Lecture 13: Async Await + Error Handling

Let’s understand something from what we’ve learnt first and then look at the optimization because of Async Await

Earlier versions of JS, we had (we still have) callback methods. And calling a callback from inside a callback was like getting into a callback hell. It looked something like this:

```
myFunctionOne(a, b, callbackFunctionOne(dataA){
        myFunctionTwo(dataA, callbackFunctionTwo(dataB){
                myFunctionThree(dataB, callbackFunctionThree(){
                        …..
                });
        });
});
```

This obviously was a difficult chunk of code to manage as it grew in size.

Promises, method chaining came to the rescue. And the syntax looked something like this:

```
//each function returns a promise
myFunctionOne(a, b)
        .then(myFunctionTwo(resultA){})
        .then(myFunctionThree(resultB){})
        .then…
        .then…
        .catch(error => {});
```

This looked so much more clean. But wait, here comes async/await. What does this look like, well it simply executes line by line, looking just like the synchronous code! Here’s the syntax:

```
// we do declare the function to be async if it has asynchronous queries which will run sequentially
let myFunctionParent = async function(){
        // what’s more! there’s just one try catch
        try{
                let resultA = await myFunctionOne();
                let resultB = await myFunctionTwo();
                …
                …
                …
                …

        }catch(err){
                console.log(err);
                return;
        }



}
```

Now, let’s get to the ‘official technical part’!

The **async function** declaration defines an **asynchronous function**, which returns an AsyncFunction object. An asynchronous function is a function which operates asynchronously via the event loop, using an implicit Promise to return its result.

SYNTAX

async function *name*([*param*[, *param*[, ... *param*]]]) {
  statements
}

Await

An async function can contain an await expression that pauses the execution of the async function and waits for the passed Promise's resolution, and then resumes the async function's execution and returns the resolved value.

Remember, the await keyword is only valid inside async functions. If you use it outside of an async function's body, you will get a SyntaxError.

Error handling

Simple try/catch can be used for handling errors when using async await.

Syntax

```
try {

  //  Block of code to try

}

catch(error) {

  //  Block of code to handle errors

}
```

Some more links to read

https://blog.hellojs.org/asynchronous-javascript-from-callback-hell-to-async-and-await-9b9ceb63c8e8

https://javascript.info/async-await

https://www.freecodecamp.org/news/how-to-master-async-await-with-this-real-world-example-19107e7558ad/