make use of DOM.

So I need to target the html tag and modify or change, so will make use of DOM

And that is interactivity so we make use of js

DOM will read entire document of HTML in the form of objects as the js treats everything like every line also as objects.

All the documents of html is taken and treated as object which represents the DOM Tree and that can be understood by javascript is called as Document Object Model

And whatever content we have that is names as content nodes or title like inside the tags we have the content <p>hello<p> so hello is a content nodes, and each and every object is represented as nodes and all the nodes have some names.

Doctype

Html

This are called as root nodes

Head , body, html, title, h1,p,p are called as

element

any element node which has attribute is called as attribute node

like this<h1 class=”h”> and <p id=”p”> so here id=”p” is attribute node

<p>hello<p> 🡪 hello is content nodes

**<html>**

**<head>**

**<title>dom</title>**

**</head>**

**<body>**

**<input type=”text” name = “username”/>**

**<h1 class =”h”> heading </h1>**

**<p id =”p” > para1 </p>**

**<p class =”p1”> para1 </p>**

**<p class=”p1”>para2</p>**

**</body>**

**</html>**

**We have six main methods in DOM**

1. **get Elements by Tag Name**
2. **get Element by ID**
3. **get Elements by class Name**
4. **get Elements by name**
5. **query selector –more important**
6. **query Select for All –more important**

**1,3,4 will return set of elements**

1. **get Elements by Tag Name:** so in this if we have multiple p tag it will return all the p tags like set of elements and that will be stored in HTML collection list, this will return HTML collection List and list is array itself

**ex:** get Elements by Tag Name(“p”);

1. **get Element by ID:** this will return only single element like <p id=”p”>

**ex:** get Element by ID(“p”);

1. **get Elements by class Name:** it returns only one elements which has class name but it can hold multiple elements and index is 0. And <h1 class=”h”>, html collection list

**ex:** get Elements by class Name(“h”);

1. **get Elements by name:** it return multiple elements which has the same name, like

<input type=”text” name= “username”> and stores in html collection list

**ex:** get Elements by Name(“username”);

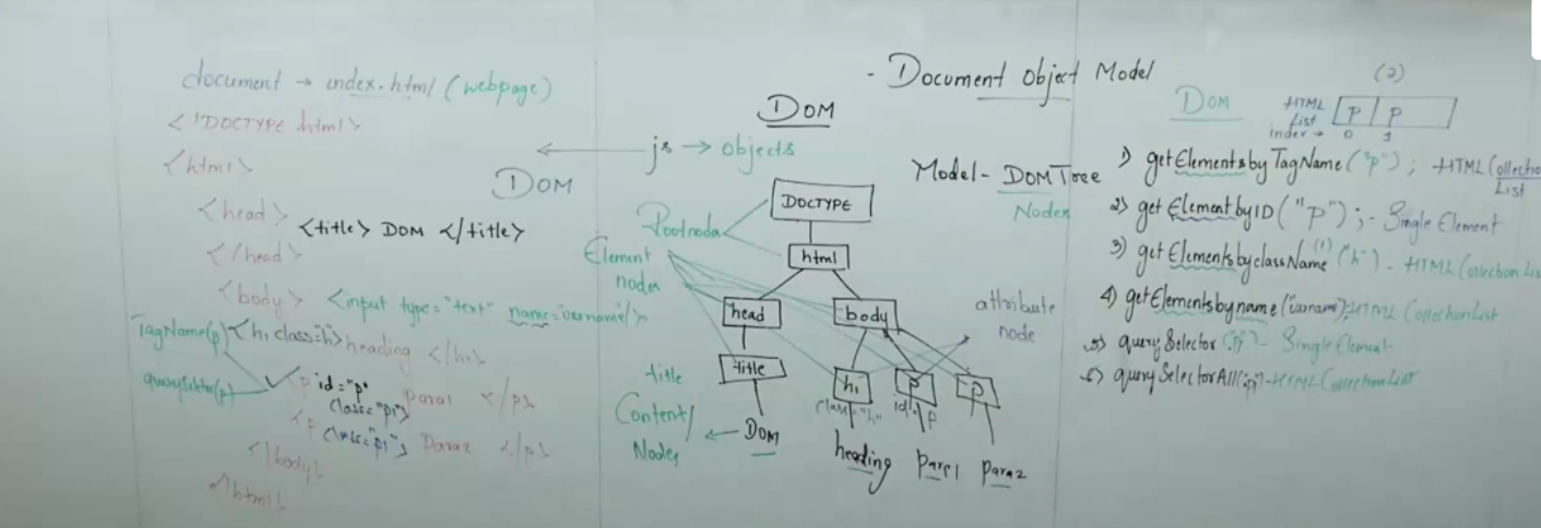
1. **Query selector:** it will return only the single element, and it will target only the 1st element

