**EXERCISE:**

1. **EXPLAIN THE FOLLOWING :-**
2. **EXECUTION CONTEXT:**

We know that whenever we run javascript code on a browser,it uses its javascript engine (for eg V8 in Google chrome) to run that code and display us the ouput based on it.

But then question arises how is the browser exactly running that code ???

So basically to run,evaluate the javascript code we need an environment , and this environment is known as EXECUTION CONTEXT .

Consider this a container in which we evalutate and execute our javascript code.

There are 2 types of execution context :

**GLOBAL EXECUTION CONTEXT :-**

This is the default execution context created by the javascript engine before it starts to execute any code. It includes variables function that is not inside any function.

A new execution context is just like any other execution context , except that it gets created by default.

A new execution (**FUNCTION EXECUTION CONTEXT**) context gets created every time a function is executed or called.

Since every function call gets its own FEC, there can be more than one FEC in the run-time of a script.

1. **CALL STACK (EXECUTION STACK) :**

Execution stack , also known as calling stack is a stack with a LIFO (LAST IN , FIRST OUT) structure, which is used to store all the execution context created during the code of execution , or,

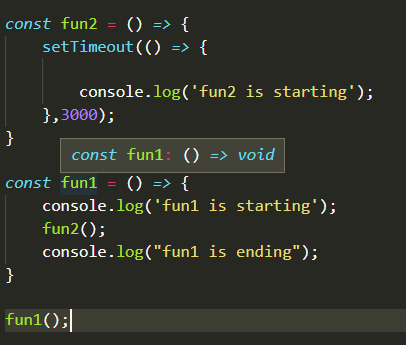
The call stack is used by JavaScript to keep track of multiple function calls.

We use call stack for memorizing which function is running right now.

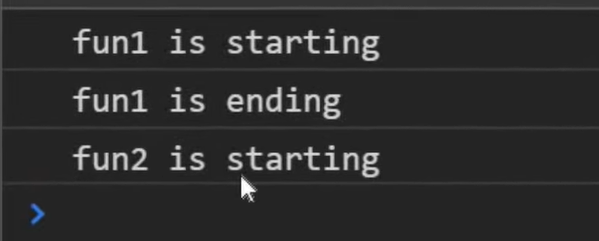
1. **EVENT LOOP :**

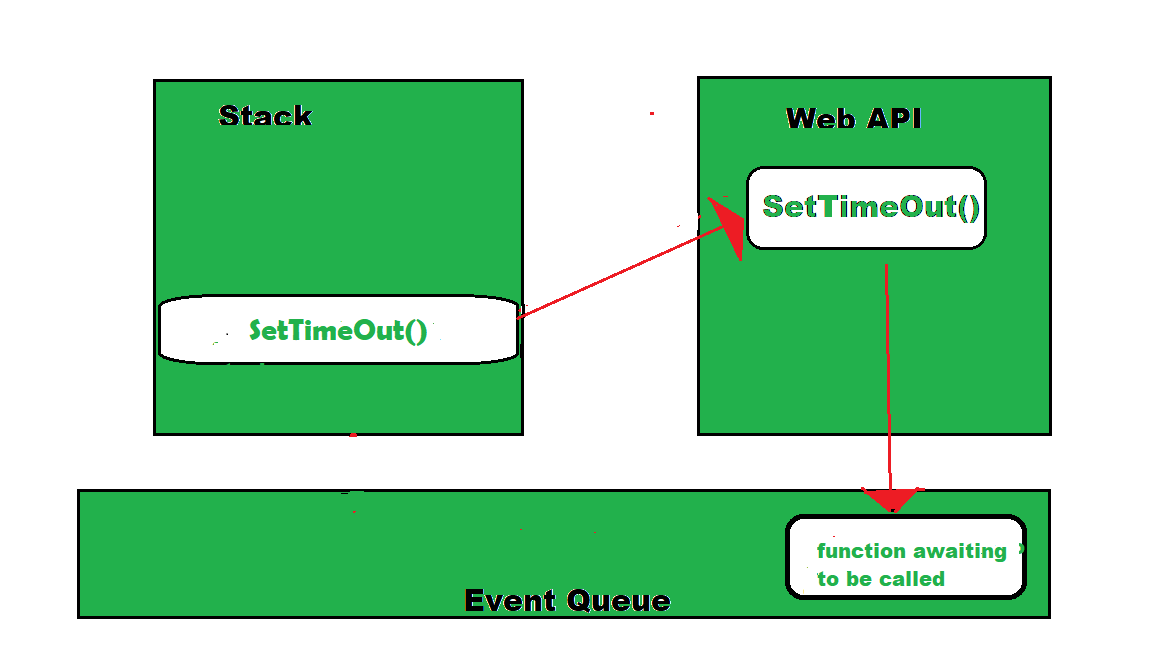
An event loop is something that pulls stuff out of the queue and places it onto the function execution stack whenever the function stack becomes empty.

