Java script

1: Word vs Keyword

anything which don't have any meaning in js is called word or those words which having some meaning in js is called key word i.e

words: beautiful, brush,

keyword: for, let, if

2: variables(var) & constant(const) & let

if we want to store any type of data in our code then the variable comes in action.

variables and constants both are English words used for storing data but the main diff is...

variables mean which change their value

constant means which do not change their value.

3: defined & un-defined & not defined

any variable which are declared or also initialized is called defined

any variable which are declared but not initialized is called un-defined

any variable which are not declared or not even initialized is called not defined.

example:

```
var a;    print(a) // un defined
var b = 12;        print(b) // defined
print(c)  // not defined
```

3: hoisting

any function or variable which is used before its declaration is called hoisting

when we declared any variable or function after their calling is moved on the top of the code which is called hoisting

4: types in JS

primitive & reference

primitive types wo hoti hain jo haqeeqi data ki copy banati hain, jab ham inhy kisi dusre variable ma assign karty hain to in ma data ki real copy hoti hai yani agar ham asal value ko modify karain to copy data par koi asar nahi hoga.

i.e

number, string, null, undefined, boolean

reference types wo hoty hain Jahan variables ma asal data k bajaye is ka reference ya address store kiya jata hai, jab ham kisi reference type ki value ko dusre variable ko assign karty hain to dono variables aik hi memory address ko point karty hain, is ka matlab ye hai k agar ham aik varaibe ki value ko modify karain to dono variables ki value change ho jaye gi q k wo asal object k sath munsalik hoty hain.

(any values which are inside the brackets "(), [], {}" is called reference types)

i.e

arrays, object, class

5: conditionals "if, else, else-if"

6**: loops (for, while)**

7: functions

its means that we provide some name to the specific code

it is mainly used for three main purposes

jab ma apna code foran na chalana chahta hun.

jab mujhy apne code ko reuse karna ho

jab mujhy apna code reuse karna ho har baar with diff data

Parameters & arguments

arguments are real values jo ham function call karty wakt pass karty hain function ma.

parameter are those variables jin ma values store hoti hain arguments waali. Jo values function catch karta hai

8: Arrays

ham aik variable ma aik value store kar sakte hain par jab hame aik se ziyada value aik hi variable ma store karni ho to ham array use karty hain (array = group of values)

i.e

var a = 12,13; // wrong

var a = [1,2,3,4] //correct

9: push pop shift unshift splice:

push will add the value after the last index in array

pop will delete the last index value from an array

unshift will add the value at first index in array

shift will delete the first index value from an array

splice will remove value from middle of an array it will take two values first is index where the value is placed second how many values we want to remove if we provide 1 it will delete only one value if we provide 2 i will delete two values and so on

10: Objects

for example we discus about persons

let suppose if we discuss about multiple persons then we use arrays

if we want to discuss about multiple things related to one person then we use objects, means objects will save details of one individual person

blank object:

```
var person = {};
```

filled object:

```
var person = {
    age: 24,
    name: "syed",
    email: "syed@gmail.com",
    contact: 03210000111
        }
```

how to call properties of objects

```
print(person.email) // syed@gmail.com
```

Methods in object:

jab ham object ki kisi b property ko function assign kar dain to wo method ban jata hai. i.e

```
var person = {
    age: 24,
    name: "syed",
    email: "syed@gmail.com",
    contact: 03210000111,
    class: function(){}
        }
```

Delete from objects

If we want to delete complete property from object not only value so we use

Delete person.email;

Advanced JS

var let const (diff)

i: JS having two versions es5, es6. in version es5 there are only var datatype but in es6 there are newly introduced data types let and const for declaration variables. It means es6 introduced let and const as new ways to declare variables, alongside it already existing var.

ii) var is function scoped means var is used any where inside there parent function.

let is braces(brackets {}) scoped means it is only used inside there braces.

i.e,

```
function abcd(){
    for(var i=1; i<12; i++)
    {
        print(i);  // 1,2,3,.....,11
    }
    print(i); // 12
}
// 1,2,3,.....,12
```

```
function abcd(){
    for(let i=1; i<12; i++)
    {
        print(i);
    }
```

```
        print(i);  // i is not defined
    }
    // 1,2,3,.....,11, i is not defined
```

so both code having diff results due to var and let keyword.

