**JavaScript Concepts**

**Console.time(‘any text’)**

**Console.log(‘to display anything in console’)**

**Console.table(‘to displat object in table form’)**

**Console.warn(‘to throw warning in console’)**

**Console.clear()-> it will clear console**

**Console.timeEnd(‘ant text’) -> it will log time to execute code between console.time and console.timeEnd**

**console.error()**

**Datatypes in Javascript**

* Boolean type.
* Null type.
* Undefined type.
* Number type.
* BigInt type. -> number greater then 2^51 -1
* String type.
* Symbol type.( symbol(), this will create unique value, (symbol() === symbol() -> false), its helps to create unique key in object.

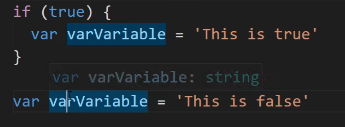
Note: this Unique Key in object can’t be use in for in loop, cannot be converted into JSON Object using JSON.Stringify() method and connot be access like obj.key1 nor obj[“key1”], we can access object values only by obj[key1].

**Undefined and null are also datatype, where in null we assign the value, but it indicates absence of data**

**Var vs let vs const:**

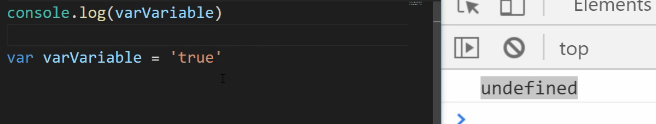
Var has functional scope if it’s defined inside function otherwise it has global scope.

var variable can be reassigned/redeclared multiple times.



However let cannot be redeclared multiple time.

Var variable can be declared before its initailize



However let will give error variable not found.

Const has local scope but cannot be modified irrespective of array and Objects

Let,const has local scope

{

let variable;

const variable

}

**Majorly developer used const and let only**

**Conversion and Coercion:**

**Coercion: two type:**

**1.Implicit coercion: It does not need explicit function fr conversion.**

**e.g ‘5’+10= 510, this will check if both operands have same datatype? If yes then it will do : OR it will convert to string.**

**1.Explicit coercion: It need explicit function for conversion.**

**e.g Number(‘5’)+10= 15**

**Note: Explicit Coercion Is also known as Conversion**

**BigInt is the number Greater then 2^53-1. Or 2n,5n etc**

All Javascript code is get into Global execution context before its compile that’s why hoisting happens.

Global Execution Context: all code which is outside the function body it will go into global execution context, this refer to global object whether is strict mode or not.

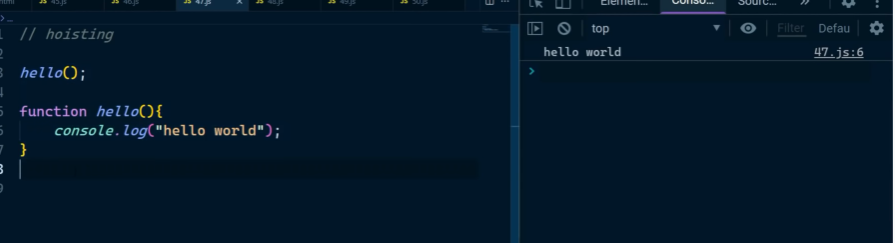
Hoisting:

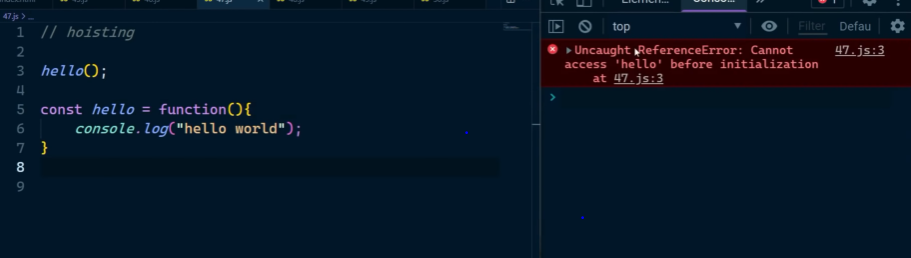
Javascript Moves the functions and variables using “var” at top of global execution context.

Functions declarations are made available.

Variable declarations are made Unavailable. Except var

It moves function declaration and variable using variable (only var) at top of the scope.





**Scope Chaining: If You don’t have the Icecream You can ask for IceCream from parents if they also dont have Icecream then You can ask icecream from grandParents.**

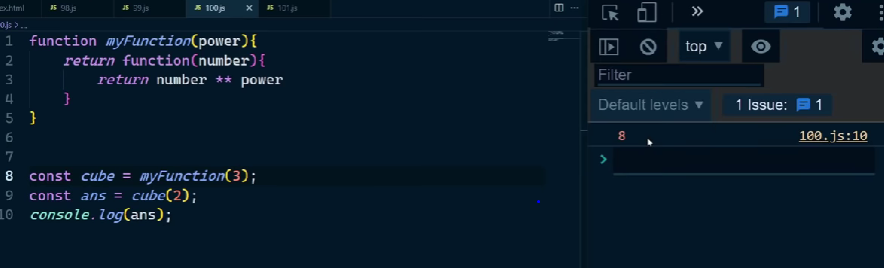
**Note : GrandParents cant ask Icecram from grandChild.**

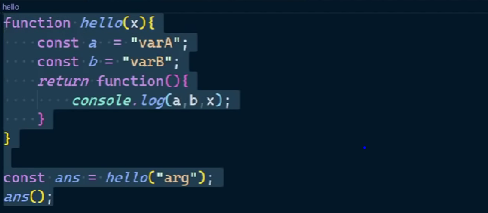
**So Similarly replace Icrecream with some variable and replace You, parents, grandParents with innerFunc, MiddleFunc, Outer Func**

**Lexical Scoping means if variable is not declared inside function scope then that function will check that particular variable is present outside that function or Not. i.e if variable not present in local scope then Javascript will check in global Scope.**

**Closures: When some function is returned inside function, then inner function returns with all the parameters having with its functional executing context and outer function parameter(lexical environment). ( we think that whenever outer function returns inside function, the whole outer function gets popped from Stack but Inner returned Function keep its functional executing scope parameters /and also its lexical environment parmeters,i.e means closure)**

