JavaScript Notes-Chai Aur Code

Lect-2: **Setting up environment in local machine for Javascript:**

- node.js is a javascript environment for running javascript files.
- install node.js on your system to get started
- use code editors like VS Code, Sublime, etc.
- use *node --version* in terminal to check if node.js is installed
- use node fileName.js to js code

Lect-3: **Save and work on Github for Javascript**

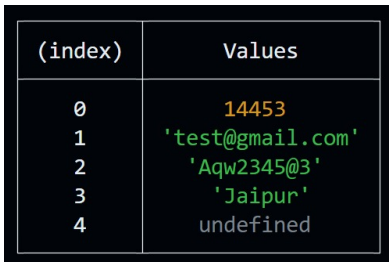
- create a GitHub repository
- To open js runtime environment at GitHub(it is same as what we have on vs code) : Click on "Code"—>"Codespaces"—>"Create Codespace on main" —>> code editor similar to vs code will open.
- Now to use this workspace we need to install node.js:- "View"—>"Command Palate"—>"Add Dev container Config file"—>"node.js and javascript"—> chose version—>Rebuild Now—>your runtime environment is ready
- Now create a Folder and test.js file inside it and write your code and run it using terminal. (But progress is not showing in repo)
- To save: Click on "Source Control icon"—>Select all the files that you want to save by clicking on + —> Write a commit message —>"Commit"—> Step-2 —> PUSH
- Always remember to close the running machine.—> In the top-left corner of GitHub.com, select Three bars—> then click Codespaces—>To the right of the codespace you want to delete, then click Delete.
- Performing above action wont delete any files. It will just close the running machine.

Lect-4: **Let, const and var ki kahani**

- const accountID = 1444553
  accountID = 2 //not allowed- value of const cant be changed or modified
-

```
const accountID = 14453
let accountEmail = "test@gmail.com"
/*
```

```
prefer not to use var because of functional scope
*/
var accountPassword = "Aqw2345@3"
accountCity = "Jaipur" //correct but not recommended
let accountState; //value is not defined --> undefined

console.log(accountID)
console.log(accountEmail)

// Instaed of using console everytime, we can use console.table([]) to get all d

console.table([accountID, accountEmail, accountPassword, accountCity, accountSta

//Note--> It is not necessary to use ; at the end of the statement in js
```



(index)	Values
0	14453
1	'test@gmail.com'
2	'Aqw2345@3'
3	'Jaipur'
4	undefined

Lect-5: **Datatypes and ECMA standards**

- "use strict"; //treat all JS code as newer version
- alert(3 + 3) //not allowed in node, allowed in Browser
- high priority should be given on code readability
- 1)use 'mdn docs' whenever you are stuck - preferable
  2) tc39 - js official documentation

```
let age = 16
let name = "Hitesh"
let isLoggedIn = false
let state
let gender = null

//number => 2^53 range
//bigint => for larger values
//string => ""
//boolean => true/false
```

```
//null => standalone value ; typeof null => Object
//undefined => value not defined
//symbol => unique

//object =>

console.log(typeof age); //number
console.log(typeof "hitesh"); //string
console.log(typeof null); //Object
console.log(typeof undefined); //undefined
```

Lect-6: **Datatype conversion confusion**

```
let score1 = 33;
console.log(typeof score1); //number

/* ------------Conversion to number ------ */
let score2 = "33";
console.log(typeof score2);
let score2InNumber = Number(score2); //Number --> string is converted into a n
umber
console.log(typeof score2InNumber);

let score3 = "33aaa";
console.log(typeof score3);
let score3InNumber = Number(score3);
//Note: this is not a number but still js convert it into a number. Its type w
ill be number, but value will be "NaN". So whenever you convert string to numb
er always check for this NaN case.
console.log(typeof score3InNumber); //number
console.log(score3InNumber); //NaN

let a = null;
let aInValue = Number(a);
console.log(typeof aInValue); //number
console.log(aInValue); // 0

let b = undefined;
let bInValue = Number(b);
console.log(typeof bInValue); //number
console.log(bInValue); // NaN
```

```
let c = "Hitesh";
let cInValue = Number(c);
console.log(typeof cInValue); //number
console.log(cInValue); // NaN

//true -> 1 ; false -> 0

/* ------------Conversion to boolean ------ */
let isLoggedIn = 1;
let booleanLoggedIn = Boolean(isLoggedIn);
console.log(booleanLoggedIn); // true
console.log(typeof booleanLoggedIn); // boolean
//1 => true; 0 => false ; 8 => true
//"" => false ; "Hitesh" => true

/* ------------Conversion to string ------ */
let someNumber = 33;
let stringNumber = String(someNumber);
console.log(typeof stringNumber); //string
console.log(stringNumber); //33
```

Lect-7: **Why string to number conversion is confusing**

```
/******************Operations************************** */
let value = 3
let negValue = -value
console.log(negValue) //-3

console.log(2+2); //4
console.log(2-2); //0
console.log(2*2); //4
console.log(3/2); //1.5
console.log(3%2); //1
console.log(2**3); //8

let str1 = "Hello "
let str2 = "Hitesh"
let str3 = str1 + str2
console.log(str3); //Hello Hitesh
```

```
console.log("1" + 2); //12
console.log(1 + "2"); //12
console.log("1" + 2 + 2); //122
console.log(1 + 2 + "2"); //32

console.log(true); //true
console.log(+true); //1
console.log(+""); //0

let num1, num2, num3
num1 = num2 = num3 = 2 + 2 //not recommended way
console.log(num1); //4
console.log(num2); //4
console.log(num3); //4

//prefix and postfix in js
let gameCounter = 100
gameCounter++
console.log(gameCounter); //101
++gameCounter
console.log(gameCounter);//102

let x = 3;
const y = x++;
console.log(`x:${x}, y:${y}`);
// Expected output: "x:4, y:3"

let a1 = 3;
const b1 = ++a;
console.log(`a:${a1}, b:${b1}`);
// Expected output: "a:4, b:4"



//++(++x);
//(x++)++
// SyntaxError: Invalid left-hand side expression in prefix operation

let x2 = 3n;
const y2 = x2++;
// x2 is 4n; y2 is 3n

/* In JavaScript, you can work with regular numbers (like 1, 2, 3) and also wi
```

```
th BigInt numbers (like 1n, 2n, 3n). BigInt is a special type of number that a
llows you to work with very large integers beyond the normal JavaScript number
limit.*/
```

Lect-8 : **Comparison of datatypes in javascript**

```
console.log(2 > 1);//true
console.log(2 >= 1); //true
console.log(2 < 1);//false
console.log(2 == 1); //false
console.log(2 != 1); //true

console.log("2" > 1); //true
console.log("02" > 1); //true

console.log(null > 0); //false --> conversion of null to a number
console.log(null == 0); //false -->here no conversion to a number
console.log(null >= 0); //true -->--> conversion of null to a number
/*Reason: an equality check '==' and comparison '>, <, >=, <=' work differentl
y.
Comaparisons convert 'null' to a number, treating it as 0. That's why null>=0
is true and null>0 is false.
*/

console.log(undefined > 0); //false
console.log(undefined == 0); //false
console.log(undefined >= 0); //false

//strict check: ===
console.log("2" === 2); //false

/* Note: Avoid confusing comparisons. Write clean and clear code.*/
```

Lect-9: **Data types of javascript summary**

```
/* Note : On the basis of how you can store the data and how you can access th
e data.
Dataypes are of two types in js: 1)Primitive 2)Non primitive or Reference type
*/
```

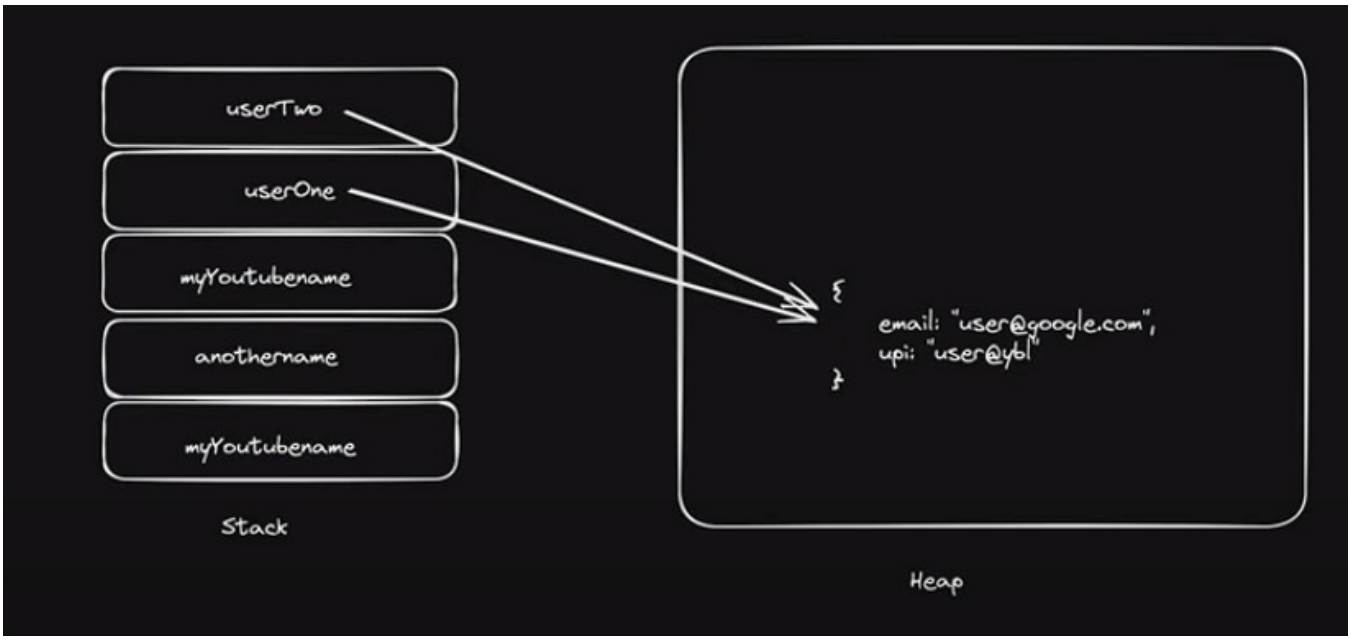
```
/*1)Primitive Types:
 --call by value
 --7 types: String, Number, Boolean, null, undefined, Symbol, BigInt
 */

let val = 123
let floatVal = 123.5
let outsideTemp = null
let userEmail;

const id = Symbol('123')
const anotherId = Symbol('123')
console.log(id === anotherId) //false

const bigNumber = 23257376746889n

/* 2) Reference (Non primitive):
 --call by refrence
 --Arrays, Objects, Functions
 */

const heros = ["Shaktiman", "Naagraj", "Doga"]

let myObj = {
    name: "hitesh",
    age: 22
}

const myFunc = function(){
    console.log("Hello World");
}

 //JavaScript is a dynamically typed language, which means you don't need to e
xplicitly specify the data type of a variable when declaring it.

 //typeof => return data type
 /*
 Return type of variables in JavaScript
1) Primitive Datatypes
       Number => number
       String  => string
       Boolean  => boolean
```

```
       null  => object
       undefined  =>  undefined
       Symbol  =>  symbol
       BigInt  =>  bigint

2) Non-primitive Datatypes
       Arrays  =>  object
       Function  =>  function
       Object  =>  object
 */
```

Lect-10: **Stack and Heap memory in javascript**

- Stack - Primitive data types ; Heap - Non Primitive

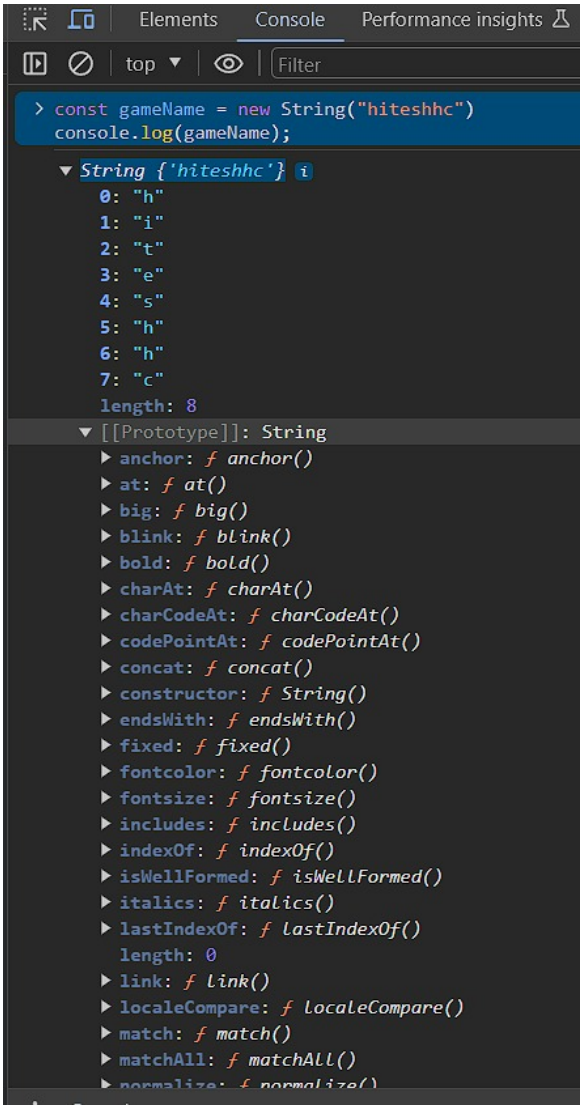
```
/*********Stack and Heap***** */
//Stack: Primitive; Heap: Non primitive

//Stack: Primitive - a copy is created
let myYoutubeName = "hiteshchoudhary"
let anotherName = myYoutubeName
console.log(anotherName);

anotherName = "chaiaurcode"
console.log(myYoutubeName);
console.log(anotherName);

//Heap: Non primitive - reference is used, since refrence is used changes will
reflect in original also. (When you take refrence back from heap , you dont ge
t a copy, you get refrence of original value)
let userOne = {
    email : "userone@gmail.com",
    upi : "user1@ybl"
}

let userTwo = userOne

userTwo.email = "hitesh@google.com"
console.log(userOne.email);
console.log(userTwo.email);
//here we made change in userTwo but it will reflect in userOne as well.
```