iii) var adds itself to the window object.

let, const doesn't add itself to the window object.

now what is window:

JS ma kafi features available hain par kafi features aise hain jo JS ma ni hain laikin wo features pir b JS use kar laity hai with the help of window, window aik box of feature hai jo browser daita hai JS ko.

i.e,

Imagine you have a pen, a piece of paper, and a box (which we'll call the window). The pen represents JavaScript.

i) Basic Task (Pen alone): You can ask the pen to make a hole in the paper, and the pen will do it. You can also ask the pen to write something on the paper, and the pen will handle that as well. These are basic tasks that the pen (JavaScript) can perform directly.

ii) Complex Task (Pen needs help): Now, suppose you ask the pen to make a mobile phone holder. The pen can't do this on its own, because it's not designed for that specific task. So, the pen will go to the box (the window object) and ask it for a mobile phone holder. The box (window) provides the holder, and the pen then gives it to you. The pen still helps you, but it relies on the box for the task.

some window functions which is used with the help of window:

alert(),

prompt(),

print(),

console(),

clearInterval(),

clearTimeout(),

document(), etc many more.

Stack:

Heap Memory:

Jitne b variables ya data hum banante hain unhe store karne k liye use hota hai heap memory.

Execution Context, Lexical Environment:

it is a container where the function code is executed and it is always created whenever a function is called

jab b ham function chalayn gy to fun apna aik khud ka imaginary container bana le ga jis ma uski 3 cheezain hon gi..

i) variables

ii) fun inside that parent fun

iii) lexical environment of that fun

ye jo imaginary container hai ise ham execution context kahty hain

what is lexical environment:

ye hame batata hai k ye fun apny andar bani kon si cheezon ko access kar sakta hai or kin ko nahi.

i.e:

```
function abcd(){
    var a = 12;
    function def(){
        var b = 21;
    }
}
```

now in lexical environment tell us that abcd() fun only access "a" variable and "def()" function but cannot access "b" variable which is inside the child func "def()", this is the work of lexical environment.

lexical environment aik chart hota hai jis ma ye likha hota hai k ap ka particular function kin cheezon ko access kar sakta hai or kin ko nahi kar sakta, matlab k it holds its scope and scope chain

i.e

We have three functions (or possibly more) that are nested inside each other, like this:

```
fun first(){
    var a = 1;
    fun two(){
        var b = 2;
        fun three(){
            var c = 3;
            }
        }
    }
```

In JavaScript, due to lexical scoping, each child function can access all the variables and properties of its parent functions. For example, function two can access the variable "a" from function first, and function three can access both "a" from first and "b" from two.

However, the parent function cannot access any variables or properties from its child functions. So, function first cannot access "b" or "c", and function two cannot access "c".

This behavior is due to the lexical environment in JavaScript, which defines the scope of variables based on where the functions are written in the code.

How to copy reference values

to copy reference values, we use spread operators denoted by three dots (...),

i.e:

```
var arr1 = [1,2,3,4,5];

var arr2 = [...arr1];

arr2.pop();

print(arr1) // 12345

print(arr2) // 1234
```

due to copy with spread operator now if we make changes on second array it will not effect on the first array.

same working for objects

```
var obj = {
    name:"syed",
    age: 25
    }


var copyObj = {...obj}
```

truthy or falsy

JS main kuch b likh lain wo mainly 2 possibilities ma se kisi aik ma belong karti hai truthy ya pir falsy

means ham jo b likhty hain wo convert ho jati hai truth ya false ma.

falsy values:

0, false, undefined, null, NaN, document.all

truthy values:

all remaining values accept falsy values

1, 1.3, "syed", true, -2, many more

For each, For in loops

ForEach loop sirf array k liye use hota hai, matlab agar ham array k har element par kuch karna chah rahy hon to ham foreach loop use karty hain.

foreach loop kabi b original array ma change ni karta means ham foreach laga kar kisi b array ma koi b kaam karty hain to original array wohi same rahti hai, means jab b foreach loop lagta hai to wo array ki aik temporary copy bana laita hai jis ma changes ho rahi hoti hain.

