# Week 2.1

1. Callbacks,
2. Async functions
3. Promises
4. JS functions (map, filter)
5. Assignment

If you already know this, feel free to skip this video!
More of a revision video
No bounties today

# Week 2.1
# Before we start

We have covered this

## Foundation
Foundation Javascript, async nature of JS
Node.js and its runtime
Databases (NoSQL/SQL)
Mongo and Postgres deep dive
Typescript beginner to advance

## Backend
Backend communication protocols
Express basic to advance
ORMs
Middlewares, routes, status codes, global catches
Zod
MonoRepos, turborepo
Serverless Backends
OpenAPI Spec
Autogenerated clients
Authentication using external libraries
Scaling Node.js, performance benchmarks
Deploying npm packages

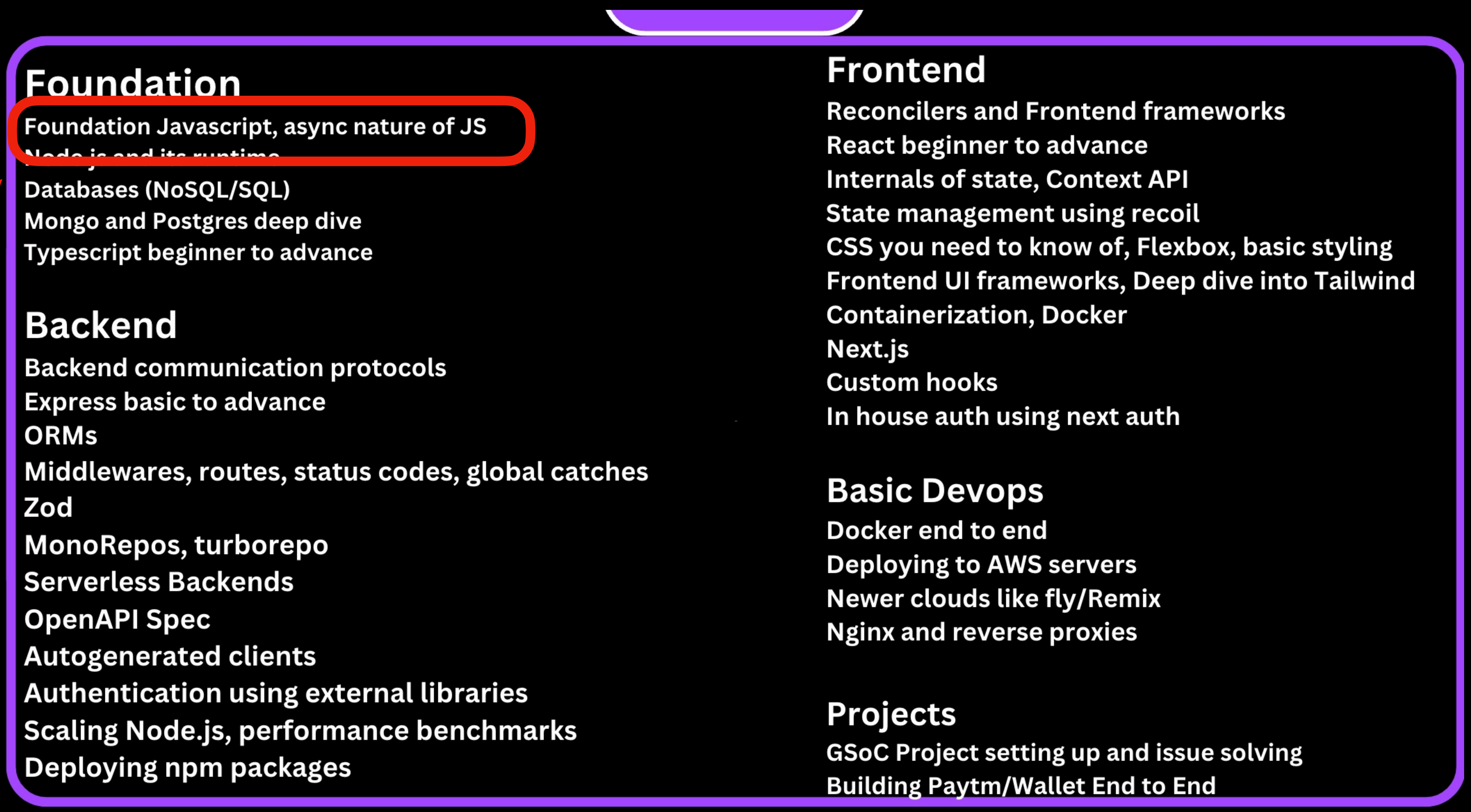
## Frontend
Reconcilers and Frontend frameworks
React beginner to advance
Internals of state, Context API
State management using recoil
CSS you need to know of, Flexbox, basic styling
Frontend UI frameworks, Deep dive into Tailwind
Containerization, Docker
Next.js
Custom hooks
In house auth using next auth

## Basic Devops
Docker end to end
Deploying to AWS servers
Newer clouds like fly/Remix
Nginx and reverse proxies

## Projects
GSoC Project setting up and issue solving
Building Paytm/Wallet End to End

# Week 2.1
# Before we start

We have covered this



**Foundation**

Foundation Javascript, async nature of JS
Node.js and its runtime
Databases (NoSQL/SQL)
Mongo and Postgres deep dive
Typescript beginner to advance

**Backend**

Backend communication protocols
Express basic to advance
ORMs
Middlewares, routes, status codes, global catches
Zod
MonoRepos, turborepo
Serverless Backends
OpenAPI Spec
Autogenerated clients
Authentication using external libraries
Scaling Node.js, performance benchmarks
Deploying npm packages

**Frontend**

Reconcilers and Frontend frameworks
React beginner to advance
Internals of state, Context API
State management using recoil
CSS you need to know of, Flexbox, basic styling
Frontend UI frameworks, Deep dive into Tailwind
Containerization, Docker
Next.js
Custom hooks
In house auth using next auth

**Basic Devops**

Docker end to end
Deploying to AWS servers
Newer clouds like fly/Remix
Nginx and reverse proxies

**Projects**

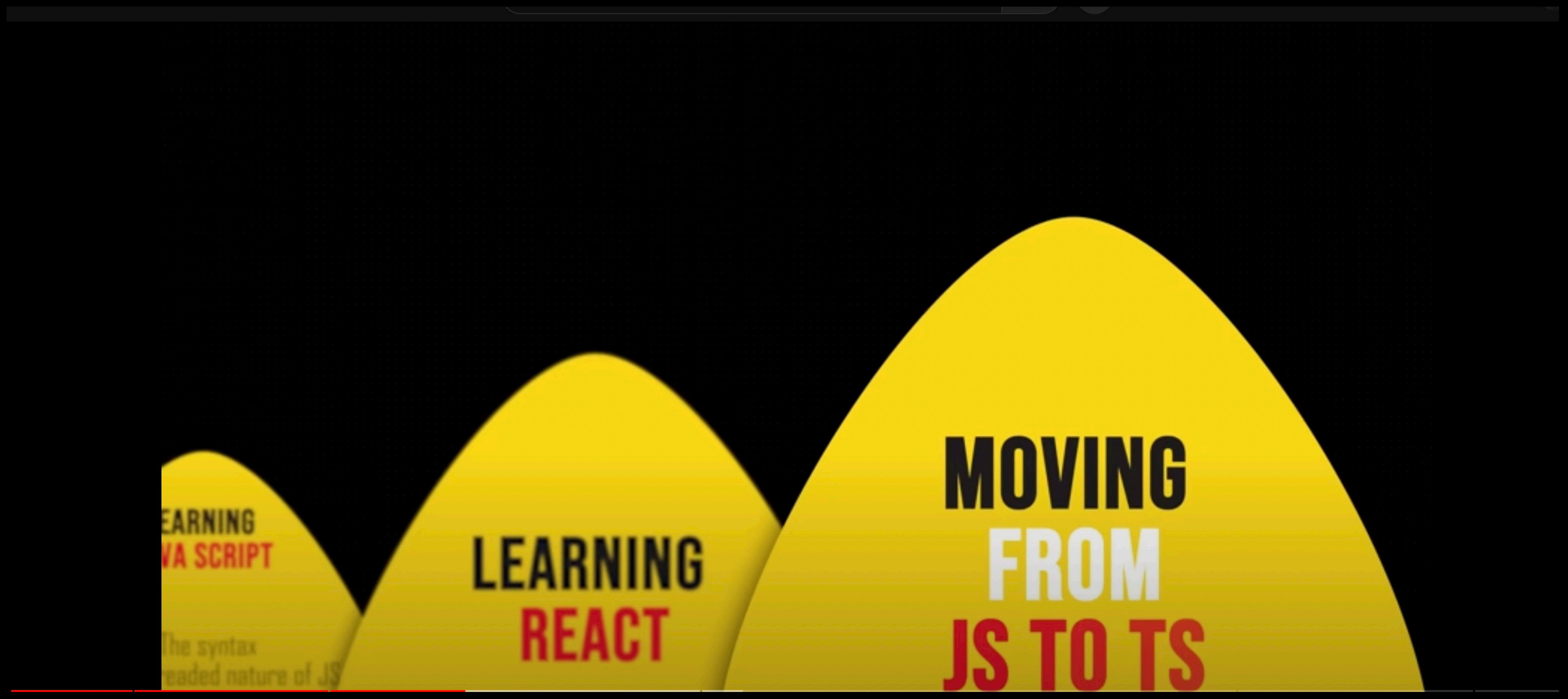
GSoC Project setting up and issue solving
Building Paytm/Wallet End to End

For people who are struggling here -

1. Try going through the offline videos
2. This (esp promises) gets better with practise
3. As the weeks progress, if you are consistent the syntax will fall into place
4. Consistency is everything, don't drop off
5. You need one eureka moment at which you get async programming and then the ride is smooth until React

There are 3 big humps in Full stack

1. Async nature of JS
2. React
3. JS to TS



We are here

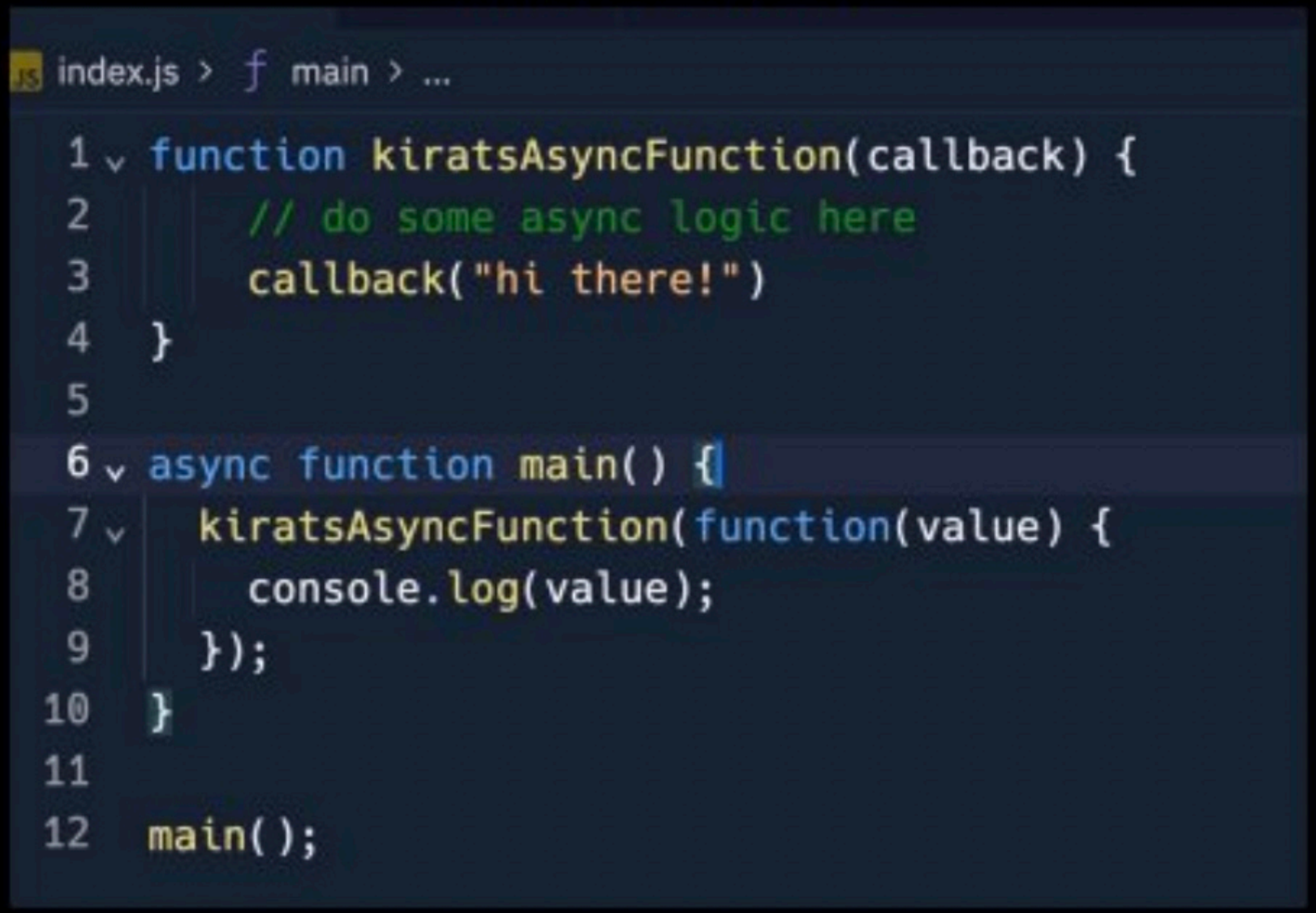
## This is the gist of the hump
## If you understand this, you are sorted
## If not, there is a knowledge gap