Lect-11: **Strings in Javascript**

```
const name = "Hitesh "
const repoCount = 50

console.log(name + repoCount + " Value") //not recommended
//use Backticks instead: (String interpolation)
console.log(`Hello, my name is ${name}and my repo count is ${repoCount}`);
```

```
const gameName = new String("hiteshhc")
console.log(gameName);
```




```
const gameName = new String("hiteshhc")
console.log(gameName);

console.log(gameName[0]);
console.log(gameName.__proto__); //we dont need to write __proto__ everytime, we can directly access all the methods as below
console.log(gameName.length);//8
console.log(gameName.toUpperCase());//HITESHHC
console.log(gameName.charAt(1));//i
console.log(gameName.indexOf('t'));//2

const newString = gameName.substring(0,4) //last index is not included ; here negetive value is ignored
console.log(newString);

const anotherString = gameName.slice(0,4) ////last index is not included ; here negetive value is also allowed
```

```
console.log(anotherString);

const newStringOne = "     Hitesh      "
console.log(newStringOne);
console.log(newStringOne.trim());

const url = "https://hitesh.com/hitesh%20choudhary"
console.log(url.replace('%20' , '-'));

console.log(url.includes('hitesh'));

console.log(gameName.split(""));
```

Lect-12: Number and Maths in Javascript

```
const score = 400
console.log(score);//400

//this gurantees it is a number
const balance = new Number(100)
console.log(balance); //[Number: 100]

console.log(balance.toString().length); //3

//toFixed() : number of digits after decimal
console.log(balance.toFixed(2));//100.00

//toPrecision(): total number of digits
const otherNumber = 23.8966
console.log(otherNumber.toPrecision(3)); //23.9
console.log(otherNumber.toPrecision(2));//24
console.log(1234.78945.toPrecision(3)); //1.23e+3

const hundreds = 1000000
console.log(hundreds.toLocaleString()); //1,000,000
console.log(hundreds.toLocaleString('en-IN'));//10,00,000

//other number methods
Number.MAX_VALUE
Number.MIN_VALUE
Number.MAX_SAFE_INTEGER
```

```
/***************Maths************** */
console.log(Math.abs(-4)); //4

console.log(Math.round(4.6));//5
console.log(Math.round(4.2));//4
console.log(Math.ceil(4.2));//5
console.log(Math.floor(4.9));//4

console.log(Math.min(4, 3, 6, 8));//3
console.log(Math.max(4, 3, 6, 8));//8

//***NOTE==> Math.random()
console.log(Math.random());//any value b/w 0 and 1

//if we want random value b/w 1 and 10
const randVal = Math.floor(Math.random() * 10) + 1;
console.log(randVal);

//if we want random value b/w min and max
const min = 10
const max = 20
const randomValue = Math.floor(Math.random() * (max - min + 1)) + min;
console.log(randomValue);
```

Lect-13: **Date and time in depth in javascript**

```
//Dates in js

/*JavaScript Date objects represent a single moment in time in a platform-inde
pendent format. Date objects encapsulate an integral number that represents mi
lliseconds since the midnight at the beginning of January 1, 1970, UTC (the ep
och).*/

let myDate = new Date()
console.log(myDate); //2024-01-17T17:03:53.379Z
console.log(myDate.toString()); //Wed Jan 17 2024 17:04:31 GMT+0000 (Coordinat
ed Universal Time)
console.log(myDate.toDateString()); //Wed Jan 17 2024
console.log(myDate.toISOString()); //2024-01-17T17:06:33.508Z
console.log(myDate.toJSON()); //2024-01-17T17:07:36.266Z
```

```
console.log(myDate.toLocaleString());//1/17/2024, 5:08:18 PM
console.log(myDate.toLocaleDateString());//1/17/2024

let myCreatedDate1 = new Date(2023, 0, 23)
console.log(myCreatedDate1.toLocaleString());//1/23/2023, 12:00:00 AM

let myCreatedDate2 = new Date(2023, 0, 23, 5, 3)
console.log(myCreatedDate2.toLocaleString());//1/23/2023, 5:03:00 AM

let myCreatedDate3 = new Date("2023-01-14")
console.log(myCreatedDate3.toLocaleString());//1/14/2023, 12:00:00 AM

let myCreatedDate4 = new Date("01-14-2023")
console.log(myCreatedDate4.toLocaleString());//1/14/2023, 12:00:00 AM


let myTimeStamp = Date.now()
console.log(myTimeStamp); //1705512093789 //this gives current time in millise
cond

console.log(myCreatedDate1.getTime());//1674432000000//time till that date in
milliseconds

console.log(Math.floor(Date.now()/1000));//1705512258 //current time in second
s

let newDate = new Date()
console.log(newDate.getDate()); //17//todays date
console.log(newDate.getDay());//3//wed, so 3rd day
console.log(newDate.getMonth());//0 // indexing starts from 0 so JAN is 0
console.log(newDate.getFullYear());//2024

console.log(newDate.toLocaleString('default', {
    weekday: "long",
    //we can add other modifications as well here
}))
//Wednesday
```

Lect-14: **Array in Javascript**

- JavaScript arrays are resizable and can contain a mix of different data types.

- JavaScript array-copy operations create shallow copies.
- A **shallow copy** of an object is a copy whose properties share the same references as those of the source object from which the copy was made. As a result, when you change either the source or the copy, you may also cause the other object to change too.
- A **deep copy** of an object is a copy whose properties do not share the same references as those of the source object from which the copy was made. As a result, when you change either the source or the copy, you can be assured you're not causing the other object to change too.

```
let myArray = [1,2,3,true, "Hitesh"]

let myHeros = ["IronMan", "Capt. America", "Thor"]

let myArr = new Array(1,2,3,4,5,6) //another way of defining array in js

console.log(myHeros[2]);//accessing array elements in js //O/P-Thor

let arr = [1, 2, 3, 4, 5]
//push()-push element at the end of the array
arr.push(6)
arr.push(7,8)
console.log(arr);

//pop()-delete element from end of array
arr.pop()
console.log(arr);

//unshift()-adds element at the start of array
arr.unshift(0)
arr.unshift(10,9)
console.log(arr);

//shift()-delete element from start of the array
arr.shift()
console.log(arr);

//includes()-true or false based on element exsits in arr or not
console.log(arr.includes(0));

//indexOf(): returns index of element, if not present then -1
console.log(arr.indexOf(7));
console.log(arr.indexOf(12));
```

```
//join() : converts array into string
let newStr = arr.join()
console.log(arr);
console.log(newStr);
console.log(typeof newStr);//string

//slice(): returns a new array containing the extracted elements.The original
array will not be modified.
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];
const newAnimals = animals.slice(2,4)
console.log(newAnimals);//[ 'camel', 'duck' ]
console.log(animals);//[ 'ant', 'bison', 'camel', 'duck', 'elephant' ] //origi
nal array is not modified

//splice(): changes the contents of an array by removing or replacing existing
elements and/or adding new elements in place. Here original array is changed.
const myFish = ["angel", "clown", "drum", "sturgeon"];
const removed = myFish.splice(2, 1, "trumpet");//remove 1 element at index 2,
and insert "trumpet"
console.log(removed); //[ 'drum' ]
console.log(myFish);//[ 'angel', 'clown', 'trumpet', 'sturgeon' ] //original a
rray is also modified
```

Lect-15: Array part 2 in Javascript

```
const marvelHeros = ["thor", "ironman", "spiderman"]
const dcHeros = ["superman", "flash", "batman"]

marvelHeros.push(dcHeros) //push into the exsiting array
console.log(marvelHeros); //[ 'thor', 'ironman', 'spiderman', [ 'superman', 'f
lash', 'batman' ] ]
console.log(marvelHeros[3][1]);//flash

//concat(): concates two or more arrays and returns a new array
const marvelHeros1 = ["thor", "ironman", "spiderman"]
const dcHeros1 = ["superman", "flash", "batman"]
const allHeros = marvelHeros1.concat(dcHeros1)
console.log(allHeros);//[ 'thor', 'ironman', 'spiderman', 'superman', 'flash',
'batman' ]
```

```
//**spread operator**
const newAllHeros = [...marvelHeros1, ...dcHeros1]
console.log(newAllHeros); //[ 'thor', 'ironman', 'spiderman', 'superman', 'fla
sh', 'batman' ]

//flattening an array: flat() method
const anotherArray = [1, 2, 3, [4, 5, 6], 7, [6, 7, [4, 5]], 10]
const flatendArray = anotherArray.flat(Infinity) //flat(depth-upto-which-u-wan
t-to-flaten-array)
console.log(flatendArray);// [1,  2, 3, 4, 5, 6,  7, 6, 7, 4, 5, 10]

//Array.isArray(): returns true or false based on passed argument
console.log(Array.isArray([1,2,3]));//true
console.log(Array.isArray("Hitesh")); //false

//Array.from() : creates a new array
console.log(Array.from("Hitesh"));//[ 'H', 'i', 't', 'e', 's', 'h' ]
console.log(Array.from({name: "Hitesh" , age: 24}));//[] //interesting case (h
ere we have to specify we want array based on keys or values, otherwise it wil
l return empty array.)

//Array.of()
let score1 = 100, score2 = 200, score3 = 300
console.log(Array.of(score1, score2, score3)); //[ 100, 200, 300 ]
```

Lect-16: **Objects in javascript**

```
//singleton
//Object.create()

//object literals
const obj = {}

const mySym = Symbol("key1")//making string "key1" a symbol

const jsUser = {
    name: "Hitesh",
    "full name": "Hitesh Chodhary",
    [mySym]: "mykey1", //using symbol as key
    age: 18,
    location: "Jaipur",
```

```
    email: "hitesh@google.com",
    isLoggedIn: false,
    lastLoginDays: ["Mon", "Tus"]
}

console.log(jsUser); //prints whole jsUser object
console.log(jsUser.email);//accessing value of "email" key
console.log(jsUser["email"]); //better way of accessing values
console.log(jsUser["full name"]);//here we cant use . to access "full name" --
>(a)
console.log(jsUser[mySym]);//to access symbol key-->(b)

/* (a) and (b) are the advantages of [] over .
*/

//modifying object values
jsUser.email = "hitesh@chatgpt.com"
console.log(jsUser.email);//hitesh@chatgpt.com

//Object.freeze(): used to freeze object, after this any changes made to objec
t will not reflect in the object
//Object.freeze(jsUser)
jsUser["email"] = "hitesh@mssoft.com"
console.log(jsUser.email);//hitesh@chatgpt.com //email was not changed bcoz we
freezed the object

//in js functions are treated as type-1 or first class citizen, because we can
use function as we use normal variable
jsUser.greetingOne = function(){
    console.log("Hello JS user");
}
jsUser.greetingTwo = function(){
    console.log(`Hello JS user, ${this.name}`);//this keyword is used to acces
s keys within that object
}

console.log(jsUser.greetingOne()); //Hello JS user
console.log(jsUser.greetingTwo()); //Hello JS user, Hitesh

console.log(jsUser.greetingTwo); //[Function (anonymous)] //here we are not in
voking function islye it is returning whole function itself
```

Lect-17: **Objects in Javascript part 2**

```
const objOne = new Object()//singleton onject
const objTwo = {} //non singleton object
//this is the only difference b/w two , banki kahani same hai

const tinderUser = {}
tinderUser.id = "123abc"
tinderUser.name = "Sama"
tinderUser.isLoggedIn = false
console.log(tinderUser);

const regualrUser = {
    email: "some@gmail.com",
    fullname: {
        userfullname: {
            firstname: "hitesh",
            lastname: "chowdhary"
        }
    }
}
console.log(regualrUser.fullname.userfullname.firstname);//hitesh

const obj1 = {1: "a", 2: "b"}
const obj2 = {3: "a", 4: "b"}
const obj3 = {5: "a", 6: "b"}

//const obj4 = {obj1 , obj2};
//console.log(obj4);//{ obj1: { '1': 'a', '2': 'b' }, obj2: { '3': 'a', '4':
'b' } } //we get obj1 and obj2 inside other obj

const obj4 = Object.assign(obj1 , obj2);
console.log(obj4);//{ '1': 'a', '2': 'b', '3': 'a', '4': 'b' }
console.log(obj1);//{ '1': 'a', '2': 'b', '3': 'a', '4': 'b' } //problem with
this is that it changes obj1 also

//if you dont want to modify any excisting object use {} with assign
const obj5 = Object.assign({}, obj1, obj2, obj3) //here we get all objects int
o {} object
console.log(obj5);
```

```
//**better way: use spread operator**
const newObj = {...obj1, ...obj2, ...obj3}
console.log(newObj);

console.log(Object.keys(tinderUser));
//[ 'id', 'name', 'isLoggedIn' ] --returns all keys as an array
console.log(Object.values(tinderUser));
//[ '123abc', 'Sama', false ]--returns all values of object as an array
console.log(Object.entries(tinderUser));
//[ [ 'id', '123abc' ], [ 'name', 'Sama' ], [ 'isLoggedIn', false ] ]--returns
[key,value] pair as an array of array

console.log(tinderUser.hasOwnProperty("isLoggedIn"));//true
```

