**Async JS Crash Course – Callbacks, Promises, Async Await**

**Traversy Media:** [**https://www.youtube.com/watch?v=PoRJizFvM7s&list=WL&index=85**](https://www.youtube.com/watch?v=PoRJizFvM7s&list=WL&index=85)

**Asynchronous Javascript.**

**Ways to deal with asynchronous data. You don’t want to wait until some task is completed to continue your program(that would be synchronous programming), you want it to continue while it is happening asynchronously. It is relevant to Javascript because we are often making requests to servers that may take time to get data back.**

**This is where callbacks come in. Callbacks where used until ES6 where promises where introduced to the language. Promises give a more elegant way to handle asynchronous data. Async Await was introduced in ES7.**

**Callbacks. Essentially a function that is passed to another function as an argument.**

**We will mimic how it is to fetch data from a server, that could take a couple seconds.k**

**Const posts = [**

**{**

**title: ‘Post one’,**

**body: ‘This is post one’**

**},**

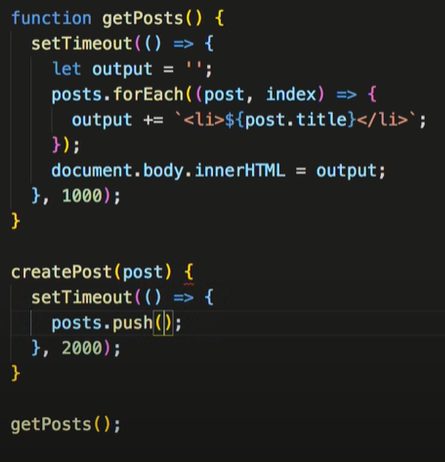
**{**

**title: ‘Post Two’,**

**body: ‘This is post two’**

**}**

**];**



(En este caso, createPost tardaría dos segundos y getPosts uno, entonces no se mostraría el ultimo post creado.)

**Function getPosts() {**

**setTimeOut( () => {**

**let output = ‘ ‘;**

**posts.forEach((post) => {**

**output+=`<li>${post.title}</li>`**

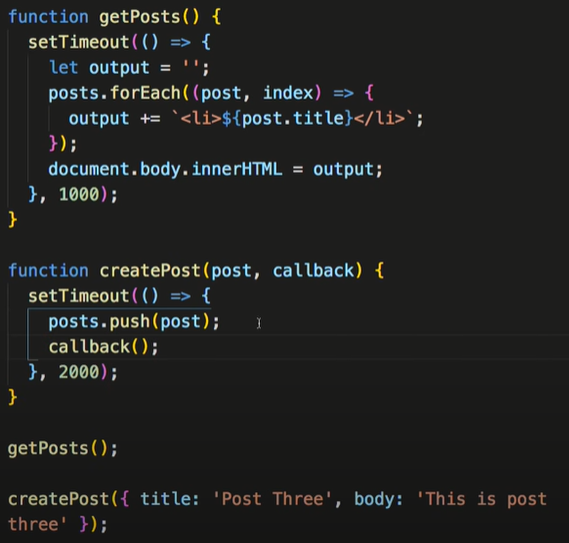
**} );**

**Document.body.innerHTML= output;**

**},2000) //Tardaria dos segundos en cargar.**

**}**

**getPosts();**



**function createPost(post) {**

**setTimeOut(() => {**

**posts.push(post);**

**}, 2000); // This will take two seconds, so this new post won’t show, because it will take longer than the getPosts.**

**}**

**createPost({title: ‘Post three’, body: ‘This is Post three’})**

**We should modificate the function so it takes a callback.**

**Function createPost(post, callback){**

**setTimeOut(()=>{**

**posts.push(post);**

**callback(); //Aquí aparecen las callbacks, para poder ejecutar el getPost después de que el post haya sido creado.**

**}, 2000)**

**}**

**createPost({title: ‘Post three’, body: ‘This is Post three’}, getPosts)**

**Promises.**

**Now we don’t run a callback. We return a promise. Axios uses promises, so we will be dealing with the response, we won’t need to create the promises.**



**function createPost(post){**

**Return new Promise((resolve, reject) =>{**

**setTimeOut(() => {**

**posts.push(post);**

**const error = false;**

**if(!error){**

**resolve()**

**} else {reject(‘error: Something went wrong’):**

**}, 2000)}**

**});**

**}**

**createPost({title: ‘Post three’, body: ‘This is Post three’}) This returns a promise.**

**.then(getPosts) //Once the promise resolves, it calls getPosts.**

**.catch(err => console.error(err));**

**Promise.all()**

**If you have a lot of different promises, you don’t wanna repeat ‘.then’ that many times.**

**Const promise1 = Promise.resolve(‘Hello World);**

**Const promise2 = new Promise ((resolve, reject) => {**

**setTimeOut(resolve, 2000, ‘GoodBye’);**

**});**

**Const promise3 = 10;**

**Const promise4 = fetch(‘${URL}’); //fetchAPI built in browsers. Also returns a promise.**

**Con fetch se necesitan dos .then, el primero para formatearlo a json.**

**.then(res => res.json());**

**Promise.all([promise1, promise2, promise3, promise4])**

**.then((values) => {console.log(values)} //Callback**



**Async/Await**

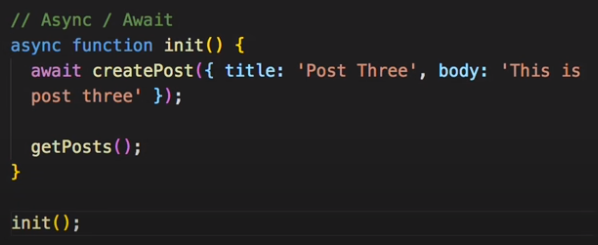
**More elegant way to handle promises. It is a way to handle responses, it is not a different way to write them. We need to have a function that is labeled asynchronous. Await waits for an asynchronous process to complete.**

**Async Function init() {**

**Await createPost({title: ‘Post three’, body: ‘This is post three’});**

**getPosts();**

**}**



**Async/Await with Fetch:**

**Async Function fetchUsers() {**

**Const res = await fetch({URL})**

**Const data = res.json();**

**Console.log(data);**

**}**

**fetchUsers();**

