**Asynchronous JavaScript - Part 3… -** [Marc Kirk](https://medium.com/@byteslovesbits?source=post_page-----864df36177f3--------------------------------) Feb 25, 2022

So the JavaScript environment executes JavaScript synchronously, on a single call stack.

So how does JavaScript do multiple things at once? How does a web-page that contains JavaScript, retrieve data from a server and perform long running calculations — all whilst dynamically updating the page and keeping the page interactive.

The answer is with the help of the web-browser. You see, whilst JavaScript is single threaded, the browser is capable of doing many things at the same time and one of those things is to give the illusion that JavaScript is asynchronous. At the most fundamental of levels, this illusion is created through the computer programming concept of a ***‘callback’***.

**Callbacks**

A callback is a function that we pass to another function as an argument. That other function will then invoke our passed in function. Callbacks are critical parts of asynchronous JavaScript! Although they are a relatively simple concept, they are central for more advanced asynchronous JavaScript concepts, such as ***promises*** and ***async await***. Callbacks come in two flavours, synchronous and asynchronous. Let’s start with synchronous callbacks.

Figure 1 shows a very basic callback. Line 1 declares a callback function. This is the function that gets ‘called back’.

Line 5 is the function that receives the callback function. A function like this, that receives another function as an argument or returns a function is known as a ***higher order function***.

One line 9, we pass the callback function into the higher order function named doSomething, which then invokes the callback function on line 6. Execution jumps to line 2, where the callback does a console.log. Note also how we can pass anonymous functions as callbacks to the higher order functions. See Figure 2.

Text

Description automatically generated

Figure 1 — synchronous callbacks

Text

Description automatically generated

Figure 2 — passing an anonymous arrow function as a callback function

**Asynchronous Callbacks**

Synchronous callbacks are useful but to do more powerful things, we need a sprinkle of asynchronicity. We get some of this asynchonicity with the help of the web-browser, more specifically, the web-browser’s APIs.

The browser supplies web-apis that extend the JavaScript language so that the language can do useful things like asynchronous operations. Figure 3. There are many powerful web-apis, such as the GeoLocation api which can find a user’s gps coordinates.

Graphical user interface, diagram, application

Description automatically generated

Figure 3— extending the core language to make it more powerful

**Setimeout**

Finally, we get to do some asynchronous programming. Open up chrome and type about:blank into the address bar. This will give you a blank page. From within the console tab of the developer tools, type window followed by enter. Figure 4. Expand the right chevron next to the word window within the console and scroll down until you reach settimeout. Figure 5.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 4 — accessing the JavaScript global object from Chrome’s developer tools

Graphical user interface

Description automatically generated

Figure 5 — settimeout is a global function

The **setTimeout()** method is part of the browsers global object. **setTimeout()**sets a timer which executes code once the timer expires. **setTimeout()** is part of a web-api that creates the illusion that JavaScript is asynchronous. We will use the **setTimeout()** method next to do some asynchronous programming.

Within the same about:blank page, type the contents of Figure 6 into the console. Figure 7. Hit enter and after three seconds the blank page will display ‘Asynchonous JavaScript’. Congratulations you are now an asynchronous programmer!

Setimeout is simple. It calls myfunction after three 3000 milliseconds, which changes the innerHTML of the document’s body to display ‘Asynchonous JavaScript’. Bonus points if you spotted the typos.

Graphical user interface, text

Description automatically generated

Figure 6 — incorrectly spelling asynchronous

Graphical user interface, text, application

Description automatically generated

Figure 7 — executing an asynchronous function

Graphical user interface, text, application

Description automatically generated

Figure 8 — web apis in action. setimeout asynchronously updates the DOM after three seconds

Stay tuned for [part 4](https://medium.com/@byteslovesbits/asynchronous-javascript-part-4-4fd493c0a994), where we explore the JavaScript event loop and gain deeper insight into how **setTimeout()**fits within the JavaScript stack. This is the most critical part of the asynchronous puzzle.