Lect-18: **Object de-structure and JSON API intro**

```
const objOne = new Object()//singleton onject
const objTwo = {} //non singleton object
//this is the only difference b/w two , banki kahani same hai

const tinderUser = {}
tinderUser.id = "123abc"
tinderUser.name = "Sama"
tinderUser.isLoggedIn = false
console.log(tinderUser);

const regualrUser = {
    email: "some@gmail.com",
    fullname: {
        userfullname: {
            firstname: "hitesh",
            lastname: "chowdhary"
        }
    }
}
console.log(regualrUser.fullname.userfullname.firstname);//hitesh

const obj1 = {1: "a", 2: "b"}
const obj2 = {3: "a", 4: "b"}
const obj3 = {5: "a", 6: "b"}
```

```
//const obj4 = {obj1 , obj2};
//console.log(obj4);//{ obj1: { '1': 'a', '2': 'b' }, obj2: { '3': 'a', '4':
'b' } } //we get obj1 and obj2 inside other obj

const obj4 = Object.assign(obj1 , obj2);
console.log(obj4);//{ '1': 'a', '2': 'b', '3': 'a', '4': 'b' }
console.log(obj1);//{ '1': 'a', '2': 'b', '3': 'a', '4': 'b' } //problem with
this is that it changes obj1 also

//if you dont want to modify any excisting object use {} with assign
const obj5 = Object.assign({}, obj1, obj2, obj3) //here we get all objects int
o {} object
console.log(obj5);

//**better way: use spread operator**
const newObj = {...obj1, ...obj2, ...obj3}
console.log(newObj);

console.log(Object.keys(tinderUser));
//[ 'id', 'name', 'isLoggedIn' ] --returns all keys as an array
console.log(Object.values(tinderUser));
//[ '123abc', 'Sama', false ]--returns all values of object as an array
console.log(Object.entries(tinderUser));
//[ [ 'id', '123abc' ], [ 'name', 'Sama' ], [ 'isLoggedIn', false ] ]--returns
[key,value] pair as an array of array

console.log(tinderUser.hasOwnProperty("isLoggedIn"));//true


/*Object de-structure and JSON API intro*/

/*
Destructuring is a JavaScript expression that allows us to extract data from a
rrays, objects, and maps and set them into new, distinct variables.
*/


const course = {
    coursename: "JS in Hindi",
    price: "999",
    courseInstructor: "hitesh"
}
```

```
//Object Destructuring in js
const {courseInstructor} = course
console.log(courseInstructor);//hitesh
const {courseInstructor: instrcutor} = course
console.log(instrcutor);//hitesh

/*
//API: API is apne kaam ko dusre ke sar pe dalna;task delegate kr dena


//JSON
{
    "name": "hitesh",
    "email": "some@gmail.com",
    "age": 18,
    "isLoggedIn": false
}

//other format of json data
[
    {},
    {},
    {}
]

//there are other formats of json as well

//use "JSON Formatter" to understand json data
*/
```

Lect-19: **Functions and parameters in javascript**

```
//Function: a sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that parti cular task should be performed

function sayMyName(){
    console.log("P");
    console.log("K");
}
sayMyName //refrence
```

```
sayMyName() //execution

//parameter: variables that we pass inside a function declaration
//argumnets: values/variables that we pass during function call

function addTwoNumbers(number1, number2){
    console.log(number1 + number2);
}
addTwoNumbers(3, 4) //7
addTwoNumbers(3, "4") //34 (so we should check if values passed is Number or n
ot)

function addTwo(number1, number2){
    let res = number1 + number2
    return res;
}
const result = addTwo(5, 5)
console.log(result)

function loginUserMessage(username){
    if(username === undefined){
        console.log("Please enter a valid username!");
        return;
    }
    return `${username} just logged in.`
}
console.log(loginUserMessage("Hitesh"));//Hitesh just logged in.
console.log(loginUserMessage());//Please enter a valid username!
```

Lect-20: Functions with objects and array in javascript

```
//rest operator '...' => this bundles all the numbers in an array and returns
an array
function calculateCartPrice(...num1){
    return num1;
}

console.log(calculateCartPrice(200, 500, 400, 2000)); //[ 200, 500, 400, 2000
]

function calculateCartPrice1(val1, val2, ...num1){
```

```
    return num1;
}

console.log(calculateCartPrice1(200, 500, 400, 2000));//[ 400, 2000 ] => here
first two values go to val1 & val2 and rest to num1

//object
const user = {
    username: "Sam",
    price: 199
}

function handleObject(anyObject){
    console.log(`Username is ${anyObject.username} and price is ${anyObject.pr
ice}`);
}

handleObject(user) //Username is Sam and price is 199

handleObject({
    username: "Hitesh",
    price: 399
}) //Username is Hitesh and price is 399

//arrays
const myNewArray = [200, 400, 100, 500]

function returnSecondValue(arr){
    return arr[1]
}

console.log(returnSecondValue(myNewArray)); //400
```

Lect-21: **Global and local scope in javascript**

```
//global scope
let a = 100
const b = 200
var c = 300  //var has global scope no matter it is defined inside block or gl
obally
d = 400 //global scope
```

```
//block scope : let & const have block scope
if(true){
    let a = 1
    const b = 2
    var c = 3
    d = 4
    console.log("Inner:", a); //1
    console.log("Inner:", b); //2
    console.log("Inner:", c); //3
    console.log("Inner:", d); //4
}

console.log("Outer:", a); //100
console.log("Outer:", b); //200
console.log("Outer:", c); //3
console.log("Outer:", d); //4
```

Lect-22: **Scope level and mini hoisting in javascript**

```
//nested scope
function one(){
    const username = "hitesh"

    function two(){
        const website = "youtube"
        console.log(username);
    }

    //console.log(website); //error =>website out of scope

    two()
}

one()


if(true){
    const username = "hitesh"
    if(username === "hitesh"){
        const website = " youtube"
```

```
        console.log(username + website);
    }
    //console.log(website); // error =>website out of scope
}
//console.log(username); //error=> username out of scope

/* ++++++++++Interesting+++++++++ */

console.log(addOne(5));//6 => correct => function declarations can be hoisted

//function declaration
function addOne(num){
    return num + 1;
}

//console.log(addTwo(5));//error=>function expressions cant be hoisted

//function expression
const addTwo = function(num){
    return num + 2;
}

console.log(addOne(5)); //6
console.log(addTwo(5)); //7
```

Lect-23: **THIS and arrow function in javascript**

```
//this: this is used to refer to current context (object)

const user = {
    username: "hitesh",
    price: 999,

    welcomeMessage: function(){
        console.log(`${this.username}, Welcome to website.`);

        console.log(this); //here this will give current object ie. 'user' obj
ect
    }
}
```

```
user.welcomeMessage() //hitesh, Welcome to website.

user.username = "sam"
user.welcomeMessage()//sam, Welcome to website.

console.log(this); //here 'this' will give an empty object {}
//Note: In browser console , here 'this' will return 'window' object


function chai1(){
    let username = "hitesh"
    console.log(this.username) //undefined
}
chai1()

const chai2 = function(){
    let username = "hitesh"
    console.log(this.username) //undefined
}
chai2()


/*************Arrow function************ */
const chaiArrow = () => {
    let username = "hitesh"
    console.log(this);//{}
}
chaiArrow()

//Explicit return: (when using return keyword)
const addTwo = (num1, num2) => {
    return num1 + num2;
}
console.log(addTwo(3, 4));//7

//implicit return
//Method-1
const addTwo = (num1, num2) => num1 + num2

//Method-2
const addTwo = (num1, num2) => (num1 + num2)
/*Note: When we use () bracket there is no need to use 'return'. But when we u
```

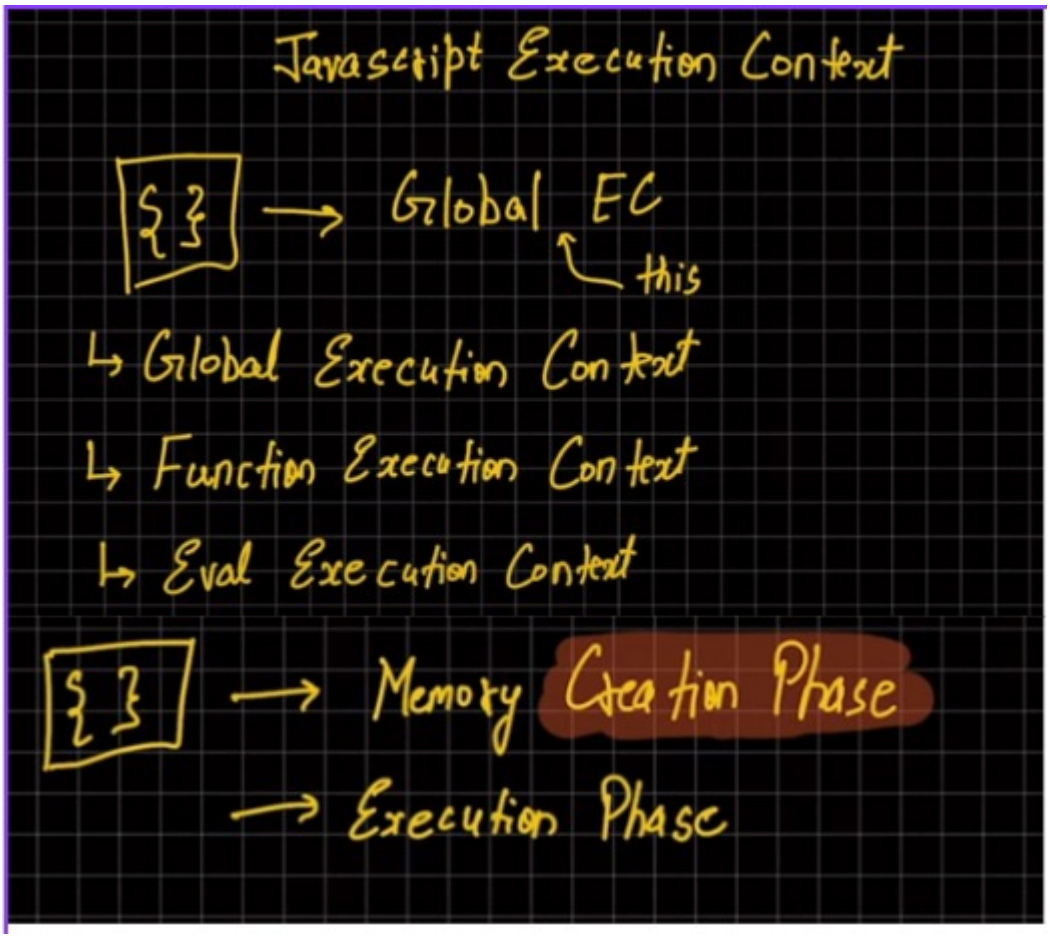
```
se {} bracket we have to use 'return' */

const add = (num1, num2) => ({username: "hitesh"})
console.log(add(3,4));//{username: "hitesh"}
```

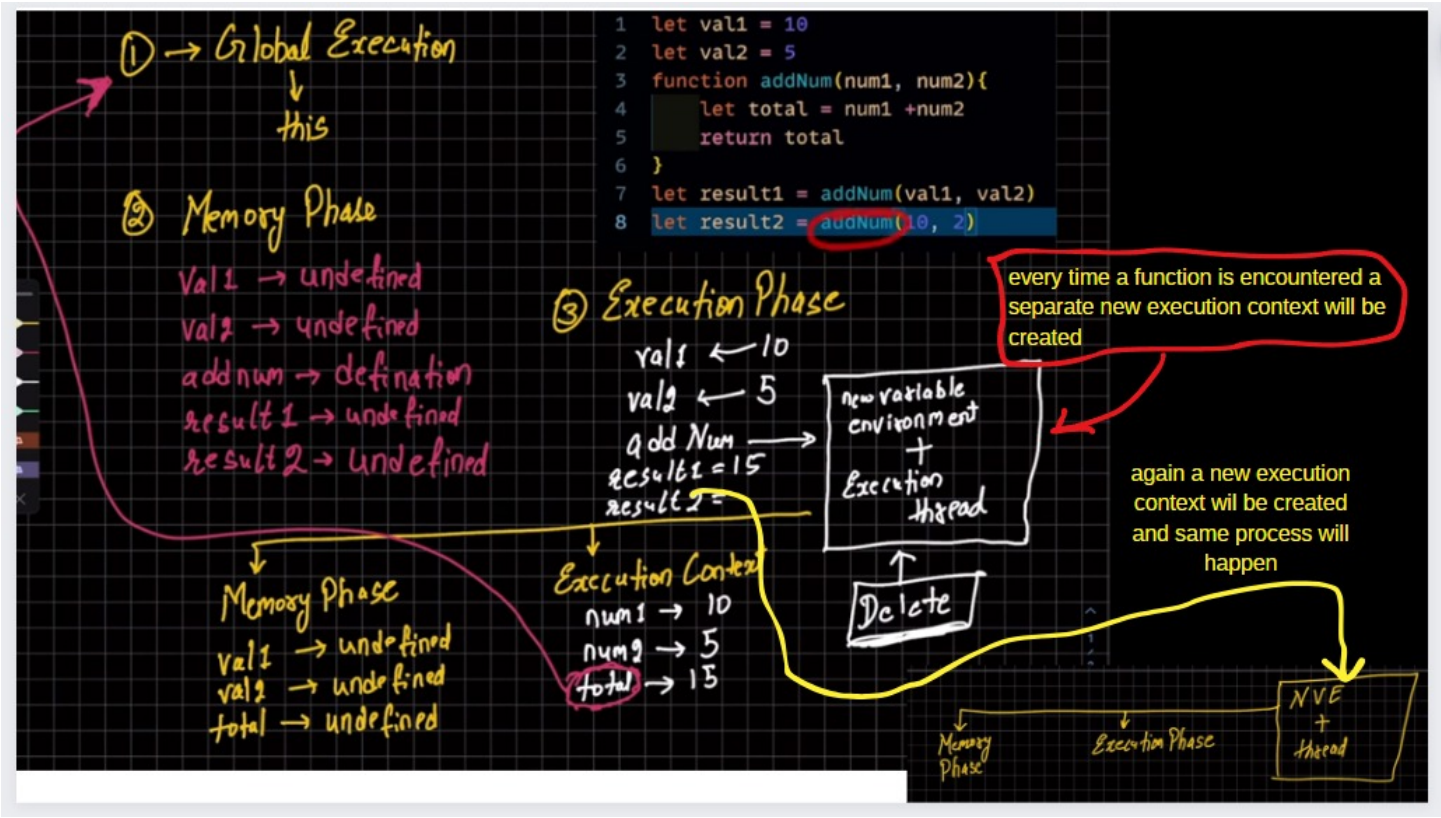
Lect-24: **Immediately Invoked Function Expressions IIFE**