**ex:** query selector(“p1”);

1. **Query selector all:** it will return all ,html collection list, and it returns all the elements which has the selector name same

**ex:** query selector all(“p1”);

**And the priority is like its equal / there is no priority and for priority of execution it will get which is written 1st.**

****

**Manipulation of DOM**

**In order to change the text in the element we have 2**

   document.*getElementById*("head").textContent="DOM in java script"; *// so here we are making use of textContent/innerText to fetch the content inside the head and it is used to display all the visible and hidden text , hidden which we make use in overflow in css*

document.*getElementById*("head").innerText="DOM in java script"; *// innerText  its same as textContent but u cant use both innerText and innerHTML at a time only 1 u can make use of*

*and innerText we can make use of only the text wehich is visible not all the hidden text*

**in order to change the text and structure of an elements we make use of innerHTML**

    document.*getElementById*("h").innerHTML = "<i> DOM JS </i>"; *//if we want tot chnage the text and the structure we make use of innerHTML*

***// adding or removing the elements and we have 3***

***So here we can add or remove the elements***

   document.*createElement*();: to create new Dom

*appendChild*(); : Adds an element as a child

*removeChild*(); : removes an element from the DOM

**adding and removing the classes**

add();

remove();

toggle();

contains();

**attributes: extra info given to html tags like for img we have attributes src,href,height**

setAttribute();

getAttribute();

removeAttribute();

**event is an action performed on the web page.**

**Events in DOM**

**--> mouse event**

mouseover : when mouse is placed

click: trigreed when an elemnt is clicked

double click : triggred when double clicked

mousemove :when move moved

mouseout: when move is out

**--> keyboard events**

Keydown: triggered when a key is pressed down

Keyup : when it is released

**--> form events**

Submit: it performs an action when the button is submitted very very imp

Change: used to know that used have changed the input

Focus : *when ever we highlight/inside the input field then its an focus*

Blur: *when u come away from the input field then it is called as blur*

Input: is used to see the i/p wht the user gives

**-->window events**

Resize: triggered when the broser window is resized

Scroll: triggered when the user scrolls

dom content loaded: this is document events, Target one element and change the color

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

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

    <title>Document</title>

</head>

<body>

    <h1>Form and input Events Demo</h1>

    <form *id*="myForm">

    <label> enter your name:</label>

    <br>

    <input *type*="text" *placeholder*="type something" *name*="username">

    <br><br>

    <button *type*="submit">submit</button>

    </form>

</body>

<script>

*//form*

*//submit --imp*

    document.*getElementById*("myForm").*addEventListener*('submit', (event) => {

   event.*preventDefault*();

*alert*("form submitted!");

    });

*//change*

    document.*getElementById*("myForm").*addEventListener*('change', (event) => {

*alert*(`Input changed to: ${event.target.value}`);

    });

*//focus*

*//when ever we highlight/inside the input field then its an focus*

    document.*getElementById*("myForm").*addEventListener*('focus', (event) => {

        console.*log*("input field focus!");

    });

