Callback function - don't get intimidated
The formal definition from MDN is a little confusing:

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.

Heh?

Don't worry, I'll try simplify it here.

A callback function is just a function which is:

- accessible by another function; and
- invoked after the first function completes

There are a few other nuances of a callback function. For example, a callback function is a function that is passed as an argument to another function. Remember our `addEventListener()` method. We passed our callback function (i.e. our event handler) inside of this function. And remember, a callback function is executed inside of the function it was passed into. Make sense?

Why do we need callbacks?

For many reasons. At the most basic level, we use callbacks to run code in response to something happening. Pretty useful right?

Another example of why we would want to use callbacks is to stop a process from blocking our other code.

Let me explain with an example.

Client-side JavaScript runs in the browser. We know this.

But this JavaScript process is a single threaded event loop.

This means that if we try to execute long-running operations within a single-

threaded event loop, these processes will be blocked. This is bad because this will stop other JavaScript code executing and we'll have to wait for the operation to complete.

Want an example?

Look at the "alert" function. The alert function is considered as one of the blocking codes in JavaScript in the browser. If you run `alert()`, you can no longer interact within the browser, until you close the alert dialog window. So, in order to prevent blocking on long-running operations, callbacks are typically used.

Conclusion

A nice way of imagining how a callback function works is that it is a function that is "**called after**" the first function (that it was passed into) executes. So, maybe a better name would have been to call it a **callafter** function. Hope this helps.

Lets move on.