Examples of closure:





output: varA, varB, arg

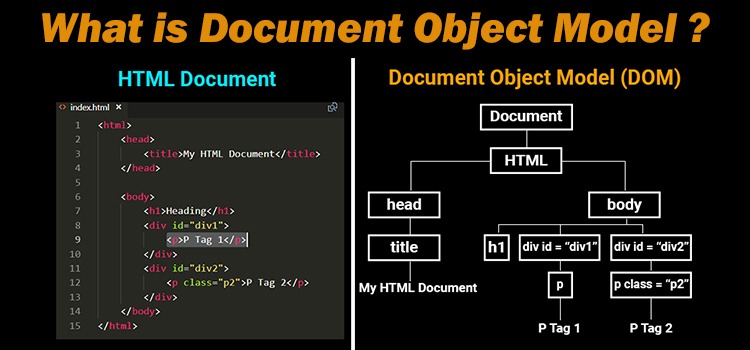
why need of Closure:

because we need to maintain self contain state of the function

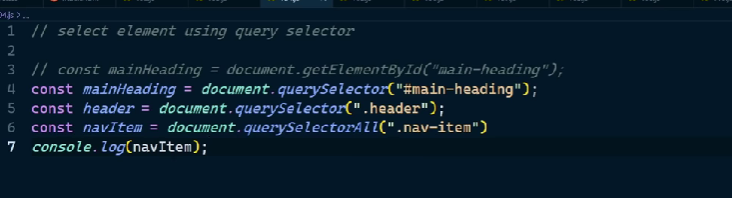
what is DOM:

It make Tree like representation of our HTML code,

Each html refers as Node, where each node store in window.document as key value pair



Query Selector:



Difference between query selector vs document.getElementById or classname:

Query selector returns Node Elements, while document.getElementById returns HTML elements

Node vs Element:

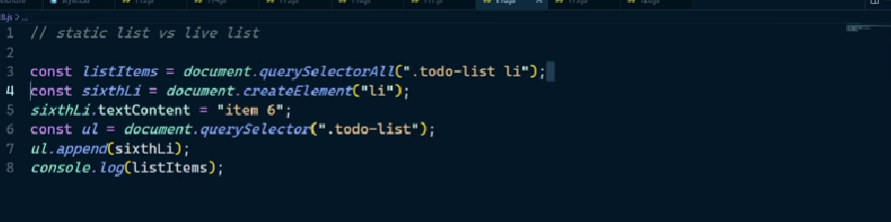
Node is everything inside HTML element including <!comment--> ,text etc..

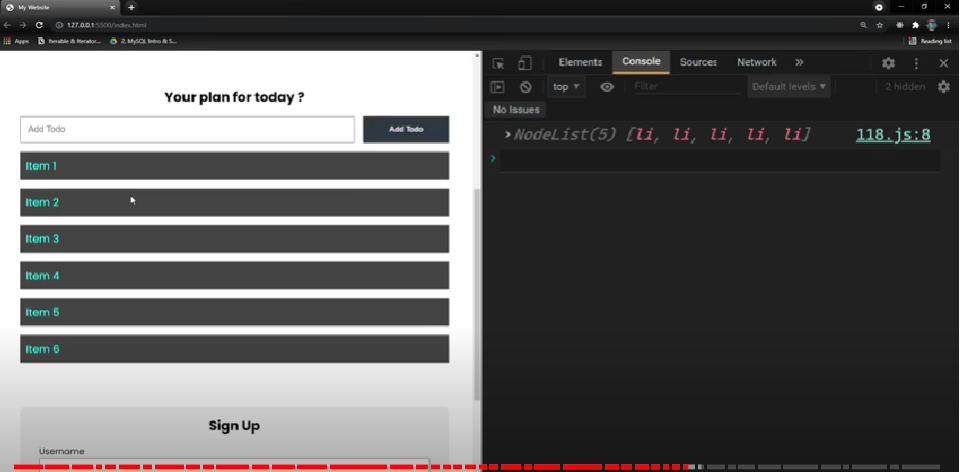
Element is special type of node which include ,span, div, <a> etc..

Query selector gives only Static List , while document.getElementById give Live list.

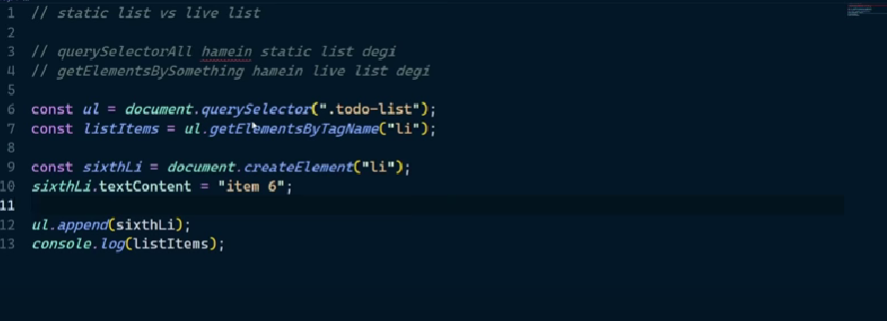
For e.g if we use query selector on console.log() it will display static list , if we use document.getElemnt then it will display live list.

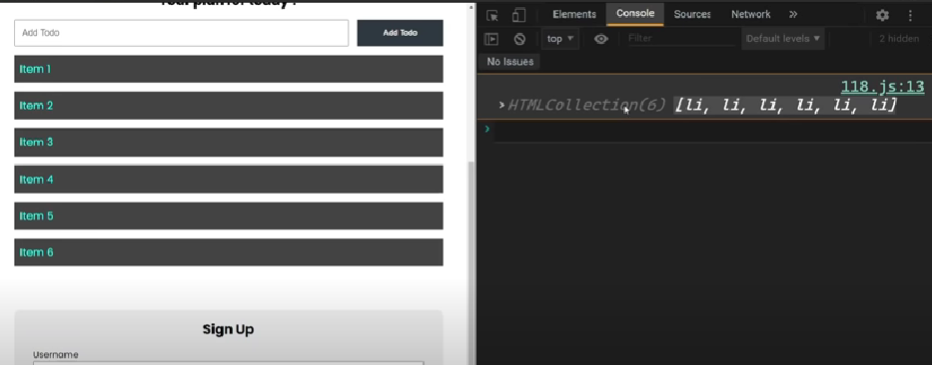
Using query Selector





Using getElement





Best way to Load Javascript file in HTML document:

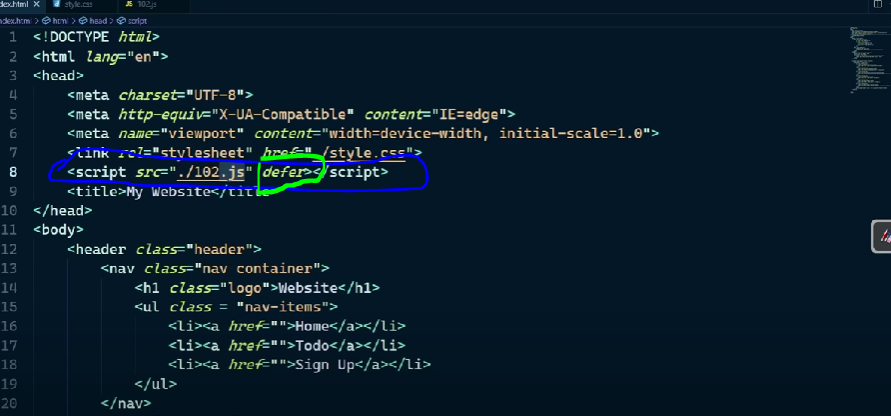
Note if You add <script src=”./path.js” ></script> at top of html file,

Then if you have used document.getElementbyId or document.Queryselector in .js file it will throw error because html file was not parsed at that point.