*//blur*

*//when u come away from the input field then it is called as blur*

    document.*getElementById*("myForm").*addEventListener*('blur', (event) => {

        console.*log*("input field lost focus!");

    });

*//input -- vv imp*

    document.*getElementById*("myForm").*addEventListener*('input', (event) => {

        console.*log*(`current value ${event.target.value}`);

    });

</script>

</html>

**HOF Hight order function**

1. Should have function that takes another function as an argument
2. It should return function as output

Either 2 even 1 condition also satisfy

**1)Should have function that takes another function as an argument**

**// so here i need like this until unless I order the coffee prepare should not execute**

Ex:

Function order (callprepare) { // so here we pass the call prepare as an argument to the order because we need the prepare to be executed only if the order is placed

Console.log(“coffee ordered”);

*callprepare*();

}

Function prepare () { //call back function

Console.log(“start preparation”);

}

Order (prepare); //and pass the arugument of one function inside the another function

**o/p:**

coffee ordered

start preparation

// order is an high order function because it has the argument as an function

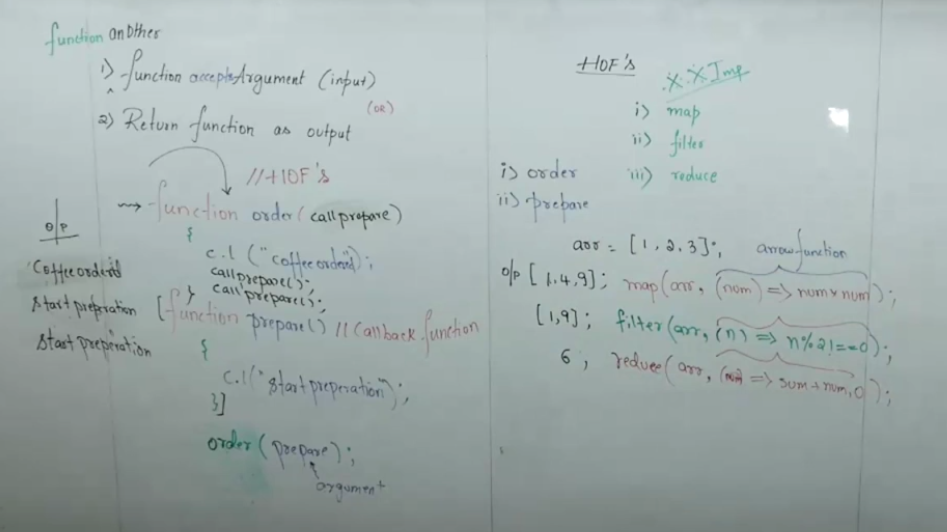
//prepare is an call back function which is passed as an argument to the hight order function, and this waits unless until it is called by an hof

**We have 3 types of HOF**

1. Map
2. Filter
3. Reduce

Why map, reduce and filter are HOF

Because it accept the another function aas an argument hence they are called as HOF

Please check the figure

**2)It should return function as output**

Function x () {

Var a =10;

Function y () {

Console.log(a);

}

return y; //so here we are returning the function as an o/p to another function

}

*x*(); *// this returns the f y () {console.log(a); } because y return to the y*

*console.log(x( ));*

**Function x() is an high order function here**

Adv of returning function like this:

**Closures**

A **closure** is formed **when a function remembers the variables from its outer function**,  
even **after the outer function has finished executing**.