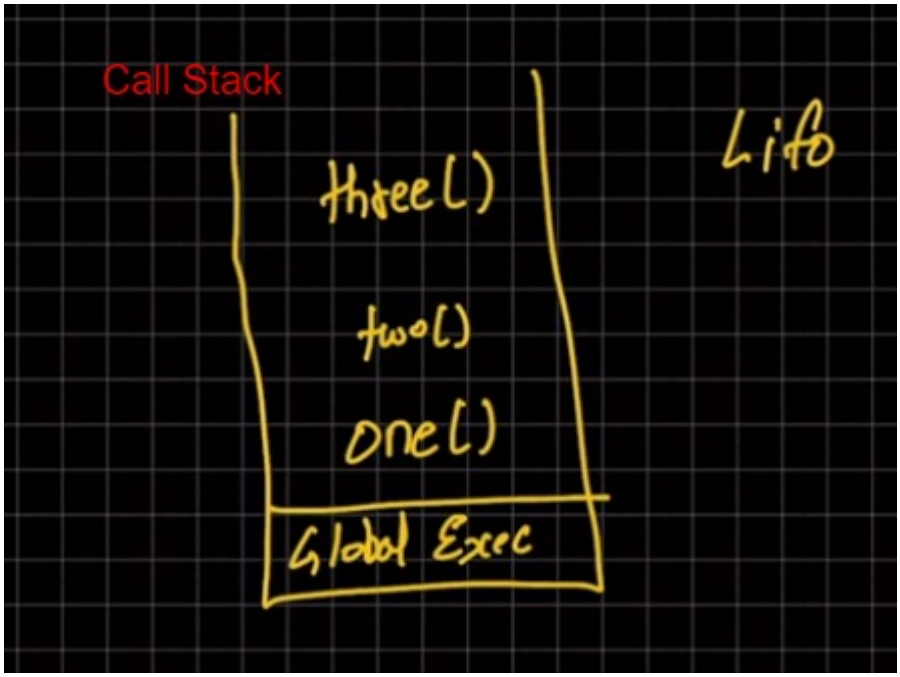
```
//An IIFE (Immediately Invoked Function Expression) is a JavaScript function t
hat runs as soon as it is defined.

/*Avoid polluting the global namespace: Because our application could include
many functions and global variables from different source files, it's importan
t to limit the number of global variables. If we have some initiation code tha
t we don't need to use again, we could use the IIFE pattern.*/

//many times we have problem with global scope pollution, so to avoid that we
use IIFE.

//**at the end of the IIFE a semicolon is necessary**

//named IIFE
(function chai(){
    console.log(`DB CONNECTED`);
})(); //DB CONNECTED

//unnamed IIFE / arrow function
(() => {
    console.log(`DB TWO CONNECTED`);
})(); //DB TWO CONNECTED

//IIFE with arguments and parameter
((name) => {
    console.log(`DB CONNECTED TO ${name}`);
})("hitesh"); //DB CONNECTED TO hitesh

//two IIFE together: write first IIFE then use ; and then write second IIFE be
low it.
(() => {
    console.log("IIFE1");
})();
```

```
(() => {
    console.log("IIFE2");
})();
```

Lect-25: **How does javascript execute code + call stack**








```
//javascript is single threaded

/*
1. JS creates a Global execution context
2. Next it creates the memory phase
3. Memory phase - In this phase, the variables are set to undefined.
  until the execution phase(next phase) and the functions are set to their def
initions.
4. Next it creates the execution phase
```

```
5. Execution phase - In this phase, the variables are initialised to given val
ues and
  when the functions are called, it creates a memory phase and execution phase
for the function
  just like the main one.
*/
```

Lect: 26: **Control flow in javascript in 1 shot**

```
/*
//if statement
if(condition){
    //statements
}

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

//if-else
if(condition){
    //if condition is true run this block
}
else{
    //if false runs this block
}
*/

const score = 200

if(score > 100){
    const power = "Fly"
    console.log(`User power: ${power}`);
}
else{
    console.log(`Score less than 100, so can't fly.`);
}

const balance = 1000

//if(balance > 500)   console.log("test1"), console.log("test2"); //correct bu
t bad code
```

```
//if else-if
if(balance < 500){
    console.log("Less than 500")
}
else if(balance < 1000){
    console.log("Less than 1000");
}
else{
    console.log("Balance greater than or equal to 1000");
}

const userLoggedIn = true
const debitCard = true
const loggedInFromGoogle = false
const loggedInFromEmail = true

if(userLoggedIn && debitCard){
    console.log("Sllow to buy course")
}

if(loggedInFromGoogle || loggedInFromEmail){
    console.log("User Logged In");
}
```

```
/*
//switch case
switch(key) {
    case value:
        //statement
        break;
    default:
        //statement
        break;
}
*/

const month = 11

switch (month) {
    case 1:
```

```
        console.log("January");
        break;
    case 12:
        console.log("February");
        break;
    case 3:
        console.log("March");
        break;
    case 4:
        console.log("April");
        break;
    case 5:
        console.log("May");
        break;
    case 6:
        console.log("June");
        break;
    case 7:
        console.log("July");
        break;
    case 8:
        console.log("August");
        break;
    case 9:
        console.log("September");
        break;
    case 10:
        console.log("October");
        break;
    case 11:
        console.log("November");
        break;
    case 12:
        console.log("December");
        break;

    default:
        console.log("Enter correct number");
        break;
}

//Note: if there is no break statement, below codes are also executed until it
```

```
encounters break.

const day = "Mon"

switch (day) {
    case "Mon":
        console.log("Monday");
        break;
    case "Tus":
        console.log("Tuesday");
        break;
    case "Wed":
        console.log("Wednesday");
        break;
    default:
        console.log("Other days");
        break;
}
```

```
const userEmail = "hitesh@gpt.ai"

if(userEmail){
    console.log("Got email ID");
}
else{
    console.log("Don't have user email.");
}

//"hitesh" =>true
//"" => false
//[] => true

//falsy values => false, 0, -0, BigInt 0n, "", null, undefined, NaN

//truthy values => "0", 'false', " ", [], {}, function(){}

const arr = []
if(arr.length === 0){
    console.log("Array is empty");
}
```

```
const emptyObj = {}
if(Object.keys(emptyObj).length === 0){
    console.log("Object is empty.");
}

/*
false == 0 //true
false == '' //true
0 == '' //true
*/


//Nullish Coalescing Operator (??) : null, undefined

let val1;
val1 = 5 ?? 10
console.log(val1); //5

val1 = null ?? 10
console.log(val1); //10

val1 = undefined ?? 15
console.log(val1); //15

val1 = null ?? 10 ?? 20
console.log(val1); //10


//Terniary operator
//condition ? true : false

const iceTeaPrice = 100
iceTeaPrice <= 80 ? console.log("less tah 80") : console.log("more than 80")
//more than 80
```

Lect-27: **For loop with break and continue in javascript**

```
//for loop
for(let i = 0; i < 10; i++){
    console.log(i);
}
```

```
//first intialization is done; then condition is checked and THEN statements i
nside block is executed and then increment happens--> then again condition is
check --> block is executed-> then increment happens and it continues till con
dition is true


let myArray = ["flash", "batman", "superman"]
for (let index = 0; index < myArray.length; index++) {
    const element = myArray[index];
    console.log(element);
}


//break and continue
for (let index = 1; index < 20; index++) {
    if(index === 5){
        console.log("Detected 5");
        break;
    }
    console.log(`Value of i is ${index}`);
}

//continue
for (let index = 1; index < 20; index++) {
    if(index === 5){
        console.log("Detected 5");
        continue
    }
    console.log(`Value of i is ${index}`);
}
```

Lect-28: **While and do while loop in Javascript**

```
//while loop
let i = 0;
while(i <= 10){
    console.log(`Value of index i is ${i}`);
    i += 2
}
```

```
//do-while loop
let score =1
do {
    console.log(score);
    score++
} while (score <= 10);
```

Lect-29: **High Order Array loops**

```
//**********for-of loop***********

const arr = [1, 2, 3, 4, 5]

for(const num of arr){
    console.log(num);
}

const greetings = "Hello World!"

for(greet of greetings){
    if(greet === " ")
        continue
    console.log(`Each char is: ${greet}`);
}



/*Maps: *key, value pair
        *uniques keys
        *maintains order in which these key value pairs are inserted
*/
const map = new Map()
map.set('IN', "India")
map.set('USA', "United States of America")
map.set('Fr', "France")

console.log(map);

//for of loop on map
for (const [key, value] of map) {
    console.log(key, ':-', value);
}
```

```
//for of loop on object: for of is not used with object
const obj = {
    game1: "NFS",
    game2: "Spiderman"
}
```

```
/************For-in loop*************/
//for object iteration we use 'for in loop'
const myObj = {
    js: "javascript",
    cpp: "C++",
    rb: "ruby",
    swift: "swift by apple"
}

for(const key in myObj){
    console.log(`${key} : ${myObj[key]}`);
}

//for-in for array
const arr = ["js", "cpp", "swift", "java", "python"]

for(const key in arr){
    console.log(key);//this will give index of array elements and not the elem
ent
    //to get array elements
    console.log(arr[key]);
}
//so avoid using for-in with arrays

//Note: for-in is not used with maps
```

```
//***********forEach loop************

const coding = ["js", "ruby",
"java", "cpp", "python"]

//Way-1
coding.forEach( function(val){
    console.log(val);
```

```
} )

//Way-2
coding.forEach( (item) => {
    console.log(item)
} )

//Way-3
function printMe(item){
    console.log(item);
}

coding.forEach(printMe) //Note:here we only pass function reference

/*
Output of all 3:
js
ruby
java
cpp
python
*/

//forEach have access to index and whole array, apart from array items

coding.forEach( (item, index, arr) => {
    console.log(item, index, arr);
})
/*
Output:
js 0 [ 'js', 'ruby', 'java', 'cpp', 'python' ]
ruby 1 [ 'js', 'ruby', 'java', 'cpp', 'python' ]
java 2 [ 'js', 'ruby', 'java', 'cpp', 'python' ]
cpp 3 [ 'js', 'ruby', 'java', 'cpp', 'python' ]
python 4 [ 'js', 'ruby', 'java', 'cpp', 'python' ]
 */


//array off objects

const myCoding = [
    {
```

```
        languageName: "javascript",
        fileName: "js"
    },
    {
        languageName: "python",
        fileName: "py"
    },
    {
        languageName: "C++",
        fileName: "cp"
    }
]

myCoding.forEach( (item) => {
    console.log(item.fileName);
})

/**
Output:
js
py
cp
*/

//The forEach() method is an iterative method. It calls a provided callbackFn function once for each element in an array in ascending-index order.

//forEach() always returns undefined and is not chainable.
//Therefore we use map() and other functions

//There is no way to stop or break a forEach() loop other than by throwing an exception. If you need such behaviour, the forEach() method is the wrong tool.

//forEach() do not allow chaining
```

