Function expression vs function statement

Function expression behaves like a variable means if we call function before declaration it will be undefined

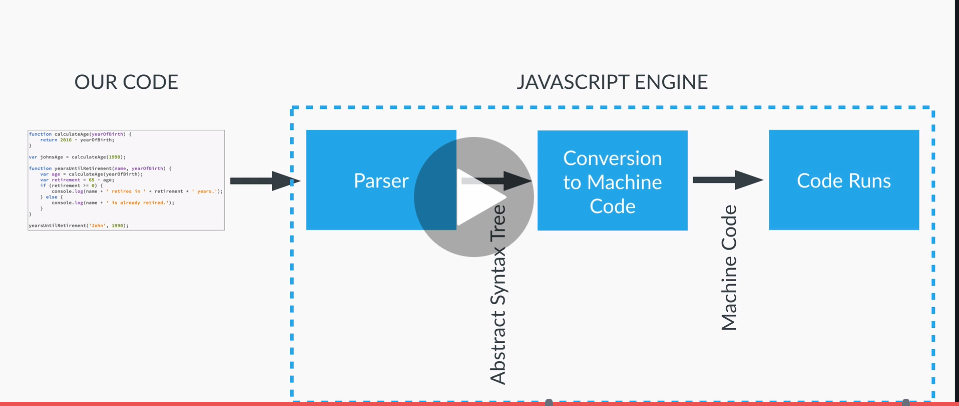
Function statement can we called before declaration

**How javascript works behind the scene**

<https://github.com/jonasschmedtmann/complete-javascript-course/tree/master/3-how-JS-works>

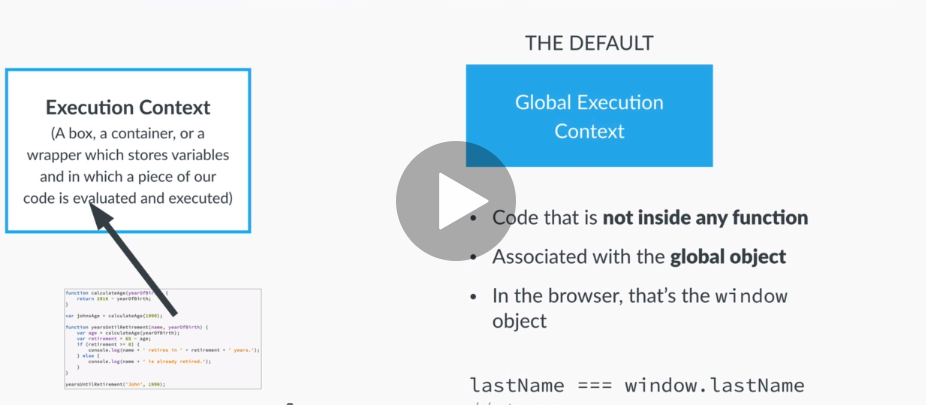
Javascript code runs on some kind of platform mainly on browsers or node.

And platforms has some javascript engine, our code goes through javascript engine it gets parsed through parser if any error code stops there if no error code converted to a **datastructure Abstract Syntax Tree(AST)** which is then converted to Machine code that runs.

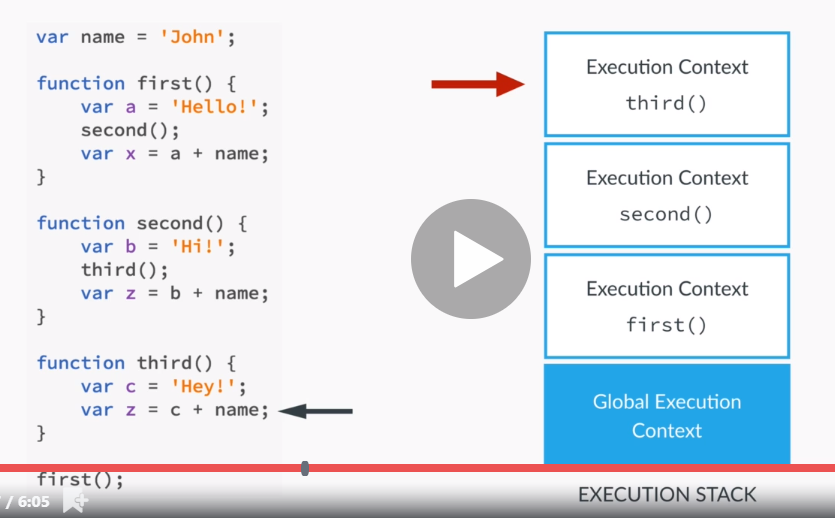


**Execution context** – A box, a container, or a wrapper where all our codes are present

In case of browser the default global execution context is window



Global execution context is for code which is not inside any function, for the code inside a function it gets its own execution context



