1.8 Javascript Web APIs and EventLoop

Table of contents:

Javascript Web APIs

A **Web API (Application Programming Interface)** is a **set of built-in functions provided by the browser** that allows JavaScript to interact with the browser and perform tasks like:

✔️ Making network requests (`fetch()`)

✔️ Setting timers (`setTimeout()`, `setInterval()`)

✔️ Manipulating the DOM (`document.querySelector()`)

✔️ Handling events (`addEventListener()`)

✔️ Working with storage (`localStorage`, `sessionStorage`)

✔️ Using advanced features (`Geolocation`, `Notifications`, etc.)

📌 **JavaScript itself does not provide these features.** They are provided by the browser as part of the **Web APIs**. Web APIs are NOT part of JavaScript (ECMAScript). They are provided by the browser or runtime environment (Node.js, Deno, etc.).

▼ setTimeOut() method

The setTimeout() method calls a function after a number of milliseconds. The `setTimeout()` method executes a block of code after the specified time. The method executes the code only once.

📌 It **does not** belong to JavaScript's core **ECMAScript specification** but is provided by the **Web APIs** in browsers and by **Node.js**.

The commonly used syntax of JavaScript setTimeout is:

```
setTimeout(function, milliseconds);
```

Its parameters are:

- **function** - a function containing a block of code
- **milliseconds** - the time after which the function is executed

The `setTimeout()` method returns an **intervalID**, which is a positive integer.

▼ clearTimeOut() method

You generally use the `clearTimeout()` method when you need to cancel the `setTimeout` () method call before it happens.

```
// program to stop the setTimeout() method

let count = 0;

// function creation
function increaseCount(){

    // increasing the count by 1
    count += 1;
    console.log(count)
}

let id = setTimeout(increaseCount, 3000);

// clearTimeout
clearTimeout(id);
console.log('setTimeout is stopped.');
```

Output

```
setTimeout is stopped.
```

▼ **setInterval() method**

The `setInterval` () method is useful when you want to repeat a block of code multiple times. For example, showing a message at a fixed interval.

The commonly used syntax of JavaScript setInterval is:

```
setInterval(function, milliseconds);
```

Its parameters are:

- **function** - a function containing a block of code
- **milliseconds** - the time interval between the execution of the function

The `setInterval()` method returns an **intervalID** which is a positive integer.

▼ **clearInterval() method**

The syntax of `clearInterval()` method is:

```
clearInterval(intervalID);
```

Here, the `intervalID` is the return value of the `setInterval()` method.