There is other way to link js file, you can use defer and async keyword:

Async will asynchronously load js file and also parallelly parsed html file, but suppose when js file loaded and html file parsed incomplete and that point Js file execute first prior to parsed remaining html file, so it also can throw error.

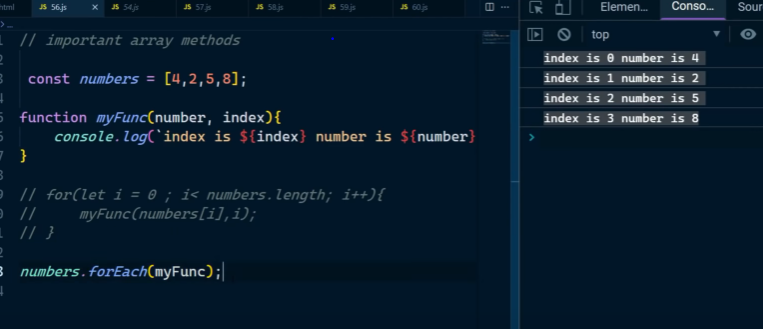
defer will also asynchronously load js file and also parallelly parsed html file, but suppose when js file loaded and html file parsed incomplete and that point remaining html file parsed first then js filed will execute, so it also cannot throw error, so defer is the best way to link js file.



For each loop used in array:

Array.forEach(Any func),

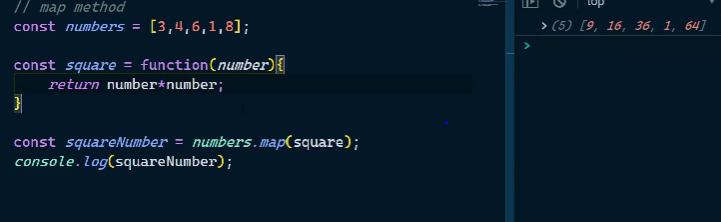
Note for each automatically passes Number and Index both to any func.and not return Anything.



map method in array

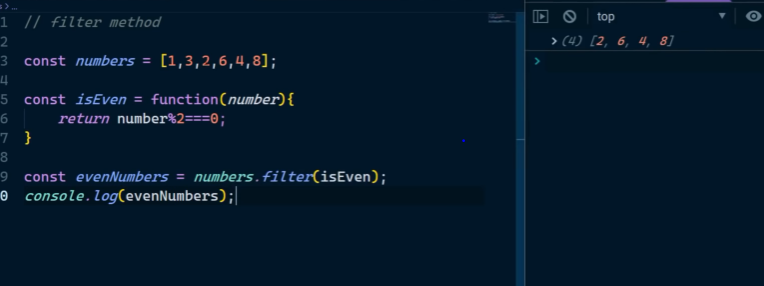
Array.map( anonymous function )

Map method run function on every element inside array and return new Array



filter method in array

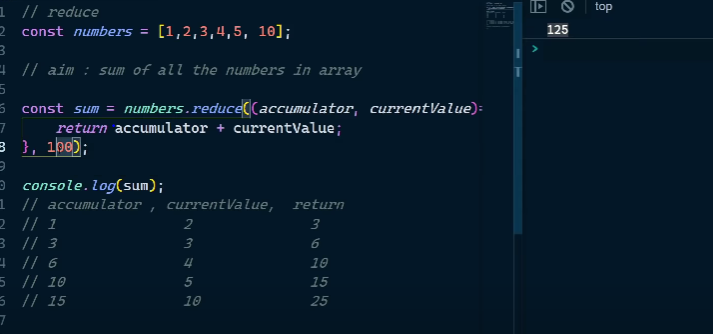
Array.filter( func ) this will retain only element where passed function return true and filter falseish elements



Reduce method in Array

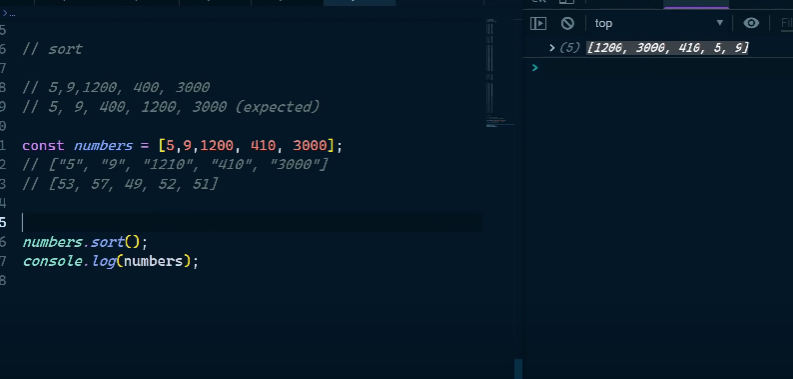
Array.reduce( func , Initial value) will reduce the Array to 1 return Value.

Here Initial value Bydeault is 0.and here accumulator = initial value



Sort in Javascript:

Note : sort method treated INT inside Array as String so it sort on basis on Ascii value.

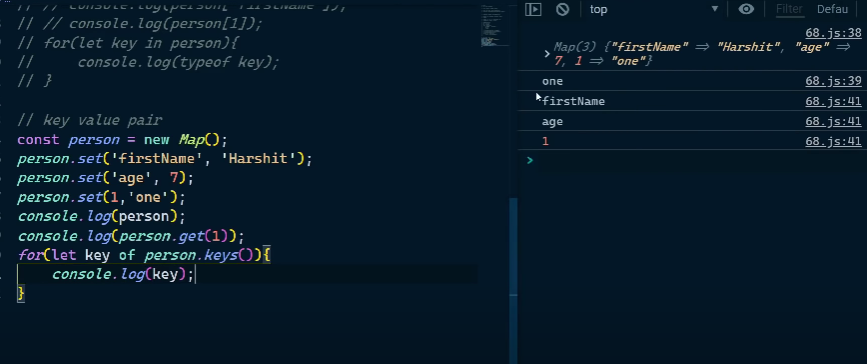


So to sort Numbers we Do



Numbers.sort((a,b)=> b-a ) for reverse Sort

Map Method to create Object



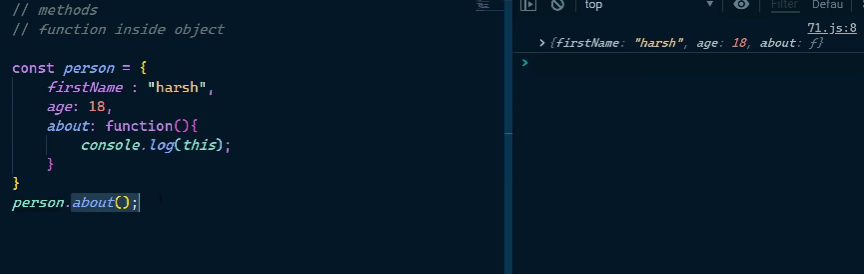
This keyword in Javascipt:

In browser-> console.log(this) -> window Object

In Node -> console.log(this) -> {} -> denotes module export object given by every node file.

this keyword inside functions indicates the object from where the function is called e.g object.function(),

value of this defines at runtime.