**1.5 | Async, Await and Promises**

Aynsc await syntax

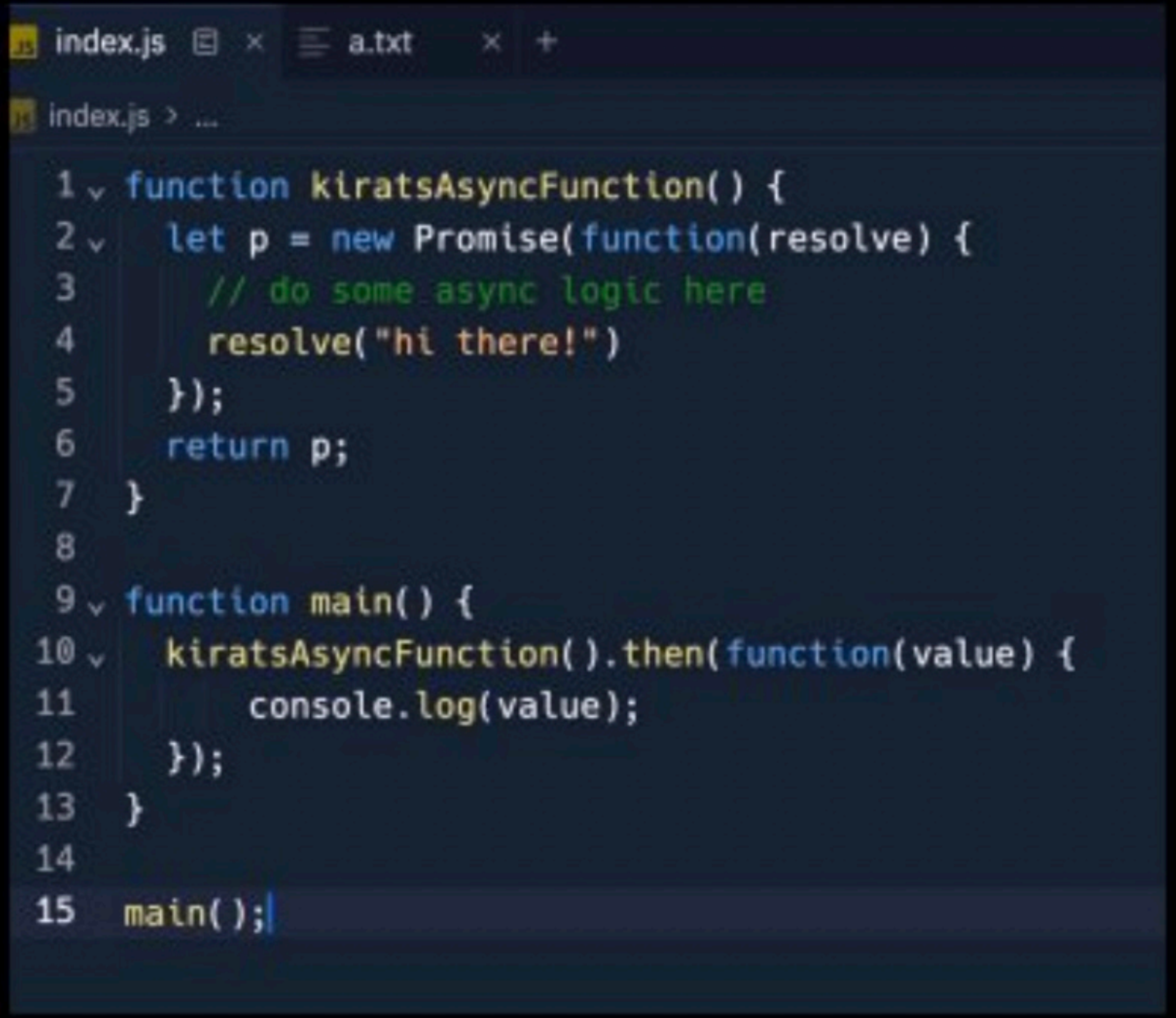
In fact, all three are very similar (becomes more manageable as you move to the right

**Callback syntax**



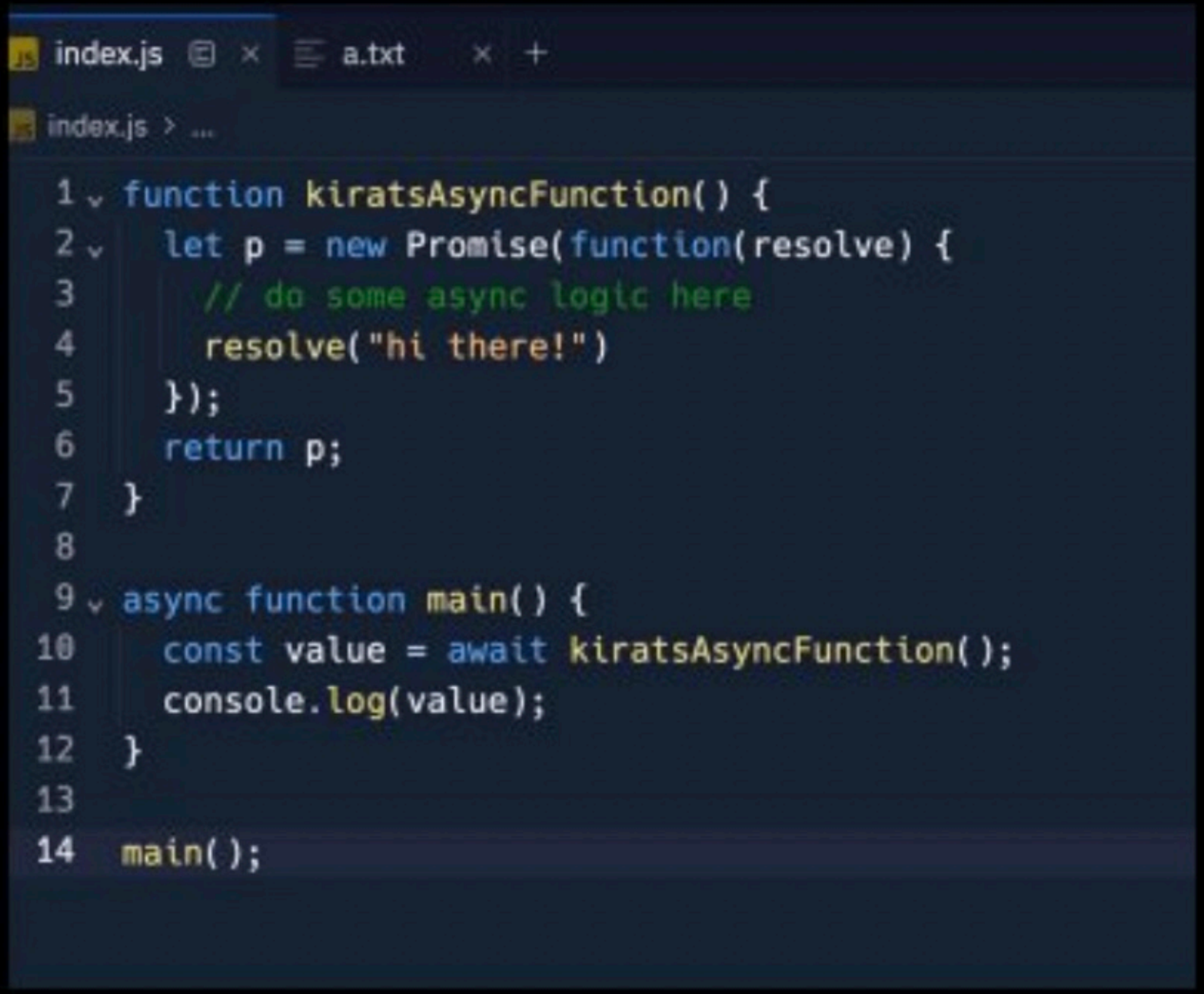
```javascript
function kiratsAsyncFunction(callback) {
    // do some async logic here
    callback("hi there!")
}

async function main() {
  kiratsAsyncFunction(function(value) {
    console.log(value);
  });
}

main();
```

**Promise (then) syntax**



```javascript
function kiratsAsyncFunction() {
  let p = new Promise(function(resolve) {
    // do some async logic here
    resolve("hi there!")
  });
  return p;
}

function main() {
  kiratsAsyncFunction().then(function(value) {
      console.log(value);
  });
}

main();
```

**Async/await syntax**



```javascript
function kiratsAsyncFunction() {
  let p = new Promise(function(resolve) {
    // do some async logic here
    resolve("hi there!")
  });
  return p;
}

async function main() {
  const value = await kiratsAsyncFunction();
  console.log(value);
}

main();
```

## Pre-requisites - Functions, var, const, data types in JS

# What did we cover until now?

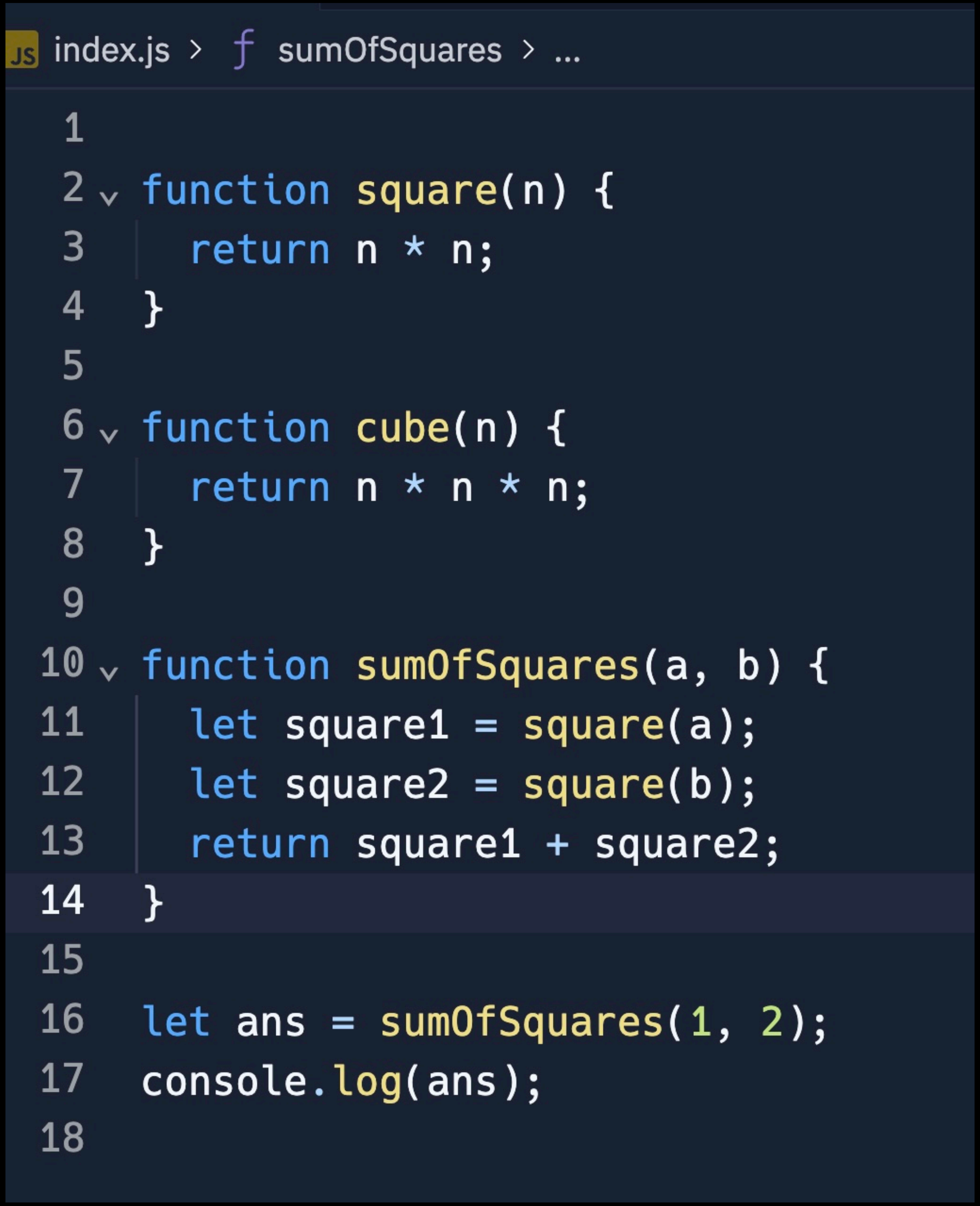
Live session 1 - Computers, Basic JS syntax, intro to callbacks
Offline session 1 - Simple JS APIs (Classes, Date class, string functions, object functions)
Offline session 2 - Loops, functions and callback functions
Offline session 3 - Async functions, JS Architecture and Promises

## What all we need to know Before we proceed to HTTP?

**Definitely - callback**
**Good to know - JS Architecture, Promises**

# Callbacks

Look at the code on the right, can you understand it?



index.js > sumOfSquares > ...

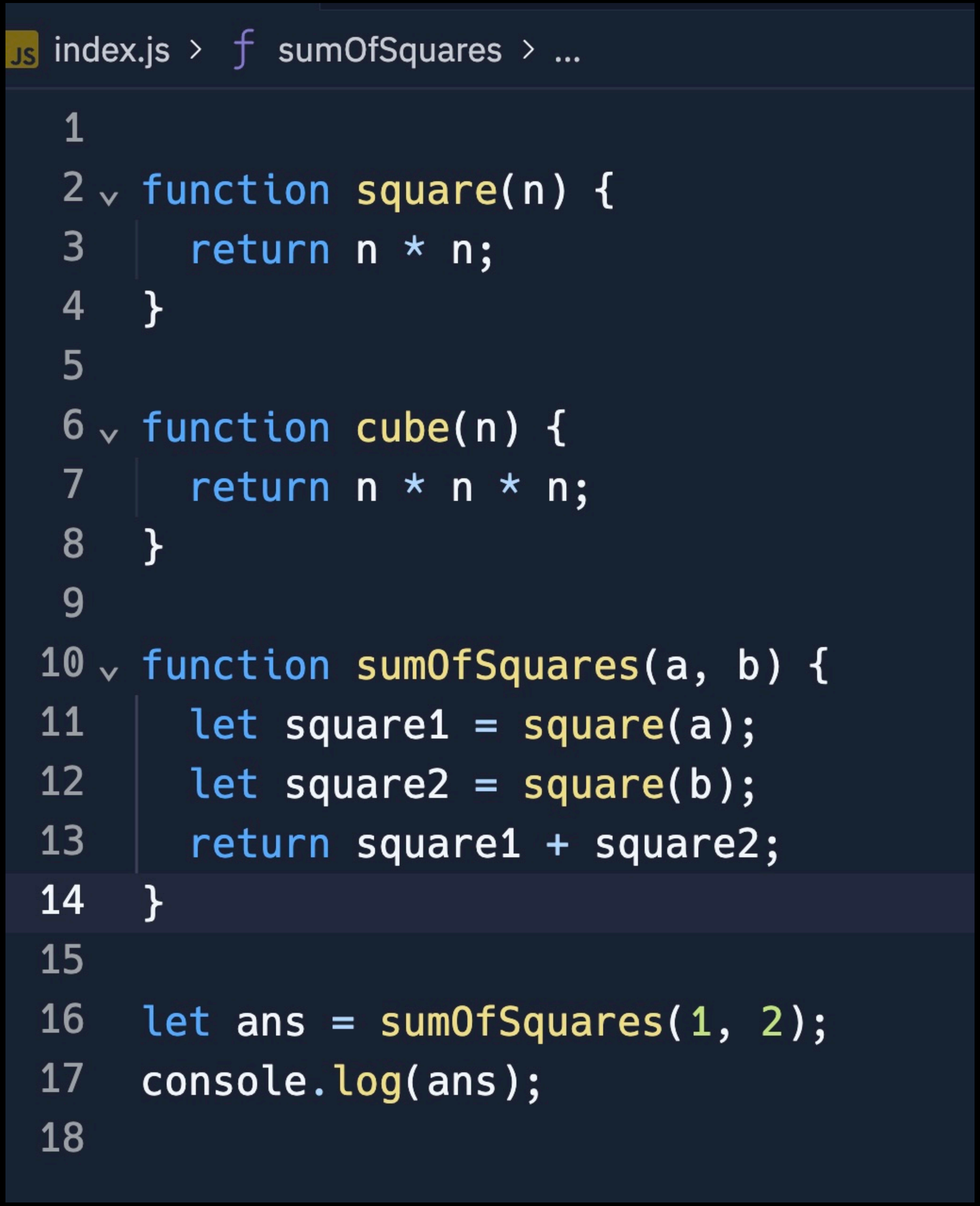
```javascript

function square(n) {
  return n * n;
}

function cube(n) {
  return n * n * n;
}

function sumOfSquares(a, b) {
  let square1 = square(a);
  let square2 = square(b);
  return square1 + square2;
}

let ans = sumOfSquares(1, 2);
console.log(ans);

```

https://gist.github.com/hkirat/1809806cb30e5dbdfafe0ee170b8437d

# Callbacks

Look at the code on the right, can you understand it?