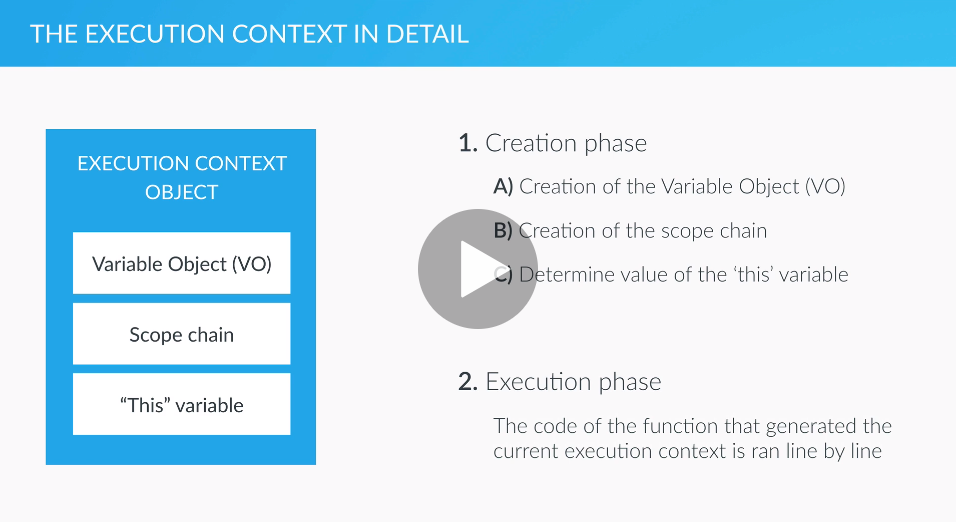
**Execution context in detail**

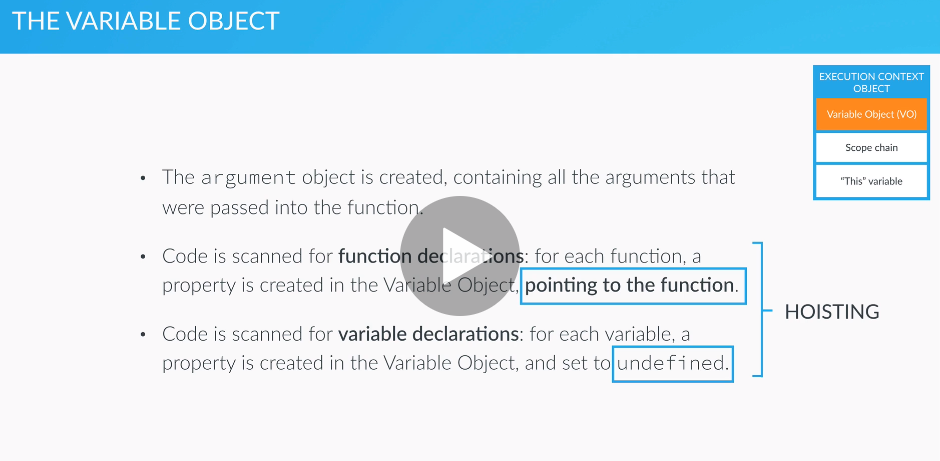
For each function execution context is created it has two phase creation phase and Execution phase

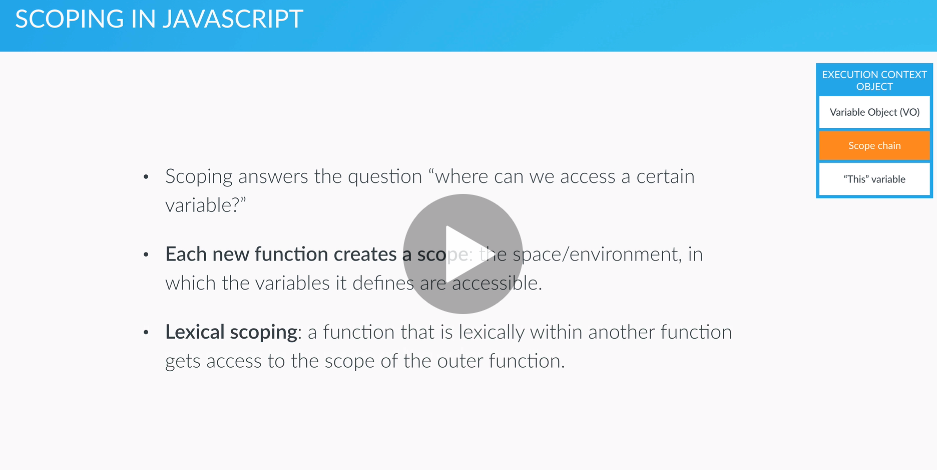
**Creation phase:**

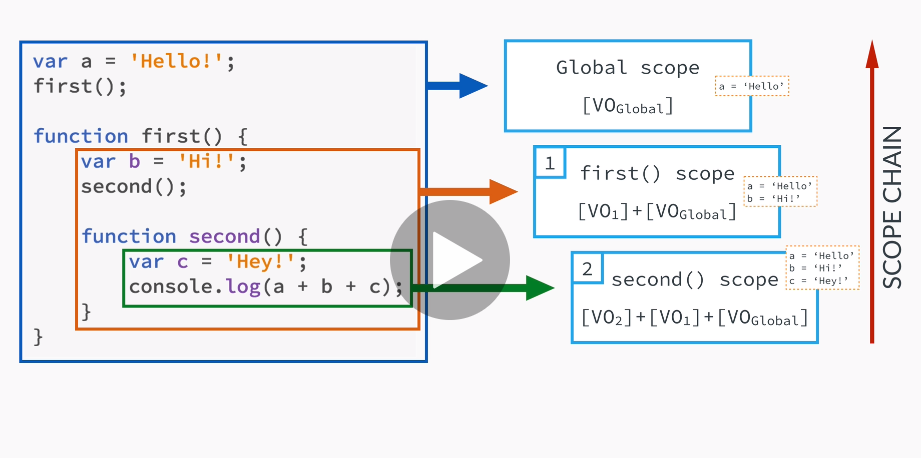
1. Creation of variable Object
2. Creation of scope chain
3. Determine value of this variable

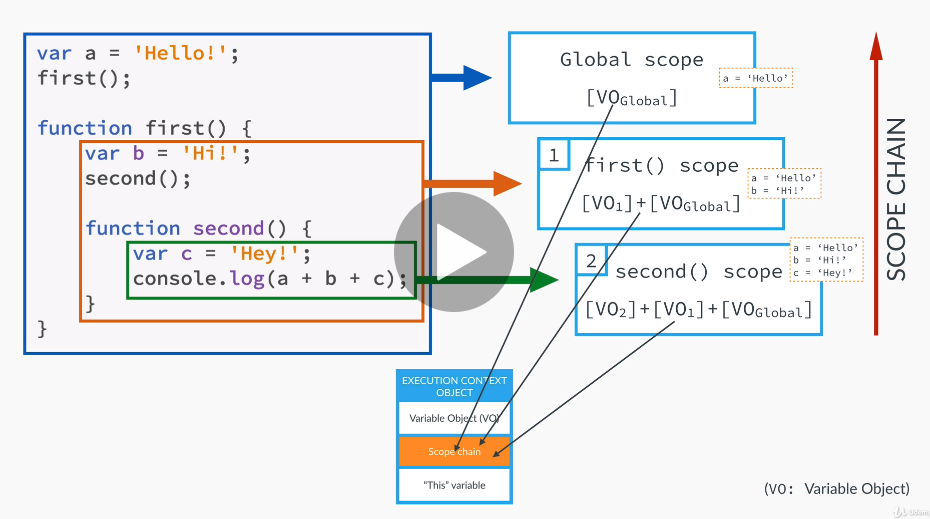
**Execution phase -** Execute code line by line