Lect-30: **Filter, map and reduce in javascript**

```
//filter() -- unlike forEach(), filter() returns values

const myNums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
const newNums = myNums.filter( (num) => num > 4)

console.log(newNums);
//[ 5, 6, 7, 8, 9, 10 ]

//whenever we use {} brackets we have to use return keyword
const newNums1 = myNums.filter( (num) => {
    return num > 4
} )

console.log(newNums1);
//[ 5, 6, 7, 8, 9, 10 ]


//doing same thing as above using forEach()

const newNums2 = []

myNums.forEach( (num) => {
    if(num > 4){
        newNums2.push(num)
    }
})

console.log(newNums2);
//[ 5, 6, 7, 8, 9, 10 ]


const books = [
    { title: 'Book One', genre: 'Fiction', publish: 1981, edition: 2004 },
    { title: 'Book Two', genre: 'Non-Fiction', publish: 1992, edition: 2008 },
    { title: 'Book Three', genre: 'History', publish: 1999, edition: 2007 },
    { title: 'Book Four', genre: 'Non-Fiction', publish: 1989, edition: 2010
},
    { title: 'Book Five', genre: 'Science', publish: 2009, edition: 2014 },
    { title: 'Book Six', genre: 'Fiction', publish: 1987, edition: 2010 },
    { title: 'Book Seven', genre: 'History', publish: 1986, edition: 1996 },
    { title: 'Book Eight', genre: 'Science', publish: 2011, edition: 2016 },
    { title: 'Book Nine', genre: 'Non-Fiction', publish: 1981, edition: 1989
},
  ];
```

```
let userBooks = books.filter( (book) => book.genre === 'History' )

userBooks = books.filter( (book) => {
    return book.publish >= 2000 && book.genre === "Science"
} )

userBooks = books.filter( (book) => book.edition >= 2005 && book.publish >= 20
00)
console.log(userBooks);
```

```
/*********map() method *********/
const myNums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

const newNums = myNums.map( (num) => num + 10 )

console.log(newNums); //[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

//chaining
const newNumChain = myNums
                    .map( (num) => num * 10 )
                    .map( (num) => num + 1)
                    .filter( (num) => num >= 50)

console.log(newNumChain) //[ 51, 61, 71, 81, 91, 101 ]

//The map() method is an iterative method. It calls a provided callbackFn func
tion once for each element in an array and constructs a new array from the res
ults.
```

```
//*************Array.reduce()********************
const array1 = [1, 2, 3, 4, 5]
const initialValue = 0
const sum = array1.reduce( (accumulator , currentValue) => accumulator + curre
ntValue , initialValue)

console.log(sum); //15

const arr = [1, 2, 3]
const total = arr.reduce( function(acc, currVal) {
    console.log(`acc = ${acc} and currVal = ${currVal}`)
    return acc + currVal
```

```
}, 0)
console.log(total)
/*
Output:
acc = 0 and currVal = 1
acc = 1 and currVal = 2
acc = 3 and currVal = 3
6
*/

const arr1 = [1, 2, 3, 4]
const arr1Total = arr1.reduce( (acc, currVal) => acc + currVal , 0)
console.log(arr1Total); //10


const shoppingCart = [
    {
        itemName: "js course",
        price: 2999
    },
    {
        itemName: "py course",
        price: 999
    },
    {
        itemName: "mobile dev course",
        price: 5999
    },
    {
        itemName: "data science course",
        price: 12999
    },
]

const priceToPay = shoppingCart.reduce( (acc, item) => acc + item.price , 0 )
console.log(priceToPay); //22996
```

Lect-31: **DOM introduction in javascript**

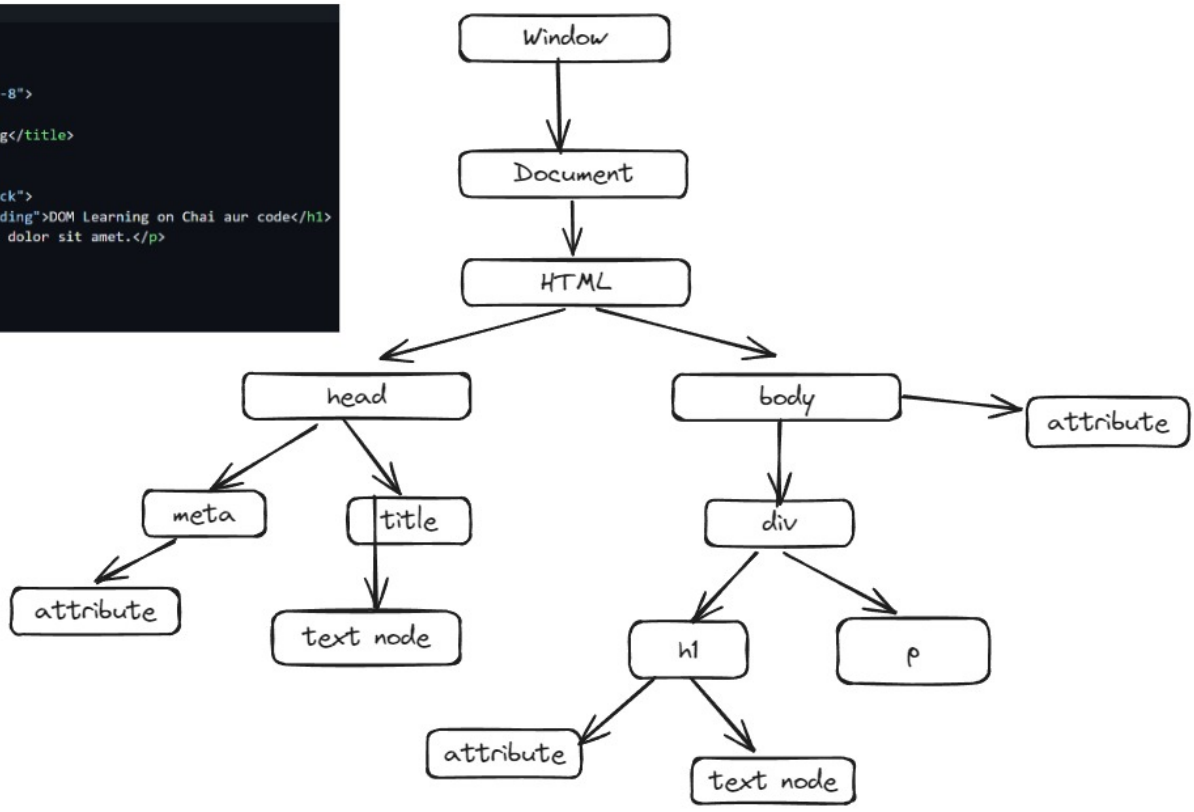
- console.log(window)
- console.log(window.document)

or
console.log(document) //gives all the HTMLs and other web page related codes

- console.dir(document) //gives hidden information as well

- document.getElementById(element_ID)
- let html = document.getElementById("myP").innerHTML; //this gives text inside of this myP ID
- Change the HTML content of an element with id="demo"
  document.getElementById("demo").innerHTML = "I have changed!";

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

    <title>DOM Learning</title>
</head>
<body>
    <div class="bg-black">
        <h1 class="heading">DOM Learning on Chai aur code</h1>
        <p>Lorem ipsum dolor sit amet.</p>
    </div>

</body>
</html>
```



- console.log(document.link) //gives all the links in the webpage.
  Note⇒ these links are not in the form of array , they are HTML Collection.

Lect-32: **All DOM selectors NodeList and HTMLCollection**

```
> document.getElementById('title')
<   <h1 id="title" class="heading">DOM Learning on Chai aur code</h1>
> document.getElementById('title').id
< 'title'
> document.getElementById('title').className
< 'heading'
> document.getElementById('title').getAttribute('id')
< 'title'
> document.getElementById('title').getAttribute('class')
< 'heading'
> document.getElementById('title').setAttribute('class', 'test') //this overrides the heading class
< undefined
```

this adds a new class to the selected element

Adding style

Console

top ▼ | Filter

```
> const title = document.getElementById('title')
< undefined
> title
<   <h1 id="title" class="test heading ">DOM Learning on Chai aur code</h1>
> title.style.backgroundColor = 'green'
< 'green'
> title.style.color = 'white'
< 'white'
> title.style.padding = '15px'
< '15px'
> title.style.borderRadius = '15px'
< '15px'
```

Adding content

```
> title
<   <h1 id="title" class="test heading" style="background-color: green; color: white;
    padding: 15px; border-radius: 15px;">DOM Learning Code</h1>
> title.textContent
< 'DOM Learning Code'
> title.innerHTML
< 'DOM Learning Code'
> title.innerText
< 'DOM Learning Code'
```

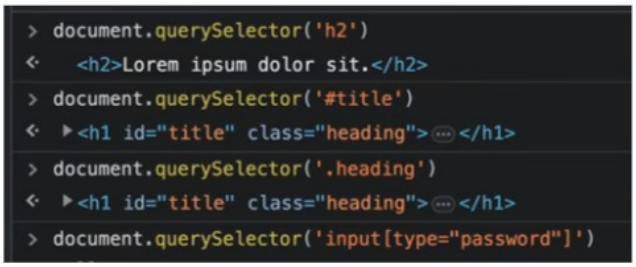
=> innerText reads text as it is rendered on screen;
it displays only visible text on the screen.

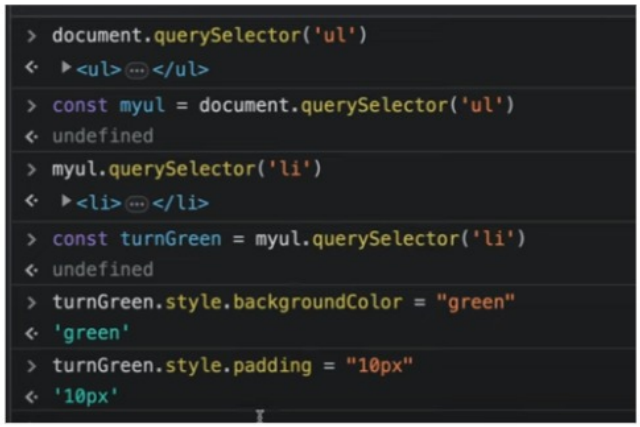
=> textContent reads text as it is in the markup.
It also returns all text, whether it's rendered on screen or not.

=> innerHTML returns text along with any tags that is present.
for ex:

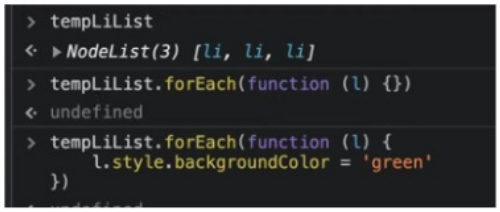
```
> title.innerHTML
< 'DOM learning on Chai aur code <span style="display: none;">test t
  ext</span>'
```

=>document.querySelector() =>this will give first element



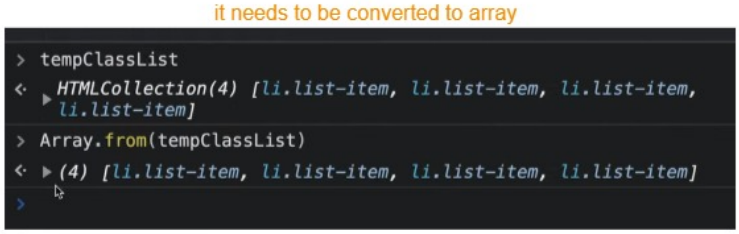


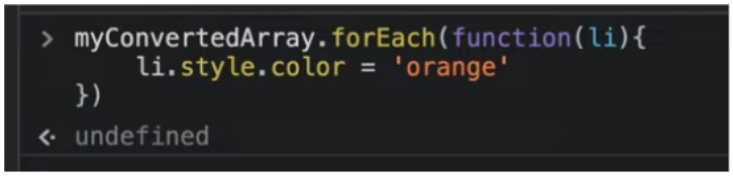
=> document.querySelectorAll() : it gives all in the form of node list (this is not an array, but it is similar to array -> forEach() can be used in node list)



=> getElementByClassName() : gives all the item with given class name, it returns a HTML collections

it needs to be converted to array





Lect-33: **How to create a new element in DOM**

div.day*4 ==> this will give 4 divs with day class

```
<div class="day">Monday</div>
<div class="day">Tuesday</div>
<div class="day">Wednes</div>
<div class="day"></div>
```

```
</body>
<script>
    const parent = document.querySelector('.parent')
    // console.log(parent);
    // console.log(parent.children);
    // console.log(parent.children[1].innerHTML);

    for (let i = 0; i < parent.children.length; i++) {
        console.log(parent.children[i].innerHTML);

    }
    parent.children
</script>
</html>
```

```
02_two.html:56
▼<div class="parent">
  <div class="day">Monday</div>
  <div class="day">Tuesday</div>
  <div class="day">Wednesday</div>
  <div class="day">Thursday</div>
</div>
02_two.html:57
HTMLCollection(4) [div.day, div.day, div.day, div.day]
  ▶ 0: div.day
  ▶ 1: div.day
  ▶ 2: div.day
  ▶ 3: div.day
    length: 4
  ▶ [[Prototype]]: HTMLCollection
Tuesday 02_two.html:58
Monday 02_two.html:61
Tuesday 02_two.html:61
Wednesday 02_two.html:61
Thursday 02_two.html:61
```

```
parent.children[1].style.color = 'orange'

console.log(parent.firstElementChild)
console.log(parent.lastElementChild)

const dayOne = document.querySelector('.day')
console.log(dayOne)
console.log(dayOne.parentElement);

console.log(dayOne.nextElementSibling)
```

```
<div class="day">Monday</div> 02_two.html:66
<div class="day">Thursday</div> 02_two.html:67
<div class="day">Monday</div> 02_two.html:70
<div class="parent">…</div> 02_two.html:71
02_two.html:73
<div class="day" style="color: orange;">Tuesday
</div>
```

Node List => parent.childNodes --->> returns a live NodeList of child nodes of the given element where the first child node is assigned index 0 .
Child nodes include elements, text and comments and even line break is counted.

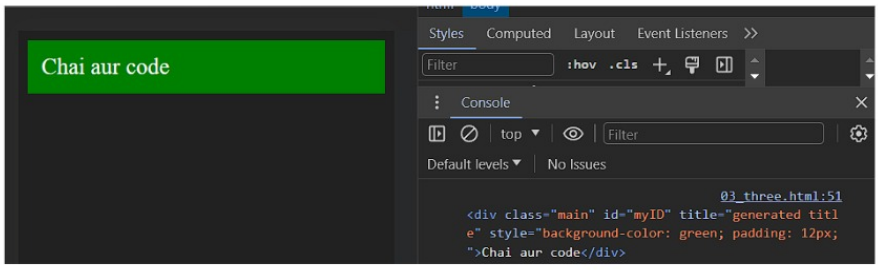
```
console.log("NODES: ", parent.childNodes);
```

```
NODES: 02_two.html:75
▼NodeList(9) [text, div.day, text, div.day, text, div.day, text, div.day, text]
  ▶ 0: text
  ▶ 1: div.day
  ▶ 2: text
  ▶ 3: div.day
  ▶ 4: text
  ▶ 5: div.day
  ▶ 6: text
  ▶ 7: div.day
  ▶ 8: text
    length: 9
  ▶ [[Prototype]]: NodeList
```

Note--> we have only four html element in parent class but length of Node list is 11. It is beacuse everything is included in the tree strcture like line break, text, comment, element

==>>Creating element using DOM