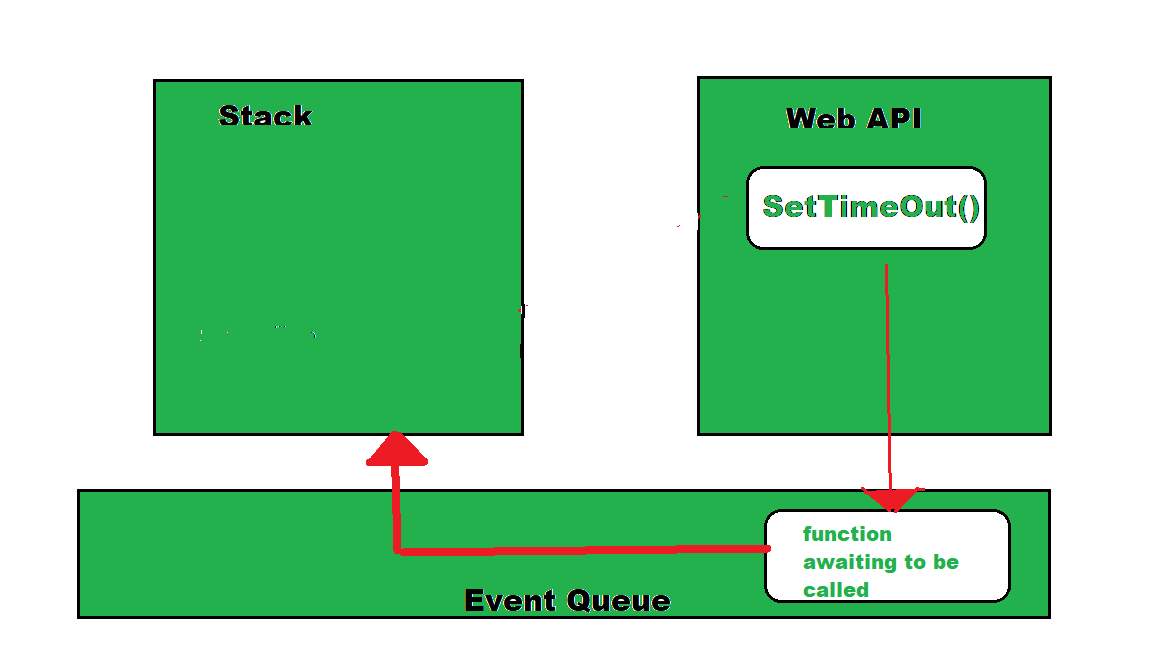
The event loop is the secret by which JavaScript gives us an illusion of being multithreaded even though it is single-threaded.