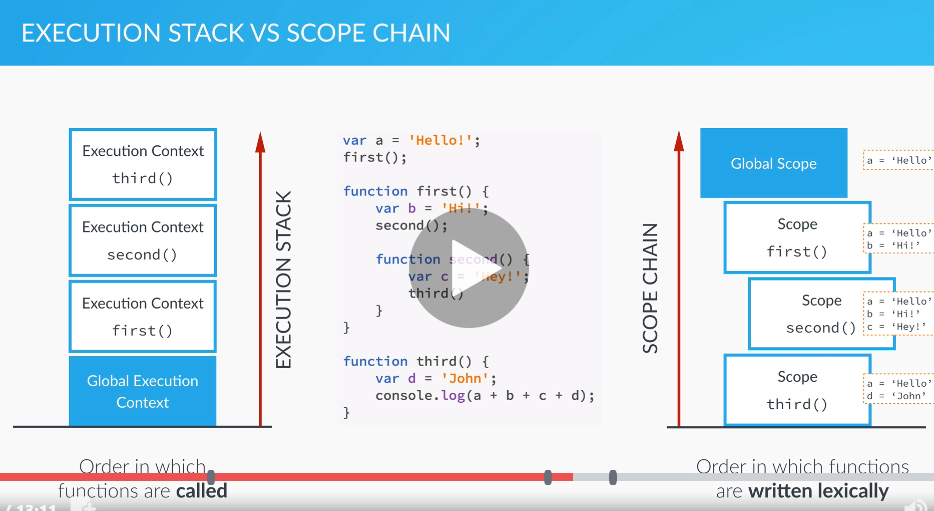












**this variable**

It is a variable that each and every execution context gets.

Now it is important to know where does the this keyword points.

Let’s see

**Regular function call:** the this keyword points to global object (window object) in case of browser

**Method call:** in this case this variable points to the object that is calling the method

**NOTE:** this keyword(of function) is never assigned a value unless function is actually called

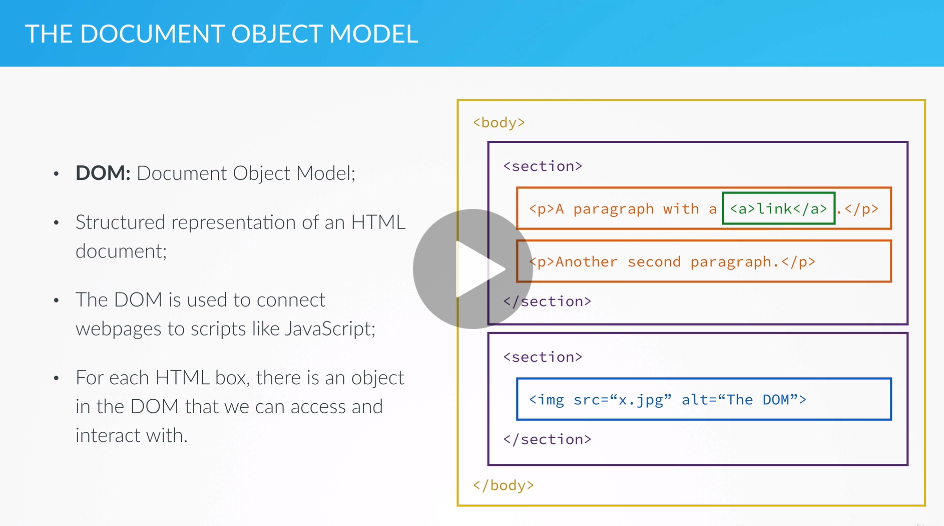
**The Document Object Model**

**DOM-** it is structural representation of HTML

DOM is used to connect web pages with scripts

For each html element there is an object in the DOM which we can access and interact with

Browser provides Document object with help of which we can interact with DOM



**Events handling:** Events are like click, focus,blur

Event listener performs an action based on certain events.

Note that Event listener happens when execution context is empty

Questions

Closure

Eventbubbling vs something

Higher order function

Curry function

Patel video

Design pattern

Data structure revision

Git interview question

Unit testing interview question jasmine karma version and differences

Unit testing for stores

Interview questions on rxjs an ngrx

C:\Users\vi1kumar\Desktop\tus\JS\0.freeCodeCamp

**Closure**

Variable from external function is available to internal functions

**Promise**

function myFunction(){

var z = new Promise(function(resolve,reject){

reject('resolve')

});

return z;

}

myFunction().then(function(val){

console.log('promise resolved',val)

},function(val){

console.log('promise rejected',val)

})

**How setitmeout works/ event loops**

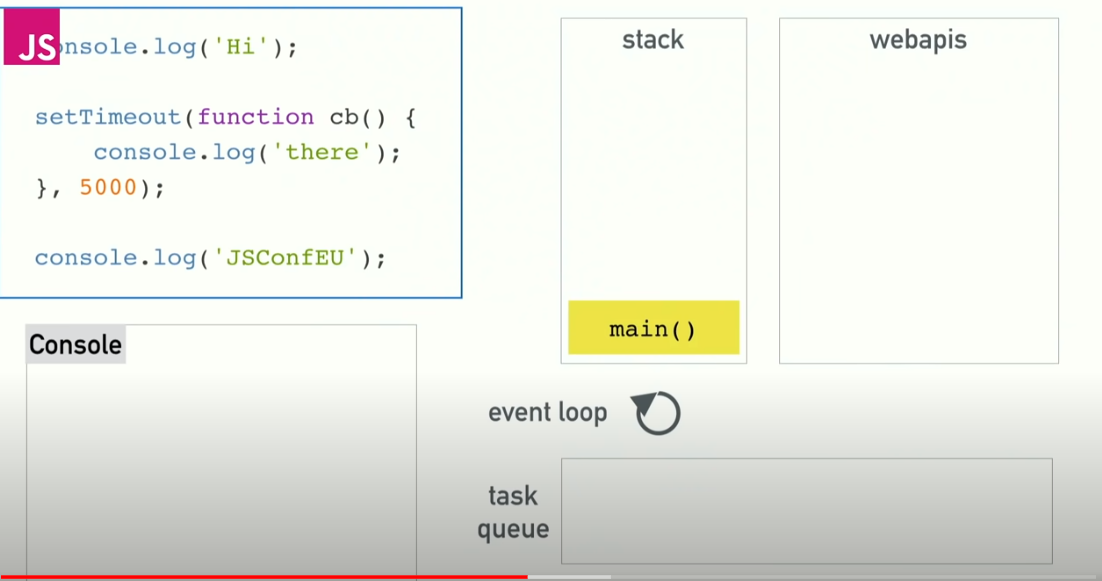
Javascript is single threaded language (it do one thing at time) and javascript is scynchronous (it works line by line)