foreach aik function laita hai hamesha jis k parameter ma array ki temporary copy ati hai or wo usy har aik elment k liye chalata hai

i.e

```
var arr = [1,2,3,4,5]
arr.forEach(function(val){
    print(val+2);
})
```

For in loop sirf use hota hai objects k liye.

```
Var obj = {
    Name: “syed”,
    Age: 24,
    City: “Peshawar”
}
For(var key in obj)
{
    Print(key);
}
```

This code of for in loop will print only the keys of object like name, age, city but we need the values of those keys as well so we achieve it like this

```
For(var key in obj)
{
        Print(obj[key]);
}
```

This will gives us the values like syed, 24, Peshawar.

This is the syntax of for in loop first we write for then we make a variable which is iterate in object using “in” keyword.

Callback functions:

Aisa code jo kuch dair baad chalta ho hum usy aik function de dait hain k jab ye fun complete ho jaye to chala daina or wo fun jo ham daity hain wo normal func hota ai or usy ham callback fun kahty hain.

Callback function wo function hota hai jo doosre function ke kaam khatam hone ke baad call hota hai. Yeh approach tab use ki jati hai jab aapko koi kaam delay ke baad ya kisi event ke hone par chalana ho.

```
function firstTask(hello) {
    console.log("Pehla kaam shuru...");

    setTimeout(function() {
        console.log("Pehla kaam mukammal ho gaya.");
        hello(); // Ye callback function call ho raha hai
    }, 2000); // 2 second delay
}

function secondTask() {
    console.log("Dusra kaam ab shuru ho raha hai.");
}

console.log(firstTask(secondTask));
```

First Class functions

JS ma aik concept hai jiska matlab hota hai k ap fun ko use kar sakty hain as a value

```
// Function ko variable mein store karna
let greet = function() {
    console.log("Hello!");
};

// Function ko call karna
greet(); // Output: Hello!

// Function ko as an argument pass karna
function executeFunction(fn) {
    fn(); // Yeh function ko call karega
}

executeFunction(greet); // Output: Hello!

// Function ko dusre function se return karna
function outerFunction() {
    return function() {
        console.log("This is a returned function!");
    };
}

let innerFunctionVar = outerFunction();
innerFunctionVar(); // Output: This is a returned function!
```

Is concept ka matlab ye hai ke functions ko JavaScript mein kisi bhi value ke tarah treat kiya ja sakta hai. Functions ko variables mein store kiya ja sakta hai, arguments ke tor par pass kiya ja sakta hai, aur doosre functions se return bhi kiya ja sakta hai. Is tarah ke behavior ko hi first-class citizens kahtay hain.

Higher Order Fun:

These are the functions which accept a function in a parameter or return a function or both.

i.e:

accept function in a parameter

```
function abc(val){


}
Abc(function(){});
```

Returning a function.

```
Function abc(){
    Return function(){}
}
Abc();
```

➔ Foreach function is also called higher order functions which takes function inside parameter.

Constructor function:

When you need to create many objects that share similar properties, you can use a constructor function.

If a function has properties defined with the “this” keyword and you call that function using the “new” keyword, it is called a constructor function.

i.e:

There is a company that manufactures remote controls. They produce a large number of remotes, and while some features are the same for all remotes, others are different. For example, the buttons are the same, but the height varies.

We need a constructor function like this:

```
function remotes(height,width){
    this.height = height;
    this.width = width;
```

```
        this.color = “black”;
        this.btn_radius = “circle”;
    }

Var remote1 = remotes(12,9);
Var remote2 = remotes(20,11);
```

So in this constructor function we provide two predefined values color and radius, but height and width is user defined when we call the function so its means that whenever the function is call every time color and radius will be same but height and width is different.

First Class functions:

A programming language is said to have first class functions when the functions in that programming language are treated as a normal values, you can save them in variables, you can pass them as an arguments to another functions.

i.e:

```
var abcd = function () {}

function abcd(fun){
    body
}
Abcd(function(){});

setTimeout(function(){});
```

New keyword:

Jab bhi new keyword lagta hai hamaisha aik blank object create hota hai.