```
<script>
    const div = document.createElement('div')
    console.log(div)
    div.className = 'main'
    div.id = 'myID'
    div.setAttribute("title", "generated title")
    div.style.backgroundColor = "green"
    div.style.padding = "12px"
    //div.innerHTML = "Chai aur code" //this is also correct or use below method- below two lines
    const addText = document.createTextNode("Chai aur code")
    div.appendChild(addText)

    //attaching div to body
    document.body.appendChild(div)
</script>
```



lect-34: **Edit and remove elements in DOM**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Chai aur Code | DOM</title>
</head>
<body style="background-color: #212121; color: #fff;">
    <ul class="language">
        <li>JavaScript</li>
    </ul>
</body>
<script>

    //Add
    function addLanguage(langName){
        const li = document.createElement('li')
        li.innerHTML = `${langName}`
        document.querySelector('.language').appendChild(li)
    }
    addLanguage('Python')
    addLanguage('typescript')

    //above function is okay, but it is not optmized. Everytime function is ca
lled it traverse whole tree structure. So we can optimize function
    function adddOptiLanguage(langName){
        const li = document.createElement('li')
        li.appendChild(document.createTextNode(langName))
        document.querySelector('.language').appendChild(li)
    }
    adddOptiLanguage("golang")

    //Edit
    const secondLang = document.querySelector('li:nth-child(2)')
    console.log(secondLang)
    const newli = document.createElement('li')
    newli.textContent = "Mojo" //newli.innerHTML = "Mojo"
    secondLang.replaceWith(newli) //replace python with 'Mojo'

    //Edit
    const firstLang = document.querySelector('li:first-child')
    // const new_li = document.createElement('li')
```

```
    // new_li.innerHTML = "Typescript"
    // firstLang.replaceWith(new_li)
    firstLang.outerHTML = '<li>Typescript</li>' //other method

    //Remove
    const lastLang = document.querySelector('li:last-child')
    lastLang.remove()


</script>
</html>
```

Lect-35: **Let's build 4 javascript projects for beginners**

Projects related to DOM

Project Link

Click Here

Solution Code

Project - 1 : colorChanger

```
const buttons = document.querySelectorAll('.button');
const body = document.querySelector('body');

buttons.forEach(function (button) {
  button.addEventListener('click', function (e) {
    if (e.target.id === 'grey') {
      body.style.backgroundColor = e.target.id;
    }
    if (e.target.id === 'white') {
      body.style.backgroundColor = e.target.id;
    }
    if (e.target.id === 'blue') {
      body.style.backgroundColor = e.target.id;
    }
    if (e.target.id === 'yellow') {
      body.style.backgroundColor = e.target.id;
```

```
    }
  });
});
```

Project - 2 : bmiCalculator

```
const form = document.querySelector('form');
//this usecase will give you empty value
//const height = parseInt(document.querySelector('#height').value)

form.addEventListener('submit', function (e) {
  e.preventDefault(); //method used to prevent or stop default action of form
  const height = parseInt(document.querySelector('#height').value);
  const weight = parseInt(document.querySelector('#weight').value);
  const results = document.querySelector('#results');
  const message = document.querySelector('#message');

  if (height === '' || height < 0 || isNaN(height)) {
    results.innerHTML = `Please Enter valid height`;
  } else if (weight === '' || weight < 0 || isNaN(weight)) {
    results.innerHTML = `Please Enter valid weight`;
  } else {
    const bmi = (weight / ((height * height) / 10000)).toFixed(2);
    results.innerHTML = `<span>${bmi}</span>`;
    if (bmi < 18.6) {
      message.innerHTML = `<span><p>You are Under Weight</p></span>`;
    } else if (bmi >= 18.6 && bmi <= 24.9) {
      message.innerHTML = `<span><p>You are in Normal Range</p></span>`;
    }
    if (bmi > 24.9) {
      message.innerHTML = `<span><p>You are Over Weight</p></span>`;
    }
  }
});
```

Project - 3 : DigitalClock

```
//const clock = document.querySelector('#clock') //use any method getElementBy
Id() or querySelector()
```

```
const clock = document.getElementById('clock');

setInterval(function () {
  let date = new Date();
  //console.log(date.toLocaleTimeString())
  clock.innerHTML = date.toLocaleTimeString();
}, 1000);
```

Project - 4: Guess The Number

```
let randomNumber = parseInt(Math.random() * 100 + 1);

const submit = document.querySelector('#subt');
const userInput = document.querySelector('#guessField');
const guessSlot = document.querySelector('.guesses');
const remaining = document.querySelector('.lastResult');
const lowOrHi = document.querySelector('.lowOrHi');
const startOver = document.querySelector('.resultParas');

const p = document.createElement('p');

let prevGuess = [];
let numGuess = 1;

let playGame = true;

if (playGame) {
  submit.addEventListener('click', function (e) {
    e.preventDefault();
    const guess = parseInt(userInput.value);
    console.log(guess);
    validateGuess(guess);
  });
}

function validateGuess(guess) {
  if (isNaN(guess) || guess === '') {
    alert('Please enter a valid number');
  } else if (guess < 1) {
    alert('Please enter a number greater than 0');
```

```
  } else if (guess > 100) {
    alert('Please enter a number less that or equal to 100');
  } else {
    prevGuess.push(guess);
    if (numGuess === 11) {
      displayGuess(guess);
      displayMessage(`Game over! Random number was ${randomNumber}`);
      endGame();
    } else {
      displayGuess(guess);
      checkGuess(guess);
    }
  }
}

function checkGuess(guess) {
  if (guess === randomNumber) {
    displayMessage(`You guessed it right. You Won!!`);
    endGame();
  } else if (guess < randomNumber) {
    displayMessage(`Number is TOO low`);
  } else if (guess > randomNumber) {
    displayMessage(`Number is TOO high`);
  }
}

function displayGuess(guess) {
  userInput.value = '';
  guessSlot.innerHTML += `${guess}, `;
  numGuess++;
  remaining.innerHTML = `${11 - numGuess}`;
}

function displayMessage(message) {
  lowOrHi.innerHTML = `<h2>${message}</h2>`;
}

function endGame() {
  userInput.value = '';
  userInput.setAttribute('disabled', '');
  p.classList.add('button');
  p.innerHTML = `<h2 id="newGame">Start New Game!</h2>`;
```

```
  startOver.appendChild(p);
  playGame = false;

  newGame();
}

function newGame() {
  const newGameButton = document.querySelector('#newGame');
  newGameButton.addEventListener('click', function (e) {
    randomNumber = parseInt(Math.random() * 100 + 1);
    prevGuess = [];
    numGuess = 1;
    guessSlot.innerHTML = '';
    remaining.innerHTML = `${11 - numGuess}`;
    userInput.removeAttribute('disabled');
    lowOrHi.innerHTML = '';
    startOver.removeChild(p);

    playGame = true;
  });
}
```

Lect-36: **Events in Javascript**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>htnml Events </title>
</head>
<body style="background-color: #414141; color: aliceblue;">
    <h2>Amazing image</h2>
    <div >
        <ul id="images">
            <li><img width="200px" id="photoshop" src="https://images.pexels.c
om/photos/3561339/pexels-photo-3561339.jpeg?auto=compress&cs=tinysrgb&w=1600&l
azy=load" alt="photoshop"></li>
            <li><img width="200px" id="japan" src="https://images.pexels.com/p
hotos/3532553/pexels-photo-3532553.jpeg?auto=compress&cs=tinysrgb&w=1600&lazy=
load" alt=""></li>
```

```
            <li><img width="200px" id="river" src="https://images.pexels.com/photos/3532551/pexels-photo-3532551.jpeg?auto=compress&cs=tinysrgb&w=1600&lazy=load" alt=""></li>
            <li><img width="200px" id="owl" src="https://images.pexels.com/photos/3532552/pexels-photo-3532552.jpeg?auto=compress&cs=tinysrgb&w=1600&lazy=load" alt="" ></li>
            <li><img width="200px" id="prayer" src="https://images.pexels.com/photos/2522671/pexels-photo-2522671.jpeg?auto=compress&cs=tinysrgb&w=1600&lazy=load" alt=""></li>
            <li><a style="color: aliceblue;" href="https://google.com" id="google">Google</a></li>
        </ul>
    </div>
</body>
<script>
    //way-1
    document.getElementById('owl').onclick = function(){
        alert("Owl Clicked")
    }

    //Before this two were also used
    //attachEvent()
    //jQuery - on

    //way-2: better way
    document.getElementById('owl').addEventListener('click', function(){
        alert("Owl Clicked Again")
    }, false)
    //Note: here 3rd parameter is true/false. y default it is false

    document.getElementById('owl').addEventListener('click', function(e){
        console.log(e)
    }, false)
    //Note: These are some of the PointerEvents we get when we console.log(e). [Study these for interview and projects]
    //type, timestamp, defaultPrevented
    //target, toElement, srcElement, currentTarget
    //clientX, clientY, screenX, screenY
    //altkey, ctrlkey, shiftkey, keyCode


    /* Event Propogation: Bubbling ->  niche se upar jata hai */
```

```
document.getElementById('images').addEventListener('click', function(e){
    console.log("clicked inside the ul")
}, false)

document.getElementById('owl').addEventListener('click', function(e){
    console.log("clicked on owl")
}, false)
//here first "clicked on owl" then "clicked inside the ul" will print


/* Event Propogation: Capturing -> goes from top to bottom   */
document.getElementById('images').addEventListener('click', function(e){
    console.log("clicked inside the ul")
}, true)

document.getElementById('owl').addEventListener('click', function(e){
    console.log("clicked on owl")
}, true)
//here first "clicked inside the ul" then "clicked on owl" will print

//When we want to prevent Bubbling we use e.stopPropogation()
document.getElementById('images').addEventListener('click', function(e){
    console.log("clicked inside the ul")
}, false)

document.getElementById('owl').addEventListener('click', function(e){
    console.log("clicked on owl")
    e.stopPropagation()
}, false)

//to prevent Capturing
document.getElementById('images').addEventListener('click', function(e){
    console.log("clicked inside the ul")
    e.stopPropagation()
}, true)

document.getElementById('owl').addEventListener('click', function(e){
    console.log("clicked on owl")
}, true)


//to prevent default behaviour we use e.preventDefault()
```

```
    document.getElementById('google').addEventListener('click', function(e){
        e.preventDefault() //to prevent default behaviour
        e.stopPropagation()//to stop Bubbling
        console.log("clicked on google")
    }, false)


    /* Todo: to make images disapper when we click on image*/

    document.querySelector('#images').addEventListener('click', function(e){
        console.log(e.target.parentNode)
        let removeIt = e.target.parentNode
        removeIt.remove() //method-1
        //removeIt.parentNode.removeChild(removeIt) //method-2
    }, false)

    //Note: Problem with above code is when we click on 'li', ul gets slected
and it removes all li. --> this is called Event Spill-Over

    //Fix of Event Spill-Over issue
    document.querySelector('#images').addEventListener('click', function(e){
        console.log(e.target.tagName)
        if(e.target.tagName === 'IMG'){
            console.log(e.target.id)
            let removeIt = e.target.parentNode
            removeIt.remove()
        }
    }, false)

</script>
</html>
```

Lect-37: **Async Javascript Fundamentals**

→ Javascript

↳ Synchronous

↳ Single threaded

Default

→ Execution Context

↳ execute one line of code at a time

→ console log → 1

→ console log → 2

each operation waits for the last one to complete before executing

CALL Stack

Memory Heap

Blocking Code VS Non Blocking code

↳ Does not block execution

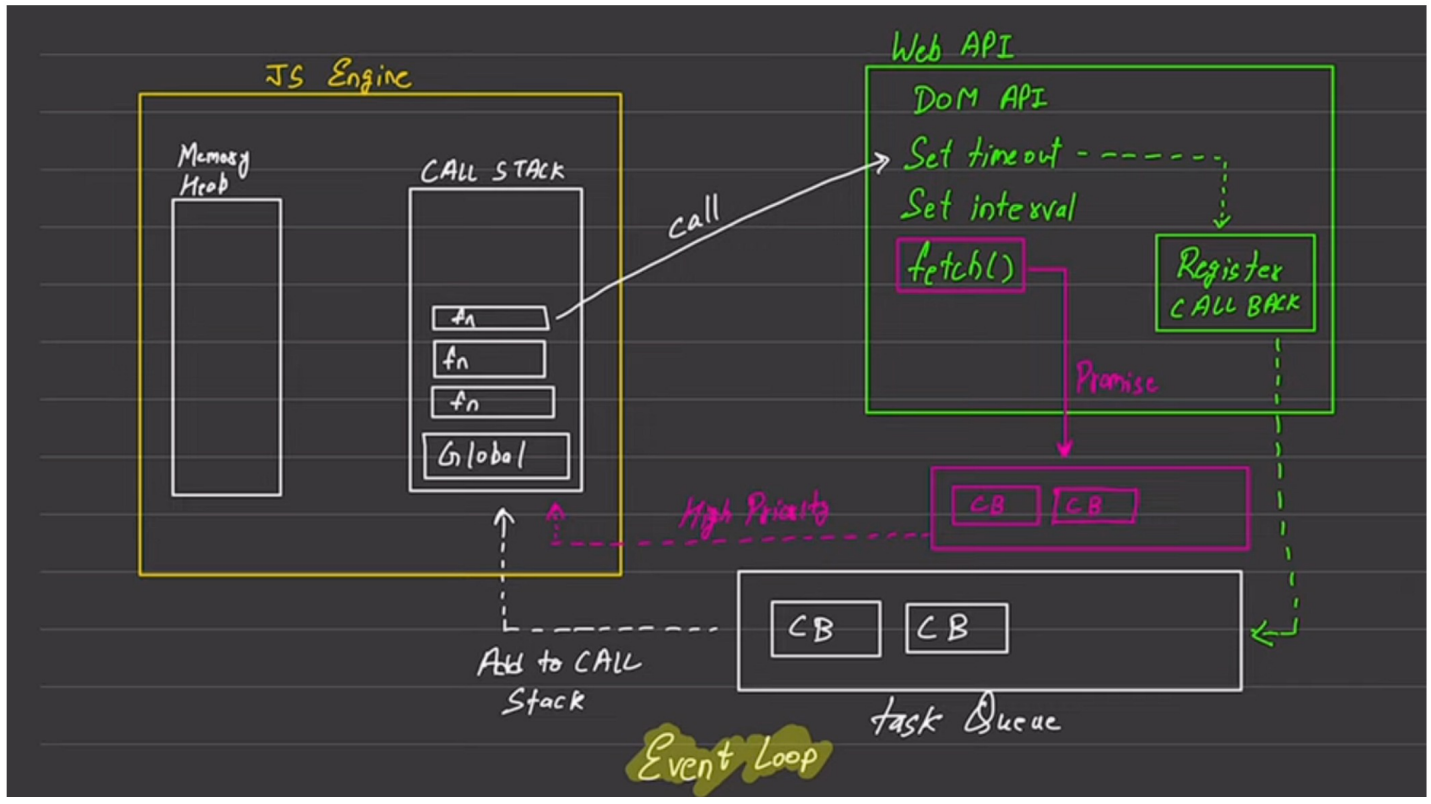
Block the flow of Program

↓

Read File Sync

↓

Read File Async



Lect-38: **2 projects with Async JS**

