ECMAScript Quick Start

Day02

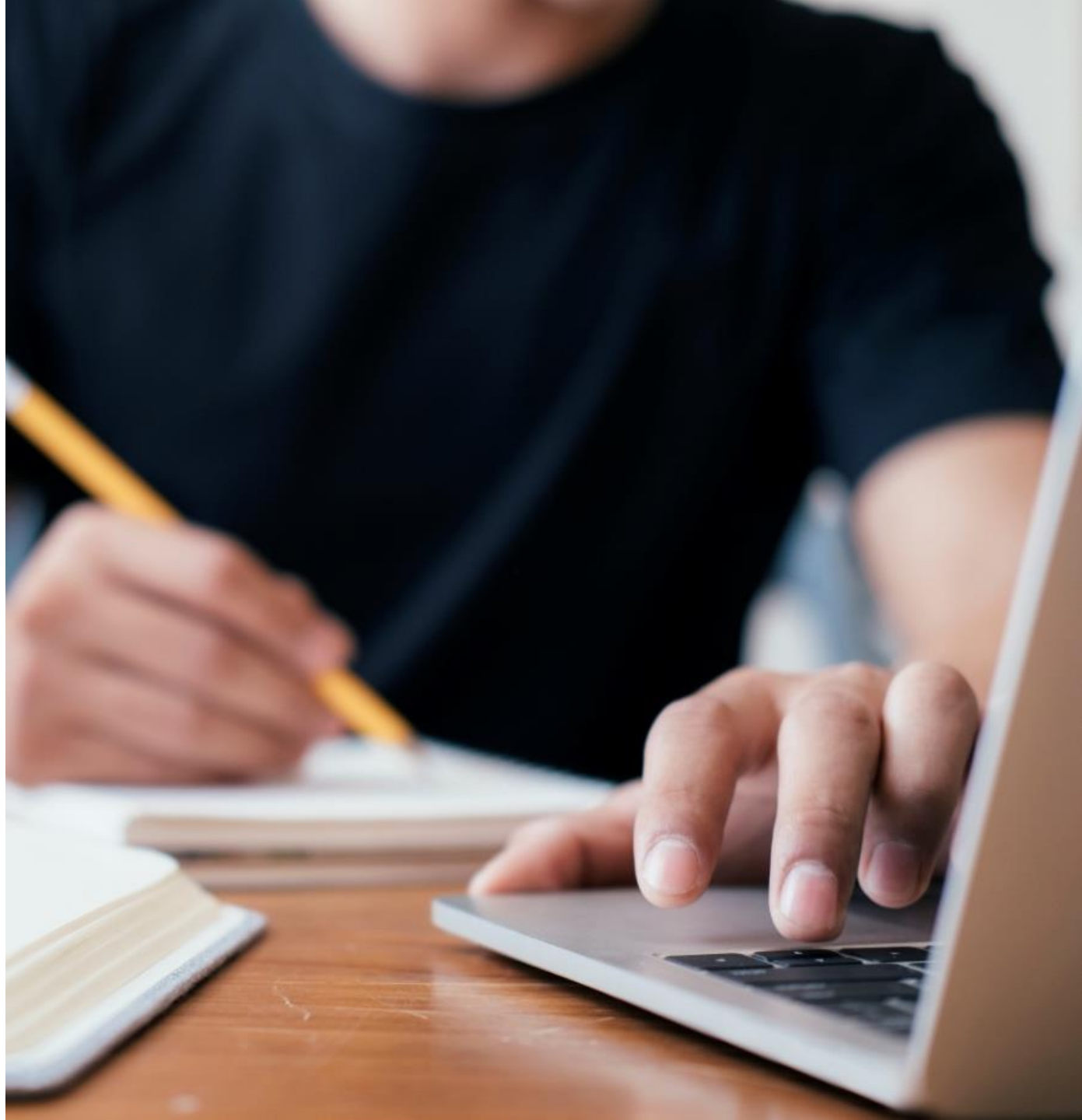
Axle Barr

Bootcamp

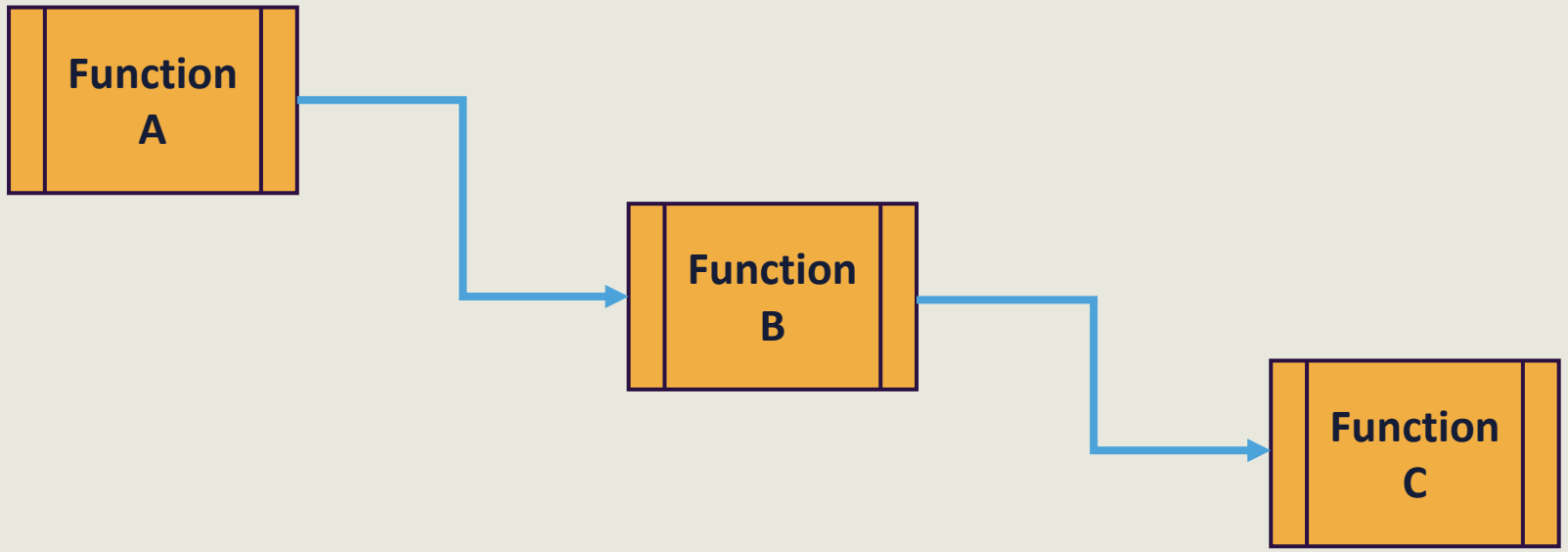
AGENDA

Day 1

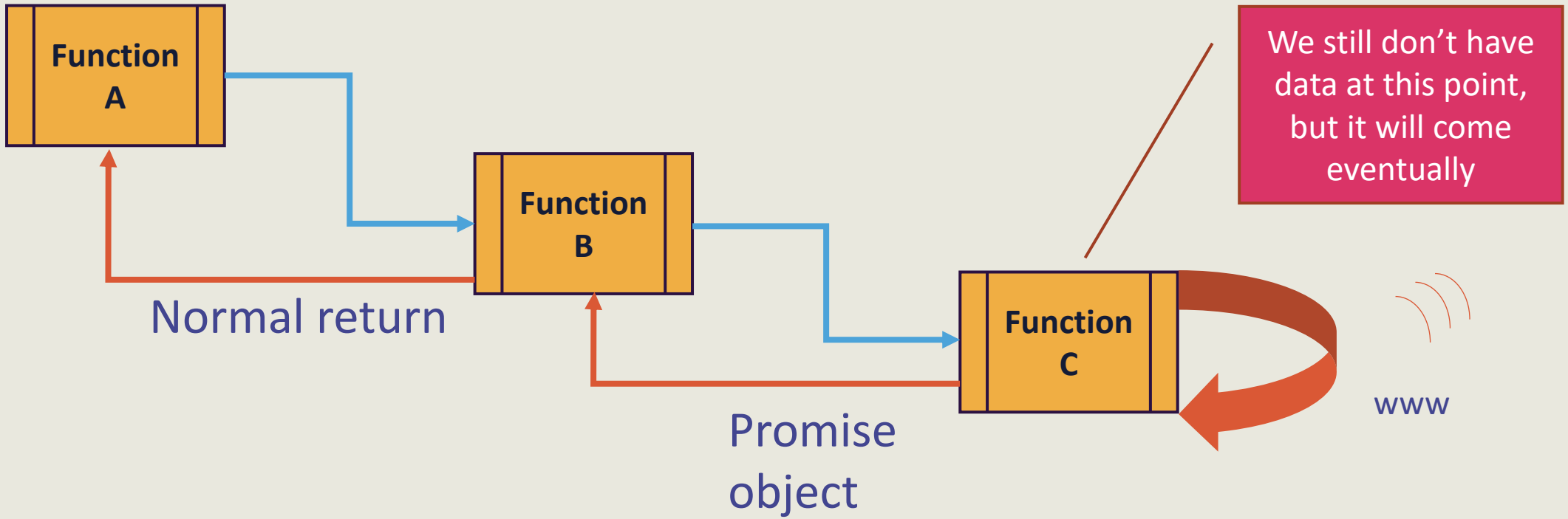
- Promisses