A closure is a function which remembers the lexical scope variable and can access when ever we need even though the parent function is not present. And that can be done by storing it in some other variable that is res we are storing here

Function x () {

Var a =10;

Function y () {

Console.log(a);

}

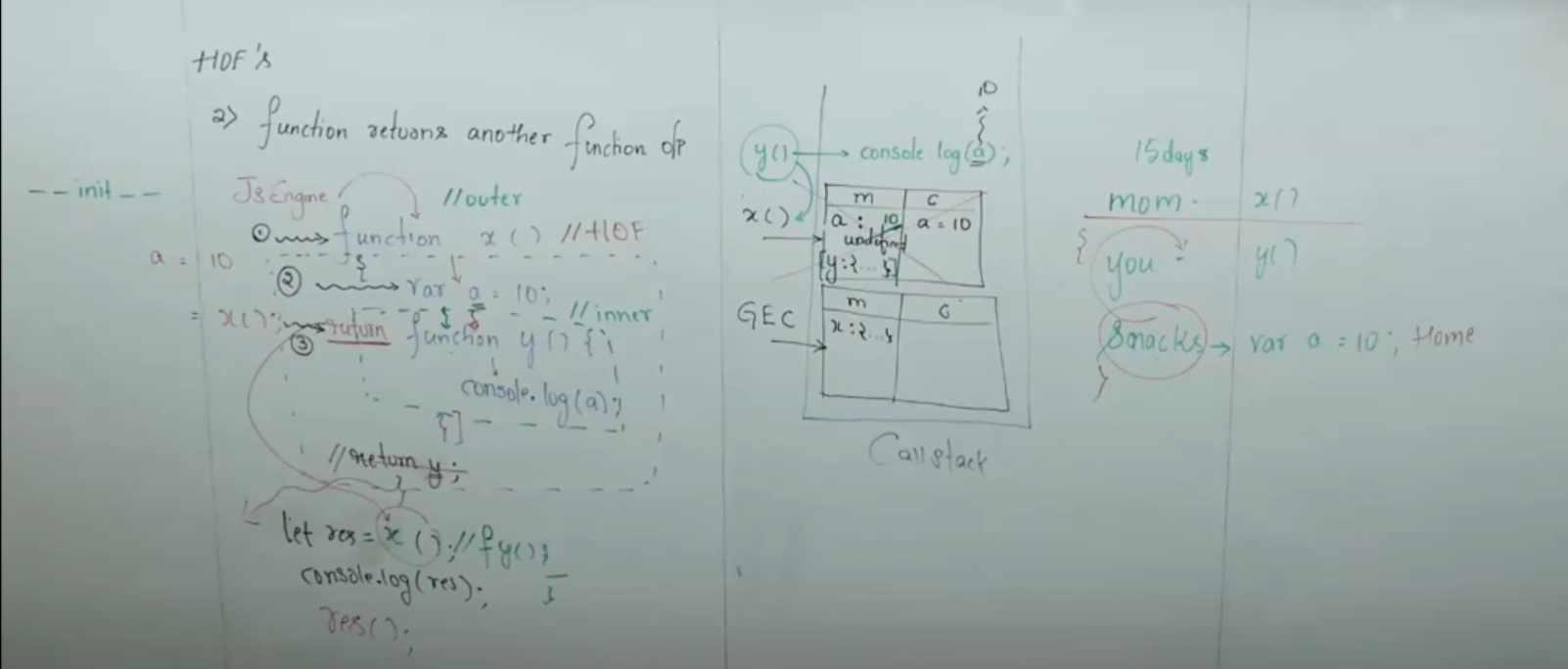
return y; //so here we are returning the function as an o/p to another function

}

let res = *x*(); // u are storing it in the res container as the value will be deleted once the memeory for x() will be deleted

console.*log*(res); *// this returns the f y () {console.log(a); } because y return to the y*

*res*(); // and u can make variable as an function as well only if we are storing the function in it



**Asynchronus**

Synchronous: which is in a sequence/steps executed one after the another

Asynchronus: which is not in a sequence / steps executed randomly

*setTimeout*(() => {

    },time in ms); //1 s =1000 ms

Ex:

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

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

    <title>Asynchronus</title>

</head>

<body>

  <script>

***//synchronus***

     console.*log*("open book my show App");

    console.*log*("search movie");

    console.*log*("select time");

    console.*log*("select seat");

    console.*log*("book ticket");

***// asynch but not in proper order because it will execute based on time which has least time***

*setTimeout*(() => {

 console.*log*("open book my show App");

