**Event Loop:**

The **event loop** is the secret behind JavaScript’s asynchronous programming. JS executes all operations on a single thread, but using a few smart data structures, it gives us the illusion of multi-threading. Let’s take a look at what happens on the back-end.

The **call stack** is responsible for keeping track of all the operations in line to be executed. Whenever a function is finished, it is popped from the stack.

The **event queue** is responsible for sending new functions to the stack for processing. It follows the queue data structure to maintain the correct sequence in which all operations should be sent for execution.

A picture containing text, electronics

Description automatically generated

Whenever an async function is called, it is sent to a browser API. These are APIs built into the browser. Based on the command received from the call stack, the API starts its own single-threaded operation.

An example of this is the setTimeout method. When a setTimeout operation is processed in the stack, it is sent to the corresponding API which waits till the specified time to send this operation back in for processing.

Where does it send the operation? The event queue. Hence, we have a cyclic system for running async operations in JavaScript. The language itself is single-threaded, but the browser APIs act as separate threads.

The event loop facilitates this process; it constantly checks whether or not the call stack is empty. If it is empty, new functions are added from the event queue. If it is not, then the current function call is processed.