Synchronous/Asynchronous code



Synchronous/Asynchronous code



Interrupting the sequence

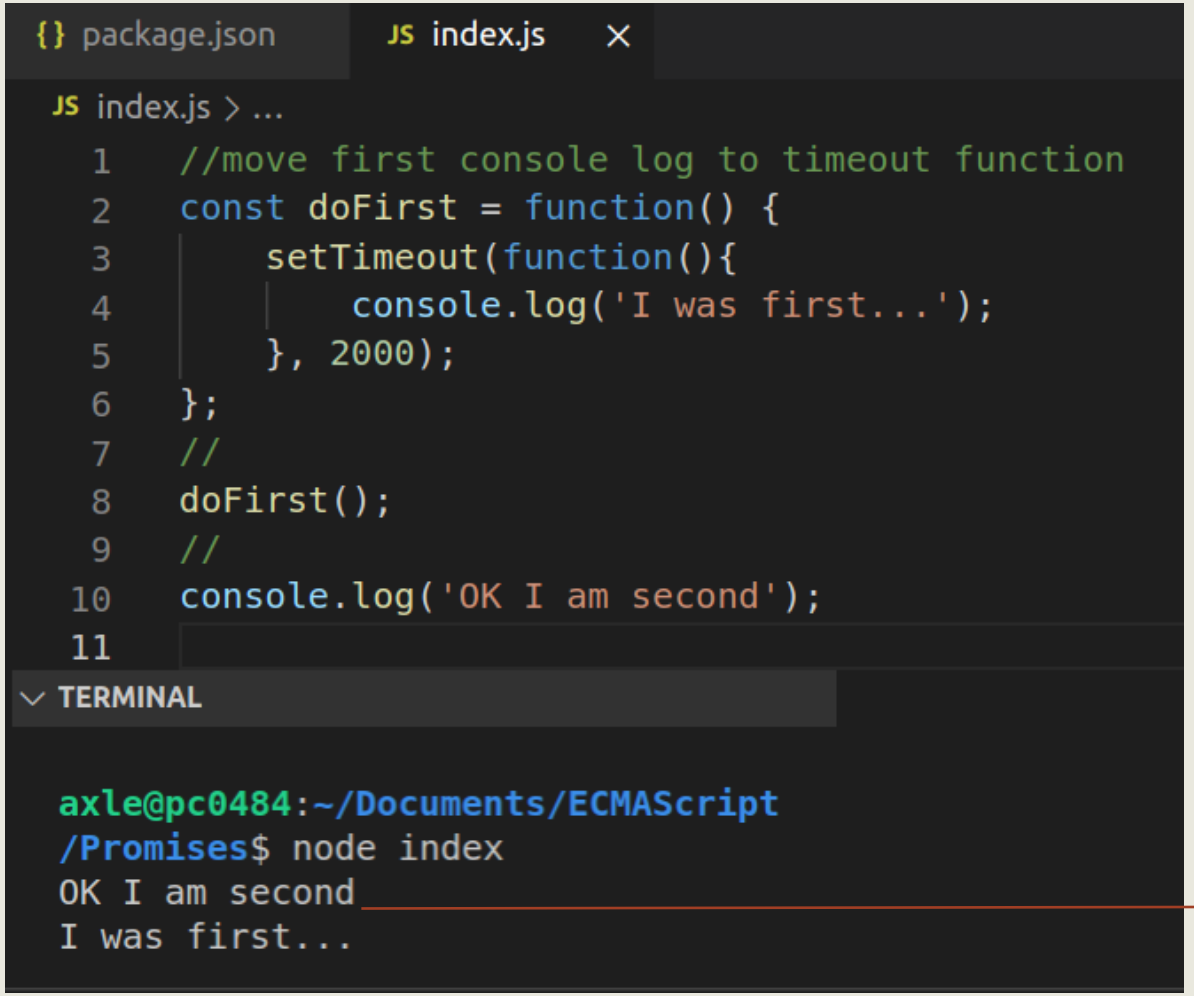
```
1   //promisses, starter code
2   const doFirst = function() {
3       console.log('I was first...
4   };
5   //
6   doFirst();
7   //
8   console.log('OK I am second');
9
```

TERMINAL

```
axle@pc0484:~/Documents/ECMAScript
/Promises$ node index
I was first...
OK I am second
```

Normal sequence, first…second..

