W2D1 - Asynchronous Control Flow

LIGHTHOUSE LABS

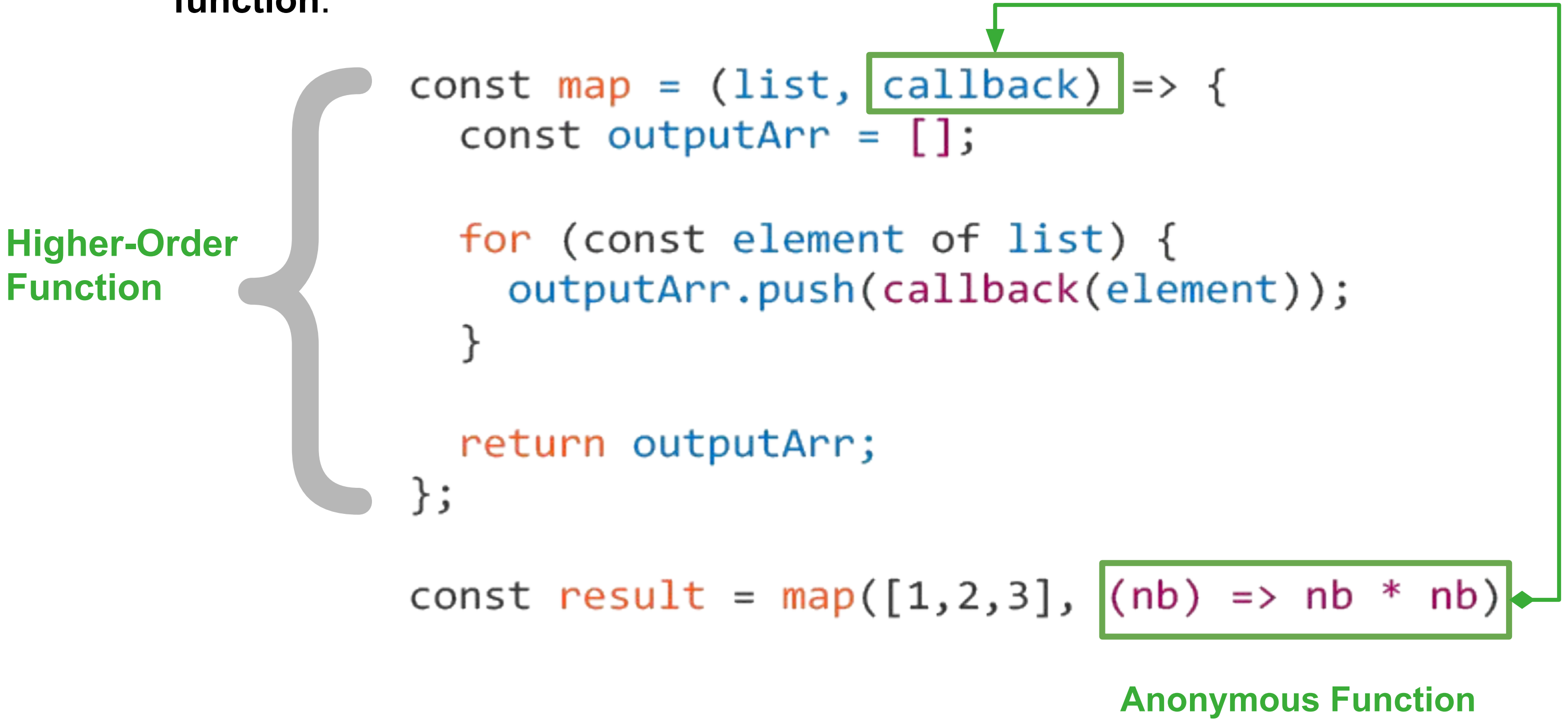
AGENDA

- Callback Recap
- Asynchronous Workflow
  - Demo
  - Event Loop
  - Events

LIGHTHOUSE LABS

Callback Recap

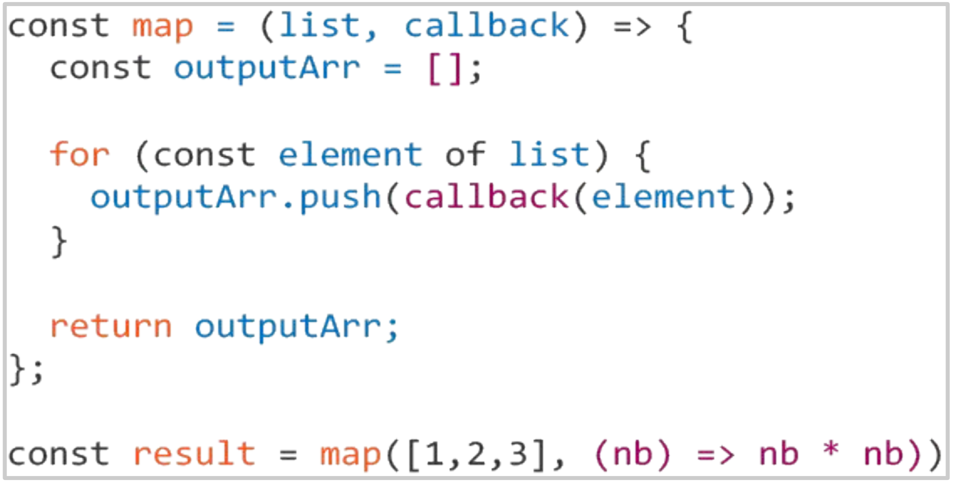
An **anonymous function** that is being passed as an argument to a **higher-order function**.



```
const map = (list, callback) => {
  const outputArr = [];

  for (const element of list) {
    outputArr.push(callback(element));
  }

  return outputArr;
};

const result = map([1,2,3], (nb) => nb * nb)
```