Now add a sumOfCube function to it



index.js > sumOfSquares > ...

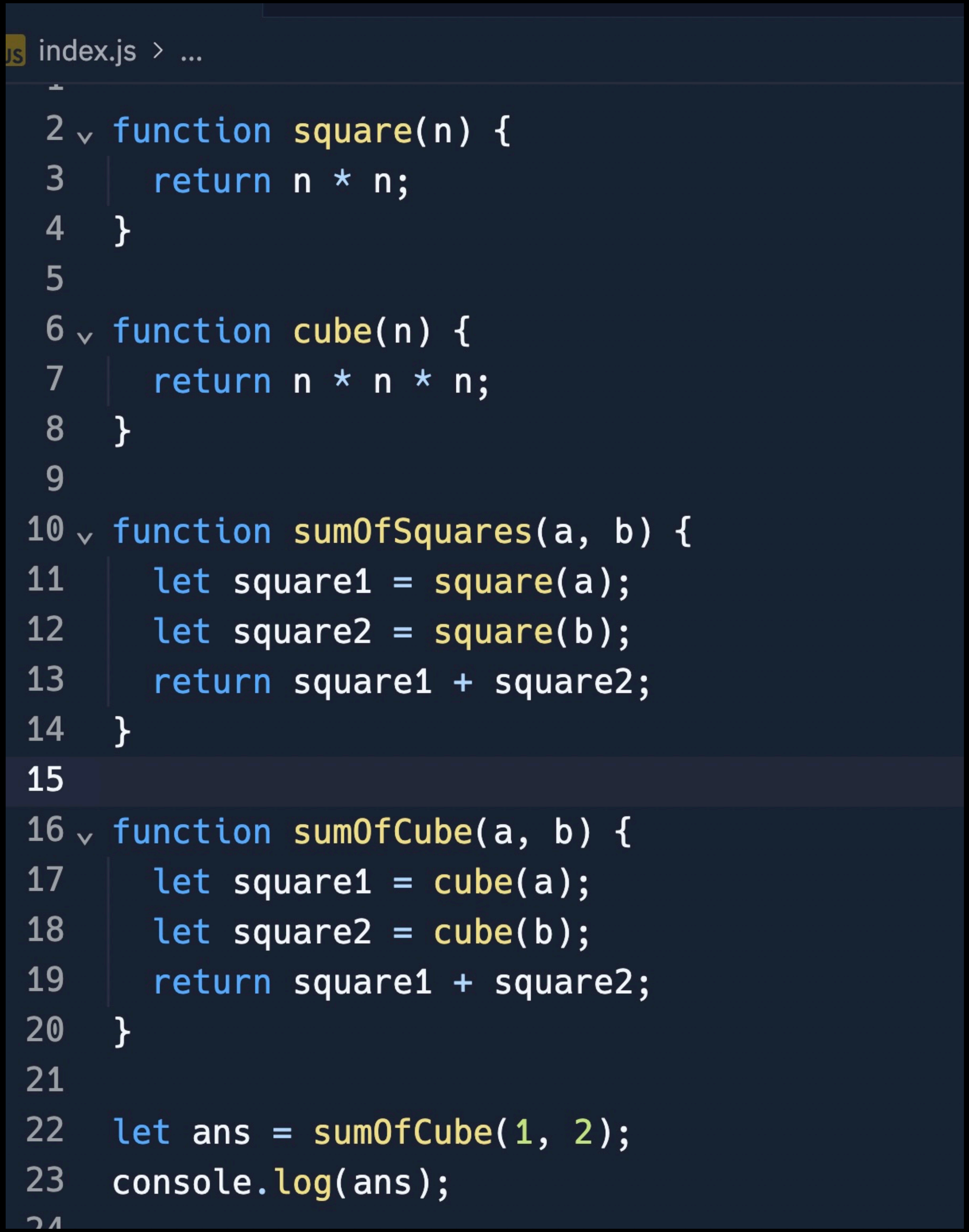
```javascript

function square(n) {
  return n * n;
}

function cube(n) {
  return n * n * n;
}

function sumOfSquares(a, b) {
  let square1 = square(a);
  let square2 = square(b);
  return square1 + square2;
}

let ans = sumOfSquares(1, 2);
console.log(ans);

```

https://gist.github.com/hkirat/1809806cb30e5dbdfafe0ee170b8437d

# Callbacks

Look at the code on the right, can you understand it?

Now add a sumOfCube function to it



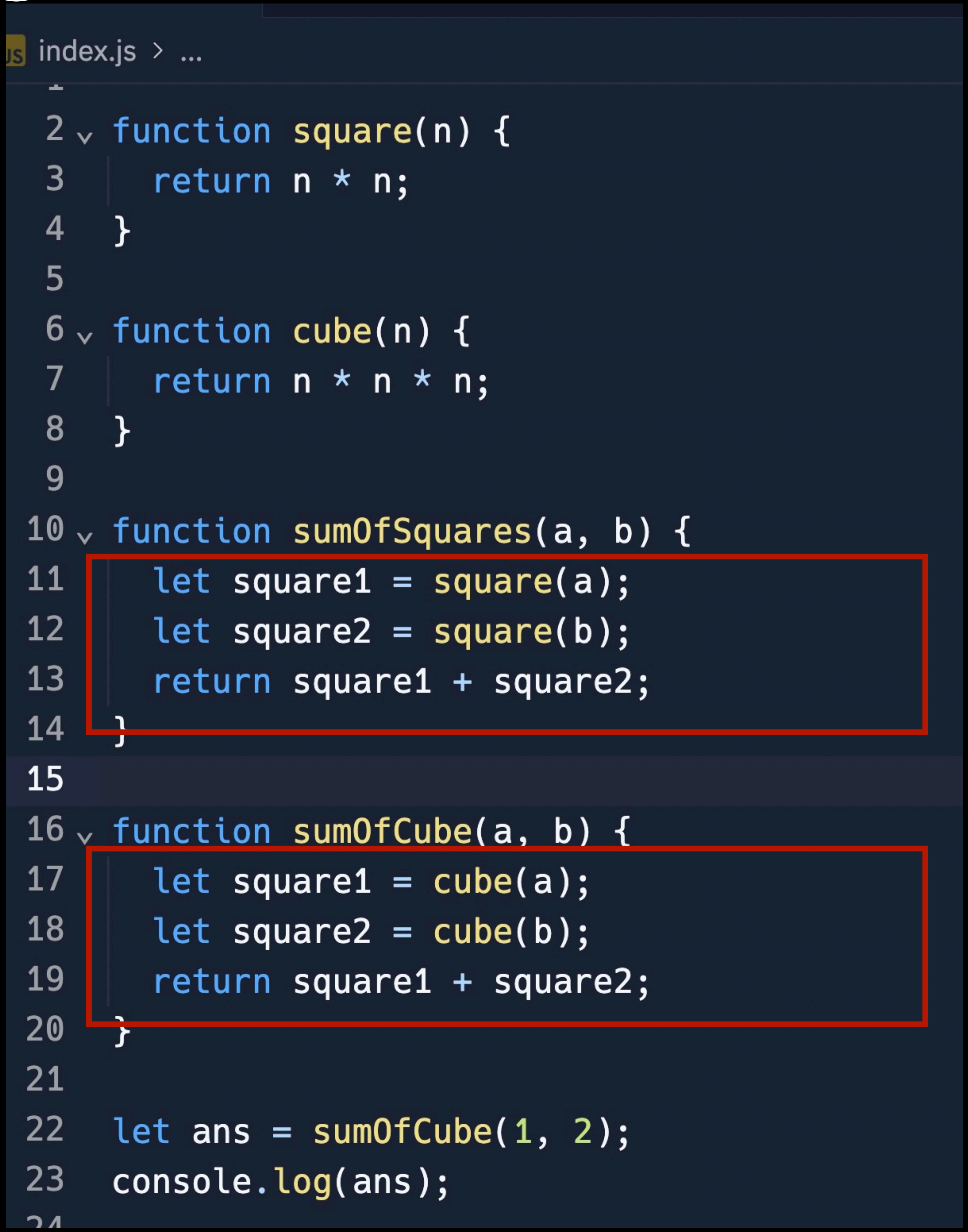
index.js > ...

```javascript
function square(n) {
  return n * n;
}

function cube(n) {
  return n * n * n;
}

function sumOfSquares(a, b) {
  let square1 = square(a);
  let square2 = square(b);
  return square1 + square2;
}

function sumOfCube(a, b) {
  let square1 = cube(a);
  let square2 = cube(b);
  return square1 + square2;
}

let ans = sumOfCube(1, 2);
console.log(ans);
```

https://gist.github.com/hkirat/aa8d262eff8bede7475e118004ba50b1

# Callbacks

Problem? Code repetition



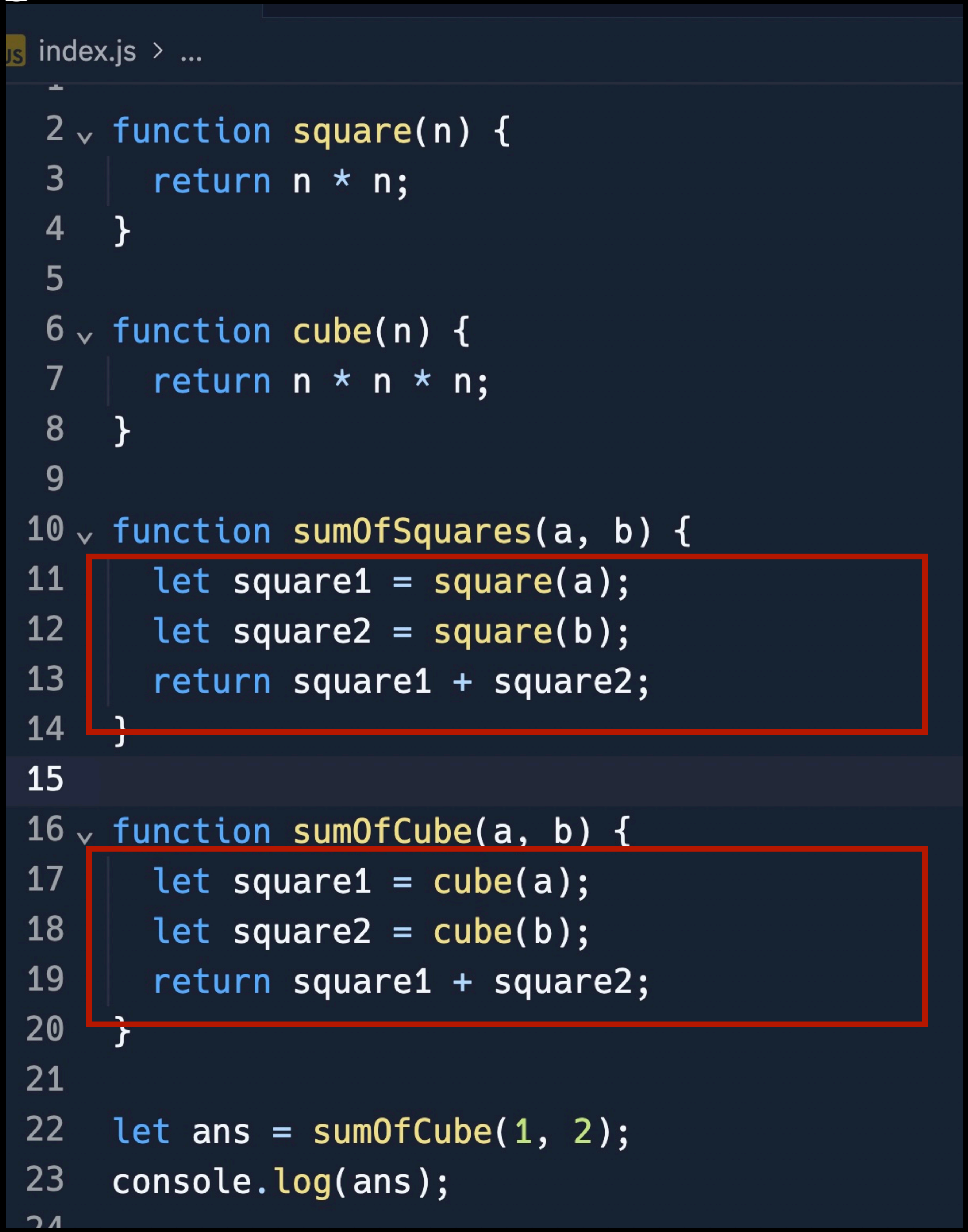
index.js > ...

```javascript
function square(n) {
  return n * n;
}

function cube(n) {
  return n * n * n;
}

function sumOfSquares(a, b) {
  let square1 = square(a);
  let square2 = square(b);
  return square1 + square2;
}

function sumOfCube(a, b) {
  let square1 = cube(a);
  let square2 = cube(b);
  return square1 + square2;
}

let ans = sumOfCube(1, 2);
console.log(ans);
```

https://gist.github.com/hkirat/aa8d262eff8bede7475e118004ba50b1

# Callbacks

**Problem? Code repetition**
**Can you create a single function (squareOfSomething) that does**
**This logic on a function it gets as an input**