    },1000);

*setTimeout*(() => {

         console.*log*("search movie");

    },5000);

*setTimeout*(() => {

         console.*log*("select time");

    },3000);

*setTimeout*(() => {

         console.*log*("select seat");

    },8000);

*setTimeout*(() => {

        console.*log*("book ticket");

    },2000);

*//if we do like this it will execute according to the time*

***// asynchrous - call back ,so will bemmaing use like this // this nested function are called as Event Loop***

*setTimeout*(() => {

 console.*log*("open book my show App");

*setTimeout*(() => {

         console.*log*("search movie"); *// this will wait to execute 1st*

*setTimeout*(() => {

         console.*log*("select time"); *// this will wait to execute 1st , 2nd*

*setTimeout*(() => {

         console.*log*("select seat"); *// this will wait to execute 1st 2nd,3rd*

*setTimeout*(() => {

        console.*log*("book ticket"); *// this will wait to execute 1st 2nd 3rd 4th*

    },2000);

},8000);

},3000);

   },5000);

},1000);

  </script>

</body>

</html>

if we write like this it leads to callback hell(like this > shape) a patteren which occurs when u write call back function where u will be converting to synchronos to asynchronos

it’s a very bad practise to do like this, to overcome this callback hell we have come with promises

**Execption handling**

The execption is an unexpected event occurred during excetion of program is called as exception

Handling the exception using the try catch, throws, finally Is called as exception handling

And we can handle the exception in 3 diff ways

1. Try and catch: we can have 1 try and multiple catch and catch will not work withour the try
2. Throws
3. Finally

**We can have multiple try catch and without try we cant make use the catch**

Console.log(“start”);

Var a =10/x; 🡨-- the exception has occurred as the x is not defined as soon as exception is occurred , exception object is created and that (EO) exception object searchs for 2 things ,the UDEH user defined execption handler and DEH default exception handler

And UDHE is the try catch, throws ,finally , if it is given it make use of this or else it goes to the DEH.

And DEH if we don’t give UDHE the EO is given to DHE and it trows and error

console.log(a);

Console.log(“end”);

// so here its given to the DEH and it throws an error

**// UDEH**

 console.*log*("start");

        try{ -- if we have error we put the try block , if the error occurred pass error to the catch

            var a=10/x;

            console.*log*(a);

            throw new *Error*("10/x is the error line"); *// if u want to customeize  ur own error*

        }

        catch(e){ // and here e is an execption and we can display that and it will not stop here only it there an error ok na but it will continue to execute each and every line

            console.*log*(“the error is”, e.message);

