Asynchronous JavaScript

JAVASCRIPT SYLHET COMMUNITY

We will cover,

- Why Asynchronous?
- JavaScript Thread
- Blocking Synchronous Code
- Asynchronous
- Call Stack, Web API, Queue, Event Loop
- Handling Async Code(Callback, Promise, Async-await)

If you know the Asynchronous of JS

- You will make API Call.
- Can use setTimeout, setInterval.
- Can read any external file.

JavaScript Thread

- JavaScript Thread process or execute each line of code.
- Traditionally JavaScript runs as single thread.
- Means execute each line of instruction at each time.
- This is also called Synchronous.

Blocking Synchronous Code

```
function delayBySeconds(sec) {
    let start = now = Date.now();
    while(now - start < (sec * 1000)) {
        now = Date.now()
    }
    console.log("done");
}

delayBySeconds(5);
```

Asynchronous

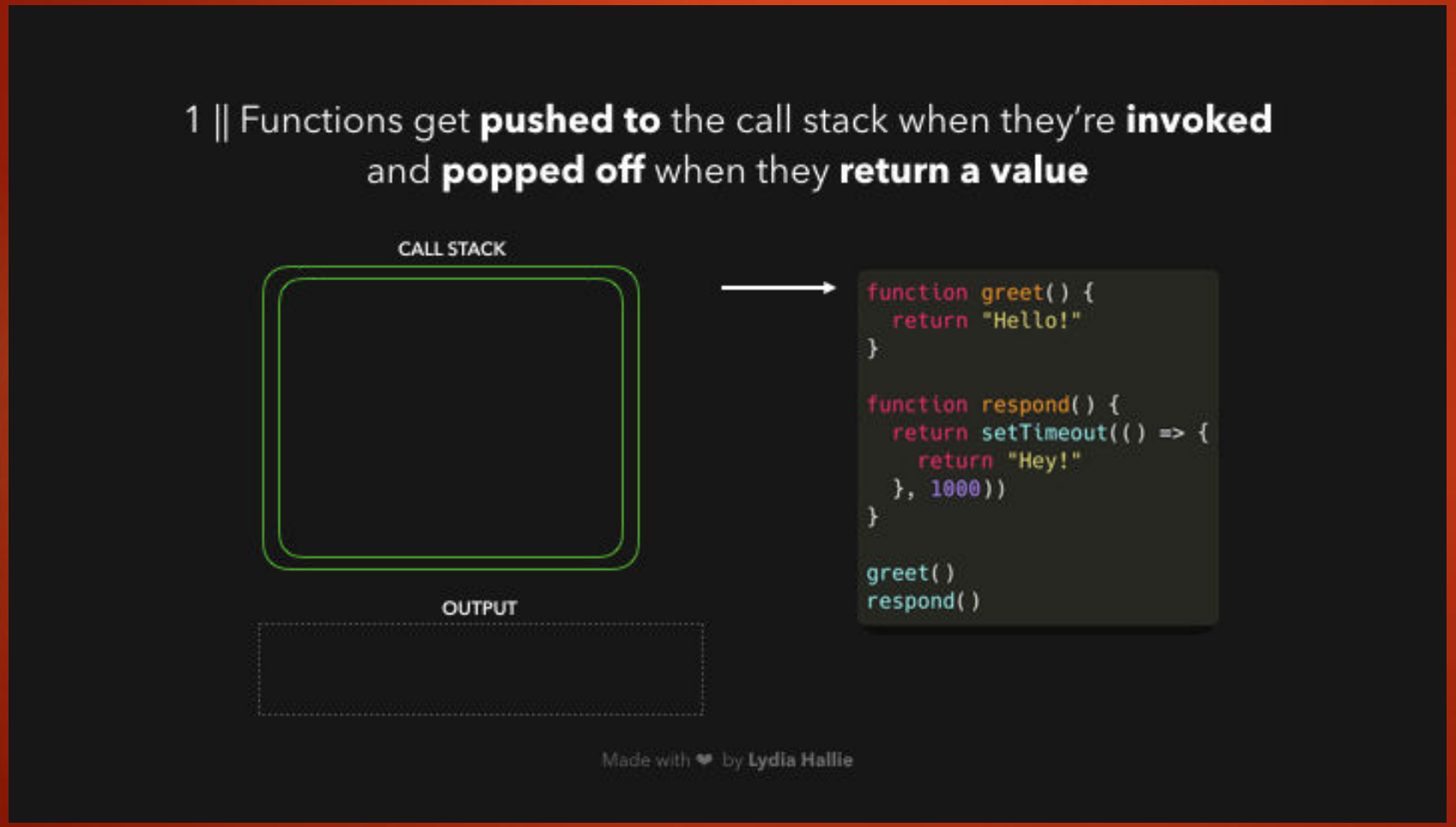
- JavaScript also handles Asynchronous nature.
- Means some code can work outside of main thread.
- With the help of call stack, web api, queue and event loop.