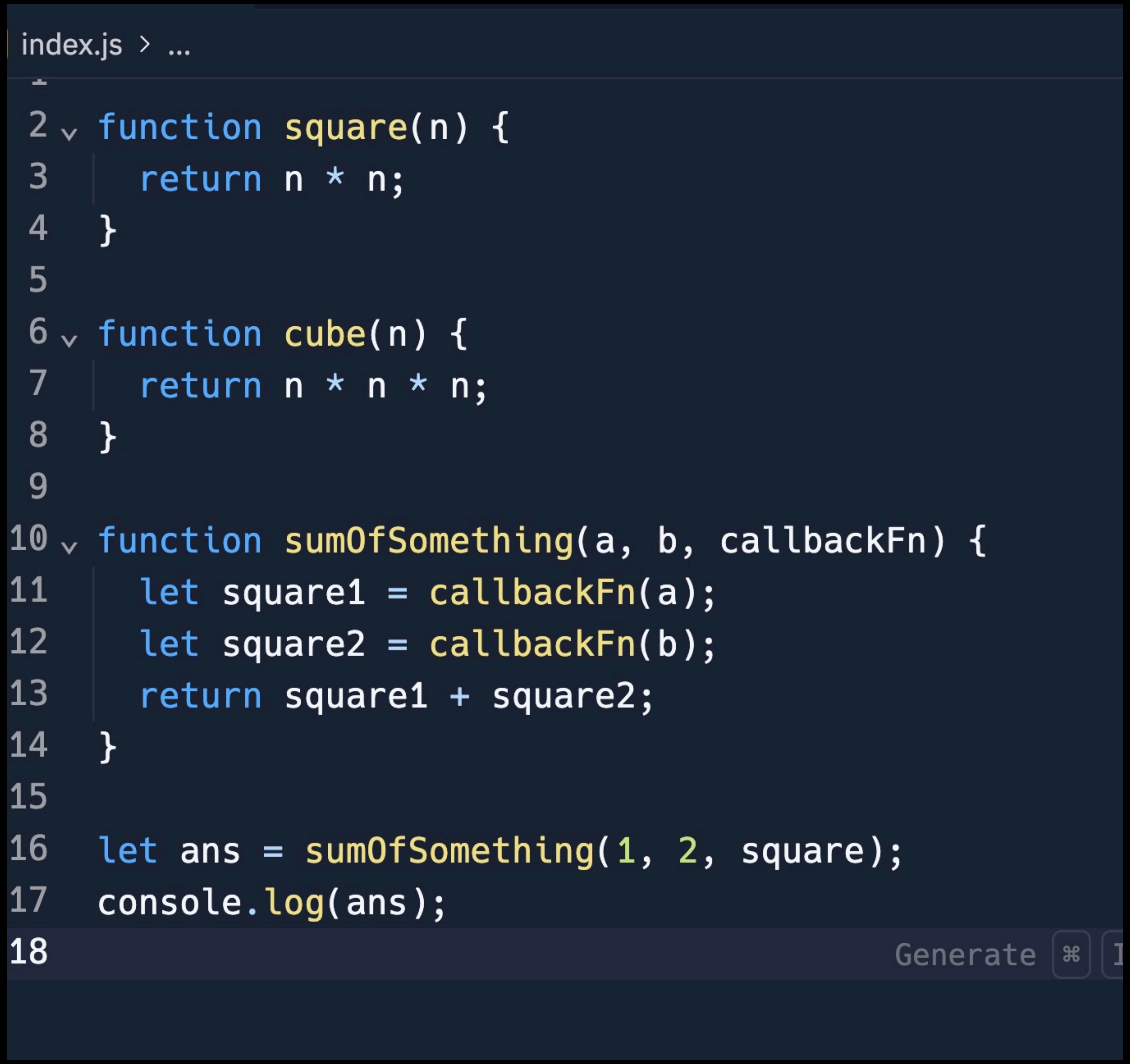

index.js > ...

```javascript
function square(n) {
  return n * n;
}

function cube(n) {
  return n * n * n;
}

function sumOfSquares(a, b) {
  let square1 = square(a);
  let square2 = square(b);
  return square1 + square2;
}

function sumOfCube(a, b) {
  let square1 = cube(a);
  let square2 = cube(b);
  return square1 + square2;
}

let ans = sumOfCube(1, 2);
console.log(ans);
```

https://gist.github.com/hkirat/aa8d262eff8bede7475e118004ba50b1

# Callbacks

**Problem? Code repetition**
**Can you create a single function (squareOfSomething) that does**
**This logic on a function it gets as an input**



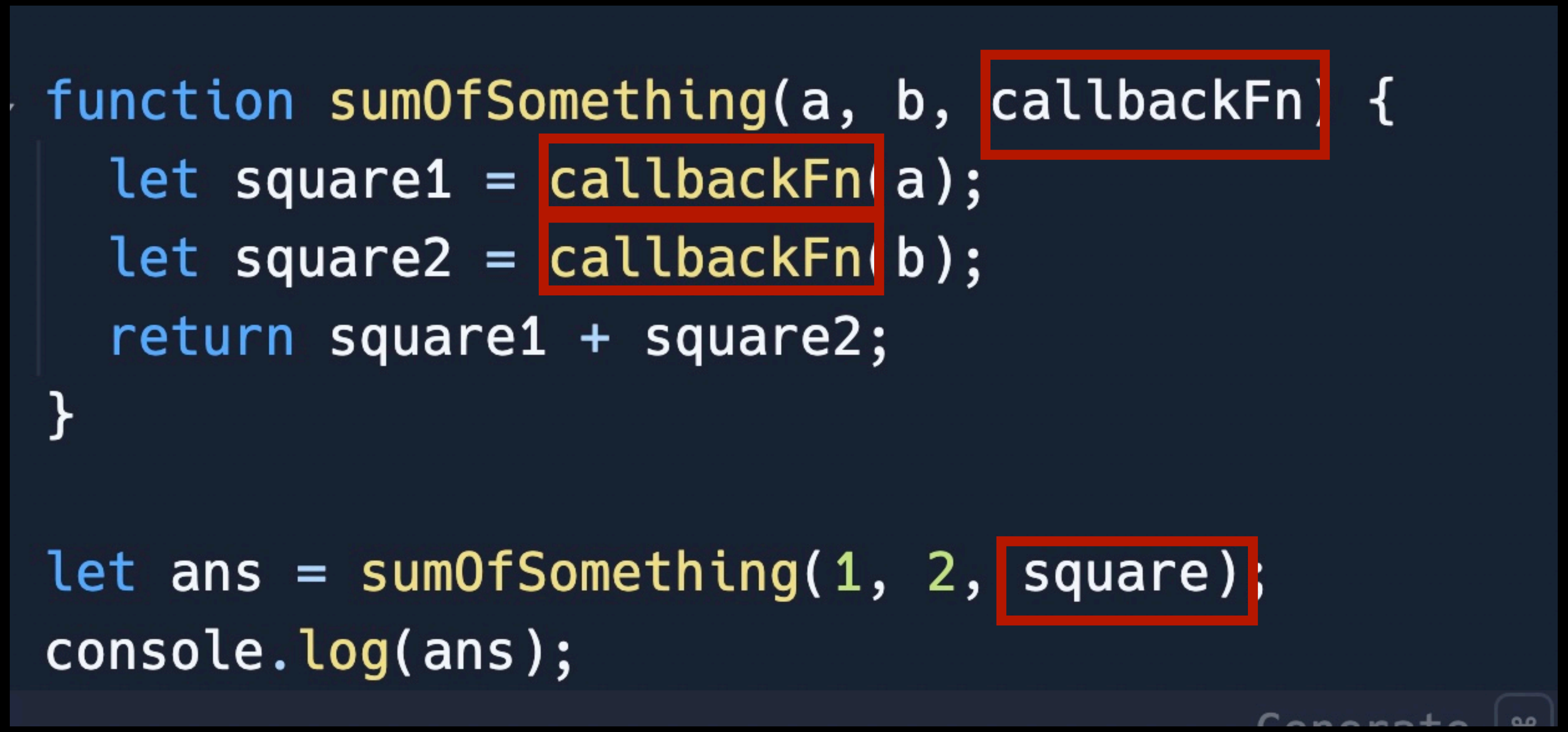
index.js > ...

```javascript
function square(n) {
  return n * n;
}

function cube(n) {
  return n * n * n;
}

function sumOfSomething(a, b, callbackFn) {
  let square1 = callbackFn(a);
  let square2 = callbackFn(b);
  return square1 + square2;
}

let ans = sumOfSomething(1, 2, square);
console.log(ans);
```

**https://gist.github.com/hkirat/aa8d262eff8bede7475e118004ba50b1**

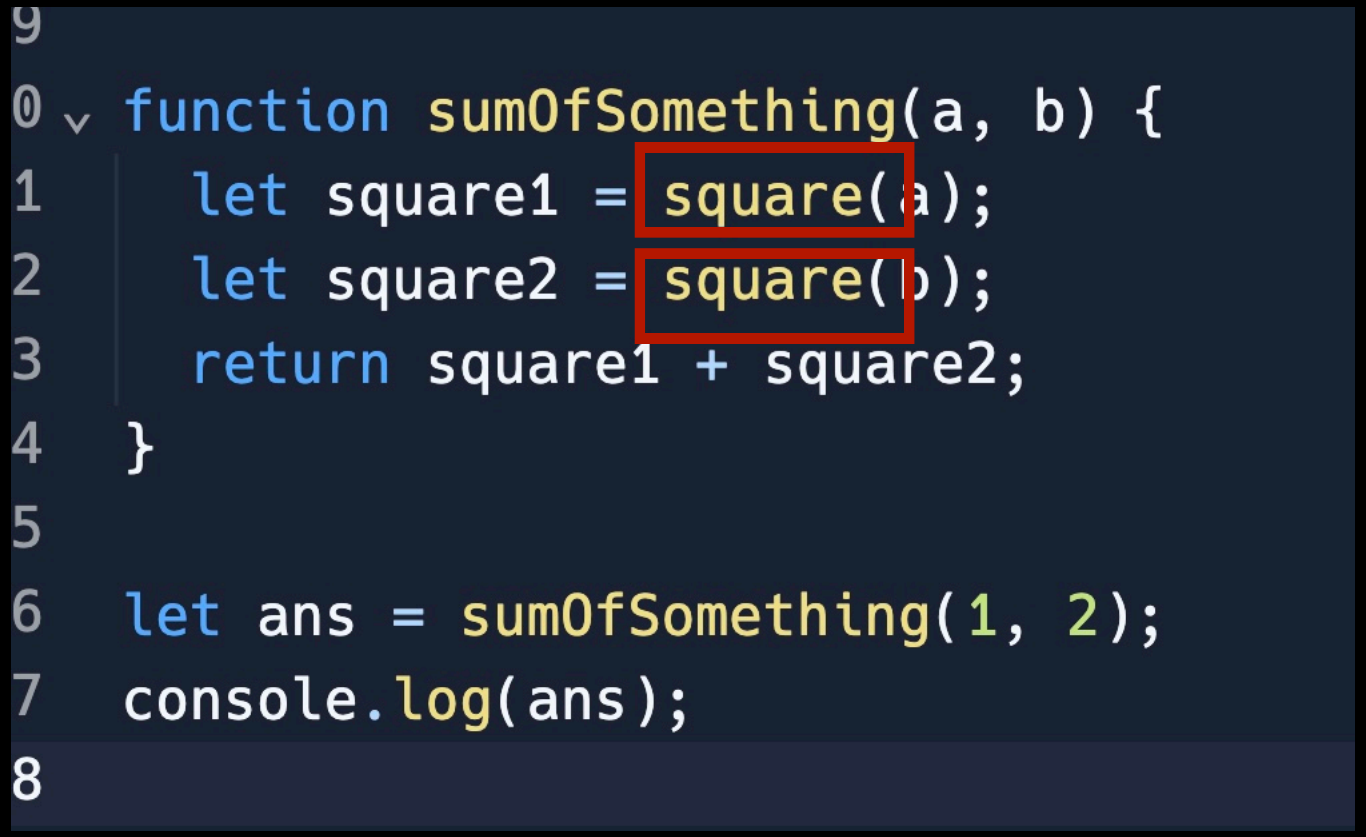
# Callbacks

**Both are the same**



```
function sumOfSomething(a, b, callbackFn) {
  let square1 = callbackFn(a);
  let square2 = callbackFn(b);
  return square1 + square2;
}

let ans = sumOfSomething(1, 2, square);
console.log(ans);
```

$$\longrightarrow$$



```
function sumOfSomething(a, b) {
  let square1 = square(a);
  let square2 = square(b);
  return square1 + square2;
}

let ans = sumOfSomething(1, 2);
console.log(ans);
```

