**Master asynchronous JavaScript NOW -** [Facundo Lavagnino](https://medium.com/@flavtech?source=post_page-----f911a10aac9a--------------------------------) Jul 28, 2022

I am going to explain in a simple way how to master asynchronous JavaScript. You will understand Callbacks, Promises and async-await.

Text

Description automatically generated

Pic from @tapasadhikary

**By default, JavaScript is synchronous**

Each function is executed sequentially line by line, this is known as the Call Stack, when a function finishes being executed, the JavaScript engine removes it from the execution stack.

That is, if we have

Text

Description automatically generated

The behavior that we can expect when executing the program is that function 1 is executed first and then function 2.

But sometimes we’ll want to get data from the server, or run functions with a delay, this takes a while, and we don’t want the JavaScript engine to stop executing sequential code, so you’ll want the code to run asynchronously

For example, we want some data from <https://api.example.com/users>, we know that the API will return a list of users, but this may take a while, milliseconds or seconds, and in that time, we don’t want to stop all the execution of our program.

**Asynchronous methods explained**

In JavaScript, we have some ways to execute asynchronous code.

We have native browser APIs that execute functions after a certain time like set Timeout, we have event handlers like click, mouse over, scroll, etc.  
We have Promises, a JavaScript object that allow asynchronous operations inside

Web APIs are based on callback functions. A callback function is a simple function that is executed when an asynchronous operation completes.  
Here is an example of how setTimeout works

Text

Description automatically generated

We have 3 functions, the behavior when executing the program would be the following:

The first synchronous function is executed, then the second, and finally the asynchronous function, since it takes a delay of 2 seconds, but the asynchronous function does not stop the execution of the following functions.

**How Promises works**

When we work with external APIs, these will return data to us with a certain delay by default, this data request to the server can either succeed or fail, but we do not know in advance the result until it ends, that is what promises are used for. We can build a promise with the native JavaScript promise constructor as follows

Text

Description automatically generated

The promise received two parameters, resolve and reject, you can define what to do when the promise succeeds by calling the resolve method or when an error occurs by calling the reject method.

When we request data from an external server, we will commonly use the native fetch API, this API requests the data from a URL and returns a promise, when the promise ends, we can obtain the result with .then or the error with .catch, here’s an example

Text

Description automatically generated

We use fetch to fetch data from the URL, in this case, to fetch a single task, this returns a promise, but as we’ve seen before, this takes some time to complete, then we fetch the result of that promise with .then , fetch will return an HTTP response but not the data we are looking for, to get the data we must use the .json() method, this is also a promise, so it will take a while to complete, we use .then again, and finally we will have the result completely asynchronously, without stopping the execution of the program.

Text

Description automatically generated

In this case we request the data of a user to a server with fetch, we have a function that deals with greeting the user, this function receives a name as a parameter, but we do not know the name yet, we must fetch the URL, obtain the user’s data, and calling the greet function with the user’s name on promise completion, this is usually what you do when working with asynchronous code and server calls.

**Async Await**

The arrival of ES6 brought incredibly useful things to JavaScript, one of them is the functions with the Async keyword, which execute asynchronous code inside, technically it is the same as promises, but it reads much better and helps to have a cleaner code and maintainable, I personally love it, and use it whenever I can.

Here is an example of how it works



We declare the function with the keyword Async, to indicate that we will execute asynchronous code inside, then we call the API with fetch placing the keyword await and save the result as a constant, we repeat the process with the conversion of the result to json, we can use destructuring of objects as in this case, and finally we call the greet function with the name received from the server. In this way the code is read a little better, it even seems synchronous

**Conclusion**

Asynchronous code is important in JavaScript, many functions depend on the result of other functions, most of the time we will need data from external APIs, this is only achieved with asynchronous code, so fully understanding this concept will make you a more complete developer And it will open up a lot of opportunities for you.

If the post was useful to you, leave me a clap and follow me, it helps me a lot. Thank you.