Means jab b ham new keyword use karty hain to hamare pas aik blank object banta hai { }, or new k baad jo b likha hota hai wo chalta hai, or usually 99.9% new k baad function call karty hain matlab function ma ja kar jahaan jahaan this keyword k sath values lagi hui hain un ko us object ma store kar do or this keyword jis function ma use hota hai usy ham constructor fun kahty hain.

if there are values or variables inside the function that do not use the “this” keyword, they will not be part of the object created by the constructor. Only the properties assigned to “this” will be added to the object.

i.e:

When we use the new keyword, a blank object { } is created.

After that, the function is called, like ”new Abc()”,

which means the code inside the function is executed.

```
Abc(){
        This.name = “syed”;
        This.age = 24;
        Var a = “hello”
        Console.log(“world”);
}
```

When new “Abc()” is called, the blank object will store only two properties: "name" and "age", because they use the “this” keyword. The variable “a” and the “console.log” statement will not be stored in the object.

Iife function:

Immediately invoked function expression.

It is a function in JavaScript that runs as soon as it is defined. It allows you to execute a function immediately after it is created, without having to call it explicitly later.

Syntax of iife fun:

```
(function() {
        var message = "Hello from IIFE!";
        console.log(message);
        }) ();
```

Prototype:

For example, if we create an object with a property name or anything in it

Var obj = {

Name: “syed”

}

and if we run it through dot operator like “obj.name” so when we write “obj.” then in suggestion’s one more property are added by default which is [[prototype]] but we didn’t create it.

So if we didn’t created these properties where they do come from that’s where the concept of prototype comes in, every created object gets a property called prototype, which means when whenever we create object it gets prototype property automatically.

Javascript by default adds a property called [[prototype]] to every object which means if we ever create an object we blindly say that it contains prototype as well.

i.e when we create an array, and if we want to find length of array so we add “.length()” property so we find the length of array so this “.length()” property is just like prototype of object.

Prototypal Inheritance:

Inheritance is basically transferring the properties of parent to child.

So, if we make two objects and we want to access properties of first object inside second object so we can use prototypal inheritance through which first object will be inherit to second object.

i.e:

```
var humanObj = {
        isWalk = true,
        isTalk = true,
        isFly = false,
        willDie = true
}

var uniStudent = {
        canStudy = true,
        givePaper = true
```

}

uniStudent.__proto__ = human;

this is the way to access parent properties inside child using double underscore "__proto__".

Sync & async

Synchronous code refers to code execution where each line runs one after the other in sequence. This means that one line of code must finish executing before the next line can start. The issue with synchronous code is that if a particular line takes a long time to execute (like waiting for a file to load or fetching data from a server), the entire program will wait for that line to complete before moving on to the next line.

Asynchronous code allows multiple lines of code to start running at the same time without waiting for the previous one to complete. When a line of code finishes its task, its result is displayed, even if other lines are still running.

For example, if you have 5 lines of code:

- The first line takes 6 seconds to complete.
- The second line takes 4 seconds.
- The third line takes 2 seconds.
- The fourth line takes 5 seconds.
- The fifth line takes 1 second.

All lines start at the same time, and the results will show as each line completes. So, the result of the fifth line will appear first, then the third, second, fourth, and finally the first.

Types of asynchronous code:

  - setTimeout()
  - setInterval()
  - promises()
  - fetch()
  - XMLHttpRequest
- All code except these ones is synchronous code.

Async JS:

Kai baar apka final code depended hota hai kisi or k server par, is case ma hame ni pata hota k answer uske server se kab laut kar aye ga, to ham is wajah se sync code ni likh sakty, is cheez se bachny k liye ham async code likhty hain taky blocking na ho or jab b us ka answer aye to us answer k respect ma chalne wala code chal jaye.

Case scenario:

When we want to display a picture from Facebook, we send a request to the Facebook server. The picture will only appear when the server responds with the data. The key point here is that we don't know exactly how long the server will take to respond—it could be a few milliseconds or a few seconds. In this case, we use asynchronous code because it allows the rest of our application to keep running while waiting for the server's response. Once the response comes back, the picture will be displayed without blocking other parts of the program.

```
Console.log("I want to see a pic from facebook");
Console.log("send request to server");
Console,log("show pic"):
Console.log("the end");
```

we don't know how much time is required for second step but the third and fourth line will be executed even the request will not come so this is a wrong case.

