**JavaScript Callbacks, Promises, Async Await in a nutshell -** [Rasanga Lakshith](https://medium.com/@rasangalakshith38?source=post_page-----79ba69b2621c--------------------------------) Mar 16, 2022

JavaScript’s asynchronous nature does not make us comfortable at all the time. I know that statement makes you confused if you have not experienced such things yet. Let me give you an example to figure it out.

Text

Description automatically generated

I created a **cars**array to save the car details. Then after I implemented two functions to get all car details (**getCars()**) and to insert a car details(**insertCar()**) to the **cars** array. In line number twenty, i am adding a new car to the cars array and then I called the **getCars()**method to display all the cars now in the **cars**array. what will be the output? here it is:

A picture containing shape

Description automatically generated

Now you might think that why the Audi car is not there… easy peasy lemon squeezy, let me explain it. I have used **setTimeout**in both **getCars()**and **insertCar()**methods. in **getCars()**,cars will be displayed after one second while the **insertCar()**method insert a car to the array after two seconds. So that JavaScript interpreter wont wait until that two seconds to be over because of its asynchronous nature. So how can we print the Audi then? Don’t worry, we have three superheros to solve that matter. They are Callbacks, Promises, and Async Await. let’s solve the above problem using one by one.

**1. Callbacks**

We need to modify our **insertCar()** method. That is because the delay exists there. let’s see the modified code and let me explain every step.

A picture containing graphical user interface

Description automatically generated

I have inserted two parameters to the **insertCar()** method which is **car**and **callback**. **car**is the same car object in the previous code snippet. The all new **callback**is a reference to our callback function. Then in line number seventeen, I have mentioned when the callback() function have to be executed. Simple right? After pushing the new car object to the car array, the callback function should execute. In line number twenty-one, I am calling **insertCar()**method by passing car object and **getCars**() method’s reference as parameters.

This **getCars**() method is our callback function. We know all the functions in JavaScript behave like objects. That is why I pass **getCars** without parenthesis. Now let’s look at the output;

Shape

Description automatically generated with low confidence

Everything works fine. That is how we use callbacks to prevent from above mentioned problem.

**2. Promises**

A promise contains two parameters. Which are **resolve**and **reject.**The promise can be either success or broken. If it is a success one we’ ll use **resolve**. If it is a broken one we’ll use **reject**. Now let’s move on to our code implementation.

A picture containing graphical user interface

Description automatically generated

As I mentioned above, promise can be a success one or a broken one. That is why I used a boolean constant in line number seventeen to demonstrate that. If the promise is a success one, which means no error, new car object will be pushed to the **cars**array.

If the promise is a broken one, then it displays “**Something went wrong**” message. After that, I have called **insertCar()** method in line number twenty-eight. There I have passed the car object along with **.then**keyword and inside parenthesis I mentioned the **getCars()** method. What is the meaning of that syntax? Easy as pie. It means, insert the car object to the **cars**array and then after execute the **getCars**method. We make the code clean and clear using promises. It is more readable than callbacks.

**3. Async Await**

This is my favorite one. And it is more readable and easy to understand. Let’s go and see the code first.

Graphical user interface, text, application

Description automatically generated

The magic is in the new **init()** function. Inside that function we declare which method should execute first and wait for the results. Here I declared that **insertCar()** method should be executed first and interpreter must wait until **insertCar()**finishes its execution. Then only **getCars()**will be executed.