1. Callbacks,
2. Async functions
3. Promises
4. JS functions
5. Assignment

## 2. Async functions

# 2. Async functions

**What is async? - Asynchronous**

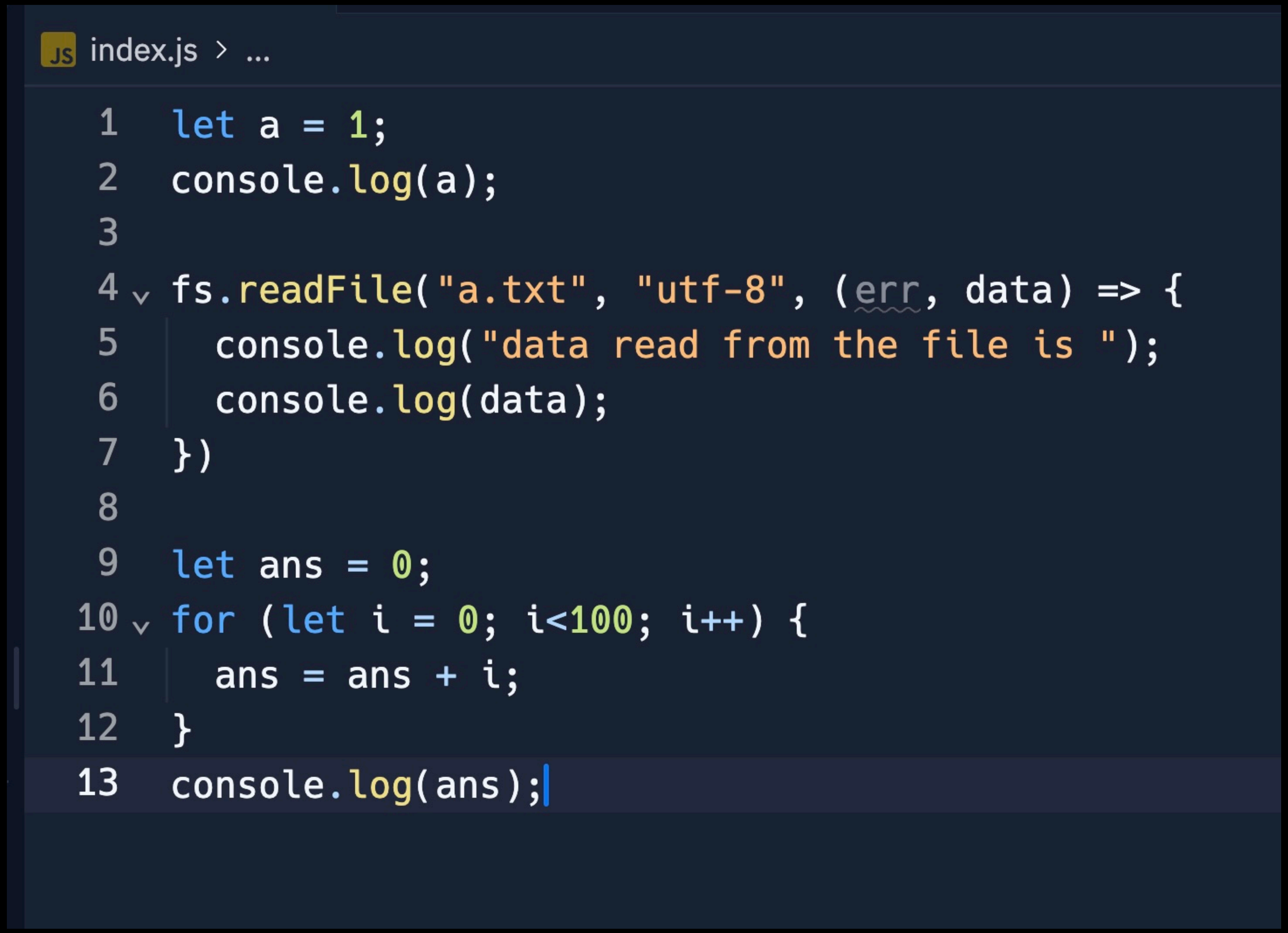
1. **Your javascript thread doesn’t have access to everything immediately**
2. **There are some tasks it needs to wait for**

**For example -**

1. **Reading a file**
2. **Sending a network request**
3. **A deliberate timeout**

# 2. Async functions

**Lets see an async function call**

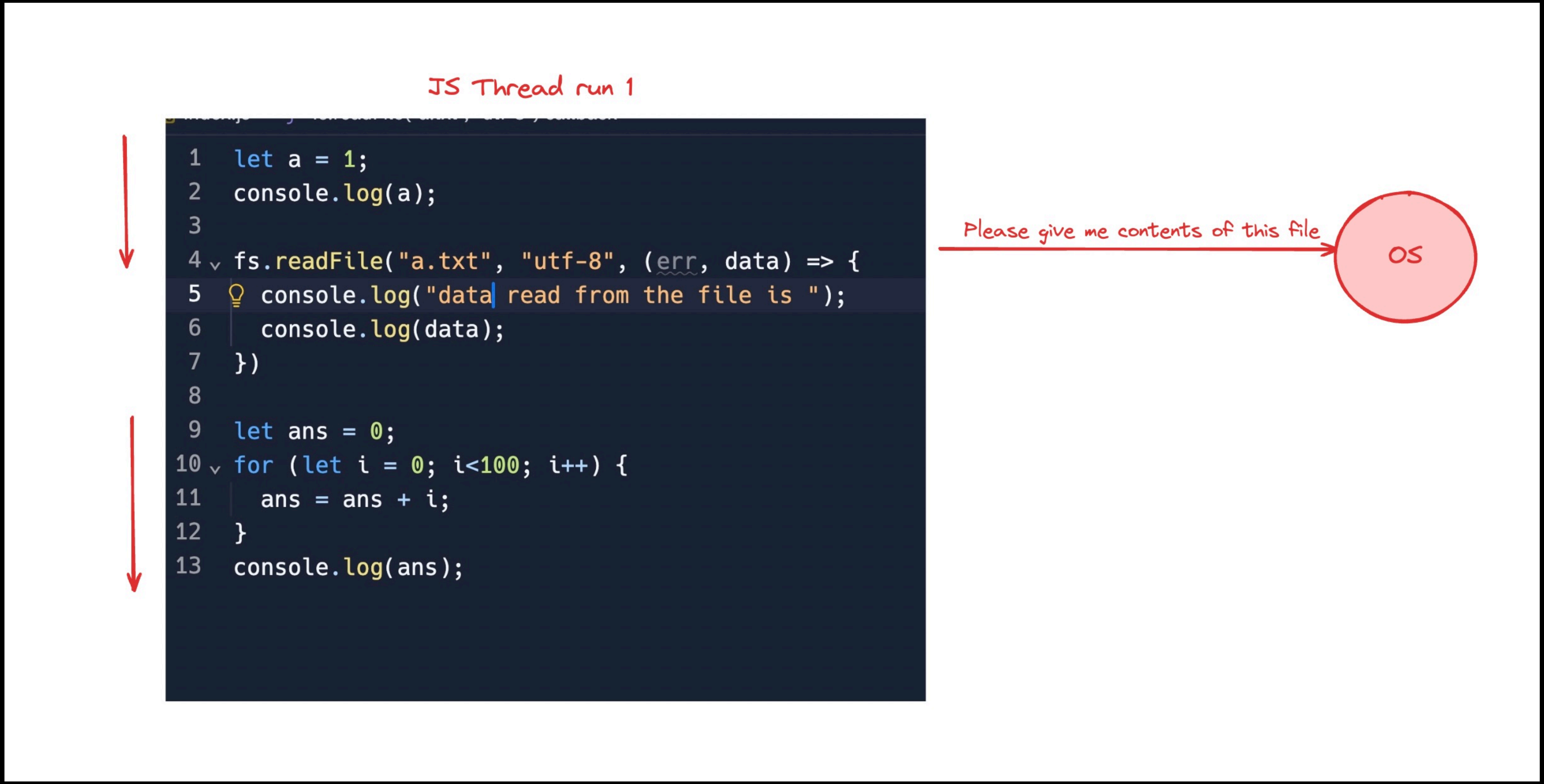


index.js > ...

```js
let a = 1;
console.log(a);

fs.readFile("a.txt", "utf-8", (err, data) => {
  console.log("data read from the file is ");
  console.log(data);
})

let ans = 0;
for (let i = 0; i<100; i++) {
  ans = ans + i;
}
console.log(ans);
```

# 2. Async functions

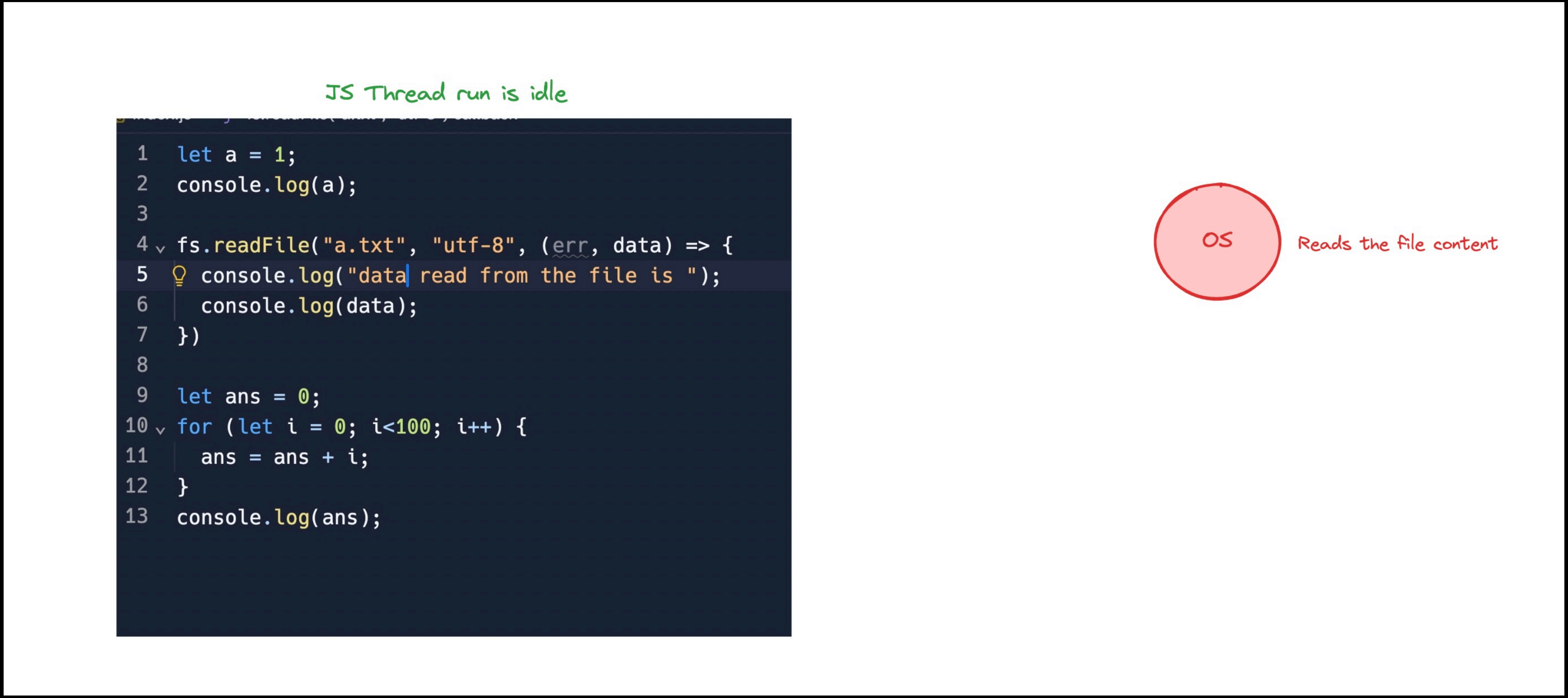
**Lets see an async function call**



```
let a = 1;
console.log(a);

fs.readFile("a.txt", "utf-8", (err, data) => {
  console.log("data read from the file is ");
  console.log(data);
})

let ans = 0;
for (let i = 0; i<100; i++) {
  ans = ans + i;
}
console.log(ans);
```

# 2. Async functions

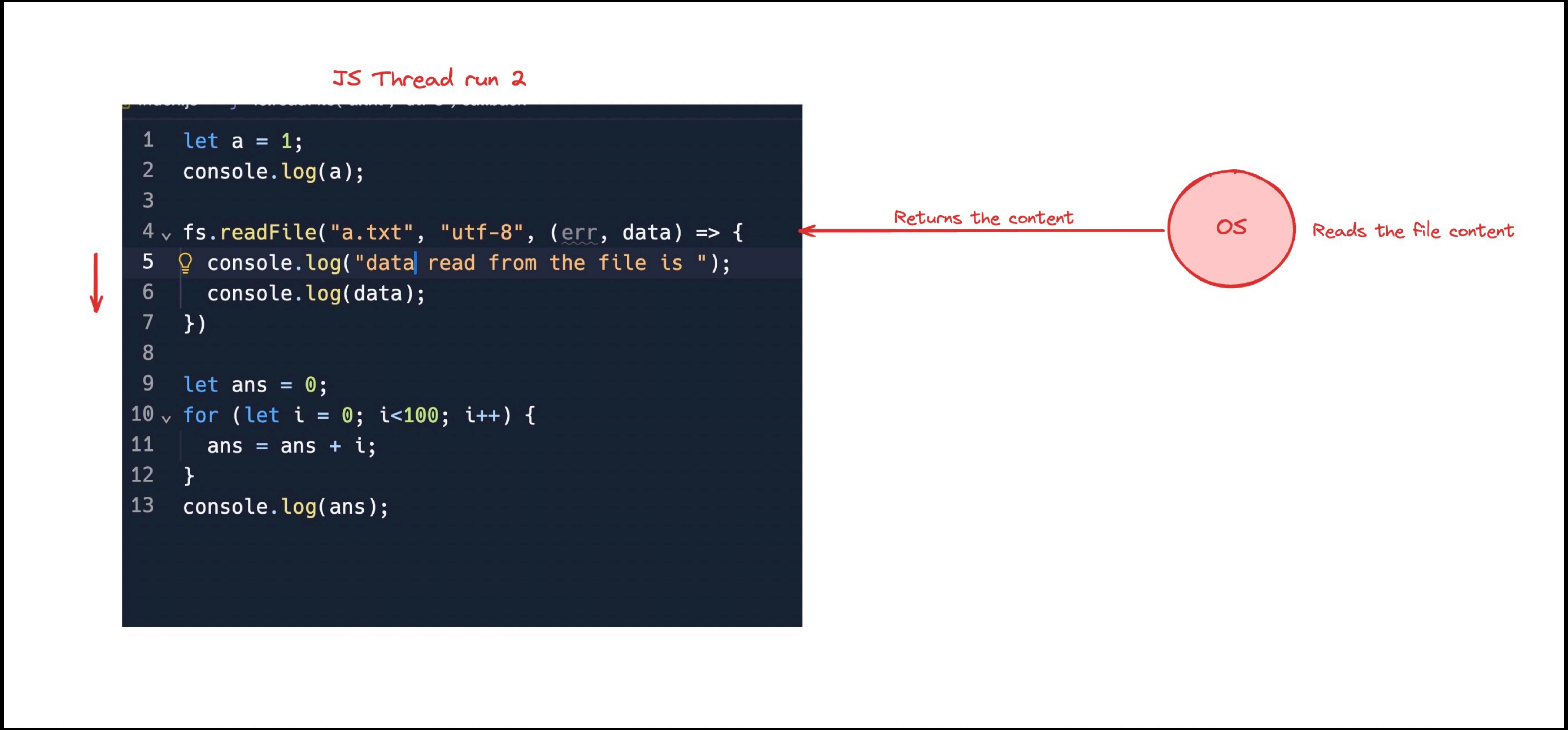
**Lets see an async function call**



```js
let a = 1;
console.log(a);

fs.readFile("a.txt", "utf-8", (err, data) => {
  console.log("data read from the file is ");
  console.log(data);
})

let ans = 0;
for (let i = 0; i<100; i++) {
  ans = ans + i;
}
console.log(ans);
```