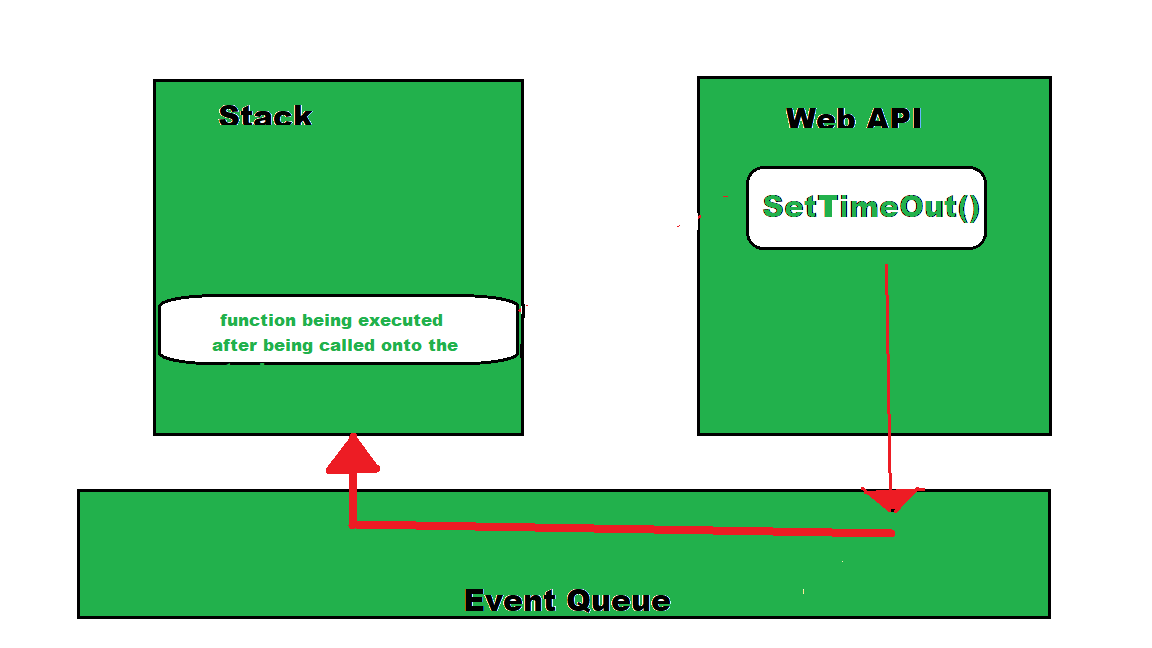


**OUTPUT:**









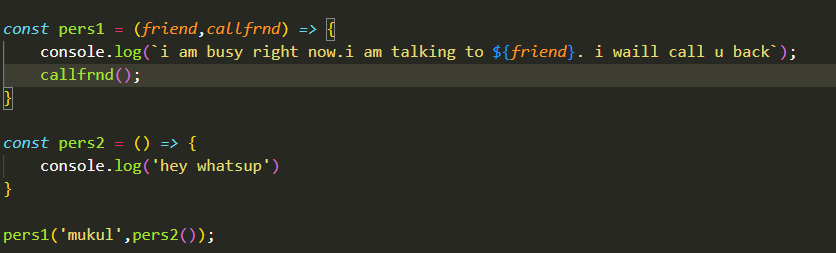
1. **WHAT ARE CALLBACK FUNCTIONS ? WHAT IS CALLBACK HELL ?**

Callback functions:

Any function that is passed as an argument is called callback function.

A callback function that is to be executed after another function has finished executing – hence the name ‘call back’.

Callbacks are a way to make sure certain code doesn’t execute until other code has already finished executing.



**Output :**

I am busy right now.i am talking to Mukul.i will call u back

hey whtsup

**CALLBACK HELL :**

This is a big issue caused by coding with complex nested callbacks. Here, each and every callback takes an argument that is a result of the previous callbacks. In this manner, The code structure looks like a pyramid, making it difficult to read and maintain.

**There are four easy ways to manage callback hell:**

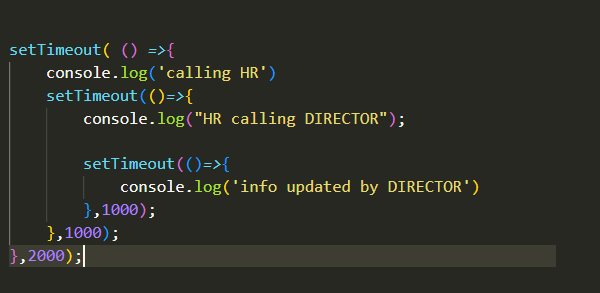
Write comments.

Split functions into smaller functions.

Using Promises.

Using Async/await.

For eg :



Nested callbacks leading to callback hell.

1. **What are promises, async/await? Explain with examples.**

**PROMISES:**

**Promises** are used to handle asynchronous operations in JavaScript. They are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.

A promise can be created using promise constructor

Syntax:

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

//do something

});

**FOR EXAMPLE:**

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

   const x = "geeksforgeeks";

   const y = "geeksforgeeks"

    if(x === y) {

     resolve();

   } else {

     reject();

   }

});

promise.

     then(function () {

         console.log('Success, You are a GEEK');

     }).

     catch(function () {

         console.log('Some error has occurred');

     });

**OUTPUT:**

Success,You are a GEEK

**ASYNC:**

It simply allows us to write promises based code as if it was synchronous and it checks that we are not breaking the execution thread. It operates asynchronously via the event- loop. Async functions will always return a value. It makes sure that a promise is returned and if it is not returned then javascript automatically wraps it in a promise which is resolved with its value.

EXAMPLE:

const getData = async() => {

    var data = "Hello World";

return data;

}

getData().then(data => console.log(data));

**OUTPUT:**

Hello World

**AWAIT:**

Await function is used to wait for the promise. It could be used within the async block only. It makes the code wait until the promise returns a result. It only makes the async block wait.

EXAMPLE:

|  |
| --- |
| const getData = async() => {      var y = await "Hello World";      console.log(y);  }    console.log(1);  getData();  console.log(2); |

**OUTPUT:**

1

2

Hello World