Interrupting the sequence



The setTimeout() function has caused line 10 to execute first, NOT what we want

Introduce the Promise

Lets create a Promise object

```
//promisses
const doFirst = function() {
    return new Promise();
    setTimeout(function(){
        console.log('I was first...');
    }, 2000);
};
//
doFirst();
//
console.log('OK I am second');
```

Introduce the Promise

The constructor of the Promise takes a function that will return the actual promise object

```
//promisses
const doFirst = function() {
    return new Promise(function(){
        //promise returned here
    });
         setTimeout(function(){
                  console.log('I was first...');
         }, 2000);
};
//
doFirst();
//
console.log('OK I am second');
```

Introduce the Promise

The Promise function itself takes two parameters.

The first parameter will handle the positive of the *doFirst*() function and the second will handle any errors or rejection of the *doFirst* function()

```
//promisses
const doFirst = function() {
    return new Promise(function(resolve, reject){
        //promise returned here
    });
         setTimeout(function(){
                  console.log('I was first...');
         }, 2000);
};
//
doFirst();
//
console.log('OK I am second');
```

Introduce the Promise

On the positive side when the **resolve** happens, we want something to happen, we want to pass some data out of the function, so wrap the setTimeout() function with this Promise:

```
//promisses
const doFirst = function() {
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            console.log('I was first...');
        }, 2000);
    });
};
//
doFirst();
//
console.log('OK I am second');
```

Introduce the Promise

Instead of logging the data, resolve the data.

It will be logged later when the *doFirst*() function is called

```
//promisses
const doFirst = function() {
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            resolve('I was first...');
        }, 2000);
    });
};
//
doFirst();
//
console.log('OK I am second');
```

Introduce the Promise

Here we are logging the return from the *doFirst*() function, which if resolved will print the line:

I was first…

If you run this code you will not get the sentence, you will get a promise.

We need to some more work to extract the sentence (data) from the Promise

```
//promisses
const doFirst = function() {
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            resolve('I was first...');
        }, 2000);
    });
};
//
console.log(doFirst());
//
console.log('OK I am second');
```

Introduce the Promise

According to the documentation, in order to handle a promise object, you need to chain on a *then*() method and handle it from there.

Note: this is a common way to get the data, there are other ways

```
//promisses
const doFirst = function() {
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            resolve('I was first...');
        }, 2000);
    });
};
//
doFirst().then();
//
console.log('OK I am second');
```

Introduce the Promise

The chained then() method takes a parameter which must be a function.

The function inside of the *then*() method takes a parameter that acts like a bucket to catch anything thrown by the Promise object, so add a parameter in here that would catch the data AND log it for now.

At this point, we still have not solved the sequencing problem, the second line still appears first.

Solving this problem means moving the second line below the return from the promise. See next slide.

```
//promisses
const doFirst = function() {
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            resolve('I was first...');
        }, 2000);
    });
};
//
doFirst().then(function(data){
    console.log(data)
});
//
console.log('OK I am second');
```

