**JavaScript Callbacks & Promises –**

[Ishini Kiridena](https://medium.com/@ishini.kiridena?source=post_page-----6ae6f5a5d236--------------------------------) May 14, 2022

JS is asynchronous, means that all I/O operations are implemented to be asynchronous.

This happens because JS is a single threaded language, and if an IO operation blocks a thread then JS will not be performing accurately.

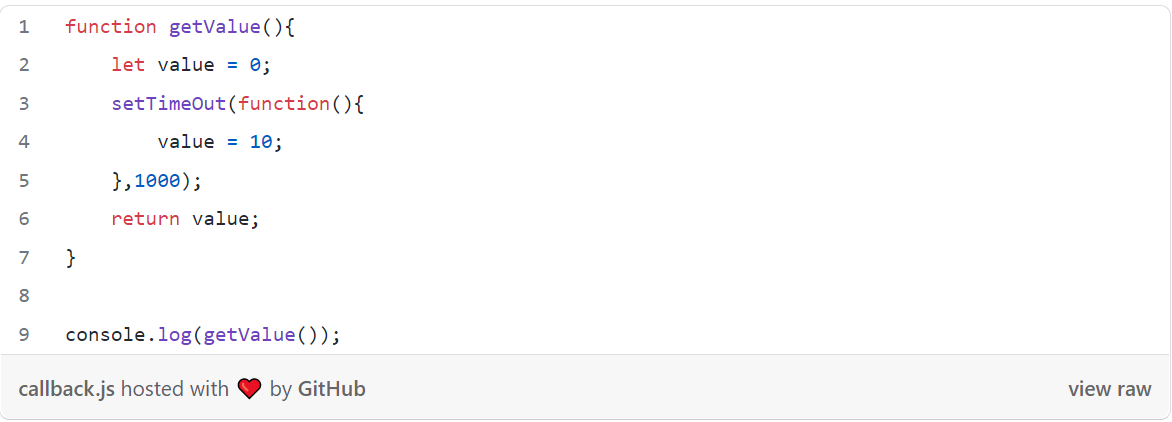
But still there are cases that we need a synchronous behavior in JS as well, where there is a need for a flow of execution.

For this we use **Callbacks**and **Promises**.

**Callbacks**

* Callbacks can be considered as functions in a way.
* This is being passed to an async task, that will be executed on the completion
* When passing the callback as a parameter it should be always the last parameter
* There can be cases where there can be a long chain of callbacks within callbacks (nested callbacks), which we refer to as “**Callback Hell**”

Lets see an example:



This will result a “0”, which is the initial value of the “value”. Why?

setTimeOut is an asynchronous operation which will be changing the value from 0 to 10 only after the 1000 milliseconds has been passed. So when the console.log happens the value comes as 0.

How we can fix this? By adding a callback



Here within this call we have whatever we are planning to do to the value, in this case a simple console.log. So here we get the output as “10”, the reason is

function (value){ console.log(value);}

this whole portion is executed at the callback, hence we overcame the synchronization issue.

**Promises**

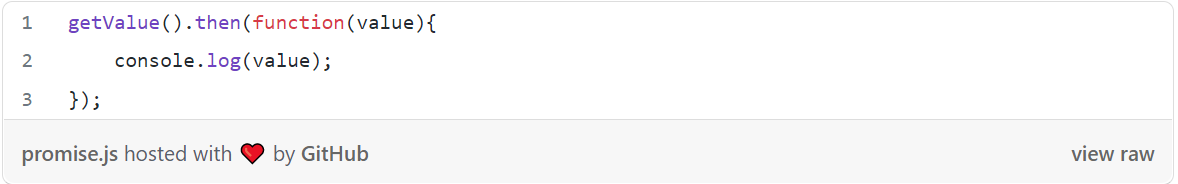
* Promise is an object that returned from an async task.
* Promises has the capability of dealing with async operations synchronously.
* Promises are introduced to overcome the issue of callback hell that we discussed previously.
* Another capability of the promise is that, it has functionalities to chain and handle complex async tasks in an accurate way.
* For chaining we can use “then” keyword.