As javascript handles them in a separate thread so they are not non-blocking, some common asynchronous stuff of javascript are,

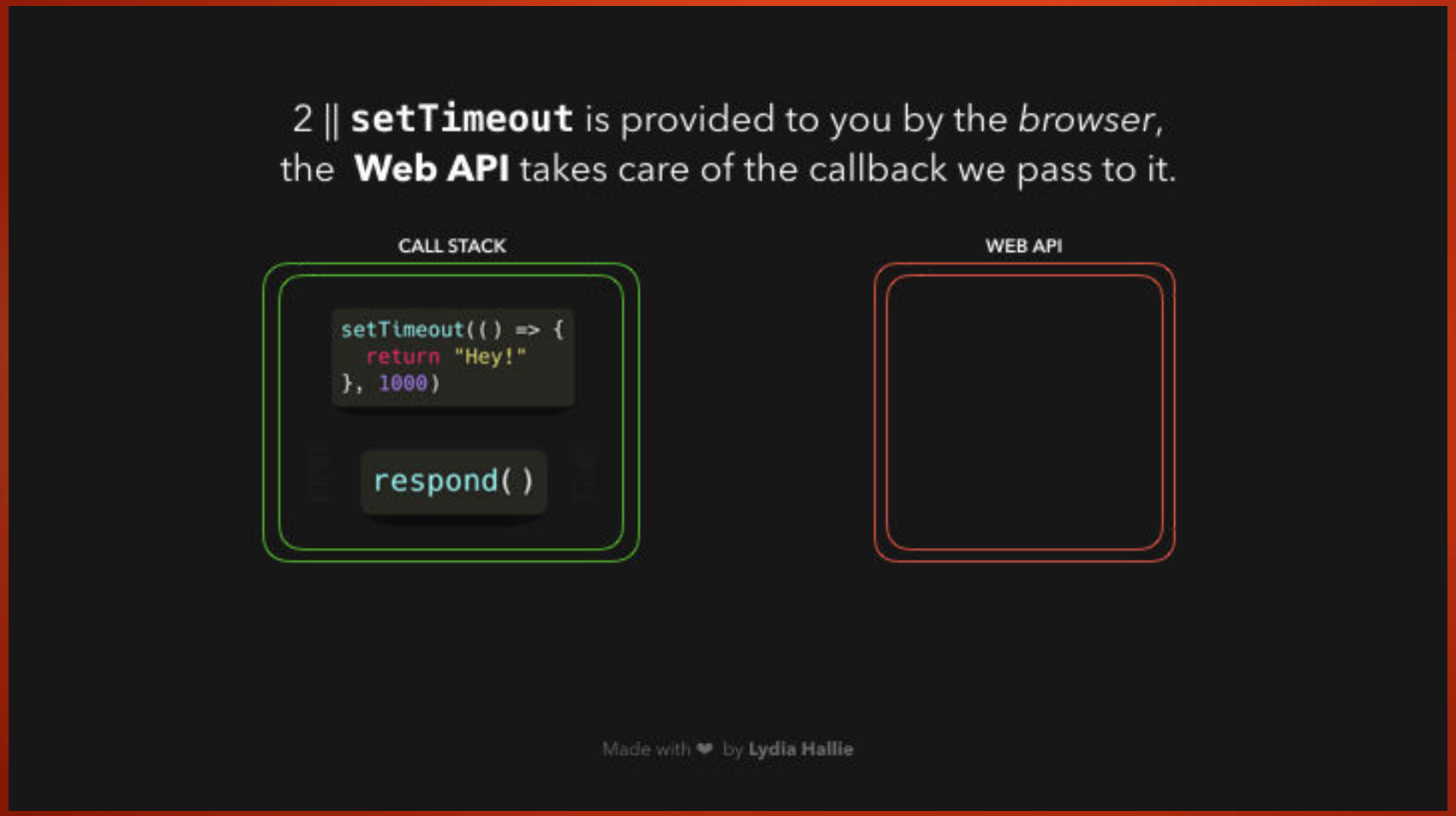
- setTimeout
- calling an API
- reading a file