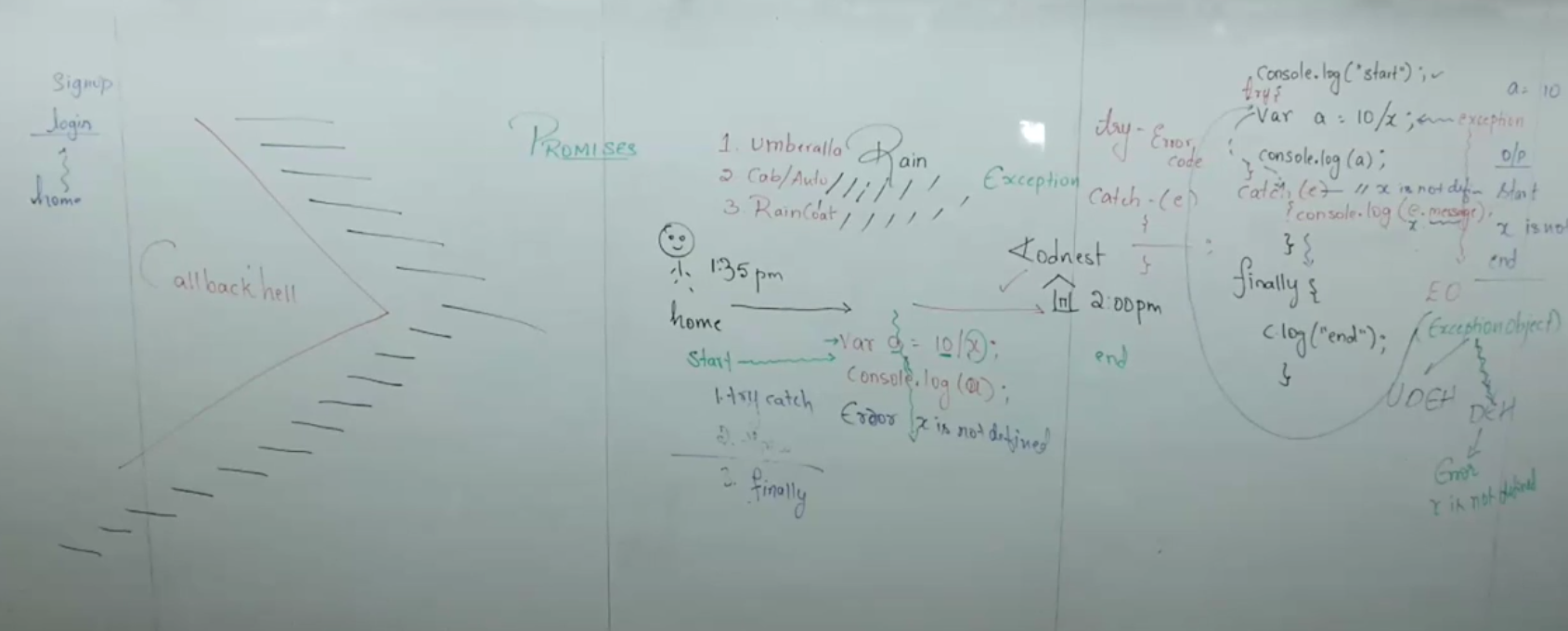
        }

        finally{ // even though the execption is occurred or not this should execute, and it can have try, catch and finally or try and finally

        console.*log*("end");

        }

// we can handle only the logical errors we cant handle the syntax errors



**ASYN ->Callback to synchronus**

setTimeout(()=>{

console.log(“step1”);

setTimeout(()=>{

console.log(“step2”);

setTimeout(()=>{

console.log(“step3”);

setTimeout(()=>{

console.log(“step4”)

},1000);

},5000);

},3000);

},2000);

**This leads to call back hell so we came up with promises**

**Promises**

As soon as the promise is made it goes to the pending state/initla state

And from pending state it can got to the resolve state or pending state

In promise we have 3 states

Pending:

Resolve: .then() , .then() resolve the promise

Reject : try catch() ,thows,finally something like that reject the promise

Everything in js is treated as object and we create a object using the new keyword

The disadvange of promise is that .then() here we will be promising each and everything so its not good to use promises and we came up with asyn/awaits

Now, **normally** in synchronous code, we use try...catch to handle errors.  
But Promises handle errors **asynchronously**, so they come with their *own mechanism* — the .catch() method.

That’s why we **don’t need** try...catch inside normal promise chains — the promise’s **rejection** automatically behaves like a “throw” that gets caught by .catch().

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

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

    <title>promise</title>

</head>

<body>

   <script>

     p = new Promise((resolve, reject)=> { *// saying the we have 2 states*

        app="BMS";

        if(app === "BMS") *// if its true*

*resolve*("1.open Bms"); *// we will pass this to then(); as an promise*

        else

*reject*("not found BMS"); *// else it acts as an try and pass this as an exception to the catch*

            })

    .*then*((msg)=> { *// to execute the 1st one we need to pass the o/p and . refers to the current object and if u want u can give p.then()*

        console.*log*(msg);

        console.*log*("2.search movies");

    })

    .*then*(()=>{

        console.*log*("3.select seat");

    })

    .*then*(()=>{

        console.*log*("4.ticket booked");

    })

