Asynchronous JavaScript

JavaScript runs code **one line at a time** — it's **single-threaded**. This means only one task can happen at any moment. Still, JavaScript can do things like wait for a timer or handle user clicks without stopping everything else.

This is because JavaScript uses **asynchronous behavior** for certain tasks.

Synchronous Code

This is how JavaScript normally works — **top to bottom**.

```
console.log("A");
console.log("B");
console.log("C");

// Output:
// A
// B
// C
```

Each line waits for the previous one to finish. That's **synchronous** execution.

Asynchronous Code

Here's where it gets interesting:

```
console.log("A");

setTimeout(() => {
  console.log("B");
}, 1000);

console.log("C");

// Output:
// A
// C
// B (after about 1 second)
```

Even though `setTimeout` is written before `"C"`, it runs **after**. Why?

What Happens Behind the Scenes

When JavaScript sees `setTimeout`, it:

1. Sends the task (along with its delay) to the **browser** (or **Node.js**, if you're running it there).
2. Continues running the rest of the code — it doesn't wait.
3. After the timer finishes, the function you passed to `setTimeout` is **sent back** to JavaScript to be run.
4. But it will only run **after** the current code is done.

This system of handling things is managed by the **event loop**.

The Event Loop (In Simple Words)

The **event loop** is a mechanism that:

- Keeps checking if JavaScript is done running your current code.
- If yes, it picks up any pending tasks (like the `setTimeout` callback) and runs them.

Real-Life Analogy

Imagine you're cooking:

- You put rice on the stove and set a timer.
- You don't just stand there — you cut vegetables.
- When the timer rings, you go back to check the rice.

Similarly, JavaScript **schedules** tasks like timers to run **later**, and continues with the rest of the code.

Another Example

```
console.log("Start");

setTimeout(() => {
  console.log("Waiting over");
}, 2000);

console.log("End");

// Output:
// Start
// End
// Waiting over (after ~2 seconds)
```

Even with 2 seconds delay, `"End"` appears **immediately** after `"Start"` — that's the power of async.

Summary

- JavaScript is single-threaded: one thing at a time.
- Functions like `setTimeout` are **asynchronous**.
- These async tasks are handled by the browser/Node and returned later.

- The **event loop** checks when JavaScript is free to run those returned tasks.

Understanding this helps you write programs that don’t get “stuck” waiting and can handle things like user input, network requests, and timers smoothly.