- All modern JavaScript engines use non-blocking or event loop approach to deal asynchronous stuff.

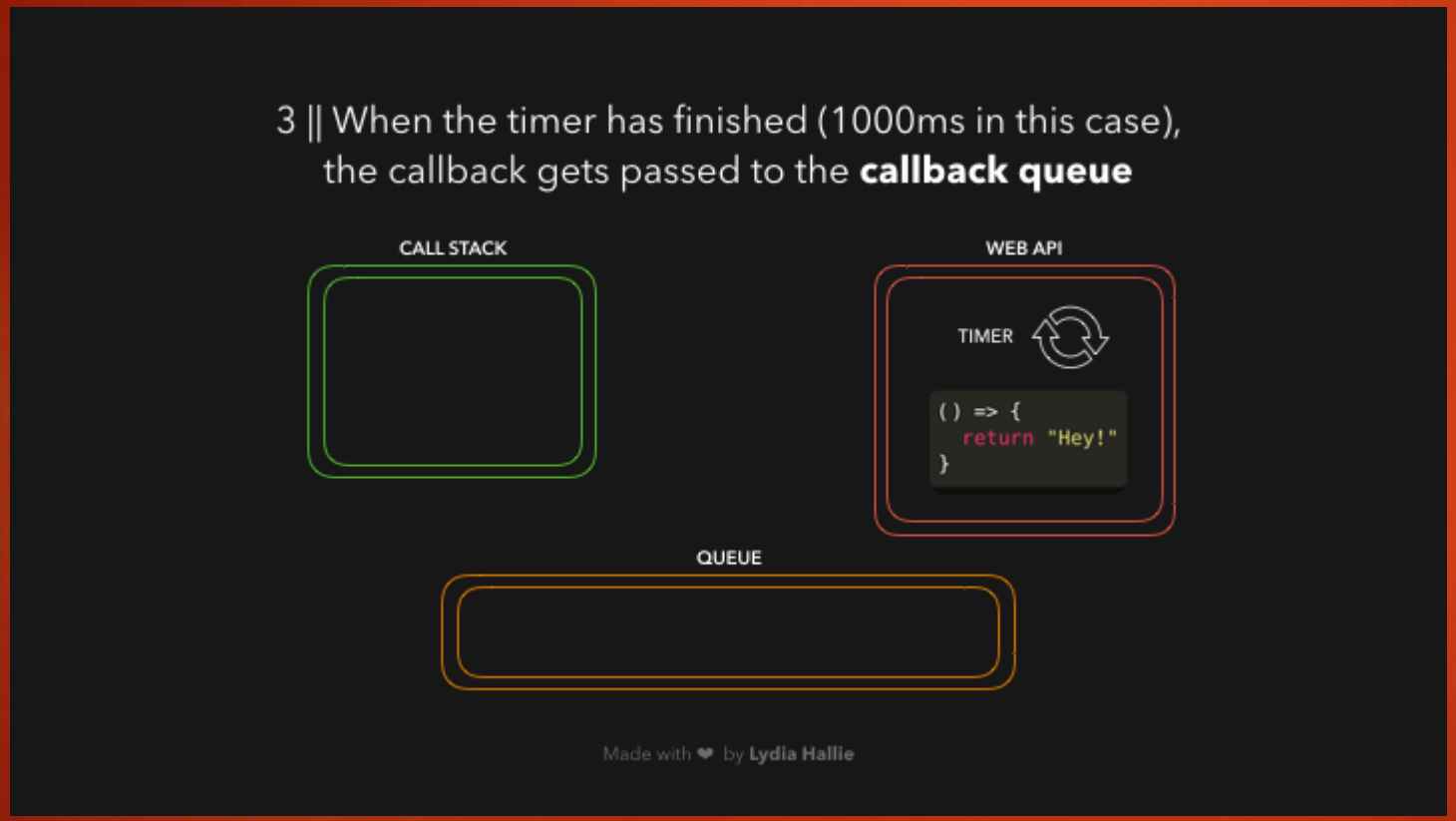
Call Stack


1 || Functions get **pushed to** the call stack when they're **invoked** and **popped off** when they **return a value**
CALL STACK
OUTPUT
function greet() {
return "Hello!"
}
function respond() {
return setTimeout(() => {
return "Hey!"
}, 1000))
}
greet()
respond()
Made with ❤ by Lydia Hallie