# 2. Async functions

**Lets see an async function call**

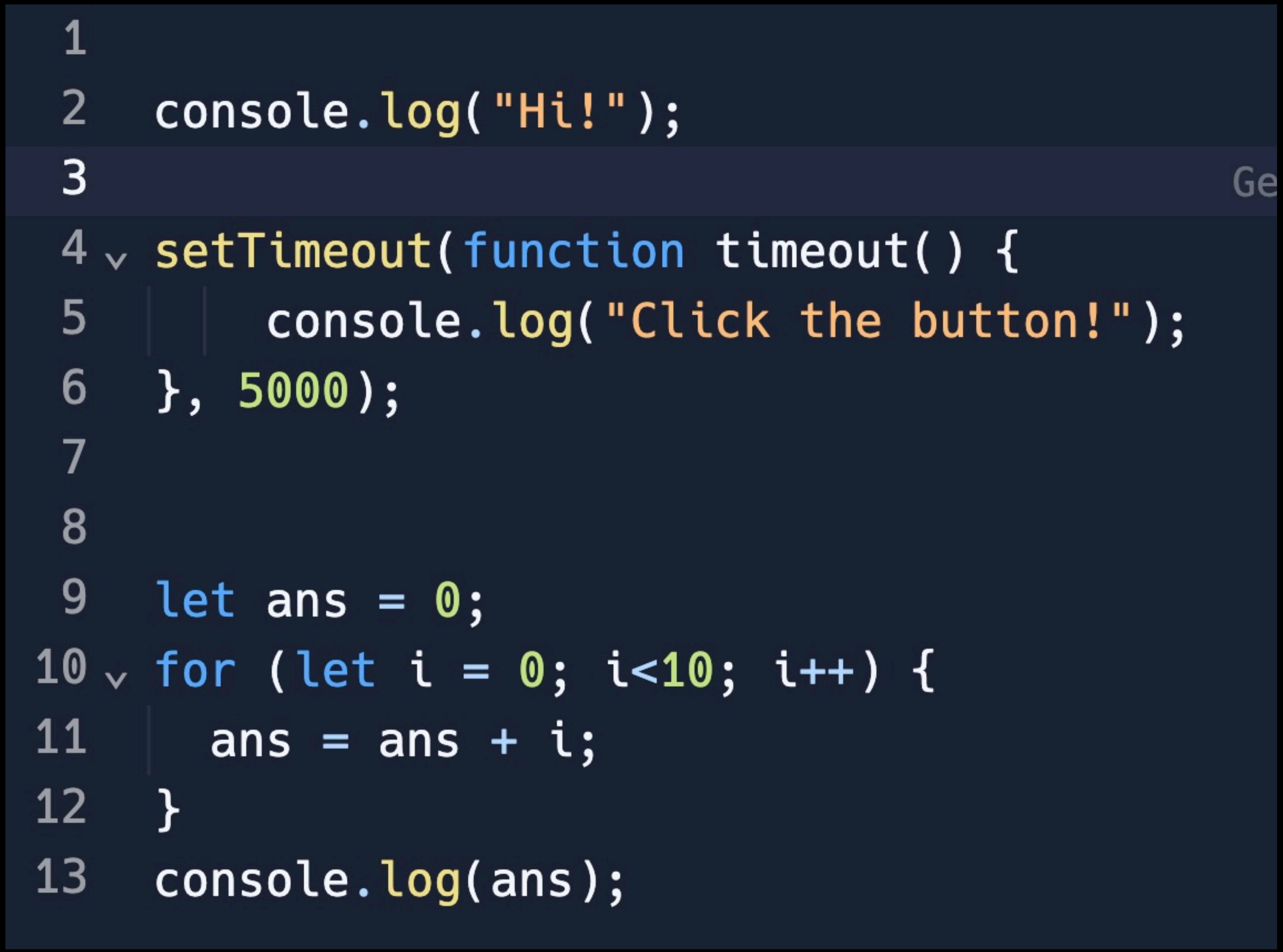


```
let a = 1;
console.log(a);

fs.readFile("a.txt", "utf-8", (err, data) => {
  console.log("data read from the file is ");
  console.log(data);
})

let ans = 0;
for (let i = 0; i<100; i++) {
  ans = ans + i;
}
console.log(ans);
```

# 2. Async functions

**Lets see it on loupe**

**http://latentflip.com/loupe/**



```

console.log("Hi!");


setTimeout(function timeout() {
    console.log("Click the button!");
}, 5000);


let ans = 0;
for (let i = 0; i<10; i++) {
  ans = ans + i;
}
console.log(ans);
```

1. Callbacks,
2. Async functions
3. Promises
4. JS functions
5. Assignment

## Promises

**Lets start with a question**

## Promises

**What are promises?**

1. **Just syntactical sugar**
2. **Just a more readable way to write async functions**

**Can you write JS without them**
**Yes - Just use callbacks**

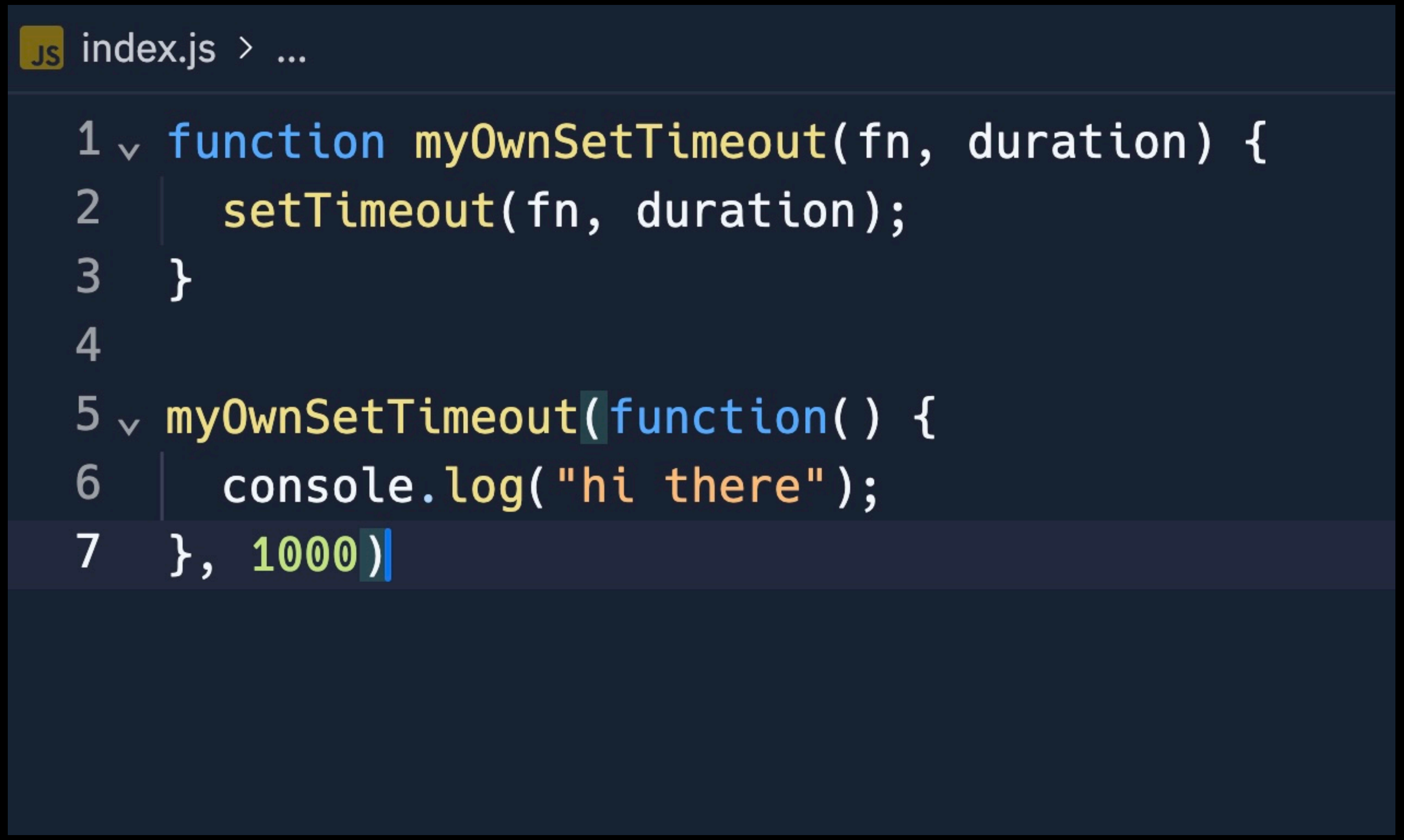
## Promises

**Lets start with a question**

**How would you create your own async function?**

# Promises

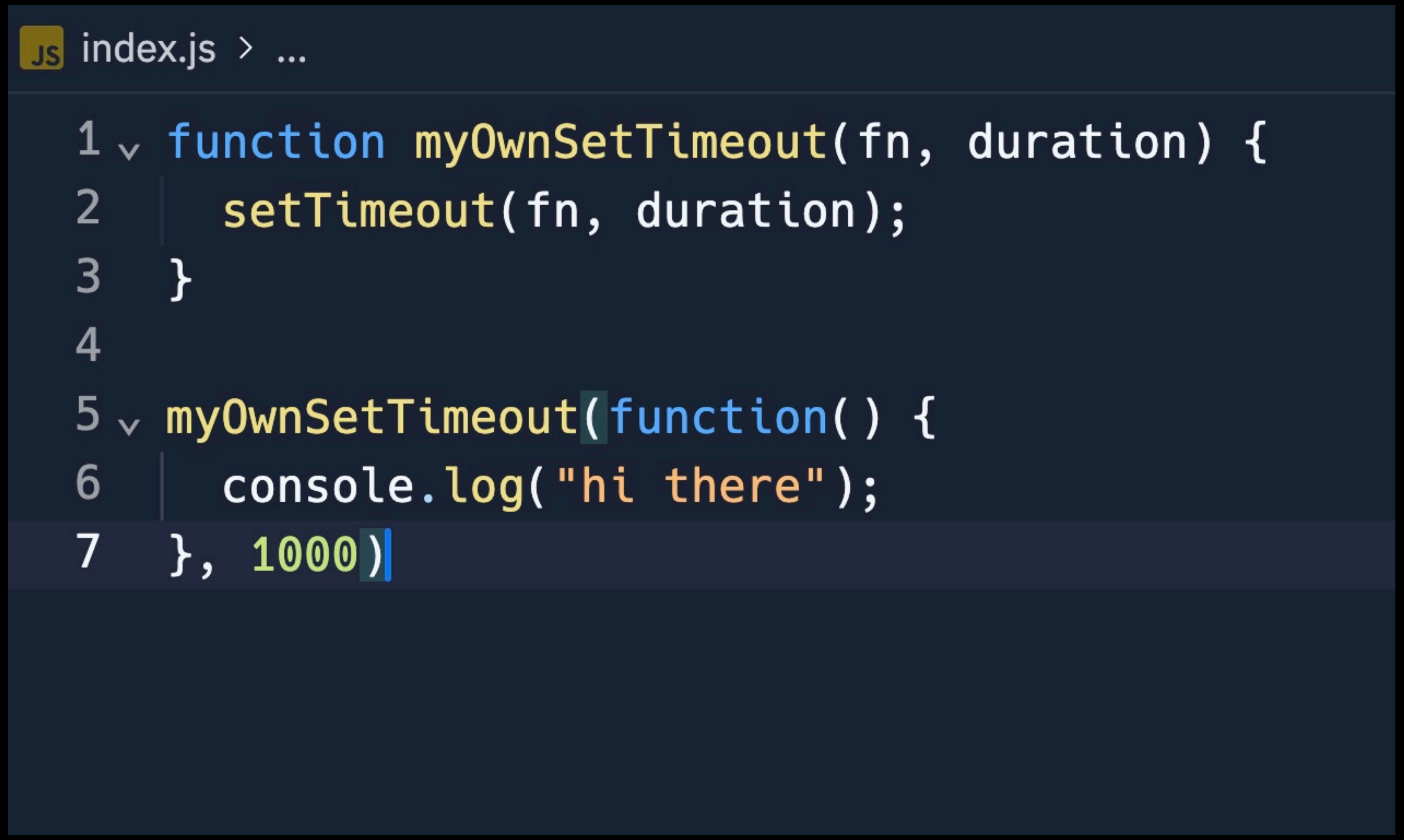
**Approach #1**
**(Wrapping another async fn)**



```javascript
function myOwnSetTimeout(fn, duration) {
  setTimeout(fn, duration);
}

myOwnSetTimeout(function() {
  console.log("hi there");
}, 1000)
```