So we write asynchronous code.

```
Console.log("I want to see a pic from facebook");
Console.log("send request to server");
setTimeout(function(){
        Console,log("show pic"):
},2000);
Console.log("the end");
```

JS is not Async:

Main stack (call stack)

Side stack (web APIs)

Forexample hamare pas aik code hai jis ma sync or async code dono likhy huye hain to JS code ko execution k liye stack par bhaijta hai, JS k pas 2 kisam k stacks hoty hai aik main stack or dusra side stack main stack sirf sync code or executions k baad ane waly outputs k liye hota hai means us par code k outputs aty hain or side stak par sirf async code chalaya jata hai means JS code ma se sync code ko direct execution k liye main stack ma bhaij daita hai or async code ko side stack ma bhaij daita hai sirf execution k liye execution k baad output k liye side stack usy main stack k pas bhaij daita hai, means agar hamare pas 5 lines of code hai jis ma teesri line async code ki hai to baki sari lines main stack ma transfer ho jayn gi or aik line side stack ma chali jaye gi or agr side stack us line of code ki execution mukamal kar le pir b wo main stack ma tab tak ni jaye ga jab tak main stack ma mojood sync line of code sara na chal jaye jab main stack sync code ko chala kar khali ho jata hai tab wo check karta hai side stack ko k us ma mojood code ki execution complete ho gai hai ya ni or ye check karne wala process event loop karta hai, to pir wo side stack se code le kar output karta hai, means chahye sync code ho ya async code dono ka output main stack ma hi hota hai par executions dono ki alag stack ma hoti hai.

Event Loop:

The event loop is responsible for managing the execution of both synchronous and asynchronous code. It constantly checks if the main stack (call stack) is empty after executing all synchronous code. If the stack is empty and there are any completed operations in the side stack (like asynchronous tasks waiting in the callback queue), the event loop brings them back to the main stack for execution.

In simpler terms:

- The event loop monitors the main stack.
- Once all synchronous code is done, the event loop looks at the side stack (callback queue).
- If any asynchronous task has completed, it transfers that task to the main stack for execution.
- This allows JavaScript to handle both synchronous and asynchronous operations efficiently without blocking the main thread.

Callbacks:

Ye hamaisha aik func hota hai, ye sirf tab chalta hai jab async code ka completion ho jata hai us k output k liye.

Async code ko likhny k liye ham in cheezon ka use karty hain

setTimeout,

setInterval,

axios,

promises

or in sab k complet hone k baad jo output ata hai us ko show karne k liye ham callback func ka use karty hain.

i.e:

setTimeout(fun(){

console.log(“output of async code”);

},2000);

Is ma set time out async code likhny k liye hai or us k andar jo function bana hai wo callback fun chai jo set time out k complete hone par chalta hai.

Promises:

In JavaScript, a promise represents the eventual completion (or failure) of an asynchronous operation. It can be in one of three states:

- Pending: This is the initial state of a promise, where it is neither fulfilled nor rejected. The operation hasn't completed yet, and this state allows us to store the promise in a variable while waiting for the result.
- Resolved (Fulfilled): When the promise successfully completes and returns the desired result, it enters the resolved state. In this case, we handle the successful output using the “.then()” method.
- Rejected: If the promise fails (for example, due to an error), it enters the rejected state. In this case, we handle the error or failure using the “.catch()” method.
- When the promise is **resolved**, the code inside “`.then()`” will execute.
- If the promise is **rejected**, the code inside “`.catch()`” will handle the error.

```
let myPromise = new Promise((resolve, reject) => {
    let success = true;  // Simulating success or failure

    if(success) {
        resolve("Promise Resolved!");
    } else {
        reject("Promise Rejected!");
    }
});

myPromise
    .then(result => {
        console.log(result);  // This runs when the promise is resolved
    })
    .catch(error => {
        console.log(error);  // This runs when the promise is rejected
    });
```

Promise will take a function, then and catch also takes a function.