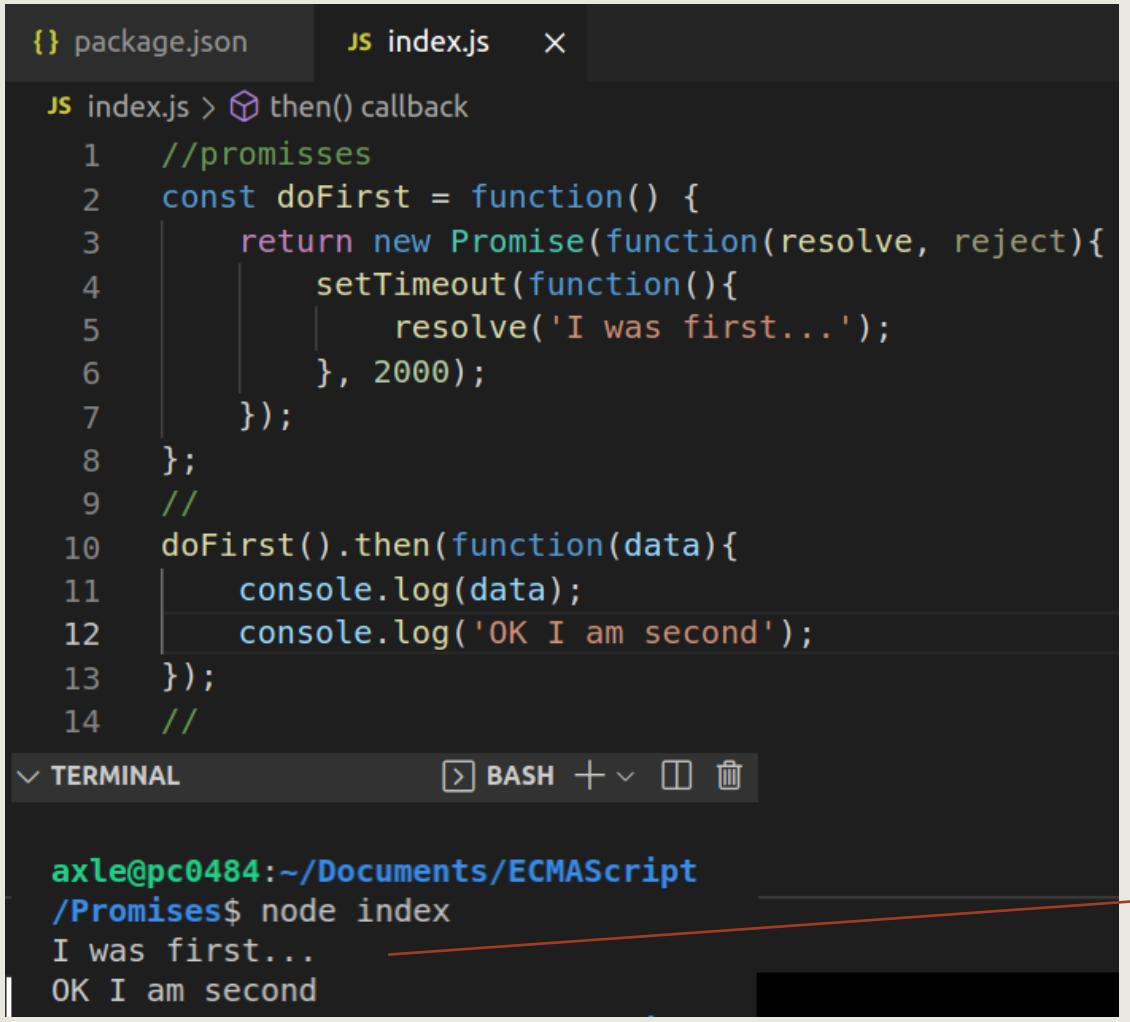
Introduce the Promise

Now the sequence is back to the way it was intended.

Both lines appear 2 seconds later, but the sequence is preserved.

```
//promisses
const doFirst = function() {
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            resolve('I was first...');
        }, 2000);
    });
};
//
doFirst().then(function(data){
    console.log(data)
    console.log('OK I am second');
});
//
```

Interrupting the sequence



Back to the original sequence, 2 seconds later but in the correct order

Real world example

At your terminal window, run the following command to install *node-fetch*:
npm install node-fetch

After running the command above, you will get a new folder called node_modules, this folder will contain the **node-fetch** module that we will import and use

Your package.json file will also change to reflect the installation we just did, but that is not important for this bootcamp

Real world example

Remove all the code from index.js and add in the following code:

```
import fetch from "node-fetch";

function getData(){
    fetch('https://jsonplaceholder.typicode.com/posts')
    .then((response) => response.json())
    .then((json) => console.log(json));
};

//

getData();
```

Error Handling

```
...
const doFirst = function() {
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            //resolve('I was first...');
            reject('error...');
        }, 2000);
    });
};
//
doFirst().then(function(data){
    console.log(data);
    console.log('OK I am second');
});
...
```

This code cannot handle a rejection

What if the promise was rejected?

Error Handling

```
...
setTimeout(function(){
        //resolve('I was first...');
        reject('error...');
    }, 2000);
  });
};
//
doFirst()
.then(function(data){
  console.log(data);
  console.log('OK I am second');
})
.catch(function(err){
  console.log(err);
});
...
```

Add a catch() function to handle any errors

Note: the then function was moved lower but was not changed

ECMAScript 2018

```
...
doFirst()
.then(function(data){
    console.log(data);
    console.log('OK I am second');
})
.catch(function(err){
    console.log(err);
}).finally(function(){
    console.log("Promise was settled");
});
...
```

ECMAScript 2018 introduced the finally function for clean up purposes