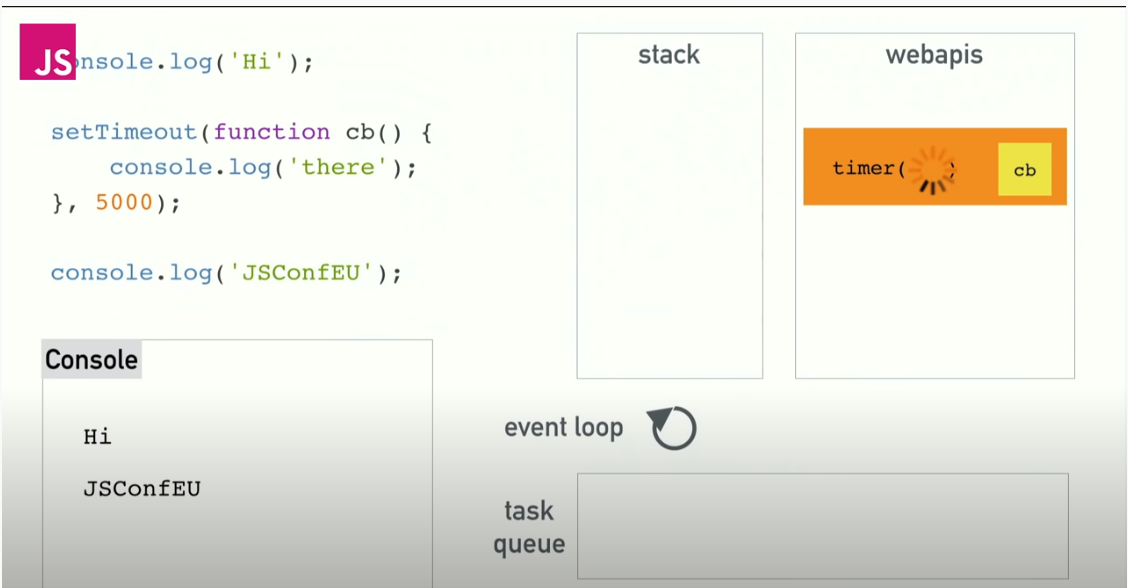
Now lets see what happens when there in some ascyncrinus task like ajax call or settimeout

Please NOTE javascript can only do one thing at a time but we have browser which provides various apis like settimeout, AJAx call

Now we can say that we have two different runtime one is javascript and other is browser, jaascript maintains call stack and so the browser(webapis)

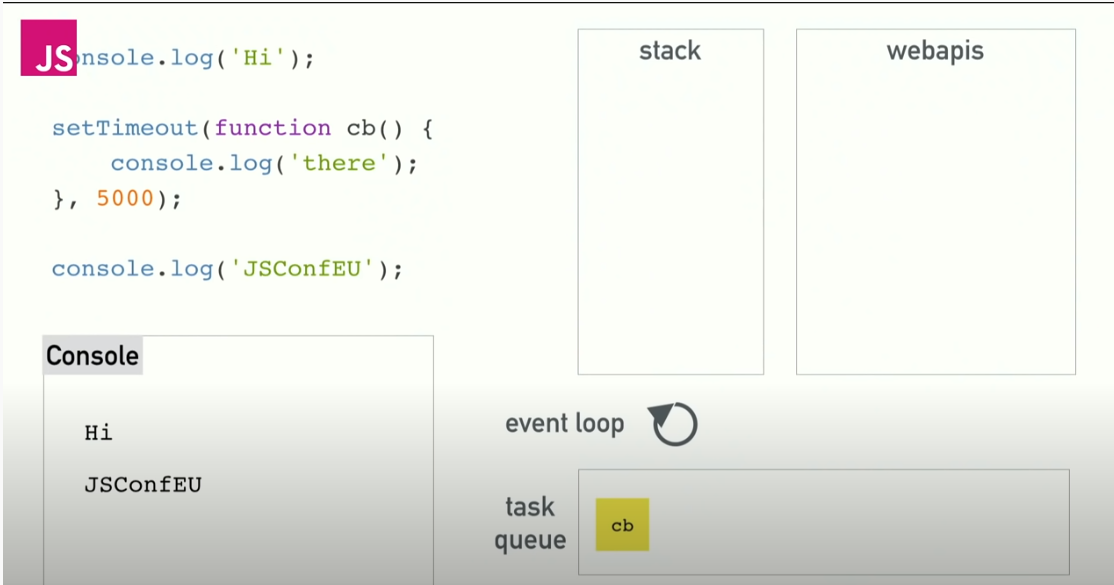
When is make a asynronus task like settimout or AJAX these are the api provided by browser so it executes on browser. With this we send a call back to be executed by js