Callback Recap

Using the callback in the context of a map function makes the function **more modular**.



```
const map = (list, callback) => {
  const outputArr = [];

  for (const element of list) {
    outputArr.push(callback(element));
  }

  return outputArr;
};

const result = map([1,2,3], (nb) => nb * nb))
```

This is all **synchronous code**:

- each statement is executed sequentially one after the other

Callback Recap

Callbacks can have another purpose:

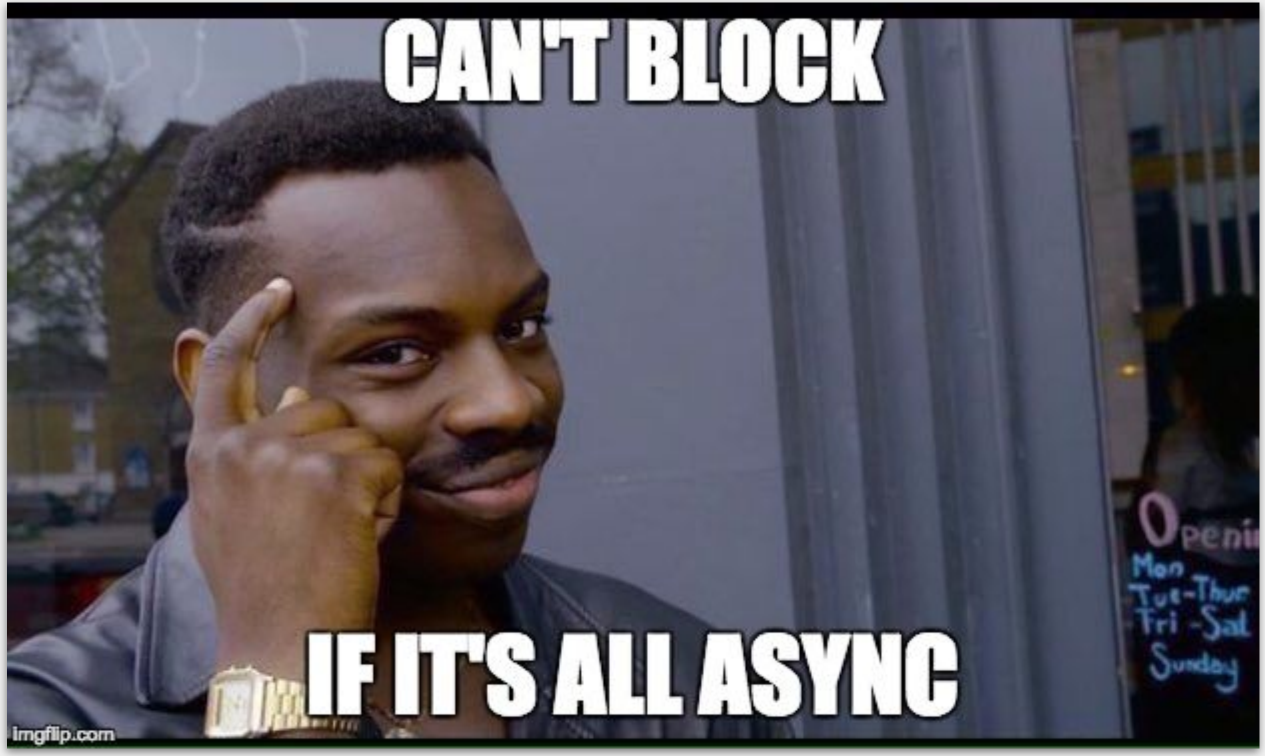
- Trigger some code execution after a delay or when an event occurs
- JavaScript can run code asynchronously!

Asynchronous Workflow

Why async code?

- Performing tasks that takes a long time such as reading a huge file, database operations, API calls, etc will **block the execution** for some time.
- It can cause performance problems.

Asynchronous Javascript allows for tasks to be performed **without blocking** the code execution.


