**ASINCRONISMO**

JavaScript sólo puede hacer una cosa a la vez, sin embargo; es capaz de delegar la ejecución de ciertas funciones a otros procesos. Este modelo de concurrencia se llama EventLoop.

JavaScript delega en el navegador ciertas tareas y les asocia funciones que deberán ser ejecutadas al ser completadas. Estas funciones se llaman callbacks, y una vez que el navegador ha regresado con la respuesta, el callback asociado pasa a la cola de tareas para ser ejecutado una vez que JavaScript haya terminado todas las instrucciones que están en la pila de ejecución.

Si se acumulan funciones en la cola de tareas y JavaScript se encuentra ejecutando procesos muy pesados, el EventLoop quedará bloqueado y esas funciones pudieran tardar demasiado en ejecutarse.

console.log("a")

setTimeout(

    function () {

        console.log("b")

    }, 1000)

console.log("c")

**CALLBACK.**

Un callback es una función que se pasa a otra función como un argumento. Esta función se invoca, después, dentro de la función externa para completar alguna acción.

Las consultas a APIs se pueden ejecutar con Jquery una librería de JavaScript.

Const API = "https://swapi.dev/api/"

const people\_URL = "people/#/"

const people = `${API}${people\_URL.replace("#", 14)}`

const opts = { crossDomain: true}

$.get(people, opts, function (persona){

  console.log(`Hola soy ${persona.name}`)

})

Para este ejemplo se está usando la API **swapi,** la cual posee toda la información de la saga de películas de Star Wars.

Se pueden generar varios callbacks en paralelo, pero el orden en que estos lleguen va a ser completamente inesperado.

obtenerPersonaje (1)

obtenerPersonaje (2)

obtenerPersonaje (3)

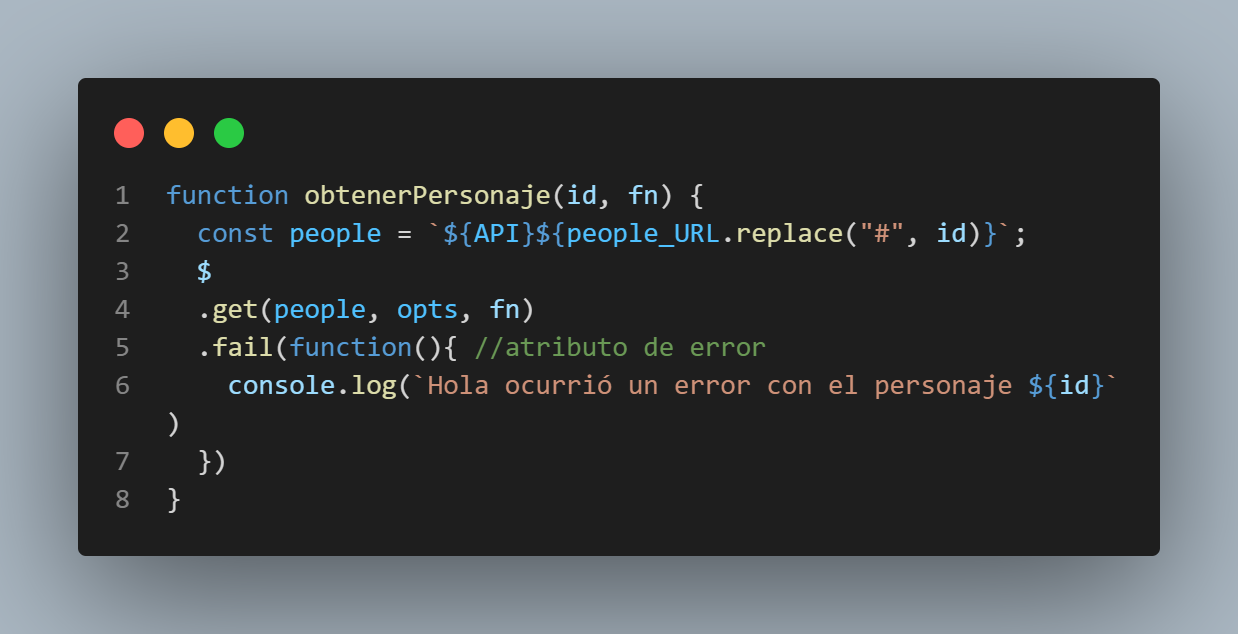
Para solucionar eso se utiliza un método y es llamar el siguiente callback como una función del anterior, con esto solo se ejecuta una vez el callback anterior ya se haya ejecutado. 