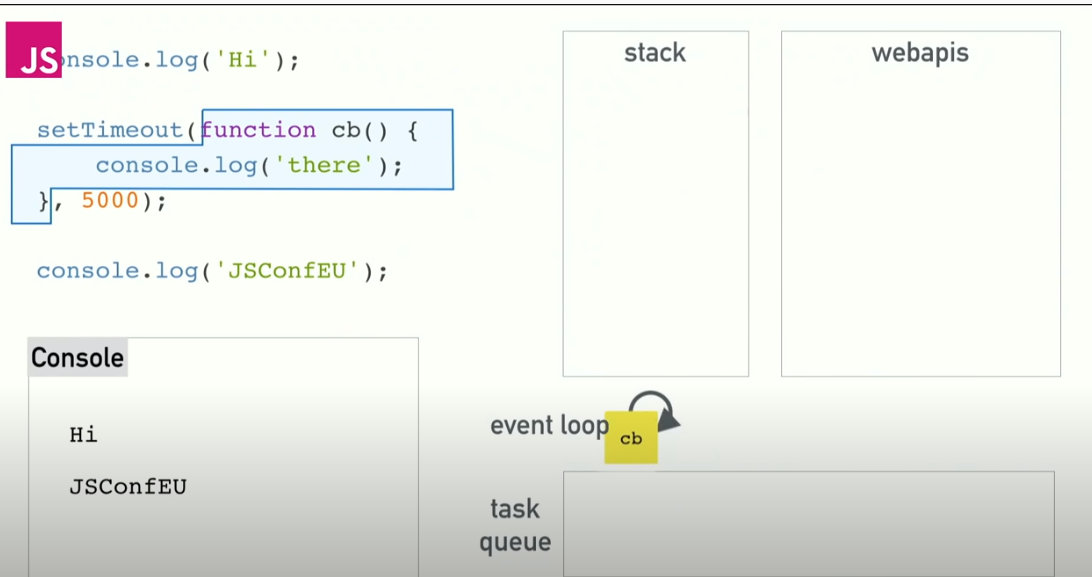


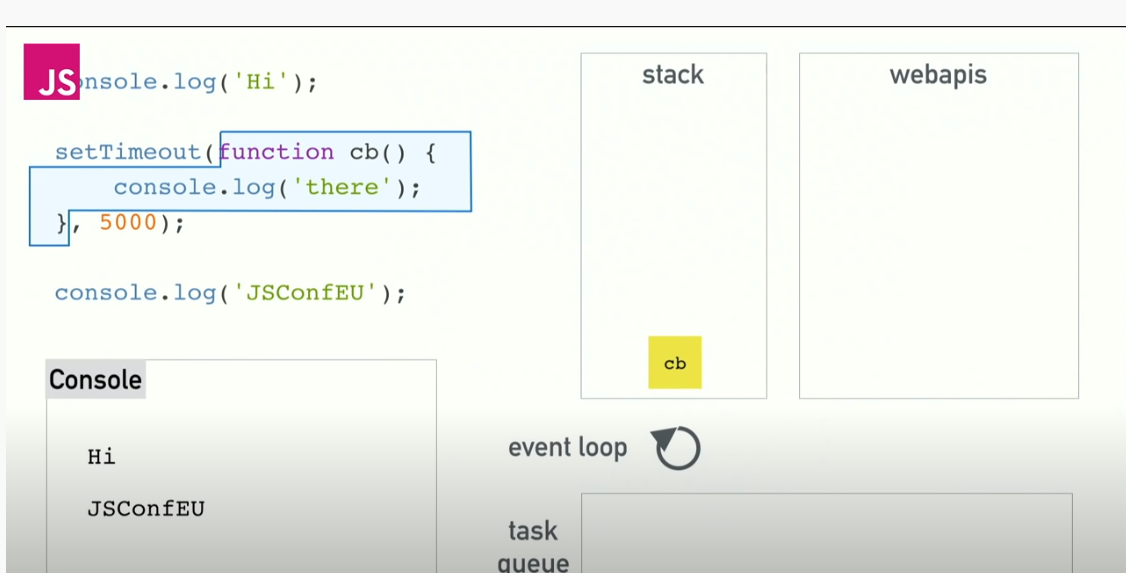


When task in webapi is done the call back is executed by js but to remember this call back will not intervene the things executing in callstack it will wait for call stack to be empty than it will execute

Not we all the task finished by webapis are maintened in **task queue and** task of event loop is to watch for call stack once call stack is empty it will move task from task queue to call stack



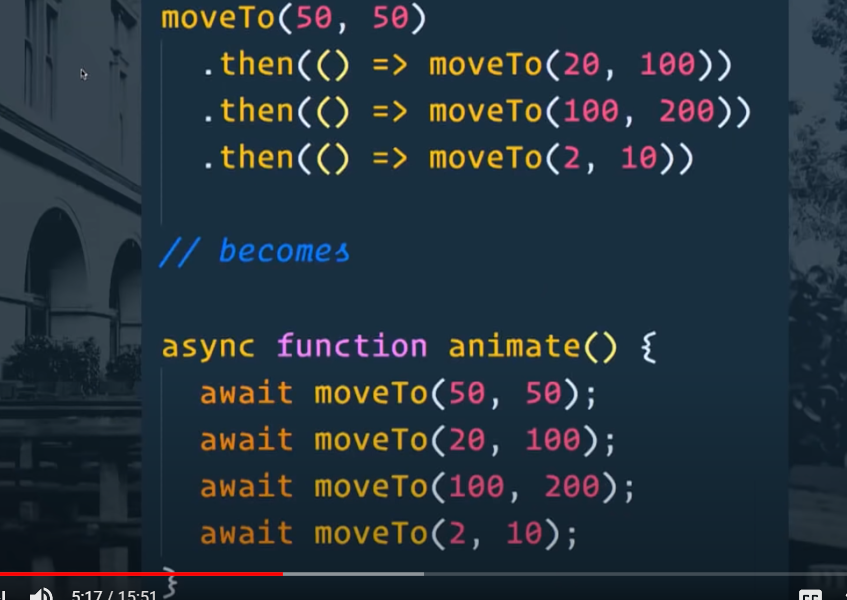




A **higher order function** is a function that takes a function as an argument, or returns a function