    .*catch*((e)=>{ *// and here we are not making use of try as we are giving reject and reject acts like an throw over here and arrow function is also not nessacry, and try is used here as its asynchronos and promises have their own error mechanism . catch();*

        console.*log*(e);

        })

        .*finally*(()=>{

            console.*log*("close the app");

        }) ;

   </script>

</body>

</html>

**With time set out time**

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

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

    <title>Document</title>

</head>

<body>

    <script>

        const p = new Promise((resolve, reject)=> { *// saying the we have 2 states*

*setTimeout*(()=>{

            let app="BMS";

        if(app === "BMS") *// if its true*

*resolve*("1.open Bms"); *// we will pass this to then(); as an promise*

        else

*reject*("not found BMS"); *// else it acts as an try and pass this as an exception to the catch*

        },6000);

     })

    p.*then*((msg)=> { *// to execute the 1st one we need to pass the o/p and . refers to the current object and if u want u can give p.then()*

        return new Promise((resolve)=>{ *// whenerver we are resolving the promise , we need to return it or else p will not know that we are resolving and it will excetue without seeing the setTimer*

*setTimeout*(()=>{

        console.*log*(msg);

        console.*log*("2.search movies");

*resolve*(); *// this is used to recive the result of the next .then()*

            },5000);

             });

    })

    .*then*(()=>{

        return new Promise((resolve)=> {

*setTimeout*(()=>{

        console.*log*("3.select seat");

*resolve*();

          },5000);

           });

        })

    .*then*(()=>{

        return new Promise((resolve)=>{

*setTimeout*(()=>{

        console.*log*("4.ticket booked");

*resolve*();

            },5000);

             });

        })

    .*catch*((e)=>{ *// and here we are not making use of try as we are giving reject and reject acts like an throw over here and arrow function is also not nessacry, and try is used here as its asynchronos and promises have their own error mechanism . catch();*

*setTimeout*(()=>{

        console.*log*(e);

        },4000) ;

            })

    .*finally*(()=>{

*setTimeout*(()=>{

            console.*log*("close the app");

        },2000)

            }) ;

    </script>

</body>

</html>

**This promises leads to the promise chaining so they came up with Async and await**

**Async and Await**

Async is a keyword given to a function which consist of asynchronus code and this keyword returns the promise by default

Await keyword can be only used inside the Async function, waiting for certain amount of time before the previous execution.

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

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

    <title>Document</title>

</head>

<body>

    <script>

*// async function main() {*

*//     await new Promise((resolve) => setTimeout(resolve,1000));*

*//         console.log("open book my show App");*

*//    await new Promise((resolve) => setTimeout(resolve,5000));*

*//          console.log("search movie");*

*//    await new Promise((resolve) => setTimeout(resolve,3000));*

*//          console.log("select time");*

*//    await new Promise((resolve) => setTimeout(resolve,8000));*

*//          console.log("select seat");*

*//     await new Promise((resolve) => setTimeout(resolve,2000));*

*//         console.log("book ticket");*

*//         }*

*//even this is not good manner to do as everytime we are giving new promise*

*//main();// we have to call this or else it will not work*

*// minimized one*

   function *wait*(ms){ *// as only the time is changing , so here we will pass the time from user and pass it as function*

    return new Promise((resolve) => *setTimeout*(resolve, ms))

   }

   async function *BookTicket*(){

    try{

        const app="BMS";

        await *wait*(2000); *// calling the function wait()*

        if(app === "BMS"){

            console.*log*("1.open book my show");

        }

        else {

            throw new *Error*("Bms not found");

        }

        await *wait*(4000);

        console.*log*("2.search for movie ");

        await *wait*(3000);

        console.*log*("3.select seats");

        await *wait*(5000);

        console.*log*("4.ticket booked");

    }

    catch(e){

        console.*log*(e);

    }

    finally{

        console.*log*("close phone");

    }

   }

*BookTicket*();

