codeburst
Bursts of code to power through your day. Web Development articles, tutorials, and news.

Follow

24K

JavaScript: What the heck is a Callback?

Brandon Morelli Follow
Jun 12, 2017 · 5 min read

Learn and understand the basics of callbacks in just 6 minutes with easy examples.

Callbacks — image via unsplash

Preface

> *Hey! I'm Brandon. I created codeburst.io and I write JavaScript tutorials and articles to help beginners better understand the inner workings of Web Development. If you have any questions about the article, leave a comment and I'll get back to you, or find me on twitter @brandonmorelli. Lastly, when you're ready to really dive into Web Development, Check out the Best Courses for Learning Full Stack Web Development*

What is a Callback?

Simply put: *A callback is a function that is to be executed* ***after*** *another function has finished executing — hence the name 'call back'.*

More complexly put: *In JavaScript, functions are objects. Because of this, functions can take functions as arguments, and can be returned by other functions. Functions that do this are called* ***higher-order functions****. Any function that is passed as an argument is called a* ***callback function****.*

```
function first(){
  console.log(1);
}
function second(){
  console.log(2);
}
first();
second();
```


As you would expect, the function `first` is executed first, and the function `second` is executed second — logging the following to the console:

```
// 1
// 2
```

All good so far.
But what if function `first` contains some sort of code that can't be executed immediately? For example, an API request where we have to send the request then wait for a response? To simulate this action, were going to use `setTimeout` which is a JavaScript function that calls a function after a set amount of time. We'll delay our function for 500 milliseconds to simulate an API request. Our new code will look like this:

```
function first(){
  // Simulate a code delay
  setTimeout( function(){
    console.log(1);
  }, 500 );
}
function second(){
  console.log(2);
}
first();
second();
```

It's not important that you understand how `setTimeout()` works right now. All that matters is that you see we've moved our `console.log(1);` inside of our 500 millisecond delay. So what happens now when we invoke our functions?

```
first();
second();
// 2
// 1
```

Even though we invoked the `first()` function first, we logged out the result of that function after the `second()` function.
It's not that JavaScript didn't execute our functions in the order we wanted it to, it's instead that **JavaScript didn't wait for a response from `first()` before moving on to execute `second()`.**
So why show you this? Because you can't just call one function after another and hope they execute in the right order. Callbacks are a way to make sure certain code doesn't execute until other code has already finished execution.

Create a Callback

Alright, enough talk, lets create a callback!
First, open up your Chrome Developer Console (*Windows: Ctrl + Shift + J*)(*Mac: Cmd + Option + J*) and type the following function declaration into your console:

```
function doHomework(subject) {
  alert(`Starting my ${subject} homework.`);
}
```

Above, we've created the function `doHomework`. Our function takes one variable, the subject that we are working on. Call your function by typing the following into your console:

```
doHomework('math');
// Alerts: Starting my math homework.
```

```
function doHomework(subject, callback) {
  alert(`Starting my ${subject} homework.`);
  callback();
}

doHomework('math', function() {
  alert('Finished my homework');
});
```


As you'll see, if you type the above code into your console you will get two alerts back to back: Your 'starting homework' alert, followed by your 'finished homework' alert.
But callback functions don't always have to be defined in our function call. They can be defined elsewhere in our code like this:

```
function doHomework(subject, callback) {
  alert(`Starting my ${subject} homework.`);
  callback();
}
function alertFinished(){
  alert('Finished my homework');
}
doHomework('math', alertFinished);
```

This result of this example is exactly the same as the previous example, but the setup is a little different. As you can see, we've passed the `alertFinished` function definition as an argument during our `doHomework()` function call!

A real world example

Last week I published an article on how to Create a Twitter Bot in 38 lines of code. The only reason the code in that article works is because of Twitters API. When you make requests to an API, you have to wait for the response before you can act on that response. This is a wonderful example of a real-world callback. Here's what the request looks like:

```
T.get('search/tweets', params, function(err, data, response) {
  if(!err){
    // This is where the magic will happen
  } else {
    console.log(err);
  }
})
```

- `T.get` simply means we are making a get request to Twitter
- There are three parameters in this request: `'search/tweets'`, which is the route of our request, `params` which are our search parameters, and then an anonymous function which is our callback.

A callback is important here because we need to wait for a response from the server before we can move forward in our code. We don't know if our API request is going to be successful or not so after sending our parameters to search/tweets via a get request, we wait. Once Twitter responds, our callback function is invoked. Twitter will either send an `err` (error) object or a `response` object back to us. In our callback function we can use an `if()` statement to determine if our request was successful or not, and then act upon the new data accordingly.

You made it

Good work! You can now (ideally) understand what a callback is and how it works. This is merely the tip of the iceberg with callbacks, there is still a lot more to learn! I publish a few articles/tutorials each week, please enter your email here if you'd like to be added to my once-weekly email list.

. . .

❤ If this post was helpful, please hit the little blue heart! And don't forget to check out my other recent articles:

codeburst
Bursts of code to power through your day. Web Development articles, tutorials, and news.

Follow

24K

JavaScript Web Development Tech Nodejs Technology

24K claps

WRITTEN BY

Brandon Morelli

Creator of @codeburstio — Frequently posting web development tutorials & articles. Follow me on Twitter too: @BrandonMorelli

Follow

codeburst

Bursts of code to power through your day. Web Development articles, tutorials, and news.

Follow

See responses (98)

More From Medium

More from codeburst More from codeburst More from codeburst

KOTLIN vs JAVA — The Great War of Android App Development

Ruchika Singh...
Jul 25 · 6 min read ★

879

How To Fetch Data From An API With React Hooks

Indrek Lasn in codeburst
Jun 21 · 4 min read ★

1K

How to Create and Deploy a Serverless React App

Frank Zickert in...
Jul 9 · 4 min read ★

18

codeburst
Bursts of code to power your coding for the day. Web Development articles, tutorials, and news.

Follow

24K