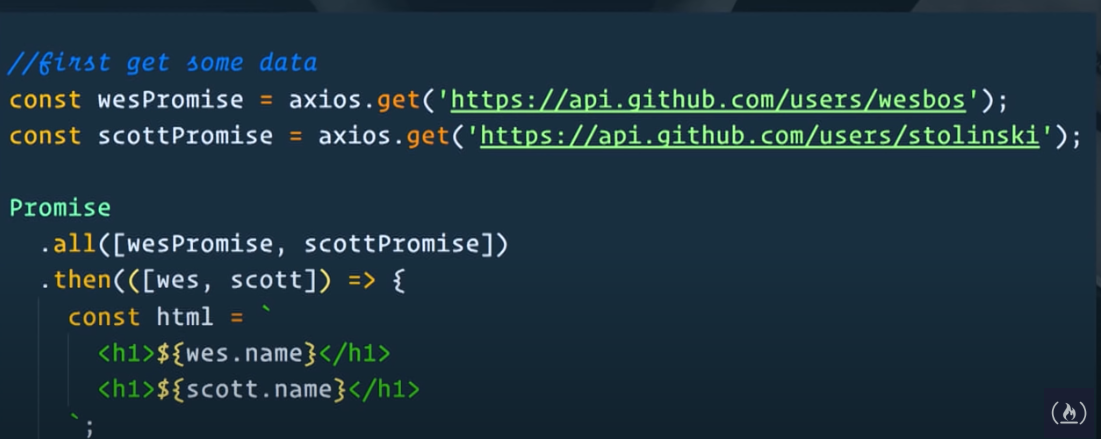
**Async await-** Async await just makes easier to write asynchronous functions





**Async is** keyword that tell this function executes ascynchronously

**Await** it is like .then but it can be used only inside async function, and this keyword tells js to not execute next line unless this line is executed

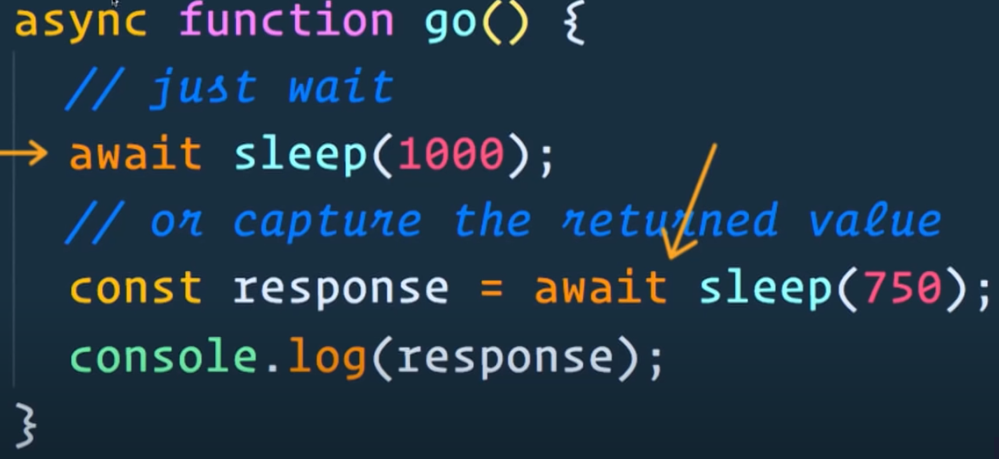


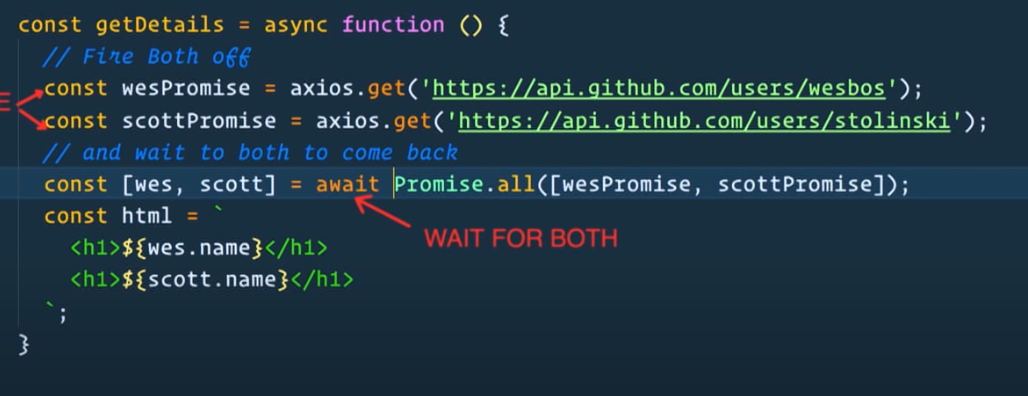
Steps to write async await

1. First mark function as async



1. Inside async function just wait by writin await keyword





**Prototype**

When we create a constructor function we can add method and values to it using prototype and those method and function will have only one copy

