event loop

What is it?

Hello, I’m Bình.

I’m an Frontend Developer/Translator at Manabie

Agenda

- 1. Multi threaded languages vs Single threaded languages
- 2. Stack and Queue
- 3. The Call Stack
- 4. The run-time environment
- 5. Event loop visualization
- 6. Conclusion, Q&A

Multi threaded languages

Languages such as **C#**, **Java** are **multi-threaded languages**. This mean they can execute **multiple things** at a time

> Imagine eating with many hands. It means you can **eat multiple things at once**.
> However, it **doesn't necessary mean that you can eat faster than with only one hand**.

Single threaded language

Javascript is a **single-threaded language**. This means it can only execute **one thing** at a time!

> Imagine if you only have one hand. You **can only eat one thing at a time**.

🤔 How can Javascript executes asynchronous code?

-> Answer: It needs some help

In order to understand how to help Javascript, we have to understand 2 concepts: **Stack** & **Queue**

Stack

Stack

Stack is an "array" of items can only be added or removed in 2 ways:

- push: Add to the top
- pop: Remove from the top

> Imagine a stack of dishes (or just look at the image here)

🤔 What does this have to do with JS code execution?

-> Answer: I'm glad you asked, but let's wait a bit to talk about the Queue first

Queue

Queue is an "array" of items can only be added or removed in 2 ways:

- push: Add to the bottom
- pop: Remove from the top

> Imagine going to the supermarket during covid

🤔 What does this have to do with JS code execution?

-> Answer: First, let's take a look at how your code is executed using **only the Stack**

The Call Stack

The Call Stack

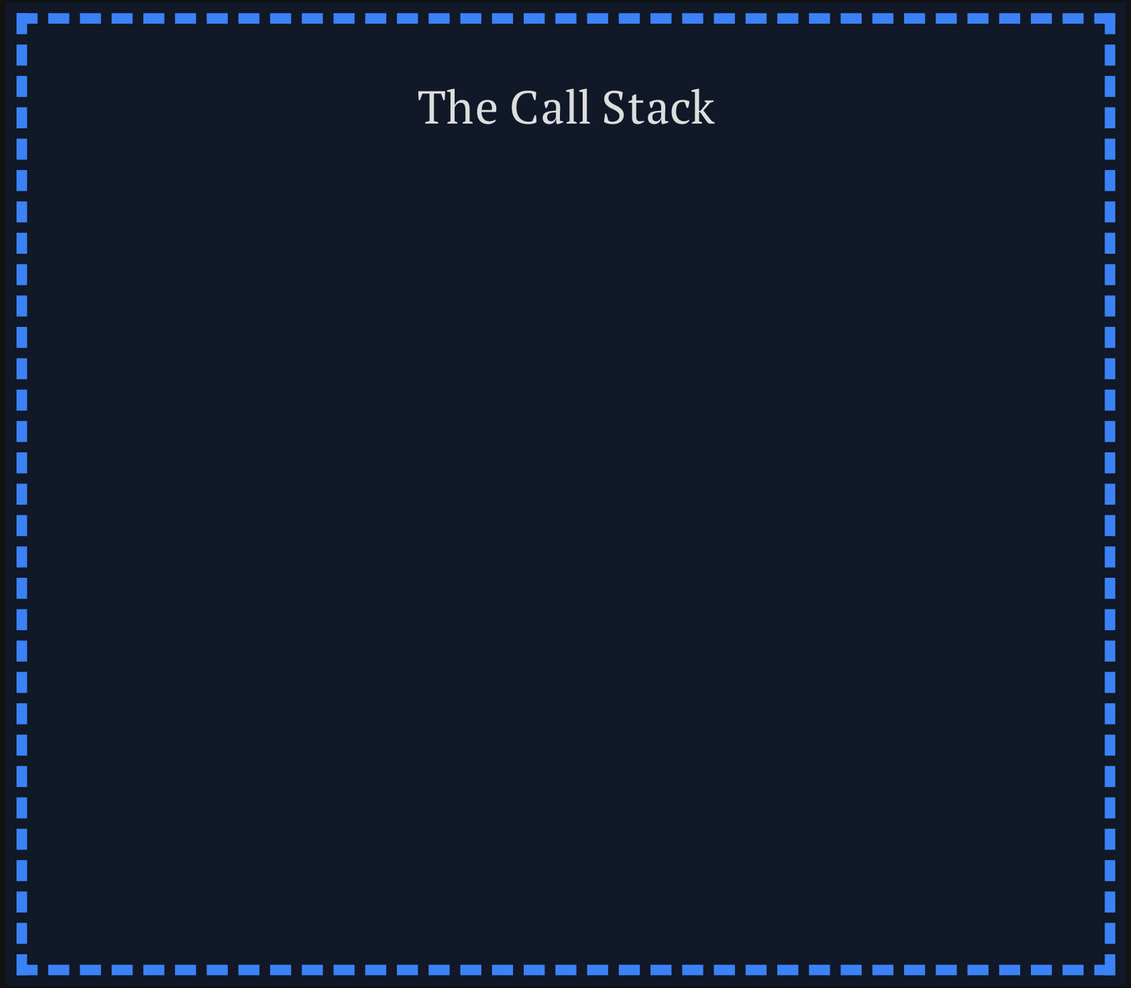
Let's take a look at this piece of code, assuming this entire piece is wrapped by a *main* function:

```
function transform(data) {
  return `Transformed data: ${data}`
}

function getFromService(data) {
  return `Data from service ${data}`
}

function fetchData(data) {
  const dataFromService = getFromService(data);
  return transform(dataFromService);
}

const data = fetchData("Books");
console.log("Render data:", data);
```