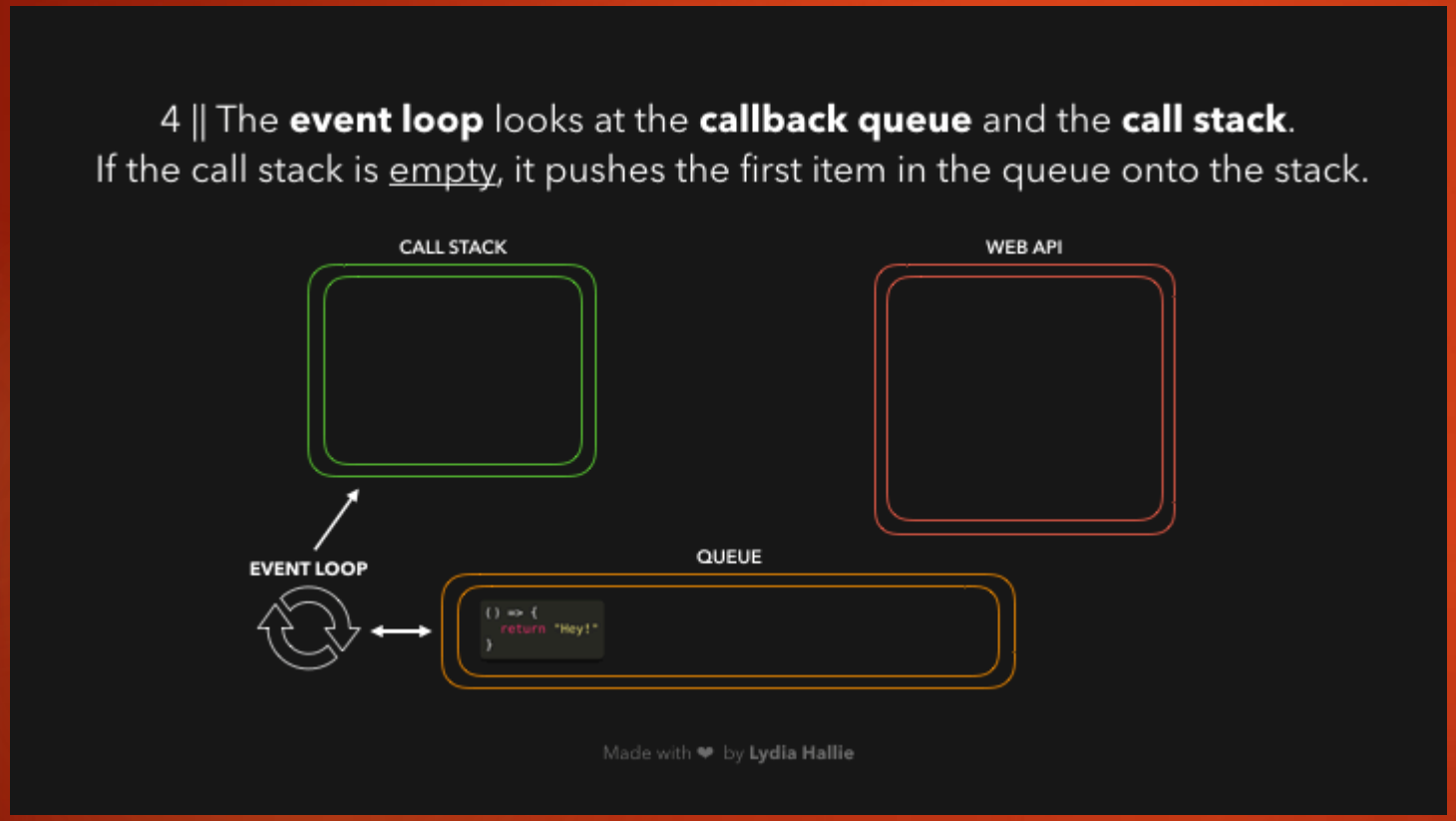
Web Api


2 || setTimeout is provided to you by the browser,
the Web API takes care of the callback we pass to it.
CALL STACK
setTimeout(() => {
return "Hey!"
}, 1000)
respond()
WEB API
Made with ❤ by Lydia Hallie

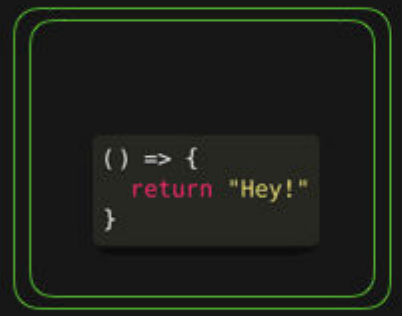
Queue


3 || When the timer has finished (1000ms in this case), the callback gets passed to the **callback queue**
CALL STACK
WEB API
TIMER
() => {
return "Hey!"
}
QUEUE
Made with ❤ by Lydia Hallie

Event Loop


4 || The **event loop** looks at the **callback queue** and the **call stack**.
If the call stack is empty, it pushes the first item in the queue onto the stack.
CALL STACK
WEB API
EVENT LOOP
QUEUE
() => {
return "Hey!"
}
Made with ❤ by Lydia Hallie

5 || The callback is added to the call stack and executed.
Once it returned a value, it gets popped off the call stack.



OUTPUT

```
function greet() {
  return "Hello!"
}

function respond() {
  return setTimeout(() => {
    return "Hey!"
  }, 1000))
}

greet()
respond()
```

Handle Asynchronous(ex: reading a file, calling an api)

These are common ways to handle asynchronous task.