# Promises

**Approach #1**
**(Wrapping another async fn)**



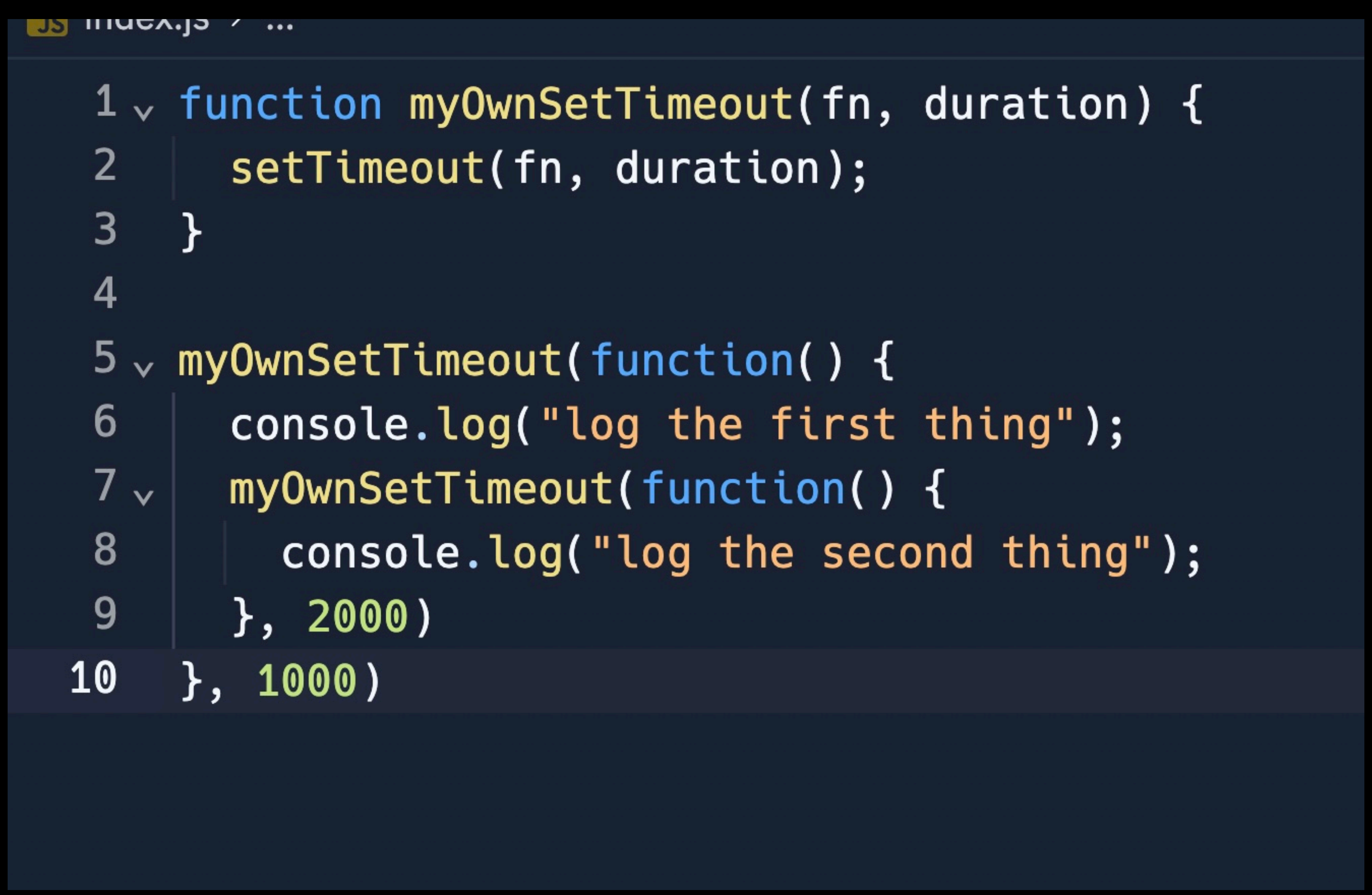
index.js > ...

```js
function myOwnSetTimeout(fn, duration) {
  setTimeout(fn, duration);
}

myOwnSetTimeout(function() {
  console.log("hi there");
}, 1000)
```

**This approach uses a callback**
**You have created a function where other people can send a callback**
**This is good, but could lead to callback hell**

# Promises

## Approach #1
## (Wrapping another async fn)



```javascript
function myOwnSetTimeout(fn, duration) {
  setTimeout(fn, duration);
}

myOwnSetTimeout(function() {
  console.log("log the first thing");
  myOwnSetTimeout(function() {
    console.log("log the second thing");
  }, 2000)
}, 1000)
```

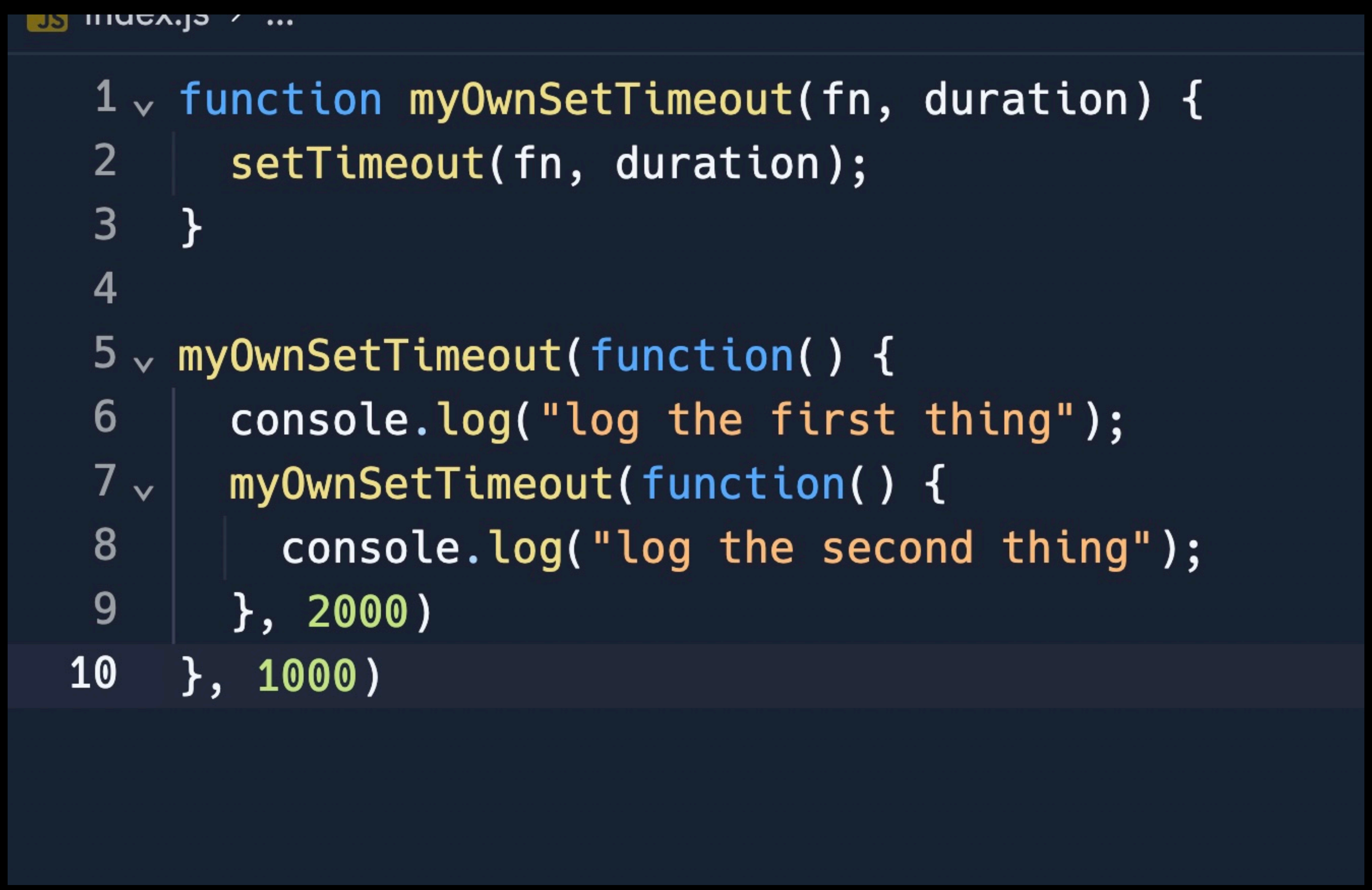
https://gist.github.com/hkirat/70eecb2d8ed0b5ef6a393e4f1e5369e1

**This approach uses a callback**
**You have created a function where other people can send a callback**
**This is good, but could lead to callback hell**

**What if I tell you -**
**Create a function that logs something after 1s**
**And then waits for 2 seconds to log another thing**

# Promises

**Approach #1**
**(Wrapping another async fn)**



```javascript
function myOwnSetTimeout(fn, duration) {
  setTimeout(fn, duration);
}

myOwnSetTimeout(function() {
  console.log("log the first thing");
  myOwnSetTimeout(function() {
    console.log("log the second thing");
  }, 2000)
}, 1000)
```

https://gist.github.com/hkirat/70eecb2d8ed0b5ef6a393e4f1e5369e1

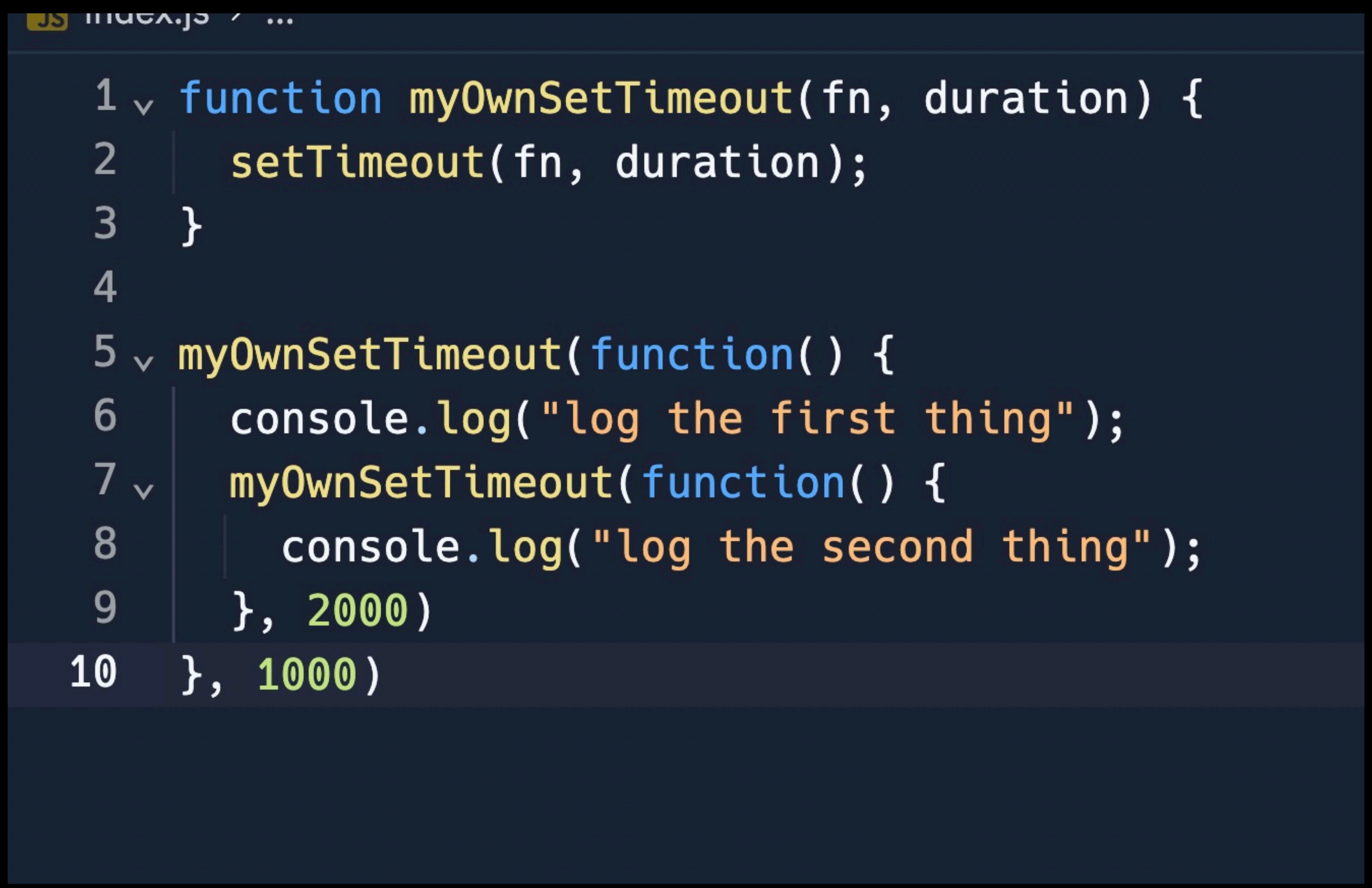
**This approach uses a callback**
**You have created a function where other people can send a callback**
**This is good, but could lead to callback hell**

**What if I tell you -**
**Create a function that logs something after 1s**
**And then waits for 2 seconds to log another thing**

**Callbacks lead to unnecessary indentation**
**This is called callback hell**

# Promises

**Approach #1**
**(Wrapping another async fn)**


```
function myOwnSetTimeout(fn, duration) {
  setTimeout(fn, duration);
}

myOwnSetTimeout(function() {
  console.log("log the first thing");
  myOwnSetTimeout(function() {
    console.log("log the second thing");
  }, 2000)
}, 1000)
```

https://gist.github.com/hkirat/70eecb2d8ed0b5ef6a393e4f1e5369e1

**This approach uses a callback**
**You have created a function where other people can send a callback**
**This is good, but could lead to callback hell**