   </script>

</body>

</html>

**FORM vadilation**

**Regex**

^ - start

$ - end

[ ] -> 0-9 OR /d numbers , A-Z,a-z

{ } -> range {0,10 } min is 0 and max 10

\* -> 1, n -repeated

. (dot) -> any character

s

**U can validate**

1. Email : [mail@name.com](mailto:mail@name.com) @ and com is must
2. Passowrd: atleast 12 char long, 1 special , uppercase,1 num
3. Mobile number: [6-9] and 10 digits
4. Username : it should not be empty, only alpha , length(min and max)
5. **Username and password:**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

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

**<title>Form Validation</title>**

**</head>**

**<body>**

**<form id="myForm">**

**<!-- Username -->**

**<label for="username">Username:</label>**

**<input type="text" id="username" placeholder="Enter your name">**

**<p style="color:red;" id="userError"></p>**

**<!-- Password -->**

**<label for="pwd">Password:</label>**

**<input type="password" id="pwd" placeholder="Enter password">**

**<p style="color:red;" id="pwdError"></p>**

**<br><br>**

**<button type="submit">Submit</button>**

**</form>**

**<script>**

**const form = document.getElementById("myForm");**

**form.addEventListener("submit", (event) => {**

**event.preventDefault(); // stop page reload**

**// Clear old errors**

**document.getElementById("userError").innerHTML = "";**

**document.getElementById("pwdError").innerHTML = "";**

**// Username validation**

**const username = document.getElementById("username").value.trim();**

**const unregex = /^[A-Za-z]+$/;**

**if (username === "") {**

**document.getElementById("userError").innerHTML = "Username can't be empty";**

**return;**

**} else if (username.length < 2) {**

**document.getElementById("userError").innerHTML = "Name must be at least 2 characters";**

**return;**

**} else if (username.length > 30) {**

**document.getElementById("userError").innerHTML = "Name can't be more than 30 characters";**

**return;**

**} else if (!unregex.test(username)) {**

**document.getElementById("userError").innerHTML = "Only letters allowed (A–Z, a–z)";**

**return;**

**}**

**// Password validation**

**const password = document.getElementById("pwd").value;**

**const passwordPattern = /^(?=.\*[A-Za-z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/;**

**if (!passwordPattern.test(password)) {**

**document.getElementById("pwdError").innerHTML =**

**"Password must have 8+ chars, include letters, numbers & a special symbol (@$!%\*?&)";**

**return;**

**}**

**// If everything is fine**

**alert("Form submitted successfully!");**

**});**

**</script>**

**</body>**

**</html>**

1. **Email:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Document</title>

</head>

<body>

<fieldset>

<legend> basic information</legend>

<form>

<label for="mail">Email:</label>

<input type="email" id="mail">

<br>

<br>

<button type="submit">Submit</button>

</form>

</fieldset>

<script>

// email validation

e = document.getElementById("email").value.trim();

let emailregex = /^[a-zA-Z0-9.\_-]+@[A-Za-z0-9.\_-]+\.[a-zA-Z]{2,}$/;

if(!emailregex.tex(email)){

alert("please enter a valid email")

}

else{

alert("valid email!");

}

</script>

</body>

</html>

1. **Phone number:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Document</title>

</head>

<body>

<form id ="myForm">

<label for ="ph">Phone number</label>

<input type="text" id ="ph">

<br>

<br>

<button type="submit" id ="submit">Submit</button>

</form>

</body>

<script>

const form = document.getElementById("myForm");

form.addEventListener("submit",(event) => {

event.preventDefault();

const pn = document.getElementById("ph").value.trim();

let regex = /^(\+91)?[6-9]\d{9}$/; // question mark means optional

if(!regex.test(pn)) {

alert("Invalid number");

}

else{

alert("valid number");

}

});

</script>

</html>

Important topics

* Let, const, var
* Hoisting
* Arrow function
* HOF’s
* DOM
* Clousers
* Callbacks / promises
* Asyc + await