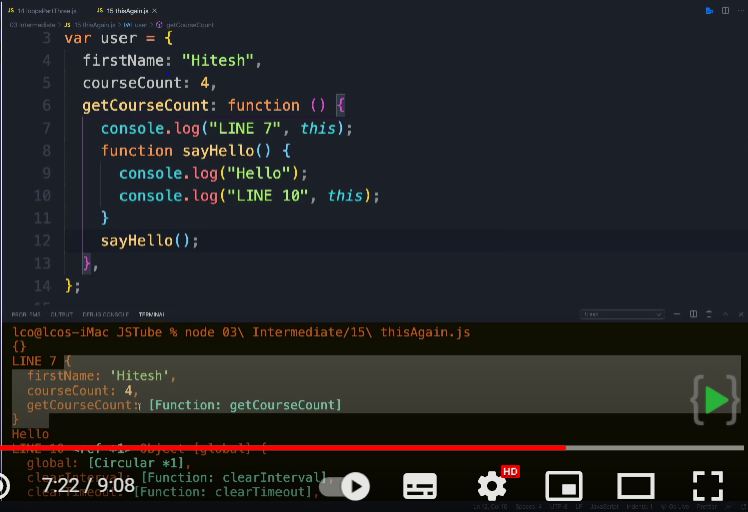
Simple console.log(this) -> print windowObject,if it runs on browser, However in Node it print empty Object {}

Everything we write on javascript page it automatically add on global window Object.



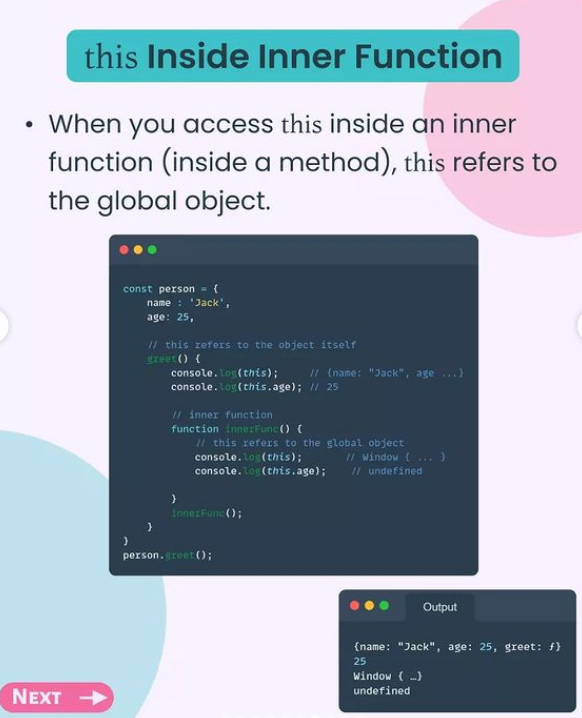
For all regular function call “this” point to window Object

Here below sayHello() is a regular function call and user.getCourseCount is a objectFunctionCall



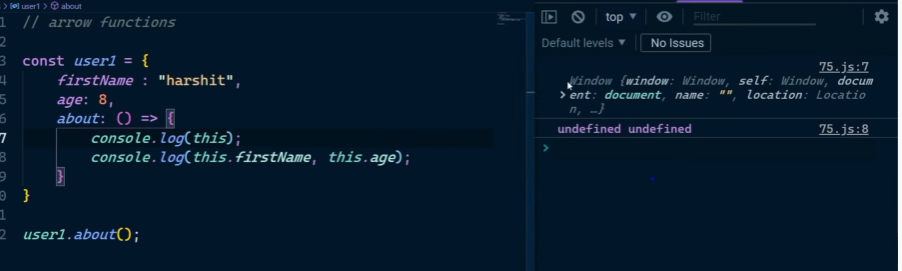
Note : In case of Arrow function “this” always refers to ‘this’ present in its surrounding / lexical Scope.

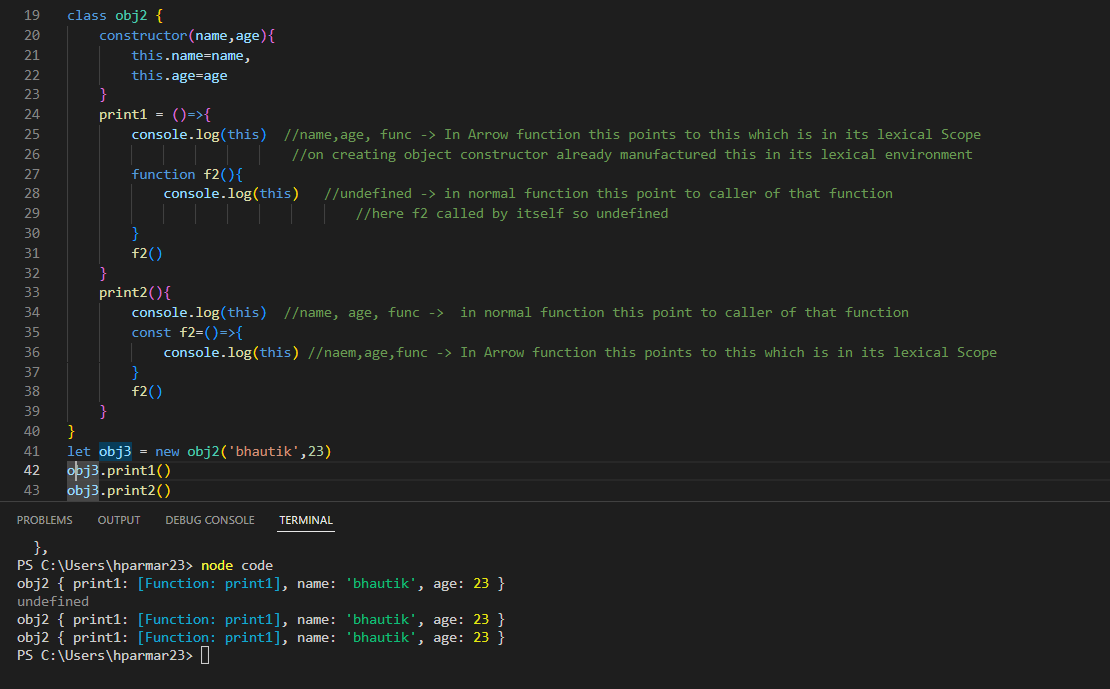
In Normal Function ‘this’ points to caller of that function