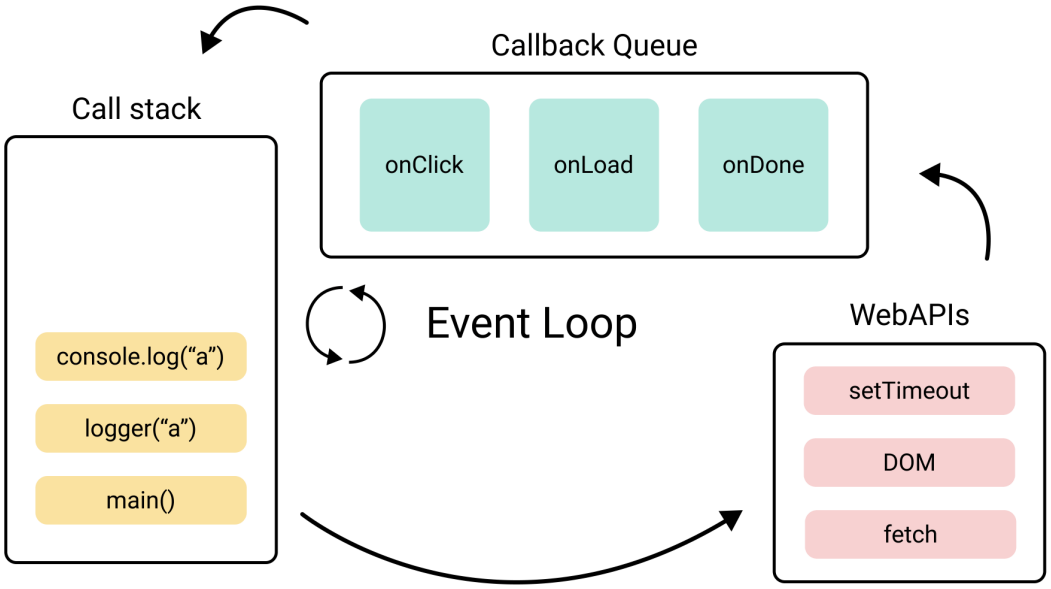
3. Event Loop in Javascript

JavaScript is **single-threaded**, meaning it can execute **only one task at a time**. But it can handle **asynchronous operations** (like timers, network requests, and user interactions) using the **Event Loop**.

👉 **The Event Loop is responsible for managing and executing JavaScript's asynchronous operations.** It ensures that tasks are executed in the correct order, without blocking the main thread.

🛠️ How the Event Loop Works?

The Event Loop continuously cycles through these steps:



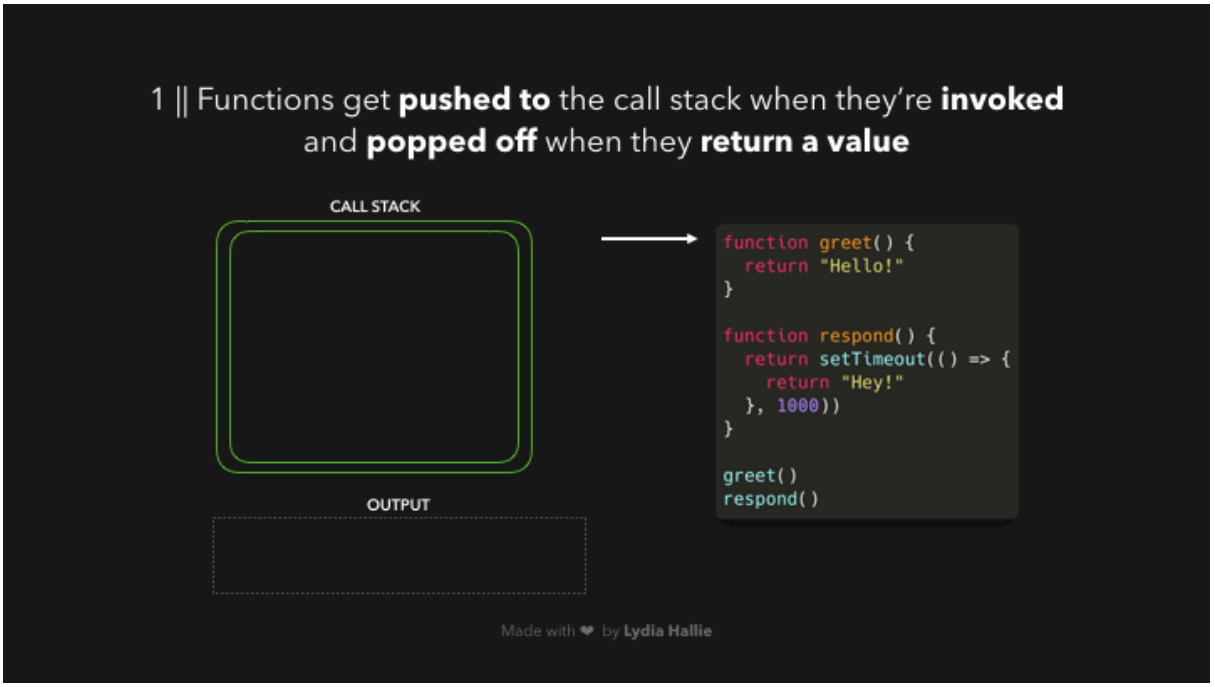
1️⃣ **Execute Synchronous Code (Call Stack)** – JavaScript runs code **line by line** in the **Call Stack**.