```
setTimeout(function(){
            console.log("Hitesh")
       }, 2000)
```

```
const sayHitesh = function(){
            console.log("Hitesh")
        }

        setTimeout(sayHitesh, 2000)
```

```
const changeHeading = function(){
            document.querySelector('h1').innerHTML = "Best JS Series"
        }

        setTimeout(changeHeading, 2000)
```

```
<!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>Chai aur Code</h1>
    <button id="stop">Stop</button>

</body>

    <script>
        /*****setTimeout()*******/
        const sayHitesh = function(){
            console.log("Hitesh")
        }

        const changeHeading = function(){
```

```
            document.querySelector('h1').innerHTML = "Best JS Series"
        }

        const changeMe = setTimeout(changeHeading, 2000)

        //clearTimeout() -> this need reference of setTimeout(), so we have to s
        document.querySelector('#stop').addEventListener('click', function(){
            clearTimeout(changeMe)
            console.log("STOPPED!")
        })
        //now if we press Stop button before 2 second then changeHeading will no


    </script>

</html>
```

```
/**setInterval()**/
        const sayHi = function(str){
            console.log(str, Date.now())
        }

        const intervalId = setInterval(sayHi, 1000, "Hi")
        //it has three parameters

        //clearInterval()
        clearInterval(intervalId)
```

```
<!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>Chai aur JavaScript</h1>
    <button id="start">Start</button>
    <button id="stop">Stop</button>
```

```
</body>

    <script>

        /******ToDo - Start printing when Start button is pressed and Stop when
        const printMessage = function(str){
            console.log(str, Date.now())
        }

        let intervalID
        document.querySelector('#start').addEventListener('click', function(){
            intervalID = setInterval(printMessage, 1000, "Hi")
        })

        document.querySelector('#stop').addEventListener('click', function(){
            clearInterval(intervalID)
            console.log("Stopped!")
        })

    </script>

</html>
```

Project: Unlimited Colors

```
//generate a random color
const randomColor = function(){
    const hex = "0123456789ABCDEF"
    let color = "#"
    for(let i = 0; i < 6; i++){
        let randNum = Math.floor(Math.random() * 16)
        color += hex[randNum]
    }
    return color
}

let intervalId;

const startChangingColor = function(){
    if(!intervalId)
        intervalId = setInterval(changeBgColor, 1000)
```

```
    function changeBgColor(){
        document.querySelector('body').style.backgroundColor = randomColor()
    }
}

const stopChangingColor = function(){
    clearInterval(intervalId)
    intervalId = null
}

document.querySelector('#start').addEventListener('click', startChangingColor)

document.querySelector('#stop').addEventListener('click', stopChangingColor)
```

Project: Keyboard Check

```
const insert = document.getElementById('insert')

window.addEventListener('keydown', (e) => {
    insert.innerHTML = `
        <div class='color'>
            <table>
                <tr>
                    <th>Key</th>
                    <th>KeyCode</th>
                    <th>Code</th>
                </tr>
                <tr>
                    <td>${e.key === ' ' ? 'Space' : e.key}</td>
                    <td>${e.keyCode}</td>
                    <td>${e.code}</td>
                </tr>
            </table>
        </div>
    `
})
```

Lect-39: **API request and V8 engine**

API:

- RANDOM USER GENERATOR: https://randomuser.me/
- JSON Formatter: https://jsonformatter.org/
  use this to read data from API

—> API Request to get follower count from GitHub

```
<script>
const requestUrl = 'https://api.github.com/users/hiteshchoudhary'
const xhr = new XMLHttpRequest();
xhr.open('GET', requestUrl)
xhr.onreadystatechange = function(){
console.log(xhr.readyState);
if (xhr.readyState === 4) {
const data = JSON.parse(this.responseText)
console.log(typeof data);
console.log(data.followers);
}
}
xhr.send();
</script>
```

Lect-40: Promise in Javascript

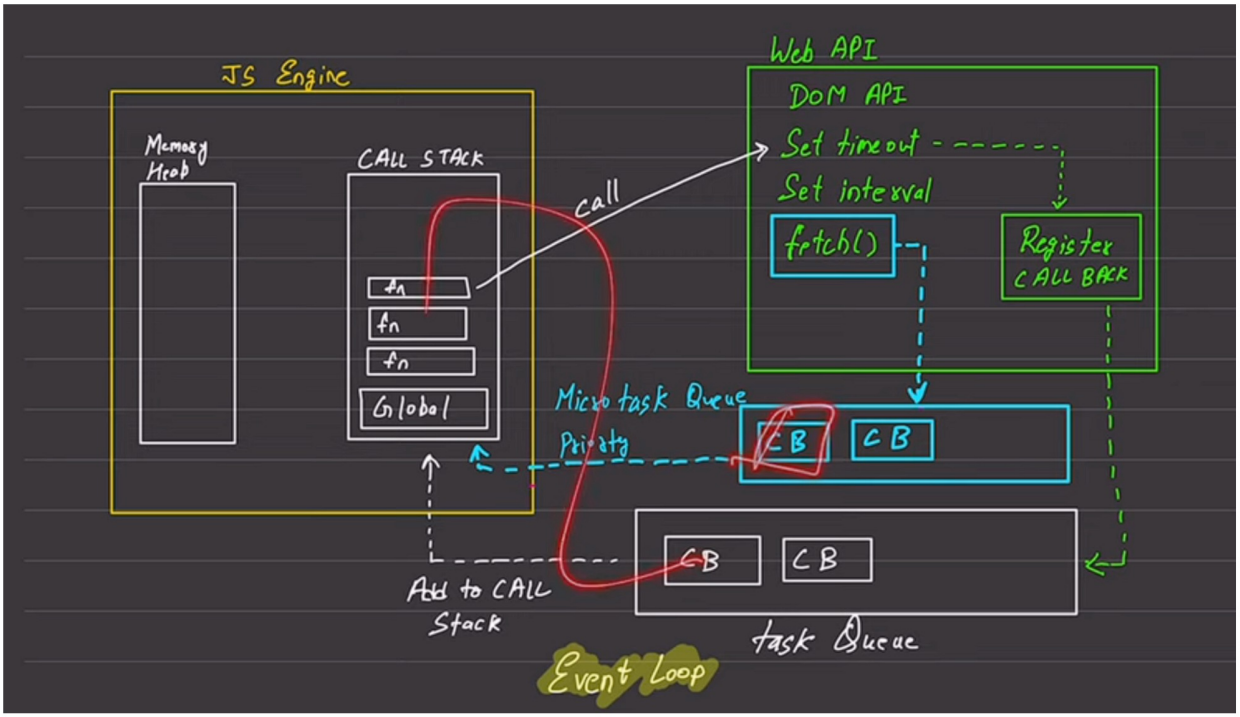
```
/*Promise: The Promise object represents the eventual completion (or failure) of

//creating Promise
const promiseOne = new Promise(function(resolve, reject){
    //Do an async task
    //DB calls, Cryptography, network

    setTimeout(function(){
        console.log('Async task is complete')
        resolve() //connecting resolve with then()
    },1000)
})

//promise consumption
//Note: resolve is connected/associated with then()
promiseOne.then(function(){
    console.log("Promise Consumed")
})
```

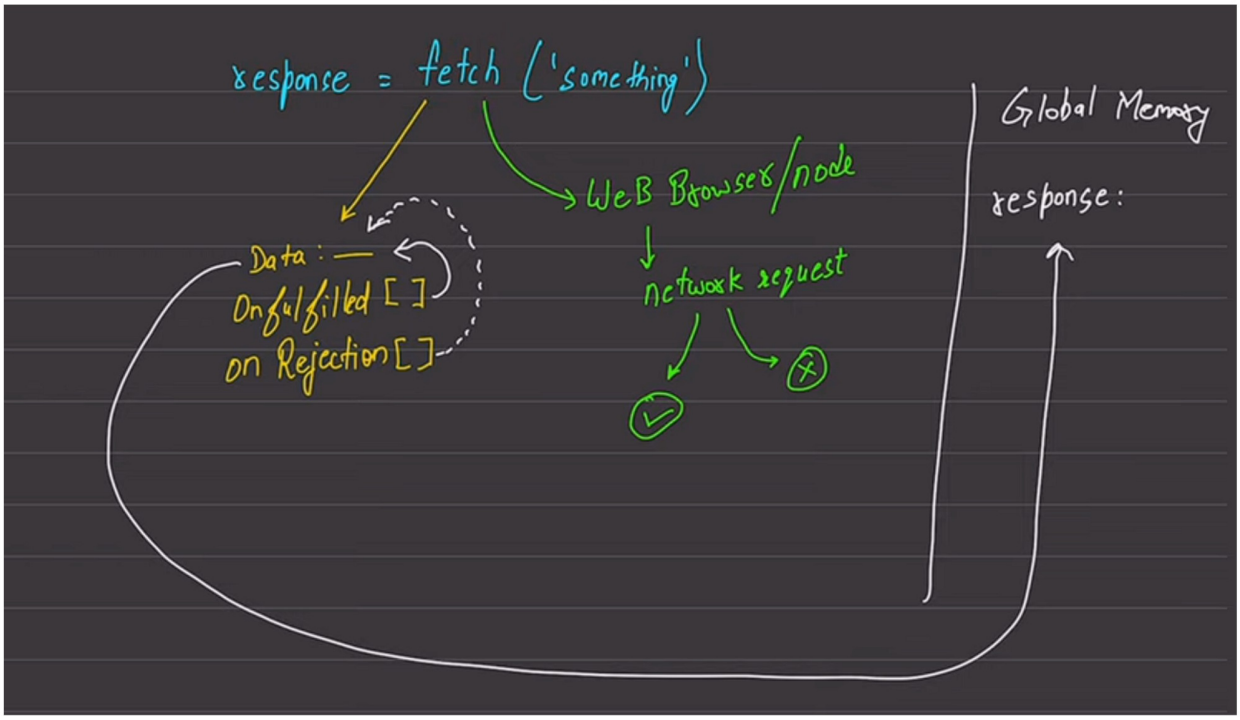
```
//creating and consuming promise together
new Promise(function(resolve, reject){
    setTimeout(function(){
        console.log("Async Task 2 is complete")
        resolve()
    }, 1000)
}).then(function(){
    console.log("Async 2 resolved");
})


//how data is passed?
//whatever argument we pass in resolve() during promise creation, it will be use
const promiseThree = new Promise(function(resolve, reject){
    setTimeout(function(){
        resolve({username: "Chai", email: "chai@example.com"})
    }, 1000)
})

promiseThree.then(function(user){
    console.log(user);
    //O/P: { username: 'Chai', email: 'chai@example.com' }
})


//use of reject(), finally() and chaining concept
const promiseFour = new Promise(function(resolve, reject){
    setTimeout(function(){
        let error = false
        //let error = true
        if(!error){
            resolve({username: "hitesh", password: "123"})
        }
        else{
            reject("ERROR: Something went wrong")
        }
    }, 1000)
})

promiseFour.then((user) => {
```

```
    console.log(user)
    return user.username //value returned from here will be used as argument in
}).then((username) => {
    console.log(username)
}).catch(function(error){
    console.log(error)
}).finally( () => console.log("The promise is either resolved or rejected"))

//reject() is connected with catch()
//catch() is executed when there is an error and resolve failed to execute
//finally(): to tell if the task is done either resolved or rejected. Ye toh hog


//using async, await
const promiseFive = new Promise(function(resolve, reject){
    setTimeout(function(){
        let error = true
        if(!error){
            resolve({username: "javascript", password: "js123"})
        }
        else{
            reject("ERROR: JS went wrong")
        }
    }, 1000)
})

async function consumePromiseFive(){
    try {
        const response = await promiseFive
        console.log(response);
    } catch (error) {
        console.log(error);
    }
}

consumePromiseFive()


//fetch()
async function getAllUsers(){
    try {
        const response = await fetch('https://jsonplaceholder.typicode.com/users
```