Lets see how we can solve the issue that we faced with setTimeOut previously using promises,

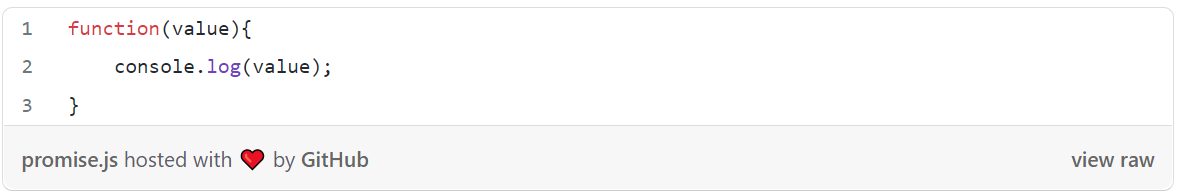
First we have to create a Promise object that will be returned in the getNewValue function



For a promise the **resolve and reject**are important variables, which provides the basic functionalities of the promise.



Here the output will be “10”.

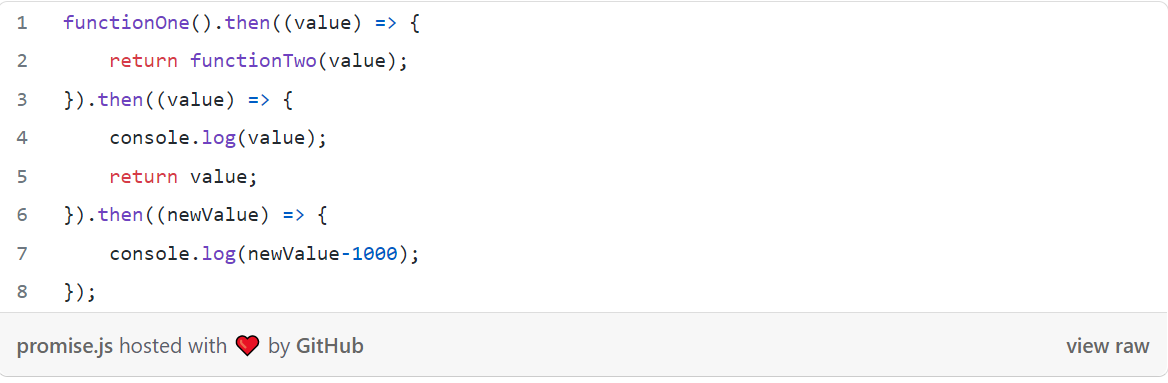


this part of the code enables you to whatever is preferred with the value, in this case its a simple console command. This part is executed by the “ resolve(value); “ code section in the getValue() method.

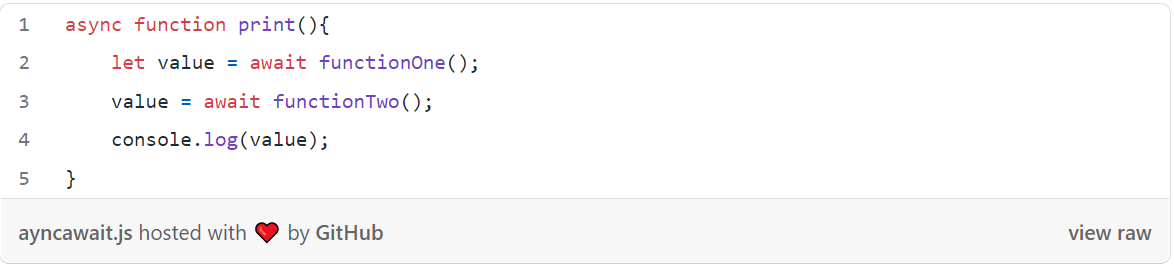
## Chaining in promises

Chaining can be done using the “then” keyword in JS and with this chaining we get the ability to perform many synchronous tasks within the same method.

Ex: Lets say that there is a function that returns some value, and using this value there are many operations are done synchronously.



Another option we have in chaining is using async/await:



Likewise we can use the same object to perform a chain in JS.