Sin embargo, este método es conocido como el Hell de los callbacks debido al desorden que genera, lo difícil que se vuelve solucionar algún problema y el hecho de que el código cada vez se vuelve mas horizontal y no vertical.

Un método más ordenado visualmente seria así:



En el caso que haya un corte en el internet u ocurra algún otro error:



**PROMESAS.**

Las promesas son valores que aún no conocemos, éstas tienen 3 estados:

* **Pending:** es el estado inicial, el estado en el que arranca la promesa.
* **Fullfilled:** es cuando la promesa se resuelve exitosamente.
* **Rejected:** es cuando ocurre algún error.

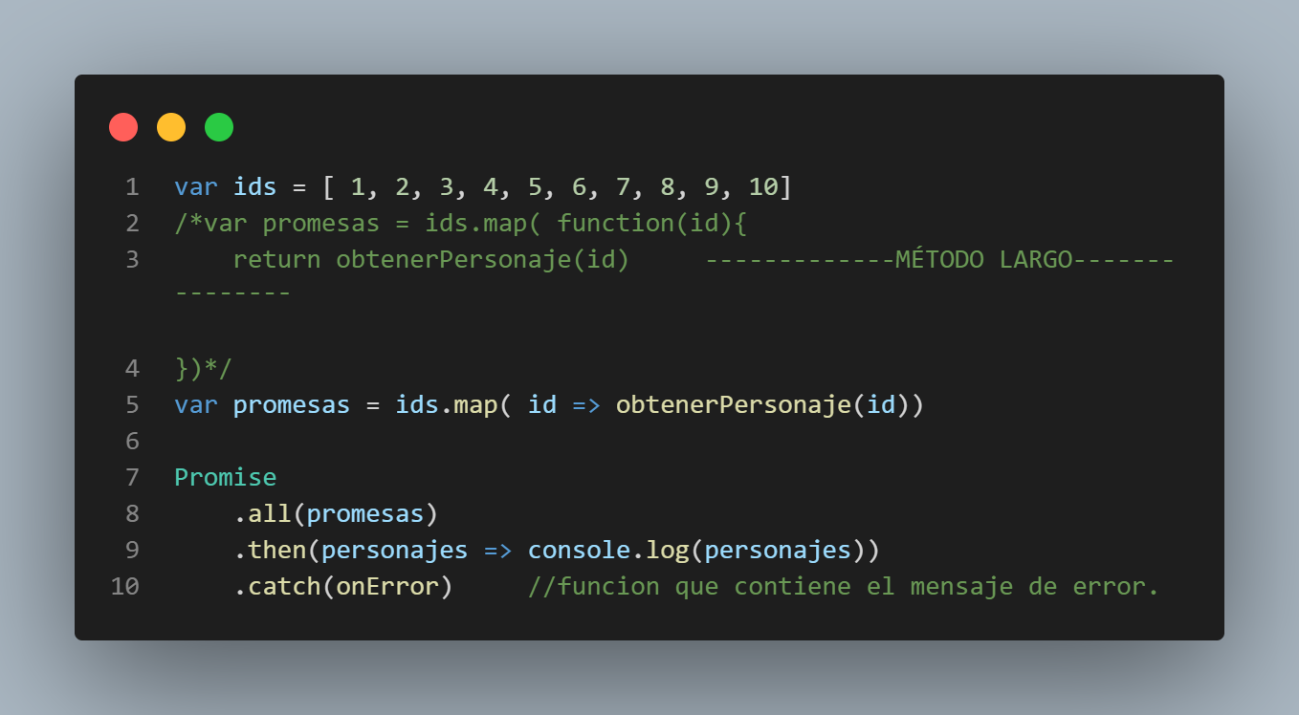


Para encadenar promesas basta con retornar el llamado de la función constructora con un nuevo **id:**



Cabe recordar que de haber un error en algún llamado el **catch** con el error lo tomara, solo con haberlo escrito una sola vez.

MULTIPLES PROMESAS EN PARALELO:



Se crea un array que contendrá las promesas y luego se llama el atributo **all** que lleva como parámetro el array que contiene las promesas.

**ASYNC-AWAIT**

Async-await es la manera más simple y clara de realizar tareas asíncronas. Await detiene la ejecución del programa hasta que todas las promesas sean resueltas. Para poder utilizar esta forma, hay que colocar async antes de la definición de la función, y encerrar el llamado a Promises.all() dentro de un bloque try … catch.