```
        const data = await response.json() //json conversion also takes time so
        console.log(data);
    } catch (error) {
        console.log("ERROR:", error);
    }
}

getAllUsers()

//doing same thing as above using then(), catch()
fetch('https://jsonplaceholder.typicode.com/users')
.then((response) => {
    return response.json()
})
.then((data) => {
    console.log(data);
})
.catch((error) => {
    console.log(error);
})
```

Lect-41: **Now you know fetch in javascript**





Lect-42: **Object Oriented in Javascript**

```
# javascript and classes

## OOP

## Object
- collection of properties and methods
- toLowerCase

## why use OOP
```

```
## parts of OOP
Object literal

- Constructor function
- Prototypes
- Classes
- Instances (new, this)


## 4 pillars
Abstraction
Encapsulation
Inheritance
Polymorphism
```

```
//object literal
const user = {
    username: "hitesh",
    loginCount: 8,
    signedIn: true,

    getUserDetails: function(){
        console.log("Got user details from database");

        //this: this refers to current context
        console.log(this); //this will give current object
        console.log(`Username: ${this.username}`);
    }
}

console.log(user.username);
console.log(user.getUserDetails());


//Constructor function
function User(username, loginCount, isLoggedIn){
    this.username = username;
    this.loginCount = loginCount;
    this.isLoggedIn = isLoggedIn;
```

```
    this.greeting = function(){
        console.log(`Welcome ${this.username}`);
    }

    return this; //by default also it returns this only
}

const userOne = new User("hitesh", 12, true)
const userTwo = new User("ChaiAurCode", 100, false)
console.log(userOne);
console.log(userTwo);

console.log(userOne.constructor) //[Function: User]

console.log(userOne instanceof User); //true


/*new keyword:
Step-1: when new is used an empty object is created, which is called an instance
Step-2: a constructor function is called
Step-3: this is injected
Step-4:you get whatever is in the function
*/
```

Lect-43: **Magic of Prototype in javascript**

```
function multipleBy5(num){
    return num * 5;
}

multipleBy5.power = 2

console.log(multipleBy5(5)); //25
console.log(multipleBy5.power); //2
console.log(multipleBy5.prototype); //{}

//Everything in javascript in an object at the end.

//function is a function. But it is also an object in javascript. Anything in ja
```

```
//injecting methods in our own created function
function createUser(username, score){
    this.username = username
    this.score = score
}

createUser.prototype.increment = function(){
    this.score++
}

createUser.prototype.printMe = function(){
    console.log(`Price is ${this.score}`);
}

const chai = new createUser("Chai", 25)
const tea = new createUser("Tea", 250)

chai.printMe()
tea.printMe()

/*

Here is what happens behind the scenes when the new keyword is used:

A new object is created: The new keyword initiates the creation of a new Javascr

A prototype is linked: The newly created object gets linked to the prototype pro

Th constructor is called: The constructor function is called with the specified

The new object is returned: After the constructor function has been called, if i

*/
```

Prototype:

```
let myName = "hitesh    "
let myChannel = "Chai      "

//we want to get true length of the word, we can do it by below method
console.log(myName.trim().length);
```

```
//[BUT we want to create a function that can perform this task]
//console.log(myName.trueLength())



let myHeros = ["thor", "spiderman"]


let heroPower = {
    thor: "hammer",
    spiderman: "sling",

    getSpiderPower: function(){
        console.log(`Spidy power is ${this.spiderman}`);
    }
}

Object.prototype.hitesh = function(){
    console.log(`hitesh is present in all objects`);
}

Array.prototype.heyHitesh = function(){
    console.log(`Hitesh says hello`);
}

// heroPower.hitesh()
// myHeros.hitesh()
// myHeros.heyHitesh()
// heroPower.heyHitesh()

// inheritance
//proto-typical inhertance: how one object can access properties of some other o

const User = {
    name: "chai",
    email: "chai@google.com"
}

const Teacher = {
    makeVideo: true
}
```

```
const TeachingSupport = {
    isAvailable: false
}

const TASupport = {
    makeAssignment: 'JS assignment',
    fullTime: true,
    __proto__: TeachingSupport
}

Teacher.__proto__ = User //Teacher can access properties of User also

//But above used syntax is outdated

// modern syntax
Object.setPrototypeOf(TeachingSupport, Teacher)
//now TeachingSupport can access all properties of Teacher also


//Todo that we wanted to do at starting of the code[Get true length of string]
let anotherUsername = "ChaiAurCode     "

String.prototype.trueLength = function(){
    console.log(`${this}`);
    console.log(`True length is: ${this.trim().length}`);
}

anotherUsername.trueLength()
"hitesh".trueLength()
"iceTea".trueLength()
```

Lect-44: Call and this in javascript

```
function SetUsername(username){
    //comples DB calls(suppose)
    this.username = username
    console.log("called");
}

function createUser(username, email, password){
```

```
    //SetUsername(username) //here SetUsername function is called but as soon as
    //Therefore we are not able to access variables inside it

    //Therefore to make it work properly we have to hold and keep its refernce,

    //SetUsername.call(username)//only call() also not sufficient, we have to pa
    SetUsername.call(this, username)

    this.email = email
    this.password = password
}

const chai = new createUser("chai", "chai@fb.com", "123@1")
console.log(chai);
```

Lect-45: **Class constructor and static**

```
//ES6

//inside a class a fucntion is called method

class User{
    constructor(username, email, password){
        this.username = username
        this.email = email
        this.password = password
    }

    encryptPassword(){
        return `${this.password}abc#`
    }

    changeUsername(){
        return `${this.username.toUpperCase()}`
    }
}

const chai = new User("Chai", "chai@gmail.com", "1234@1")

console.log(chai.encryptPassword());
```

```
console.log(chai.changeUsername());


//behind the scene
//if class is not available this is how codes were before

function User1(username, email, password){
    this.username = username
    this.email = email
    this.password = password
}

Object.prototype.encryptPassword = function(){
    return `${this.password}abc#`
}

Object.prototype.changeUsername = function(){
    return `${this.username.toUpperCase()}`
}

const tea = new User1("Tea", "tea@gmail.com", "123tea@")

console.log(tea.encryptPassword());
console.log(tea.changeUsername());
```

```
//inheritance

class User{
    constructor(username){
        this.username = username
    }
    logMe(){
        console.log(`USER NAME is ${this.username}`);
    }
}

class Teacher extends User{
    constructor(username, email, password){
        super(username)
        this.email = email
        this.password = password
```

```
    }

    addCourse(){
        console.log(`A new course is added by ${this.username}`);
    }
}

const chai = new Teacher("Chai", "chai@gmai.com", "123")

chai.addCourse()
chai.logMe()

const masalaChai = new User("masalaChai")

masalaChai.logMe()
//masalaChai.addCourse() //no access

console.log(chai instanceof Teacher); //true
console.log(chai instanceof User); //true
console.log(masalaChai instanceof User); //true
console.log(masalaChai instanceof Teacher); //false
```

```
//static properties

//static: to prevent access to a method we use static

class User{
    constructor(username){
        this.username = username
    }

    logMe(){
        console.log(`Username: ${this.username}`);
    }

    static createId(){
        return `123`
    }
}

const hitesh = new User("hitesh")
```

```
hitesh.logMe()///logMe() is accessible

//console.log(hitesh.createId()); //createId() method cant be accessed now. Beca

class Teacher extends User{
    constructor(username, email){
        super(username)
        this.email = email
    }
}

const iphone = new Teacher("iphone", "i@phone.com")

iphone.logMe() //logMe() is accessible

//console.log(iphone.createId()); //createId() method cant be accessed now. Beca

//Note: static is used to prevent inheritance
```

Lect-46: **Bind in javascript**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>React</title>
</head>
<body>
    <button>Button Clicked</button>
</body>
<script>
    class React {
        constructor(){
            this.library = "React"
            this.server = "https://localhost:300"

            //requirement
            document
                .querySelector('button')
```

```
                .addEventListener('click', this.handleClick.bind(this))

        }
        handleClick(){
            console.log("button clicked");
            console.log(this.server);
        }
    }

    const app = new React()
</script>
</html>
```

Lect-47: **Now you know Objects in Javascript**

```
//Can we change the value of PI? If yes, how? If no, why?

//Ans: We cant change the value of PI

console.log(Math.PI); //3.141592653589793
Math.PI = 5
console.log(Math.PI); //3.141592653589793

//Why? beacuse
const descripter = Object.getOwnPropertyDescriptor(Math, "PI")

console.log(descripter);
/*
{
  value: 3.141592653589793,
  writable: false,
  enumerable: false,
  configurable: false
}
*/
//beacuse writable is set to false and we cant change this writable property. So



const chai = {
    name: "ginger chai",
```

```
    price: 250,
    isAvailable: true,

    orderChai: function(){
        console.log("chai nahi bani");
    }
}

console.log(Object.getOwnPropertyDescriptor(chai, "name"));
/*
{
  value: 'ginger chai',
  writable: true,
  enumerable: true,
  configurable: true
}
*/

//here we can enumerate chai object. Beacuse enumerable is true
for(let [key, value] of Object.entries(chai)){
    if(typeof value !== 'function'){
        console.log(`${key} : ${value}`);
    }
}

//here we disable 'name' property
Object.defineProperty(chai, 'name', {
    enumerable: false
})

console.log(Object.getOwnPropertyDescriptor(chai, "name"));
/*
{
  value: 'ginger chai',
  writable: true,
  enumerable: false,
  configurable: true
}
*/

//here we cant enumerate 'name' property of chai object. Beacuse enumerable is s
for(let [key, value] of Object.entries(chai)){
```

```
    if(typeof value !== 'function'){
        console.log(`${key} : ${value}`);
    }
}
```

Lect-48: **Getter Setter and Stack Overflow**

```
//getter and setter
//name of the method is same as property for both get and set
//both get and set comes together

//Note: this._password is used inside get and set. If we use this.password it wi

//class based syntax--> mostly this method is used 95% time
class User{
    constructor(email, password){
        this.email = email
        this.password = password
    }

    get email(){
        return this._email.toUpperCase()
    }

    set email(value){
        this._email = value
    }

    get password(){
        return `${this._password}@123`
    }

    set password(value){
        this._password = value
    }
}

const chai = new User("chai@ai.com", "abc")
console.log(chai.email)
console.log(chai.password)
```

```
//Proposal: ES2022 - use # to make private
```

```
//funcstion based syntax for getter and setter
//old way of doing

function User(email, password){
    this._email = email;
    this._password = password

    Object.defineProperty(this, 'email', {
        get: function(){
            return this._email.toUpperCase()
        },
        set: function(value){
            this._email = value
        }
    })
    Object.defineProperty(this, 'password', {
        get: function(){
            return this._password.toUpperCase()
        },
        set: function(value){
            this._password = value
        }
    })

}

const chai = new User("chai@chai.com", "chai")

console.log(chai.email);
```

```
//object based syntax
//old way
const User = {
    _email: 'h@hc.com',
    _password: "abc",
```

```
    get email(){
        return this._email.toUpperCase()
    },

    set email(value){
        this._email = value
    }
}

const tea = Object.create(User)
console.log(tea.email);
```

Lect-49: **Lexical scoping and Closure**

```
<!--
A closure is the combination of a function bundled together (enclosed) with re
ferences to its surrounding state (the lexical environment). In other words, a
closure gives you access to an outer function's scope from an inner function.
In JavaScript, closures are created every time a function is created, at funct
ion creation time.
-->


<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Closure</title>
</head>
<body style="background-color: #313131;">
    <button id="orange">Orange</button>
    <button id="green">Green</button>


</body>
<script>
    //Lexical scoping: username of outer function can be accessed by inner fun
ction
    function outer(){
        let username = "chai"
```

```
        function inner(){
            console.log("Inner:",username);
        }
        inner()
    }
    outer()
    //console.log("TOO OUTER:", username); //here username cant be accessed. O
ut of scope


    //CLOSURE concept:
    function makeFunc(){
        const name = "Mozilla"
        function displayName(){
            console.log(name)
        }
        return displayName;
    }

    const myFunc = makeFunc()
    myFunc()

    /*
    - jab displayName return krenge toh sirf displayName ka reference nh jyeg
a, uska outer function bhi if exist krta h, uska bhi pura scope jayega. Becaus
e of Lexical Scoping.
    - Sirf execution context nahi jaata hai , pura ka pura lexical scope jata
hai
    - Closure ka matlab hai jab aap pura ka pura function hi return kr dete ho
toh sirf function return nh hota h, Lexical scope return hota hai
    */
</script>

<!--Note : you can write as many script tag as you want-->
<script>
    // document.getElementById('orange').onclick = function(){
    //     document.body.style.backgroundColor = 'orange'
    // }
    // document.getElementById('green').onclick = function(){
    //     document.body.style.backgroundColor = 'green'
    // }
    //what if you have 500 buttons? Writing again and agian is not good idea.
```

```
We are breaking DRY(Dont Repeat Yourself) principle


    /*Closure concept in practical*/

    //this wont work as desired. So we have to use Closure here
    // function clickHandler(color){
    //     document.body.style.backgroundColor = `${color}`
    // }

    // document.getElementById('orange').onclick = clickHandler("orange")

    //Using Closure
    function clickHandler(color){
        return function(){
            document.body.style.backgroundColor = `${color}`
        }
    }

    document.getElementById('orange').onclick = clickHandler('orange')
    document.getElementById('green').onclick = clickHandler('green')

</script>

</html>
```

Lect-50: **Javascript ends with a story**

- Story of **Michael Phelps: prepare for unforseen circumstances.**
- Preparation is the key.
- You can’t be taught everything you learn by doing things on your own.