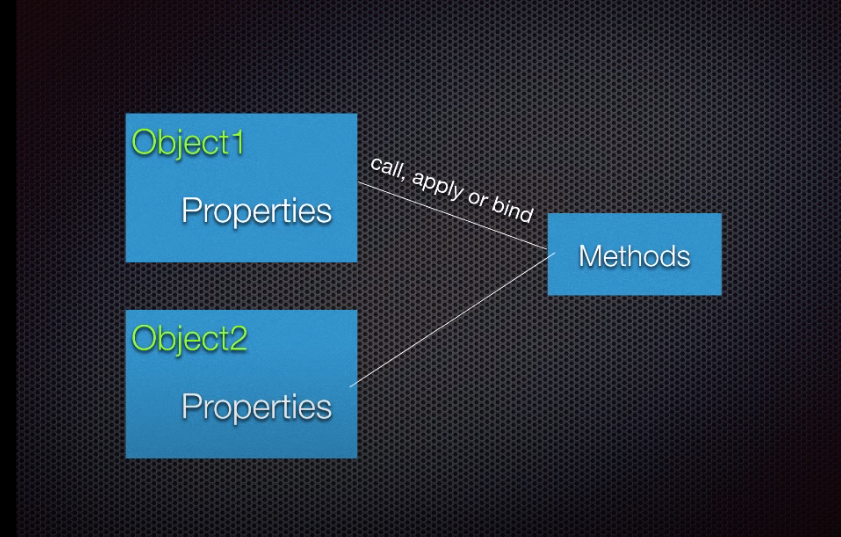


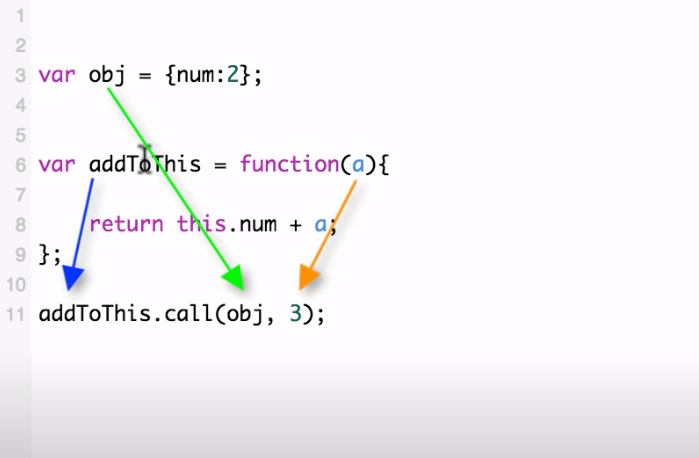
**Prototype inheritance:**

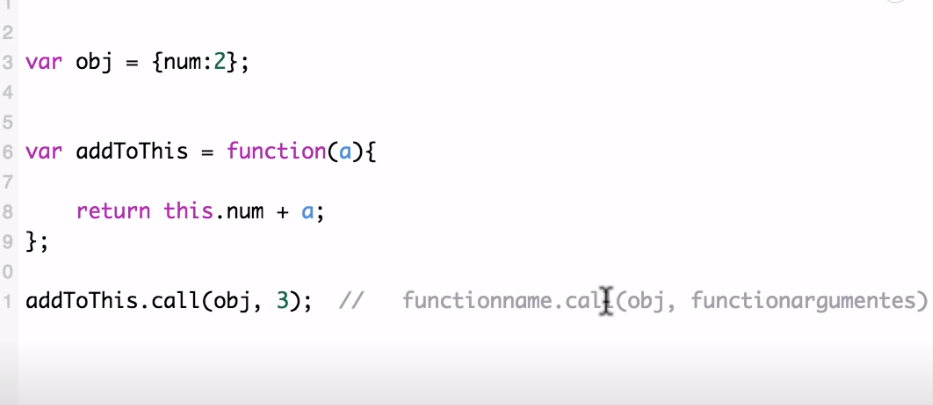
By default every function in js has property called prototype and we can add method and property to the function using prototype, when we create abjects from this function it will inherit this property and methods that is defined using prototype.

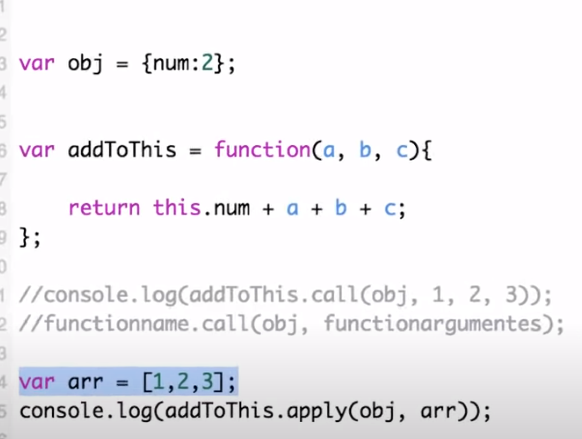
Note : Object will not have it own copy of prototype method it will only inherit

