JavaScript Async Await – An Introduction

The Javascript async await statement provides a way to express asynchronous logic in your code in a sane and readable manner

In this article, you will learn how to master the Javascript async await statement. Before async await came along, the only thing we had was promises and callbacks. The async-await statement syntax is easier to understand, read, and implement in your code.

Async Function

The **async function** is the code we use to define an asynchronous function. The async function returns a Promise as its result. Async functions run in an order that's separate from other code in the event loop. An async function does not need to contain an await expression.

Await

An **await** expression causes an async function to pause until the passed promise has resolved before executing the rest of the code in a function. While the async function pauses, any calling function will continue to run. Await expressions only work inside an async.

```
function sleep (milliseconds) {
  return new Promise(resolve => setTimeout(resolve, milliseconds));
};

function goToSleep (milliseconds) {
  console.log('Going to bed now...');

  await sleep (milliseconds);

  console.log('Finished sleeping!');
};

goToSleep(3000); // Uncaught SyntaxError: await is only valid in async function
```

javascript-async-await-intro-1.js hosted with ♥ by GitHub view raw

```
function sleep (milliseconds) {
  return new Promise(resolve => setTimeout(resolve, milliseconds));
};

async function goToSleep (milliseconds) {
  console.log('Going to bed now...');

  await sleep (milliseconds);

  console.log('Finished sleeping!');
};

goToSleep(3000);
```

javascript-async-await-intro-2.js hosted with ♥ by GitHub view raw

```
async function getData (dataUrl) {
  const response = await fetch(dataUrl);
  return await response.json();
};


console.warn('about to fetch the json...');

const myJson = await getData('https://www.mocky.io/v2/5de9064331000023ac6b166d');

console.warn('here is the json:');
console.dir(myJson);
```

javascript-async-await-intro-3.js hosted with ♥ by GitHub view raw

Example 1

Example 1 demonstrates that we can only use the await expression inside an async function. Any attempts to use the await expression outside of an async function will throw an exception.
If you run the Example 1 code in your Javascript console, you will see that you get an error that states, “Uncaught SyntaxError: await is only valid in an async function.” The code will not run or produce any result other than the error.

Example 2

Example 2 demonstrates how to correct the error generated in Example 1. To fix the error, we need to change the standard function **goToSleep** into an async function. You can see by looking at the code in Example 2 that the only necessary change is to add the word **async** before the word **function** in our goToSleep function declaration.

If you run this new code in your Javascript console, you will see that it works and correctly pauses the goToSleep function for 3000ms without throwing an exception.

We can now see that in this example, the async function goToSleep first uses console.log to alert the user that we are going to bed now. Next, we tell the function goToSleep to **await** while we call the standard non-async function **sleep**, and wait for it to finish. Once the sleep function ends, goToSleep starts back up and uses consol.log again to alert the user that we finished sleeping.

Example 3

Example 3 gives us a perfect real-world example of how a Javascript Async Await expression is useful in programming. This example uses an asynchronous function along with the await expression to notify a user that it is about to fetch some data, fetches the data, then tells the user when completed.

If you break down the code, you can see it’s pretty straightforward.

1. The console uses its warning expression console.warn to let us know that it's about to fetch some json.

    a. Console.warn is the same as console.log but with a highlighted background to show importance.

    b. The warning lets us know that it might take some time to fetch the data. So this code is functioning almost like a loading screen.

2. With the warning issued, we call the async function **getData** and pass it the dataURL https://www.mocky.io/v2/5de9064331000023ac6b166d

    a. The first **await** expression pauses the function and waits for a response from the website to let us know we established a connection.

    b. Once connected, the second await expression pauses the function again until it can download the contents into the **myJson** constant.

3. With the async function getData finished, we use the consoles warning statement again to let the user know that the code is finished loading and is ready to present the json.

4. We use another console statement, console.dir, to present the json to the user.

    a. Console.dir is similar to console.log, but is for printing objects and properties instead of strings.

Conclusion

Now that you have had a chance to look over these examples, we hope that you have a better understanding of how a Javascript async await statement works and have some ideas on how you can add it to your code. They are a great way to keep things moving and prevent the browser from locking up while your code works on large tasks. They are also an excellent way to provide feedback to users in the form of progress updates, but these ideas only scratch the surface of what is possible using this tool.

I recommend trying your hand by expanding on the above examples, to help solidify your understanding of Javascript async await statements. For example, instead of one long 3000ms wait as we see in Example 2, why not create a code that gives us three 1000ms pauses with progress updates in-between breaks printed to console.log.

If you enjoyed reading over these javascript async await examples and you have learned something new, please share them to Facebook and Twitter.