We also make nested promises as well but we don't say it nested we say it promise chaining.

i.e:

we make a promise and for this promise we make then and catch and if we want promise chaining then we make another promise inside ".then()" and so on.

Async Await:

Koi b aisa func jis me ham asyn code likhengy, q k asunc code likhny ki wajah se ham promise use kar sakty hain, or us k output k liye hame ".then()" lagana pare ga, us "then" se bachny k liye ham async await ka use karty hain.

i.e

```
function abc(){
    let rawData = await fetch('https request sent');
    let ansData = await rawData.jason();
    console.log(ansData);
}
```

Arrow Functions:

An arrow function in JavaScript is a more concise way to write function expressions. It was introduced in ES6 (ECMAScript 2015) and is often used because of its simpler syntax

There are three types of arrow functions

Fat Arrow function:

```
Var a = () => {};
```

Fat means brackets, arrow means sign, and their body.

Fat arrow function with one parameter:

```
Var b = (param) => {};
```

If we have only one parameter so JS allows us to write parameter without brackets example

```
Var b = param => {};
```

If we have multiple parameters then we use brackets with them

```
Var b = (param1, param2) => {};
```

Fat arrow with implicit return:

It means that return the function value without using return keyword, if we write any thing after arrow but without brackets of body then this value will automatically return, i.e

Var c = () => 12;

Template Literals:

In ES5 (Concatenation): To combine strings and variables or calculations, we had to use the "+" operator for concatenation, which could get a bit messy for long sentences. **In ES6 (Template Literals):** With **template literals**, we use **backticks** " ` " instead of quotes, and we can directly embed variables or calculations inside the string using "${}". This makes the code more readable and cleaner.

i.e.:

ES5:

```
var name = "Ali";
var age = 25;
var sentence = "My name is " + name + " and I am " + age + " years old.";
console.log(sentence);
// Output: My name is Ali and I am 25 years old.
```

ES6

Ex 1:

```
const name = "Ali";
const age = 25;
const sentence = `My name is ${name} and I am ${age} years old.`;
console.log(sentence);
// Output: My name is Ali and I am 25 years old.
```

Ex 2:

```
const num1 = 10;
const num2 = 20;
const result = `The sum of ${num1} and ${num2} is ${num1 + num2}.`;
console.log(result);
// Output: The sum of 10 and 20 is 30.
```

Default Parameters:

Forexample hamare pas aik parameterized function hai jis ko ham call karty wakt 3 values pass kar rahy hain or dusri baar call karne par ham usy 2 values pass karain or aik na karain to JS by default us parameterized variable ko undefined le laita hai, isi undefined ko khatam karne k liye ham parameterized variables ko by default 0 assign kar daity hain taky agar un ma se kisi ki value na aye to usy 0 consider kar liya jaye.

i.e.:

```
function add(a, b, c) {
    return a + b + c;
}

console.log(add(1, 2, 3)); // Output: 6
console.log(add(1, 2));    // Output: NaN (because c is undefined)
```

```
function add(a = 0, b = 0, c = 0) {
    return a + b + c;
}

console.log(add(1, 2, 3)); // Output: 6
console.log(add(1, 2));    // Output: 3 (because c is 0 by default)
```

Rest & Spread Operators:

Spread use hota hai jab hame kisi b array ya object k elements ko copy karna ho.

Rest use hota hai jab ham kuch bachi hui values aik variable ma dalna chahty hon.

i.e.:

Spread:

```
var arr = [1,2,3,4,5,6,7,8];

var copyArr = [...arr];
```

Rest

for example we make a function with 3 parameters and user pass more then three parameters so the function will accept first three parameters and ignore the rest values so therefore if we dint know how many values will pass to the function then we use rest operator in the end

```
function abc(a,b,c,...d){
    console.log(a,b,c,...d);
}
Abc(1,2,3,4,5,6,7);
```

In this example of function, we make a function with three parameter and the 4rth parameter is rest operator if user pass more then three values those values will save in d variable in the form of an array.

Destructuring:

Arrays ye objects se un ka data baher nikal kar kisi variable ma store karny ko destructuring kahty hain.