calls main()

the stack is empty

The Call Stack

Let's take a look at this piece of code, do you see any problems?

```
function getFromService(data) {
  const start = new Date().getTime();
  while (new Date().getTime() < start + 10000);
  return `Data from service: ${data}`;
}

function fetchData(data) {
  const dataFromService = getFromService();
  return `Test ${dataFromService}`
}

const data = fetchData("Books");
console.log(data);
```

-> We need help from **the Run-time Environment**

The run-time environment

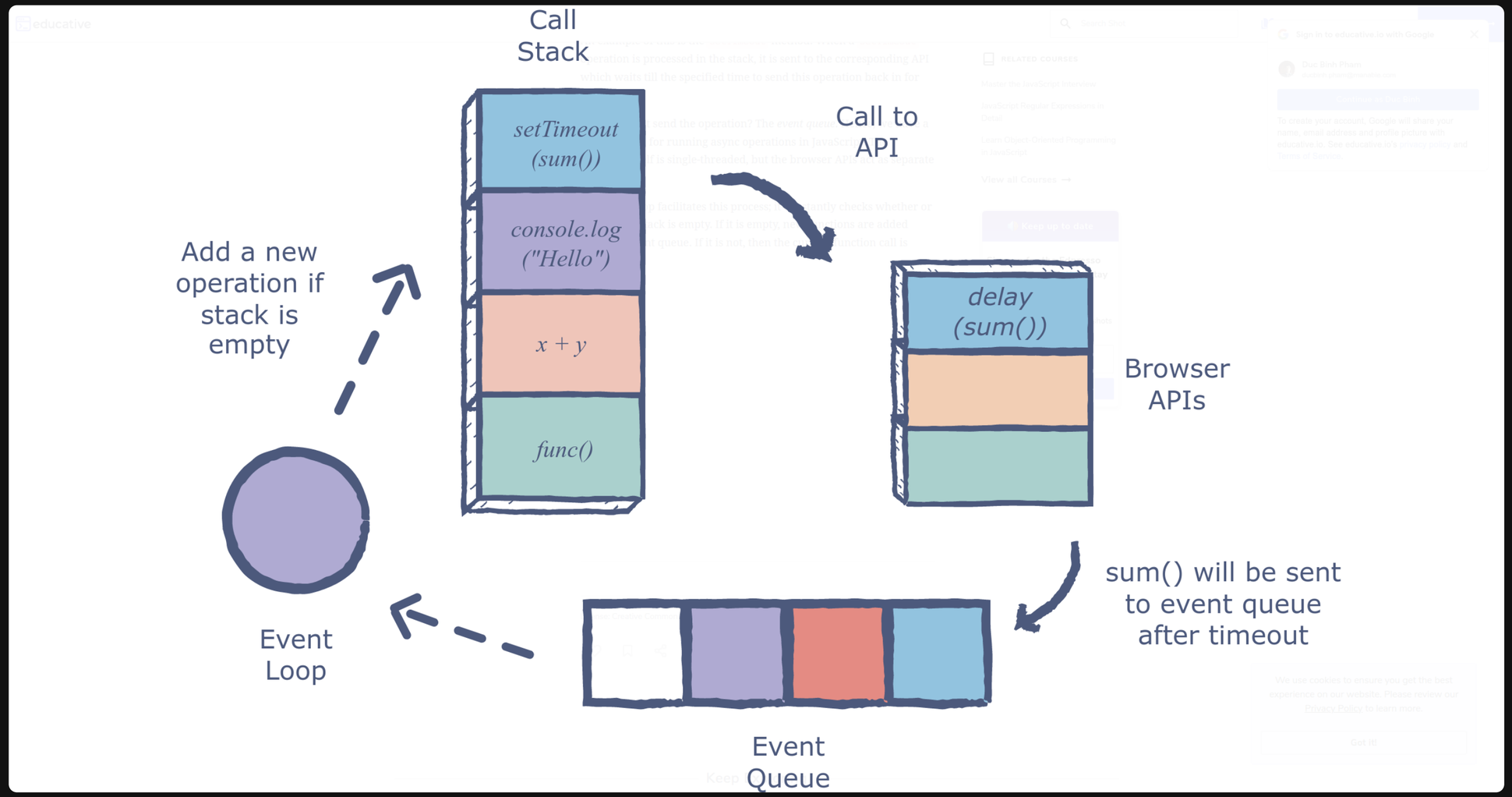
Our **JS code gets executed by a Javascript Engine**, each browser/environment uses a different engine

Javascript Engine	Browser
V8	Chrome and NodeJS
SpiderMonkey	Mozilla FireFox

The Javascript Engine runs inside a run-time environment which consists of the following components:

- ⚙️ JS Engine: executes our code using the **Call Stack**
- 🌐 Web API: provides DOM manipulation, AJAX, timer functions
- 📞 Callback/Event/Message Queue: holds the messages/events waiting to be executed
- 🔁 Event Loop: **constantly checking** if the **Call Stack** is empty or not. **If it's empty**, it pushes an event from the **Callback Queue** to the **Call Stack**

Event loop visualization



Educative.io

Event loop visualization

Let’s try to visualize this piece of code:

```
function fetchData(data, callback) {
    setTimeout(function fetchDataFromServer() {
      callback("Fetched Data" + data)
      },
      5000
    );
}

fetchData("Books", function log(fetchedData) {
  console.log(fetchedData)
})
fetchData("Students", function log(fetchedData) {
  console.log(fetchedData)
})
```

Magic Link

Conclusion, Q&A

The JavaScript Engine (Chrome V8/Mozilla SpiderMonkey) executes our code.
However, in order to perform what we usually ask it to do (DOM manipulation, AJAX, timer functions), it relies heavily on the run-time environment (Browsers/Node/Deno) Read more

Event loop is a **design pattern** or a piece of program that the run-time environment provides for us (and the engine) to use to give **the illusion of concurrency (things happening at the same time) when executing Javascript code**.
It does it by using the call stack, event queue and run-time API ~~, and probably more but I'm not aware of them (yet)~~

Please don't hesitate to ask if you have any questions

Acknowledgement

What the heck is the event loop anyway? | Philip Roberts | JSConf EU

JavaScript Internals: JavaScript engine, Run-time environment & setTimeout Web API

What is an event loop in JavaScript?