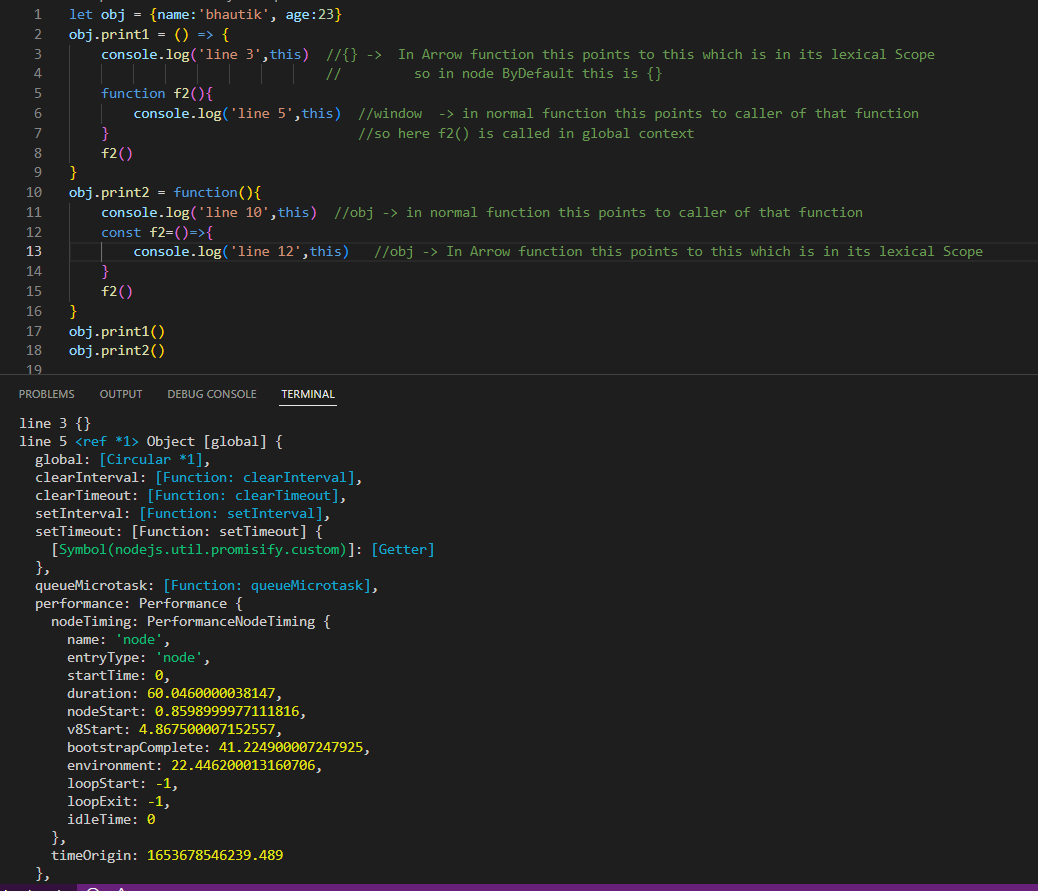


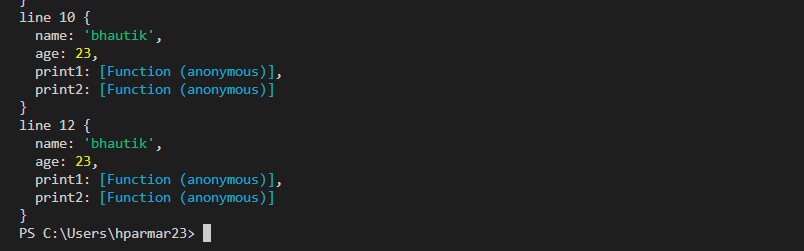
Difference between the Normal function and the Arrow function?

* Arrow Function is Introduced to handle the problem of **this** keyword, because in normal function this keyword sometimes automatically binds to window Object, sometime binds to its own function.
* So now in arrow function, nothing is binded neither arguments neither this.
* In a normal function call, the **arguments** keyword is supported, and in arrow function **arguments** keyword is not supported.
* regular functions are constructible, they can be called using the new keyword.  However, **the arrow functions are only callable and not constructible**, i.e arrow functions can never be used as constructor functions.
* Arrow function solve problem of binding this in normal Function.







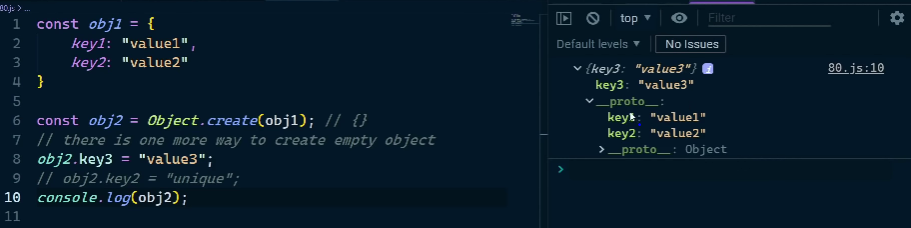


Prototype:

Prototypes are the mechanism by which JavaScript objects inherit features from one another.

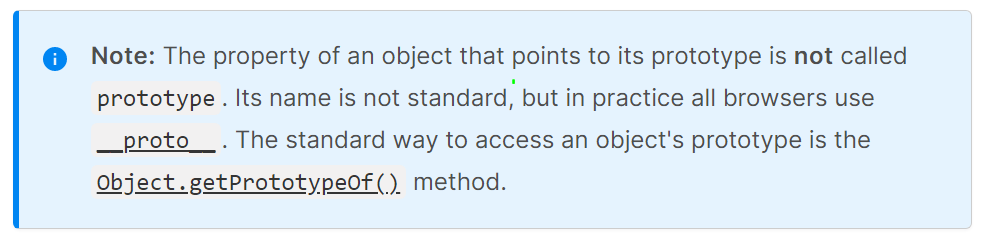
Every object/Function in JavaScript has a built-in property, which is called its **prototype**. The prototype is itself an object, so the prototype will have its own prototype, making what's called a **prototype chain**. The chain ends when we reach a prototype that has null for its own prototype.