Arrays:

```
Var arr = [1,2,3,4];
Var [a,b] = arr;          // a=1, b=2
Var [ ,a, ,b] = arr;      // a=2, b=4
```

Objects:

```
Var obj = { name: “syed”, age: 24};

Var {age} =obj;              // age = 24
```

In objects we make a same name variable which we want to take out value from object

Try & Catch:

Java script is an interpreted language which mean it runs line by line so when we have a code of 3 lines foreample

```
Console.log(“hey1”);

Console.log(hey2);

Console.log(“hey3);
```

In this code line one will be print and line 2 got an error which is “hey2” is not defined so the next line of code which is correct but not going to be print due to the error of line 2, so for avoid this type of error we use try and catch in which we tell to the code if there is an error in line 2 check it out but also run the next line of code and don’t stop on second line, i.e.

```
Console.log(“hey1”);

Try(){
        Console.log(hey2);
}

Catch(err){
        Console.log(err);
}

Console.log(“hey3);
```

So in this case all line of code is running.

Java Script DOM:

Document Object Memory

4 Pillars of DOM:

- Selecting html element in JS
- Changing in html tag
- Changing in css
- Event listener

Selection of html elements:

When we want to select any tag, class form html file we write document for it because in JS we say document to html file, and tags in side the file we say it query so if we want to select it we can say queryselector, like

Document.queryselector(h1)

Document.queryselector(“.divClass”)

We can also save them in variables like

Var a = Document.queryselector(h1)

Console.log(a);

When using document.querySelector(), it only selects the first matching element of the specified selector. If there are multiple elements with the same tag (like multiple <h1> tags), only the first one in the DOM will be selected. However, if you want to select all elements that match the selector (like all <h1> tags), you need to use document.querySelectorAll(). This will return a NodeList containing all the matching elements.

Key Difference:

- querySelector(): Selects only the first element that matches the given selector.
- querySelectorAll(): Selects all matching elements and returns them as a NodeList.

, i.e.

Document.queryselectorAll(h1)

Ham html tag ko un ki id or class k through b utha sakty hain jaise ham “queryselector()” laity hain usi tarha hamare pas “queryselectorById(“#idDiv”)” hota hai jo id k through tags ya divs ko uthata hai or isi tarha “queryselectorByClassName(“.classDiv”)” hota hai jo class name k through div ya tag ko uthata hai.

Change in html:

Now if we want to change anything in html tags for example there is a tag of h1 in which already written “I am h1 tag”, and if we want to change this text through JS first we select it through query selector then we change it by using “innerHtml”, i.e.

First way:

```
Var a = Document.queryselector(h1)

a.innerHTML = “I am html tag which is changed by JS”
```

second way:

```
Var a = Document.queryselector(h1).innerHTML = “I am html tag which is changed by JS”
```

innerHTML ki jagah ham textContent b likh sakty hain like

```
Var a = Document.queryselector(h1).innerHTML = “<h1>hello</h1>”

    //   hello

Var a = Document.queryselector(h1).textContent = “<h1>hello</h1>”

    //   “<h1>hello</h1>”
```

In dono ma diff ye hai k agar ham innerHTML use kar k tag k sath koi text likhty hain to wo usy tag ma convert kar k text show kar daita hai, par jab ham textContent use karty hain to wo usy as it is likh diata hai tag k element k sath wo usy tag ma convert ni karta.

Change in css:

Now if we want to change any style of any tag in html using JS so first we select the tag and store it in variable then use style property with dot extension and change the property, i.e.

```
Var a = Document.queryselector(h1)
a.style.color = “red”                // text color change to red
a.style.backgroundColor = “black”    // text background will become black
```

in style we can not provide multiple styles in one line we provide it in multiple lines.

Event Listener:

An **event listener** in JavaScript is a way to detect and respond to user interactions or actions on a webpage, such as **clicking**, **dragging**, **pressing a button**, or other actions (events). When an event occurs, the **event listener** triggers a specified function (called a **callback function**) that performs a certain task in response to the event.

```
Var a = Document.queryselector(h1)
a.addEventListener(“click”, function(){
    console.log(“hello”);
    a.innerHTML = “I am html tag which is changed by JS”
    a.style.color = “red”                // text color change to red
    a.style.backgroundColor = “black”    // text background will become black
})
```