CAN'T BLOCK
IF IT'S ALL ASYNC
imgflip.com
Openi
Mon
Tue-Thur
Fri -Sat
Sunday

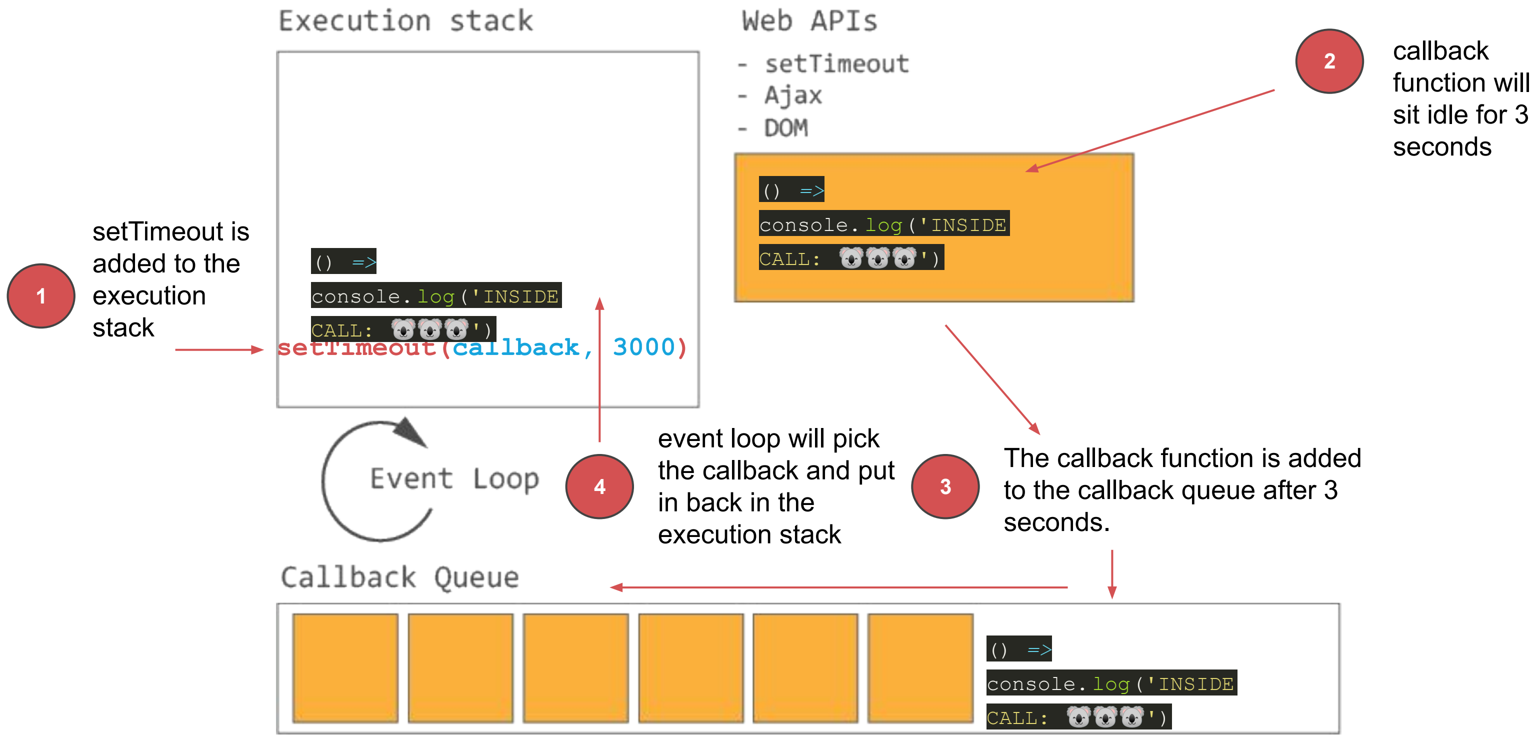
Asynchronous Workflow

A few examples...

```
const displayLater = (callback) => {

  setTimeout(() => {
    callback();
  }, 3000);

}

displayLater(() => console.log("Executing the console.log after 3000ms"));
```

Asynchronous Workflow

What is actually happening under the hood



```
() =>
console.log('INSIDE
CALL: 🐨🐨🐨')
setTimeout(callback, 3000)
```

1 setTimeout is added to the execution stack

2 callback function will sit idle for 3 seconds

3 The callback function is added to the callback queue after 3 seconds.

4 event loop will pick the callback and put in back in the execution stack

Asynchronous Workflow

Why JavaScript is using the event loop:

- It is single threaded
    - The execution of the instructions are done in a single sequence. One statement is processes at a time.
    - The opposite is multithreaded. Multiple parts of a program can be executed at the same time.
- It’s event based

Asynchronous Workflow

JavaScript is event-based:

- Events are actions or occurrences that happen in the app (ex. click on a button)
- Whenever an event occurs, we can programmatically code what we want to execute
- We're using callbacks again to trigger the execution of what we want to happen

Asynchronous Workflow

The event loop

Event Loop Demo: http://latentflip.com/loupe/

Questions?

LIGHTHOUSE LABS