Const Obj2 = Object.create(obj1) -> create \_\_proto\_\_ value for obj2 which contain obj1

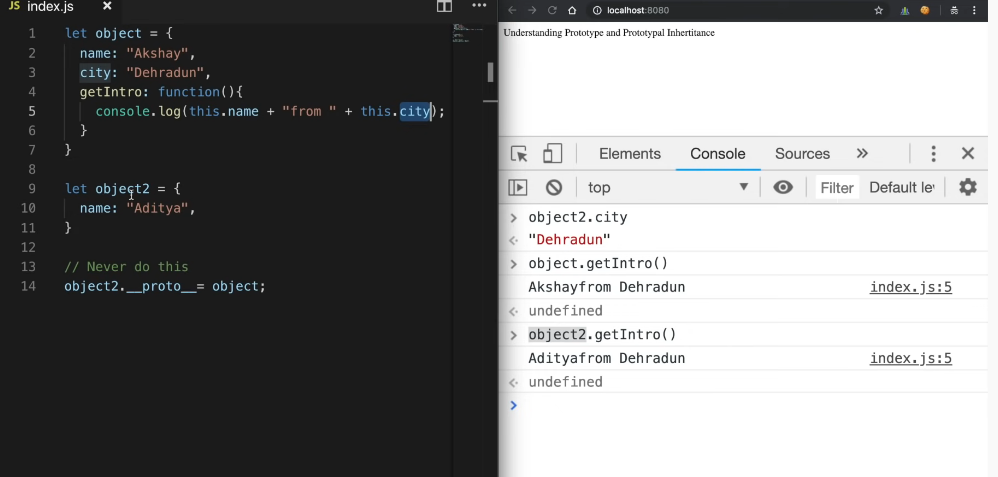
C

\_\_proto\_\_ is same as [[prototype]] , but \_\_proto\_\_/[[prototype]] is not same as PROTOTYPE,

\_\_proto\_\_ present in Objects, Prototype presents in only functions



Prototype Inheritance:



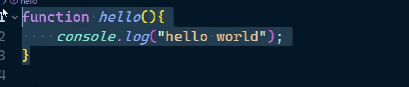
Here object2 is inheriting the property and functions of object, because we \_\_proto\_\_ of object2 points to object.

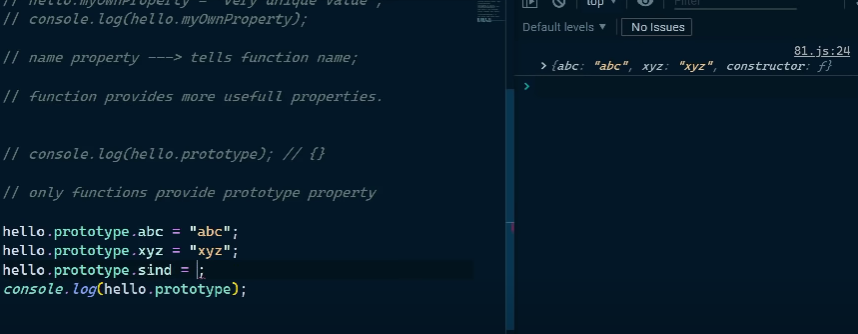
In Javascript function -> function + Object, In short in javascript everything act as object

So function also act like objects we can use function as:

Function.name, function.bind,function.call,function.bind

There is also something like function.proptotype where proptotype is just as empty object { },so we can use this prototype to add key:value pair.





New Keyword:

Suppose ,

Function CreateUser(firstName,age){

this.firstName=firstName;

this.age = age

}

const user1 = new CreateUser(“harshit”,6);

here new keyword do following things:

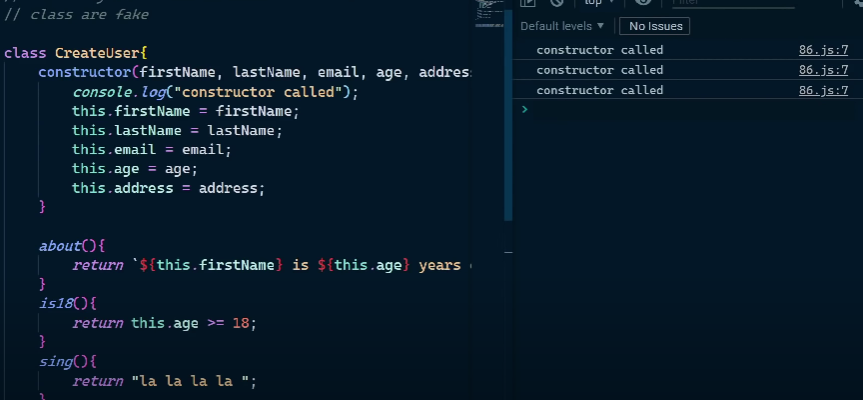
1. Create empty Object i.e this={}
2. Add following parameter inside this object and return that object.
3. So New keyword help us to create new object but The new object’s internal ‘Prototype’ property (\_\_proto\_\_) is set the same as the prototype of the constructing function.

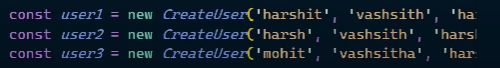
Here CreateUser is known as Constructer function, Constructer functions helps us to create object,

Convention-> Constructer function Name always start with capital letter.

Class keyword:

Class make easy to do all above things: Instead of writing CreateUser Function we make class and call the constructer method using new keyword and create Objects.

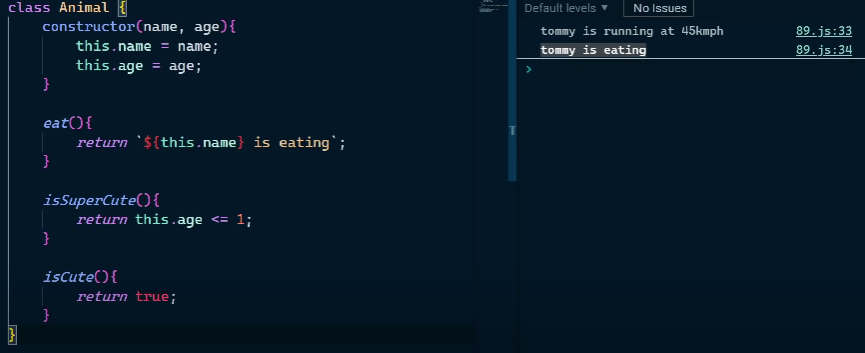


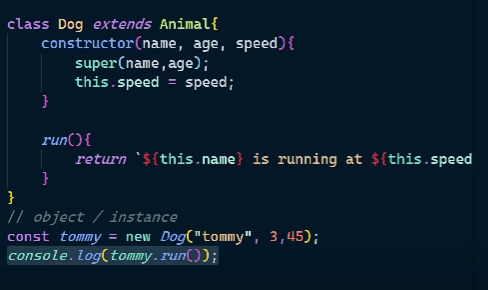


We can make child class also using extends keyword:

e.g class B extends A{ }

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

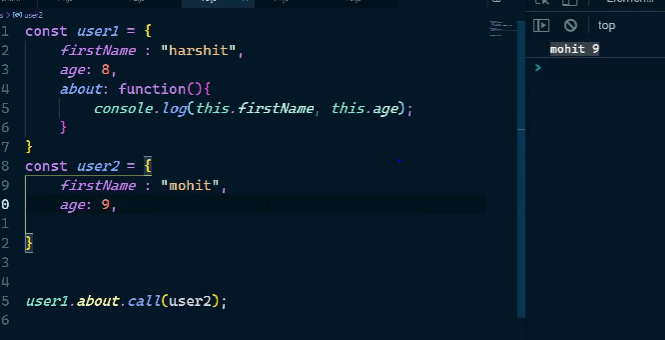




Difference Between Call, Bind, apply:

Call:

User1.call(user2) is used to call function which is present in user1 Object as value or outside the user obj as shown in 2nd Image, but In that function you have to used parameters present in user2 Object. So here call method is used.(irrespective user2 doesnot have that function)



You can also passed parameter to that function



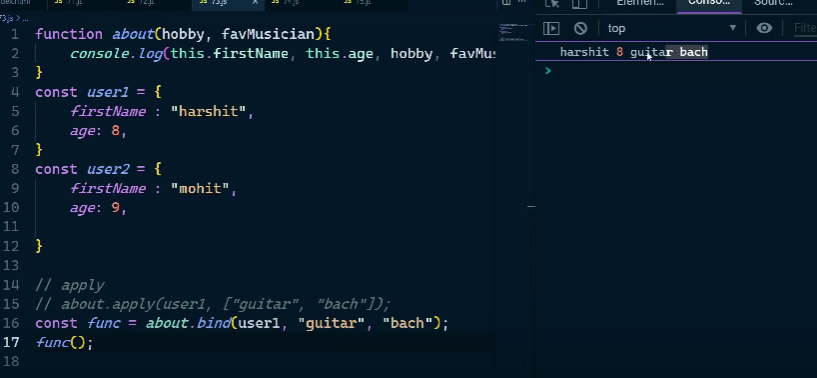
Apply:

Apply is same as call but just it passes parameter in form of Array as shown below

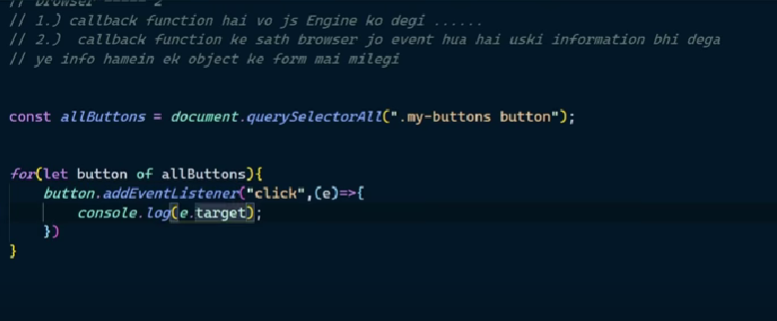


Bind:

Bind is also same as call but it does not run the function wherever its called, but it returns a new function, In short it bind the function with the variable where it returns, so in future we can call that binded function.



Event Listener Function takes two parameters i.e eventName and also event object which can use for further information.



***Stack memory stores primitive datatypes(string, Numbers) and the addresses of objects. The Reference datatype(objects, Array) are stored in heap memory.***

Event Loop:

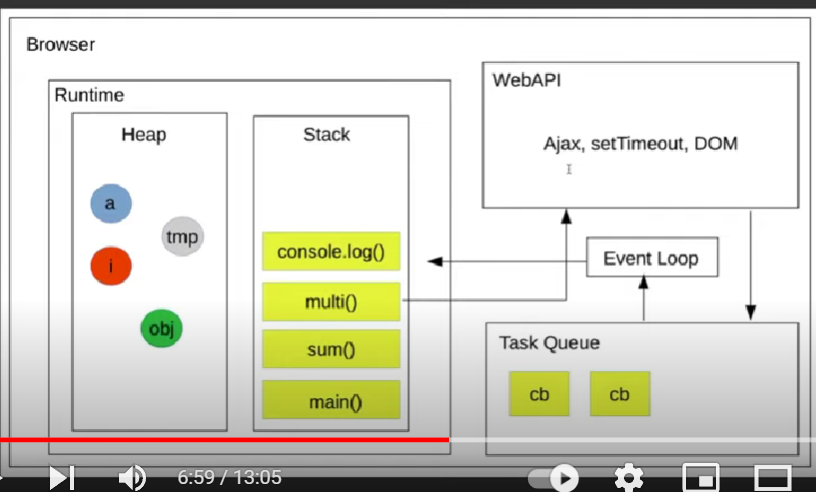
Every function on Javascript page on runtime goes intside stacks i.e (global execution context) and if they have inner functions they will get upon each other inside stack, when respective function execution overs those functions start getting popped out from stack.

Suppose if there is some Asynchronous functions are present then they will treated as “callbacks” and get stored in WebApi container. Suppose those callback function execution completes, then they will shift to task/Event Queue.

Here Event loop comes in picture, event loop first check and wait till all function in stack gets executed , after stacks get empty then only callback value returns from the queue.

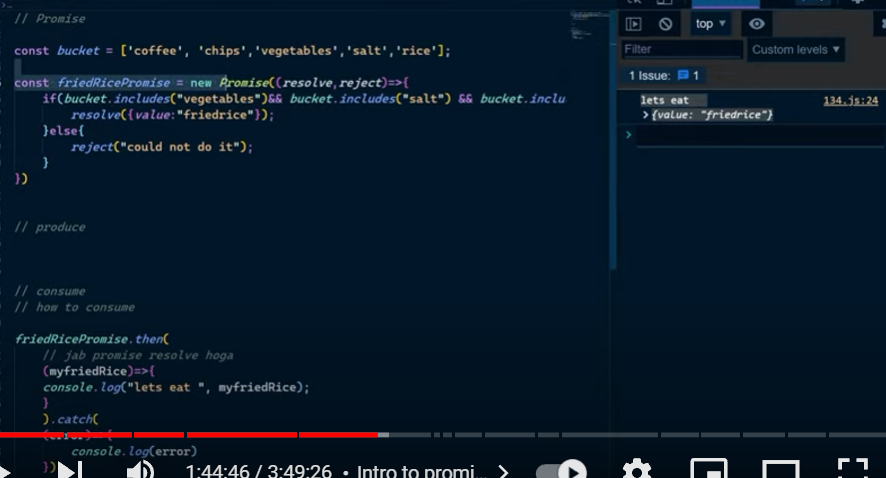
It is called event loop because it continuesly( in loop ) checking the callStack and callback Queue.

Note: for every callback give promise e.g fetch will store result in MicroQueue instead of callback Queue, and MicroQueue priority is higher than callback Queue.