**Call, bind and apply**

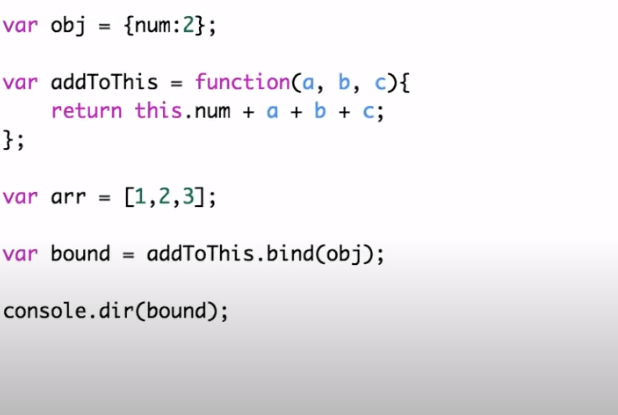




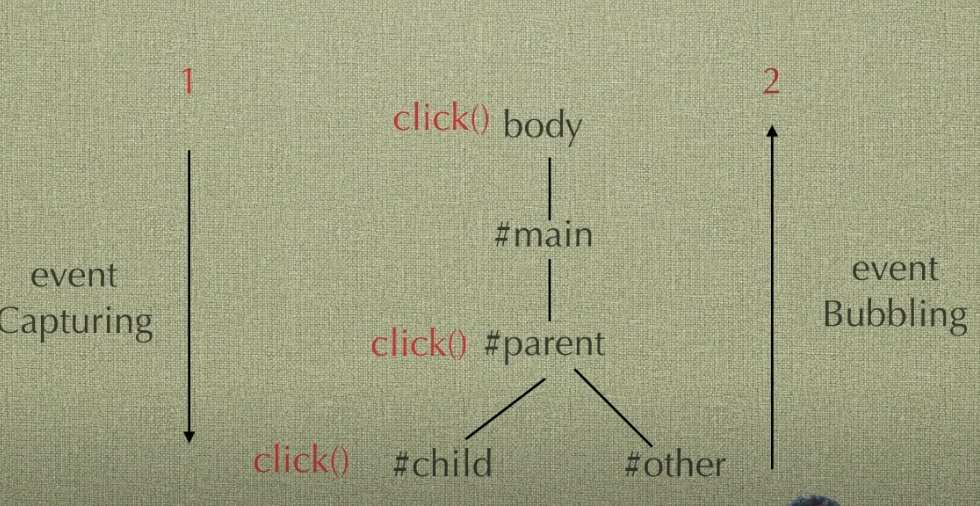


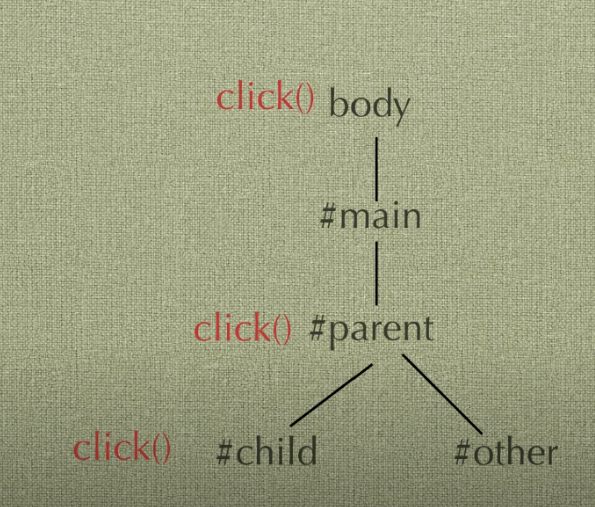


**Apply is same as calloOnly difference between call and apply in apply we can pass array of arguments**



**Event bubbling:** whenever we have event in page it goes from lowest element to top, when we click on some element menas click happened on its parent also till body so all click event will be called





The **event propagation** mode determines in which order the elements receive the **event**. Bubbling or capturing

<https://stackoverflow.com/questions/22939130/when-should-i-use-arrow-functions-in-ecmascript-6>

**Web worker** is a mechanism to build comminucation between page and worker. Javascript is single threaded. Web workers are simple means for running web content to run scripts in background threads.

Worker thread runs in background without interfering UI

### Spawning a dedicated worker

Creating a new worker is simple. All you need to do is call the [Worker()](https://developer.mozilla.org/en-US/docs/Web/API/Worker/Worker) constructor, specifying the URI of a script to execute in the worker thread ([main.js](https://github.com/mdn/simple-web-worker/blob/gh-pages/main.js)):

var myWorker = new Worker('worker.js');

The magic of workers happens via the [postMessage()](https://developer.mozilla.org/en-US/docs/Web/API/Worker/postMessage" \o "The postMessage() method of the Worker interface sends a message to the worker's inner scope. This accepts a single parameter, which is the data to send to the worker. The data may be any value or JavaScript object handled by the structured clone algorithm, which includes cyclical references.) method and the [onmessage](https://developer.mozilla.org/en-US/docs/Web/API/Worker/onmessage" \o "The onmessage property of the Worker interface represents an EventHandler, that is a function to be called when the message event occurs. These events are of type MessageEvent and will be called when the worker's parent receives a message (i.e. from the DedicatedWorkerGlobalScope.postMessage method).) event handler. When you want to send a message to the worker, you post messages to it like this ([main.js](https://github.com/mdn/simple-web-worker/blob/gh-pages/main.js)):

first.onchange = function() {

myWorker.postMessage([first.value,second.value]);

console.log('Message posted to worker');

}

In the worker, we can respond when the message is received by writing an event handler block like this ([worker.js](https://github.com/mdn/simple-web-worker/blob/gh-pages/worker.js)):

onmessage = function(e) {

console.log('Message received from main script');

var workerResult = 'Result: ' + (e.data[0] \* e.data[1]);

console.log('Posting message back to main script');

postMessage(workerResult);

}

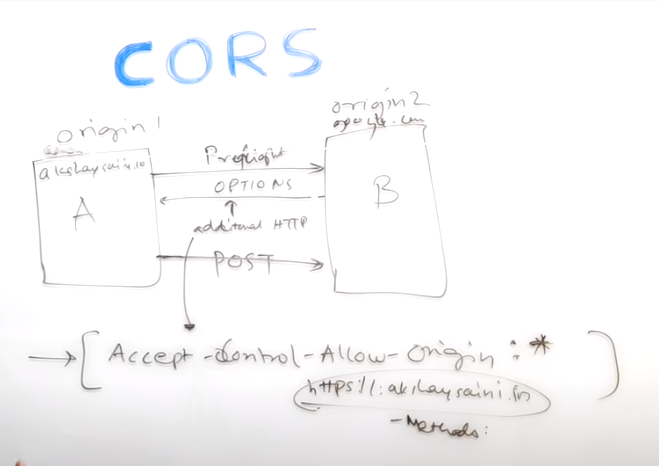
**cors error**

**Cross origin resource sharing:**

When we make a a call from one origin(domain) to another origin browser first make a options call which checks that whether another domain has enabled cors or not if not than request is not allowed .

Cors policy is enabled than final call is completed successfully

Does all call need pre flight request – No not all calls need it browser decides it based on some mechanism



**Throtling and debouncing**

Both are the ways to restrict expensive operations to happen frequently, in Debouncing we follow the approach when we call a function on key press only if there is gap of some time on key press.

throttling is the way when call function after certain fixed time

**async and defer attribute in script tag**

if none if the attribute is present than as soon as script encountered it is loaded and executed and during that tim html parsing stops

async – in this case script and html are loaded parallelly and during execution of script html is parsing is paused

defer script tag is loaded and executed once html parsing is done

**function currying –** when a function return a function, we can use the returned function for specific task based on outer function variables

e.g function add(a){

return function addInner(b){

return a+b

}

}

Var addToThree = add(3);

A **higher order function** is a function that takes a function as an argument, or returns a function.

Difference between JS and TS

TS is object oriented programming language whereas js is scripting language

Ts is superset of JS it basicall is JS with all es6 feature plus more imporvements

TS is strongly typed we cannot change type of variable on fly

Difference between ES6 and ES5

Class syntax

Let, const

Fat arrow function

Rest and spread

Difference between compiler and interpretor

Compiler takes a code converts it into another program(usually machine language) completely and than executes code

Whereas interpretor takes the code translates one line of code run it than translates next line and run it

**What is inversion of control and dependency injection** -

Suppose we have library like underscore and we call a function of library for any help we need, but in framework like angular we write a function and framework calls that function, this is achieved by design pattern called IOC, in this case flow of control is fully managed by framework,

and we can have dependency injection Constructor injection. In this case, the dependency is passed as a parameter in the constructor method.

**Delegates and events**

Instead of assign an onclick handler to each <td> (can be many) – we’ll setup the “catch-all” handler on <table> element.