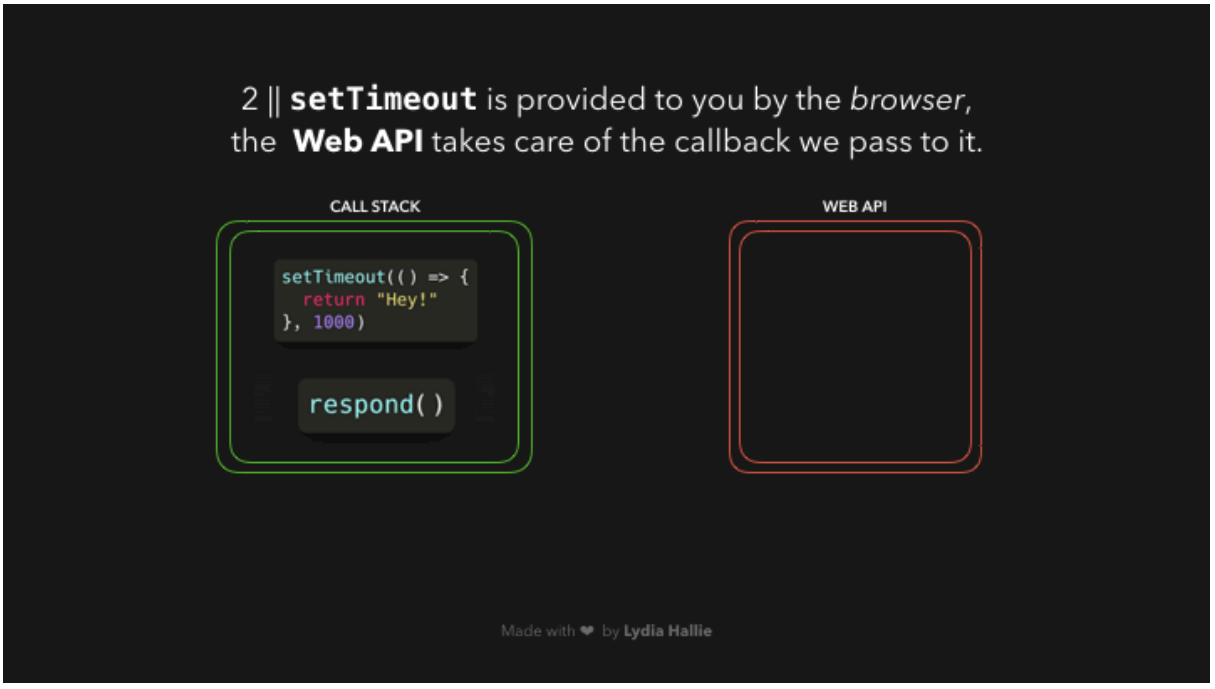
2️⃣ **Handle Web APIs (Async Tasks)** – If an async task (like `setTimeout` or `fetch`) is called, it is **handed over to the Web API**.

3️⃣ **Move Completed Tasks to the Callback Queue** – Once async tasks finish, their callbacks are placed in the **Callback Queue** (or the **Microtask Queue** for Promises).

4️⃣ **Check the Call Stack** – If the **Call Stack is empty**, the Event Loop moves tasks from the **Queue to the Stack** and executes them.

5️⃣ **Repeat the Process** – This cycle continues **forever** while JavaScript is running.





Let's understand it all again!

```
console.log("1️⃣ Start");

setTimeout(() => {
    console.log("3️⃣ Inside setTimeout");
}, 2000);

console.log("2️⃣ End");
```

What Happens Under the Hood?

1️⃣ `console.log("1️⃣ Start")` is added to the **Call Stack** → executed immediately.

2️⃣ `setTimeout(callback, 2000)` is added to the **Call Stack**.

- The timer is **handled by the Web APIs**, and the callback is **moved out** of the stack.3️⃣ `console.log("2️⃣ End")` is added to the **Call Stack** → executed immediately.4️⃣ After 2 seconds, the callback function is moved to the **Callback Queue**.5️⃣ The **Event Loop** moves the callback to the **Call Stack** once it's empty.