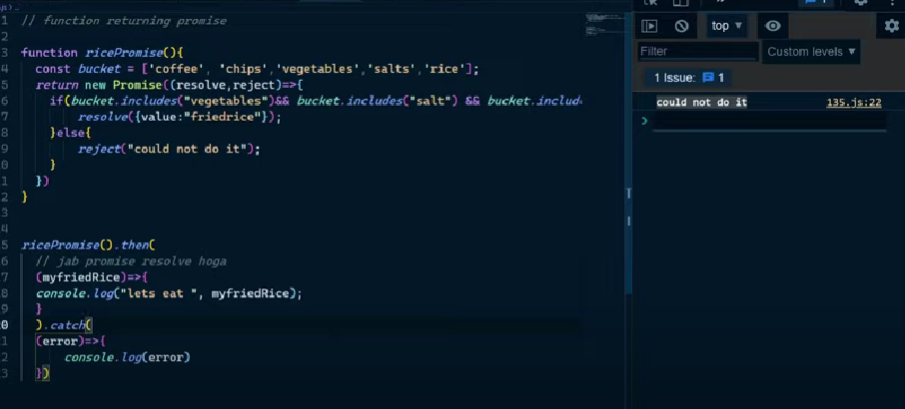


***Promise:***

Promise is an asynchronous task, that take over by Task/Event Que as shown above and it get executed after Javascript executes all Lines of code.

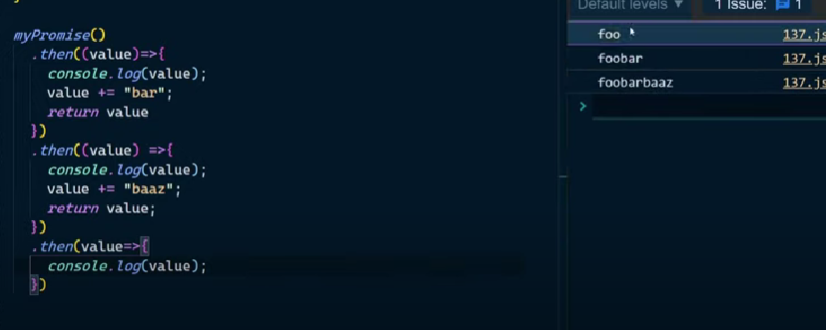


Function returning promise:

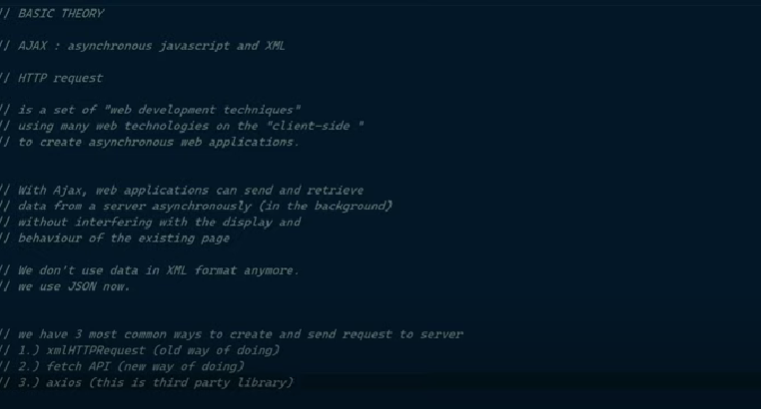


Note: then and catch methodF always return promise

So you can do chaining of then e.g:



AJAX: Aysnchronous javascript and xml, but now we use JSON Instead of xml(extensible markup language)

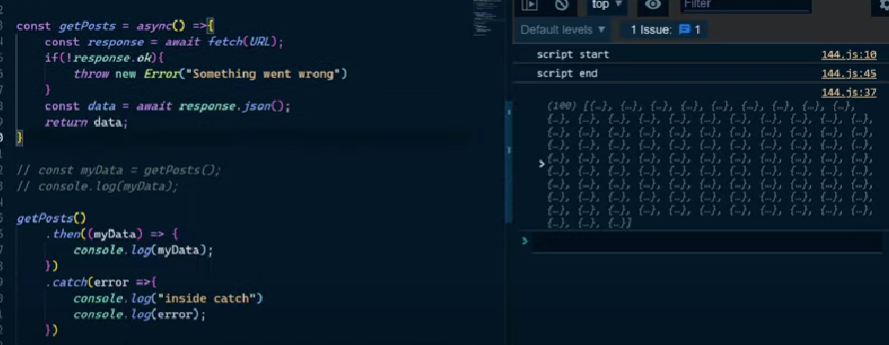


ASYNC and AWAIT:

Aysnc keyword always returns Promise.

Await keyword make async function body to run synchronously.

Inshort Await will wait till we get response.



Map and weak Map in JS

1. Map is similar to object, but the difference is in map we can store object and function as key also, which is not possible in object (object can contain only string and numbers as key).

Let a = new Map()

a.set(obj1,obj2) i.e obj1->obj2

1. We can loop the object while accessing both [key, value] at same time like e.g using for(let [key,value] of mapobj){}

Weakmap

A **WeakMap**accepts only objects as keys whereas a **Map**,in addition to objects, accepts primitive datatype such as strings, numbers etc. and There is no size property exists in **WeakMap**.

Set and weak Set in JS

1. Set method removes duplicate element

Weak set

The WeakSet is similar to a Set. However, WeakSet can only contain objects whereas a Set can contain any data types such as strings, numbers, objects, etc.

Exception Handling:

Try, catch , finally, throw

As we know try catch means the same as it names.

finally -> this runs everytime, whether code throw error or it get resolved.

Throw -> throw expression -> throw 10/0 -> divide by zero error

Es6 Modules:

You can Divide your large code in multiple files to make linearity in your project using Impost and export functionality.

For that you have to add type=”module” in script file while attaching script file to index.html

<script src=”./app.js” type=”module” ></script>

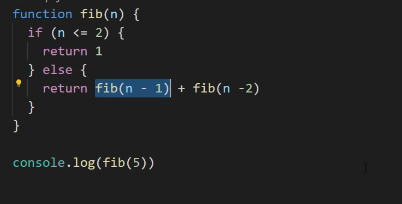
Note : while adding type=”module” it automatically add defer in script, you don’t have to add Manually again.

Memoization and Dynamic Programming

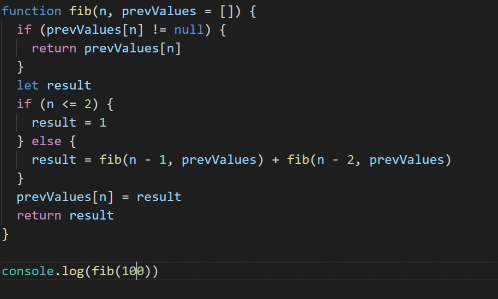
It is Technique so that program we will cache the response in some memory so that we do not execute the function for the same request.

Consider the below program for printing n fibonnaci number.

Fibonacci sequence is 1,1,2,3,5,8 I.e next is sumof previous 2 numbers



we can Improve this code efficiency using Memoization i.e remembering the previous computed value.



Programming Paradigms:

It’s the way how You organize code.

Javascript is versatile supports all 3.

