[Individuals]

[Mina Sameh][Link]
Document

[Adham Magdy][Link]
Event Loop Explained
ESLint vs Prettier

[Ahmed Abu Qahf][Link]

The event loop is what makes Asynchronous JavaScrip possible because JS is single-threaded which means it can only run one thing at a time. JavaScript executes a script from top to bottom by organizing the order of execution through a **call stack**.

```
function blockingTask(){
     for(let t = 0; t < 10000; t++) {};
     console.log('blockingTask is done';
}

console.log('Script Start');
blockingTask();
console.log('Script end');
```

In the previous example, Javascript pushes the first console statement on top of the call stack and executes it, then places the blockingTask function on top of the stack right after. The computer takes 10 seconds to execute the blockingTask, prints the message `blockingTask is done` and then pops the function from the stack and pushed the last console statement;

Blocking the execution of the code is not a good thing as it may lead a webpage to hang while the function is done executing...etc. Hence, came the idea of callbacks. Basically, the idea is to have a non-blocking function that will run and

won't block the execution of the rest of the script. In order to achieve this goal, a callback function will be attached to an async function to be called whenever the async function has finished execution. But then came an issue of how to organize the execution of the callbacks, hence came the callback queue to keep record of which callback to be executed first. Again, another issue when the callback function will be moved from the callback queue to the call stack, hence came the event loop. The event loop acts as a gatekeeper, it manages what goes in and out from the callback queue and when a callback should be pushed back to the call stack. The event loop manages to perform such functionality though a set of phases called: timers, pending callbacks, idle-prepare, poll, check & close callbacks.

[Team]

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.

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.