Js is a synchronous single threded language, it has a single call stack.and it can do only one thing at a time.

This callstack is present inside js engine.and every code of js will get executed at this place in callstack.

Eg:

// Js is a synchronous single threded language, it has a single call stack.and it can do only one thing at a time.

// This callstack is present inside js engine.and every code of js will get executed at this place in callstack.

// Eg:

//suppose we want to execute the below piece of code

// function a() {

//   console.log("a");

// }

// a();

// console.log("End");

// whenever any js code runs, a global execution context gets created , and then this global econtext will get pushed into the callstack.

//now in thsi gecontext , this whole will run line by line.

//

//go to first line , it is function def of a(), so a will get memory and

this func def gets stored into it.

//next we have function invocation a() -  then a execution context gets created for funtion a(), to execute the code of this function.

//this a() again gets pushed into callstack

//now code of a() function gets executed line by line.- that means it will print a to the console.

//so at present - callstack has => GEC,a()

//console has => a

//then it goes to ending parenthesis of function } -it sees that there is nothing more to execute inside a() fucntion.

//so a() will comes out of callstack.

// now the control moves to line - console.log("End");

// and it prints 'end' to console.

//after this there is nothing more to print to console-  so  our GEC also gets out of callstack

//so callstack becomes empty of executing whole program.

//so main task of callstack is to execute wtever comes intoit.

//it does not wait for anything, if there is something into it, it executes directly without waiting

//this is how it works

//but suppose we wants to wait, suppose we have some code to run after 5000 ms.

//can we do that? no we cannot do that,wt ever comes inside callstack will automatically quickly executed

//if u say to execute after 5 seconds, it cant. as it has no timer

//but suppose if we need to keep track of time and to execute some piece of code after some delay, we need some extra super powers, the super powers of timers,

// how to get that powers lets see

// this callstack wil be present in js engine.

// and  this js engine will be inside a browser

//so browser has a js engine inside it, and inside js engine, we have callstack and inside callstack, our pgm runs

//browser also has some local storage inside it to store something

//u can also have a timer inside browser

//u can also have url - https:////

//so browser has  the powers to communicate with external world.

//like suppose we have netflix server, the browser can request data from it and can recieve the data.

// and after recieving data it can display it on browser window.

//it even has many different access - bluetooth, geolocation

//suppose the code running inside callstack need access to these powers of browsers

//so js engine should have some way to access those powers of browsers

//to access all those powers of browsers, we can use web apis

//let see  some web apis  - settimeout,dom apis,fetch(),localstorage,console,location

//setTimeout() - it is not a part of js, the DOM apis is  not a js ,fetch(),localstorage,console,location these all are also not a part of js.

//these are basically some kind of super powers which browser has.

//this browser gives access to js callstack /js engine

//to use timer of browser - it gives access to settimeout()

// to access html dom tree- the browser gives access to dom tree using dom api

//fetch() help us to make connection to other servers /external servers

//and we get it inside callstack bcoz of the global object

//wt is  global obj- it is small window

//browsers gives facility to js engine to use all these powers by using a keyword window.

//to access timer power-  window.settimeout() similarly all webapis we can use along with window keyword

//if u do window.localstorage in ur js code, it gives access to localstorage of js

//similarly , if u do window.console.log() - it allow u to console something on window

//this is how u can access browser powers

//similarly we have lot more web apis

// as window is global obj, instead of window.function we can directly access , as it is in global scope

//browser wraps all these apis into  a global objb - window and gives access of this window to this js engine

//now u can use all these funtions in js code

// eg: practical

//try to execute the below piece of code

console.log("Start");

setTimeout(function cb() {

  console.log("callback");

}, 5000);

console.log("End");

//lets see how it works using the web api powers - settimeout,dom apis,fetch,console

//as we know, at line 1 of this execution, a global econtext gets created and pushed into callstack

//now whole code runs line by line

//first line - console.log("start") ; will use console api actually and  it calls internally this api and it print it into console.

//so we can access this console and print just because we have console api

//this console is plugged to our js code( which is executing in callsstack global execution context) through window.

// so presently callstack has - GEC

//console has - start

//now go to next line,settimeout() -  it will go and call settimeout api and it gives  access to timer of browser ,it takes some callback func and some delay

//so when u pass the callback func to settimeout- it basically registers the callback, and we passed  delay of 5000ms, so it starts 5000 ms inside the timer

//and js moves to next line - sees console.log("End ") - and console as 'end'

//