- Callback
- Promise
- Async-await

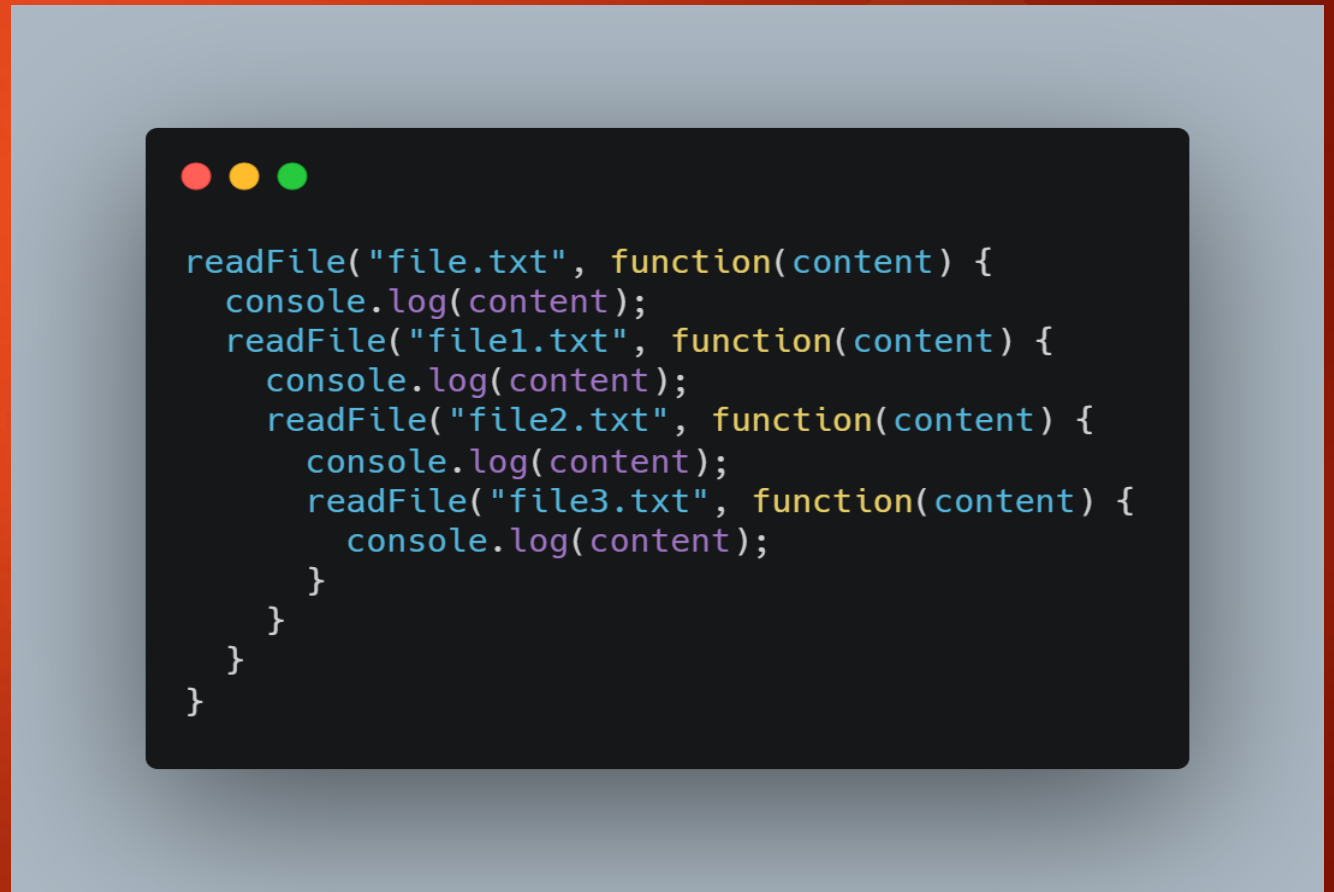
Callback

- Just pass a function that will be called when a task is completed.

```
readFile("file.txt", function(content) {
  console.log(content);
});
```

Callback Hell

- Callback(nested) can be hard to read.
- No proper Error Handling.



```
readFile("file.txt", function(content) {
  console.log(content);
  readFile("file1.txt", function(content) {
    console.log(content);
    readFile("file2.txt", function(content) {
      console.log(content);
      readFile("file3.txt", function(content) {
        console.log(content);
      }
    }
  }
}
```

Promise

Solves some issues of Callback.

- Chaining
- Better Error Handling

```
new Promise((resolve, reject) => {
    resolve("Your request is fine");
}).then(res => {
    console.log(res); // Your request is fine
}).catch(err => {
    console.log(err);
})
```

Practical Example of promise

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(function(response) {
    return response.json();
  })
  .then(function(data) {
    console.log(data)
  })
  .catch(function(err) {
    console.log('Fetch problem: ' + err.message);
  });
```

Async-await

- Another one, which is just syntactic sugar of Promise.
- Async Function returns a promise, await is used to call an api and wait for it to resolve or reject.

```
const fetch = require('node-fetch');

async function getUsers() {
    try {
        const response = await fetch("https://jsonplaceholder.typicode.com/users")
        let data = await response.json()
        console.log(data)
    } catch(err) {
        console.log(err);
    }
}

getUsers()
```