## **ASYNC**

### **Introduction**

Since JavaScript is the language of the web, there are some functions that by necessity are going to take a decent amount of time to complete, such as fetching data from a server to display on your site. For this reason, JavaScript includes support for asynchronous functions, or to put it another way, functions that can happen in the background while the rest of your code executes.

### **Learning Outcomes**

1. What is a callback?
2. What’s a promise?
3. What are the circumstances when promises are better than callbacks?
4. What does the .then() function do?

### **Callbacks**

In the recent past, the way that these were most commonly handled were with callbacks, and even now they are still used quite a lot in certain circumstances.

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action. [MDN](https://developer.mozilla.org/en-US/docs/Glossary/Callback_function)

Callbacks are simply functions that get passed into other functions. For example:

myDiv.addEventListener("click", function(){ // do something! })

Here, the function addEventListener() takes a callback (the “do something” function) and then calls it when myDiv is clicked.

You will likely recognize this pattern as something that happens *all the time* in JavaScript code. Unfortunately, though they are useful in situations like the above example, using callbacks can get out of hand, especially when you need to chain several of them together in a specific order. The rest of this lesson discusses patterns and functions that will help keep you out of [Callback hell](http://callbackhell.com/).

Take a moment to skim through [this article](https://github.com/maxogden/art-of-node#callbacks) before moving on. Or, if you prefer a video, watch [this](https://www.youtube.com/watch?v=QRq2zMHlBz4).

### **Promises**

There are multiple ways that you can handle asynchronous code in JavaScript, and they all have their use cases. Promises are one such mechanism, and they’re one you will see somewhat often when using other libraries or frameworks. Knowing what they are and how to use them is quite useful.

Essentially, a promise is an object that might produce a value at some point in the future. Here’s an example:

Lets say getData() is a function that fetches some data from a server and returns it as an object that we can use in our code:

const getData = function() { // go fetch data from some API... // clean it up a bit and return it as an object: return data }

The issue with this example is that it takes some time to fetch the data, but unless we tell our code that, it assumes that everything in the function happens essentially instantly. So, if we try to do this:

const myData = getData() const pieceOfData = myData['whatever']

We’re going to run into trouble because when we try to extract pieceOfData out of the returned data, the function getData() will most likely still be fetching, so myData will not be the expected data, but will be undefined. Sad.

We need some way to solve this problem, and tell our code to wait until the data is done fetching to continue. Promises solve this issue. We’ll leave learning the specific syntax for the articles you’re about to read, but essentially Promises allow you to do this:

const myData = getData() // if this is refactored to return a Promise... myData.then(function(data){ // .then() tells it to wait until the promise is resolved const pieceOfData = data['whatever'] // and THEN run the function inside })

Of course, there are many more occasions where one would want to use Promises beyond fetching data, so learning these things now will be very useful to you.

### **[Assignment](https://www.theodinproject.com/paths/full-stack-javascript/courses/javascript/lessons/async#assignment)**

1. Read [this article](https://davidwalsh.name/promises). It’s a good starting place and it’s short and to the point.
2. Watch [this video](https://www.youtube.com/watch?v=2d7s3spWAzo). It’s a good place to get a feel for how one might actually use promises in the wild. Feel free to watch the other videos in the series, but they aren’t strictly needed at this point. The video also mentions the ES5/ES6 issue, don’t worry about that at this point either. All major browsers support Promises and we will teach you how to support older browsers in a later lesson.
3. Watch [this video](https://www.youtube.com/watch?v=8aGhZQkoFbQ) to understand how asynchronous code works in JavaScript.
4. Read [Chapter 2: Callbacks](https://github.com/getify/You-Dont-Know-JS/blob/1st-ed/async%20%26%20performance/ch2.md) and [Chapter 3: Promises](https://github.com/getify/You-Dont-Know-JS/blob/1st-ed/async%20%26%20performance/ch3.md) from You Don't Know JS. In *Chapter 2*, the author explains the problems with callbacks and why callback hell will be your worst enemy (hint: it’s the inversion of control and non-linear nature of callbacks). In *Chapter 3*, you go deep into the how and why of promises. This chapter is not the easiest read, but you’ll be a promising professional if you take the time to properly digest it. It’s worth the effort.

### **Additional Resources**

1. [This](https://www.sitepoint.com/demystifying-javascript-closures-callbacks-iifes/) is another useful article about Callback functions in JavaScript.
2. The [MDN Documentation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) for Promises. It might not be the best resource for *learning* all about them, but once you’ve read a more friendly article or tutorial, this will probably be the place you return to for a refresher.
3. [This video](https://www.youtube.com/watch?v=vQ3MoXnKfuQ) and [this one too](https://www.youtube.com/watch?v=yswb4SkDoj0) are both nice introductions to Promises if you need more repetition.
4. [This tutorial](https://scotch.io/tutorials/javascript-promises-for-dummies) is another good introduction.