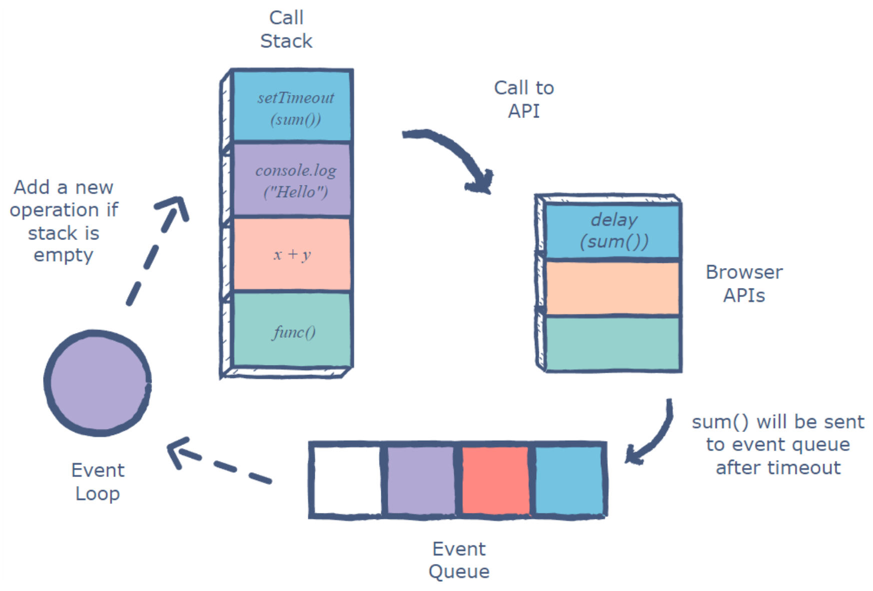
JS Event Loops

What is it:

Event Loop is the reason why java script looks like it executes async functions using multi-threading despite it using a single thread normally by using a "**Call stack**" and an "**Event/Callback queue**" linking between them. The call stack is responsible for keeping track of the functions called, and the event queue is responsible for sending the functions to the stack.

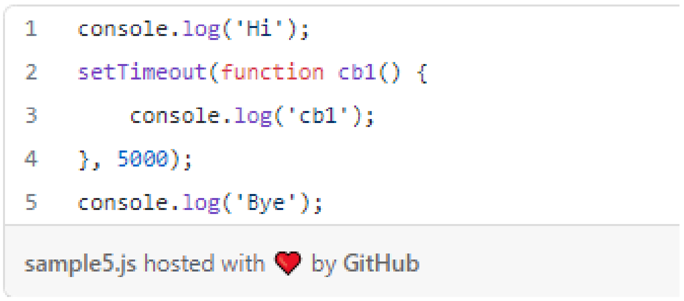
How it works:

- The event loop monitors both the stack and the queue. If the stack is empty, the event loop takes the first event in the queue and pushes it in the stack for execution and it does this every iteration which is called "Tick".
- When using an async function ,for example the setTimeout function, it is sent to the "Web API" in the browser with the API handling the "Multi-Thread illusion".
- Then JS will continue sequentially finishing the execution of the rest of the code sending the calls to the call stack in order to execute in order until.
- Once the return value is returned or the timer is done the async function is then called again into the queue and then called in the stack in order to be executed



Example:

The following code snippet image is executed using the “setTimeout” callback function.



```javascript
console.log('Hi');
setTimeout(function cb1() {
    console.log('cb1');
}, 5000);
console.log('Bye');
```

- Before execution, the call stack, api and event queue are empty.
- Console.log(‘Hi’) is inserted to the call stack and executed then removed from the stack.
- Once the setTimeout function is added to the stack it is executed, but it requires a timer. So it’s sent to the api in order to handle the countdown of 5 seconds while the rest of the code continues executing, then setTimeout is removed from the stack.
- Then console.log(‘Bye’) is pushed to the stack, executed and then removed, same as the earlier console.log
- Once the countdown is finished, the callback is then pushed into the event queue and the event loop checks if the call stack is empty, and since its empty the callback function is pushed to the stack along with console.log(‘cb1’)
- The callback function is executed and then removed, and finally console.log(‘cb1’) is executed.

References:

- https://blog.sessionstack.com/how-javascript-works-event-loop-and-the-rise-of-async-programming-5-ways-to-better-coding-with-2f077c4438b5
- https://www.educative.io/answers/what-is-an-event-loop-in-javascript