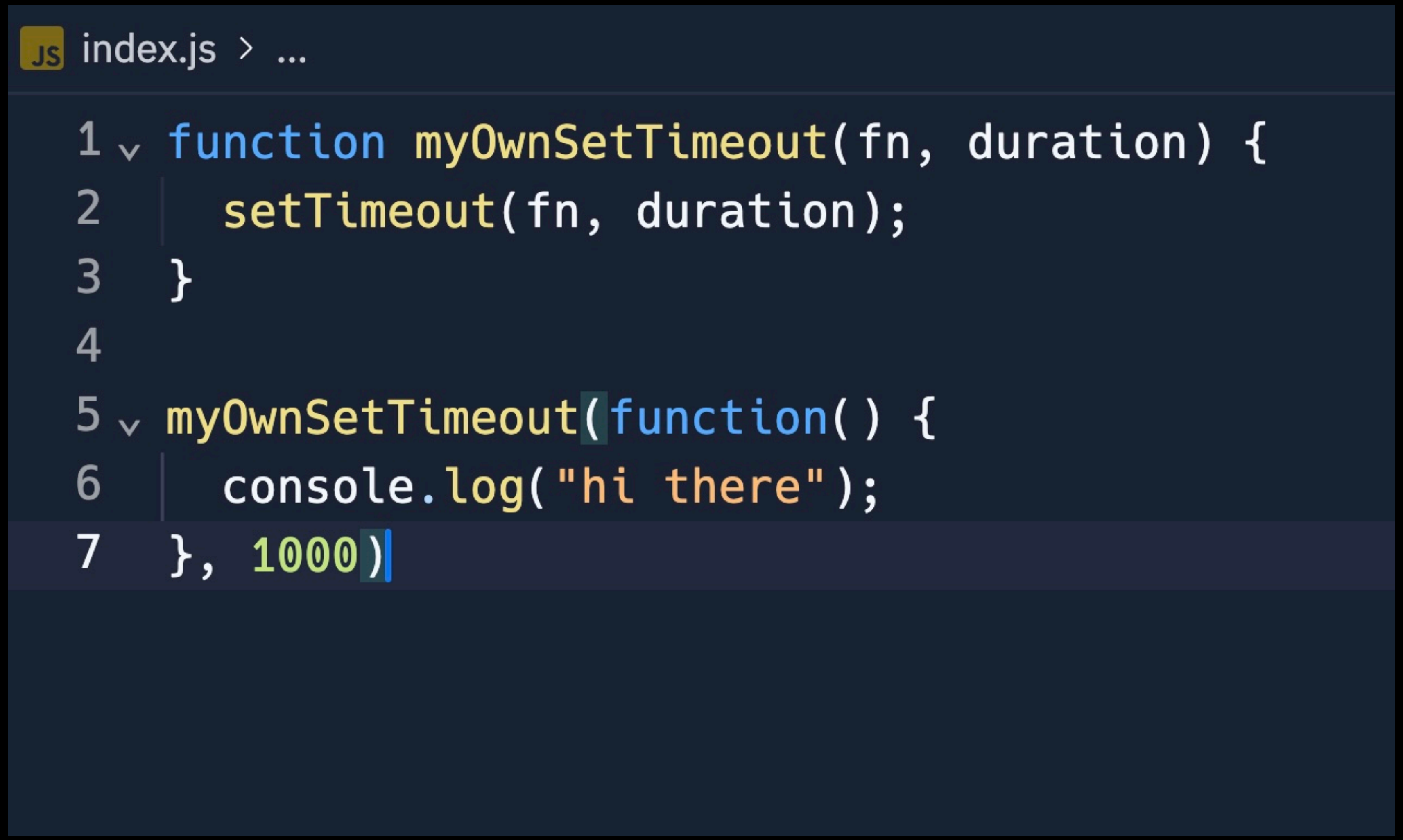
**What if I tell you -**
**Create a function that logs something after 1s**
**And then waits for 2 seconds to log another thing**

**Callbacks lead to unnecessary indentation**
**This is called callback hell**

**Lets see how promises fix this**

# Promises

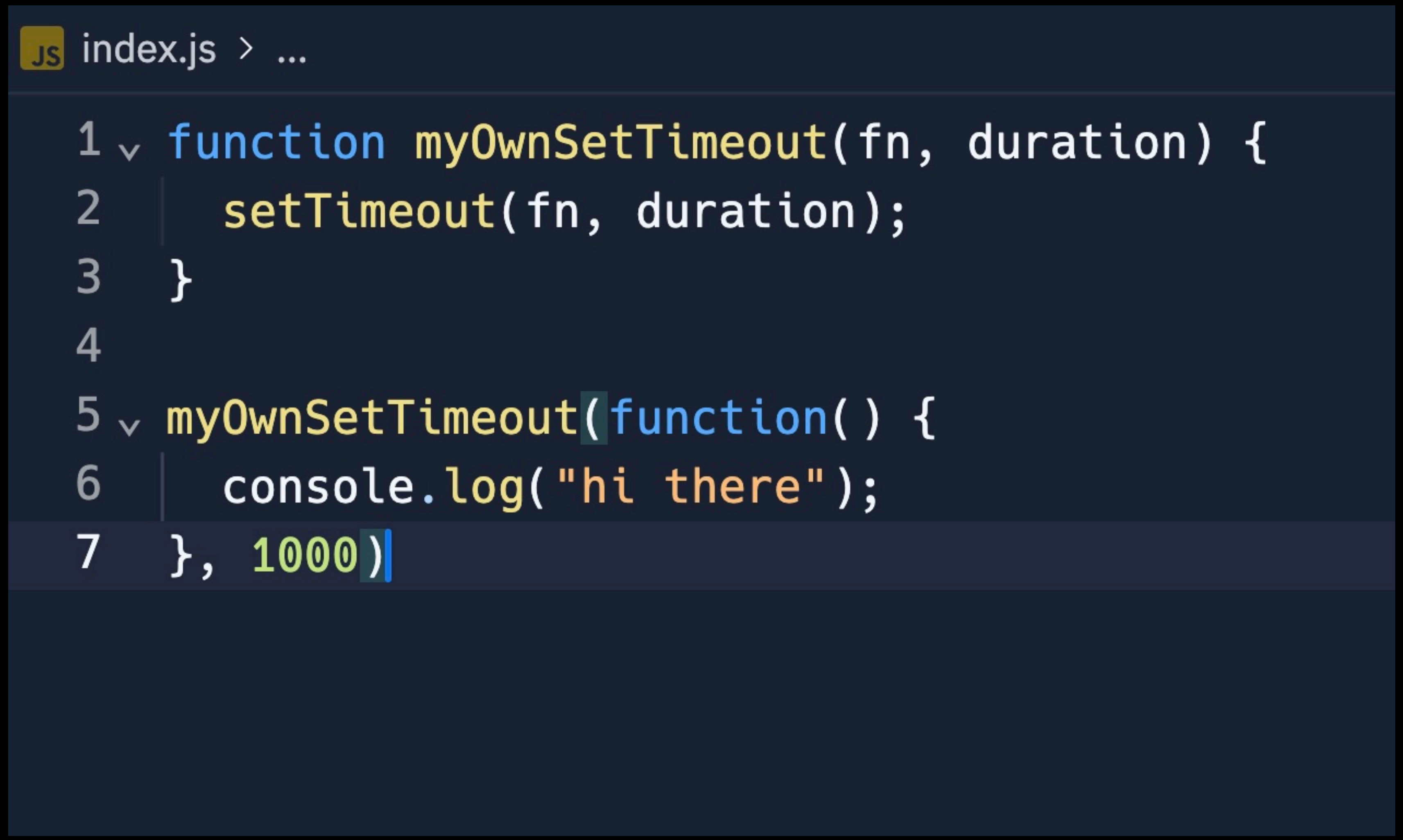
**Approach #1**
**(Wrapping another async fn)**


index.js > ...

```javascript
function myOwnSetTimeout(fn, duration) {
  setTimeout(fn, duration);
}

myOwnSetTimeout(function() {
  console.log("hi there");
}, 1000)
```

# Promises

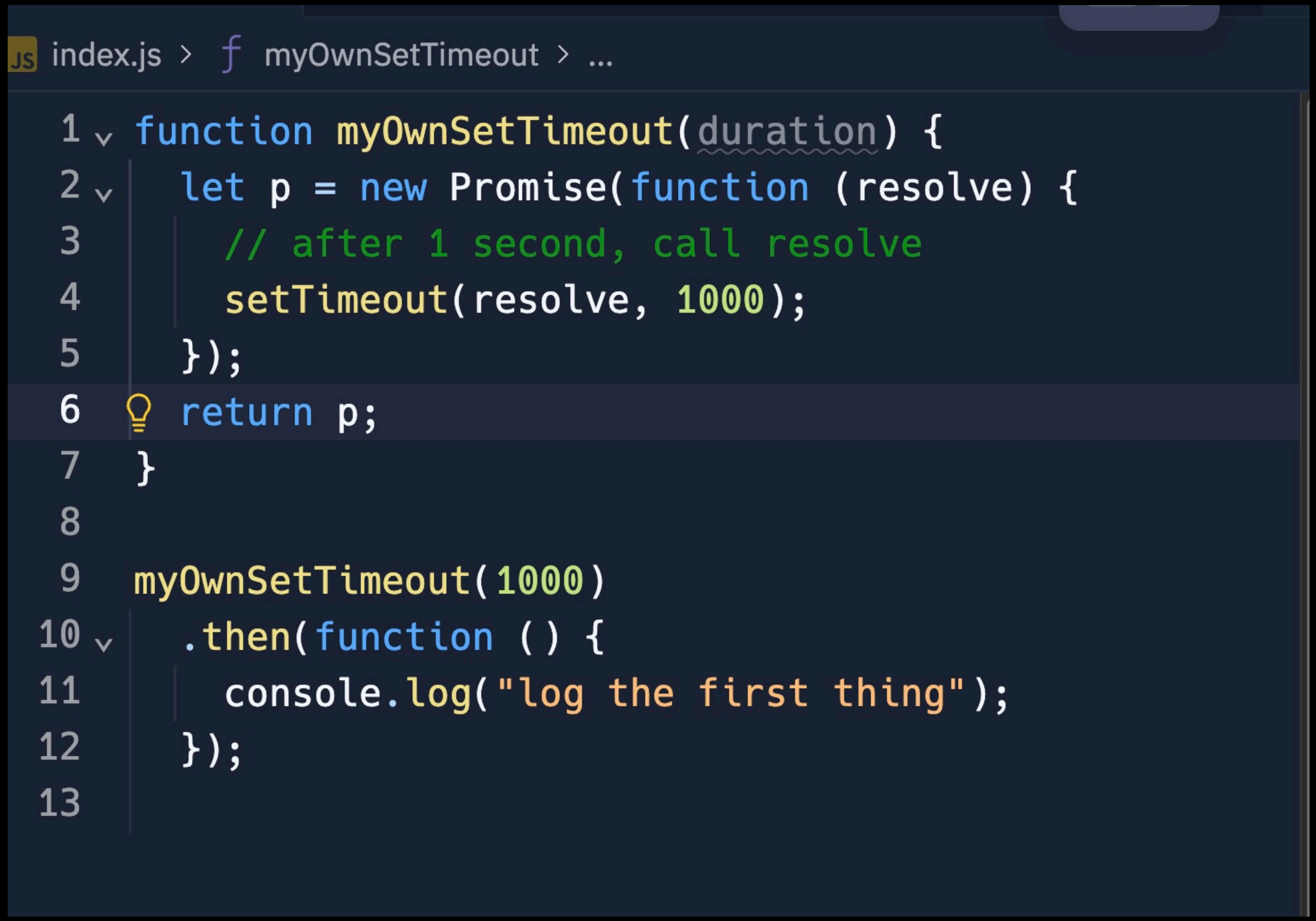
**Approach #1**
**(Wrapping another async fn)**



index.js > ...

```
function myOwnSetTimeout(fn, duration) {
  setTimeout(fn, duration);
}

myOwnSetTimeout(function() {
  console.log("hi there");
}, 1000)
```

**Approach #2**
**(Using promises)**



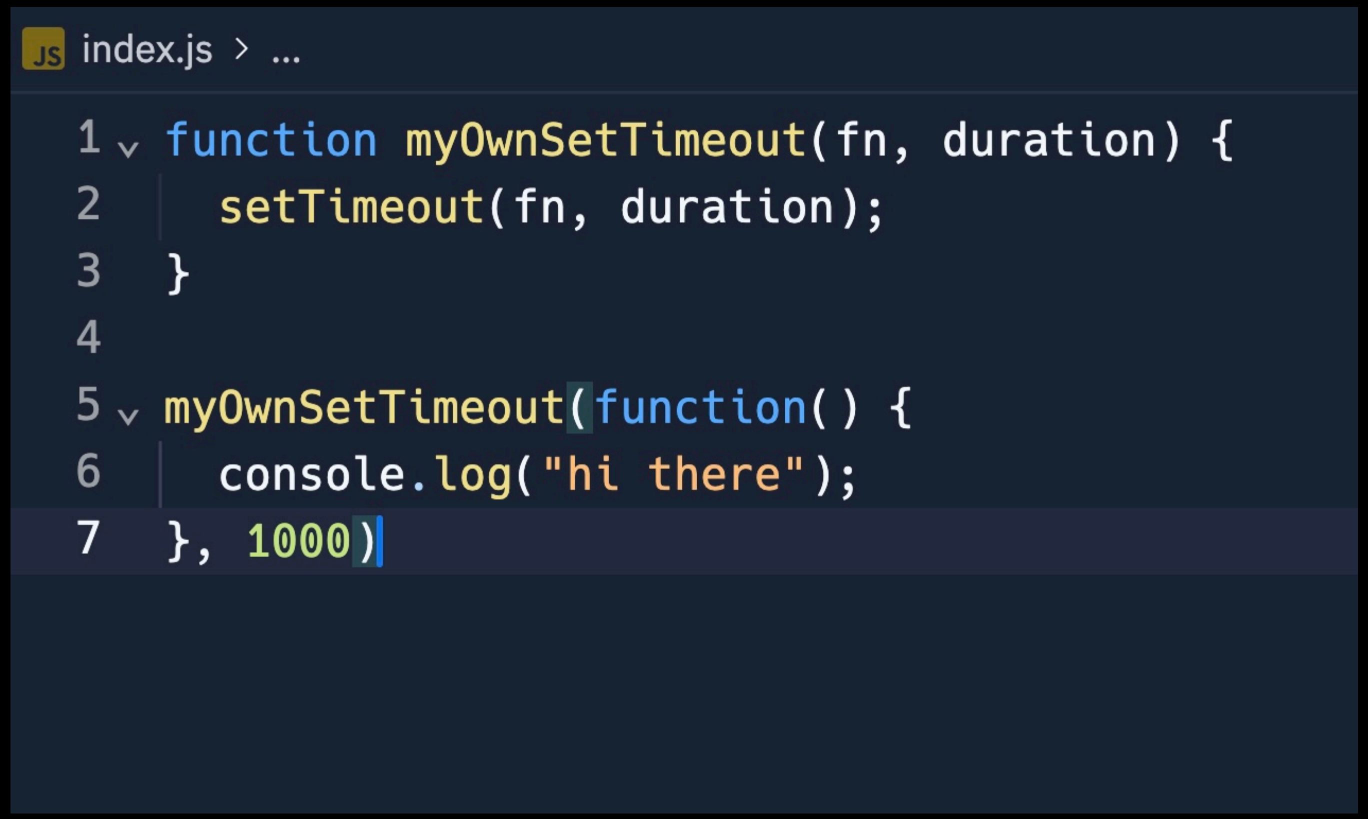
index.js > ƒ myOwnSetTimeout > ...

```
function myOwnSetTimeout(duration) {
  let p = new Promise(function (resolve) {
    // after 1 second, call resolve
    setTimeout(resolve, 1000);
  });
  return p;
}

myOwnSetTimeout(1000)
  .then(function () {
    console.log("log the first thing");
  });

```

https://gist.github.com/hkirat/9a7a0ef9ad6788f645497a2cd2b92106

# Promises

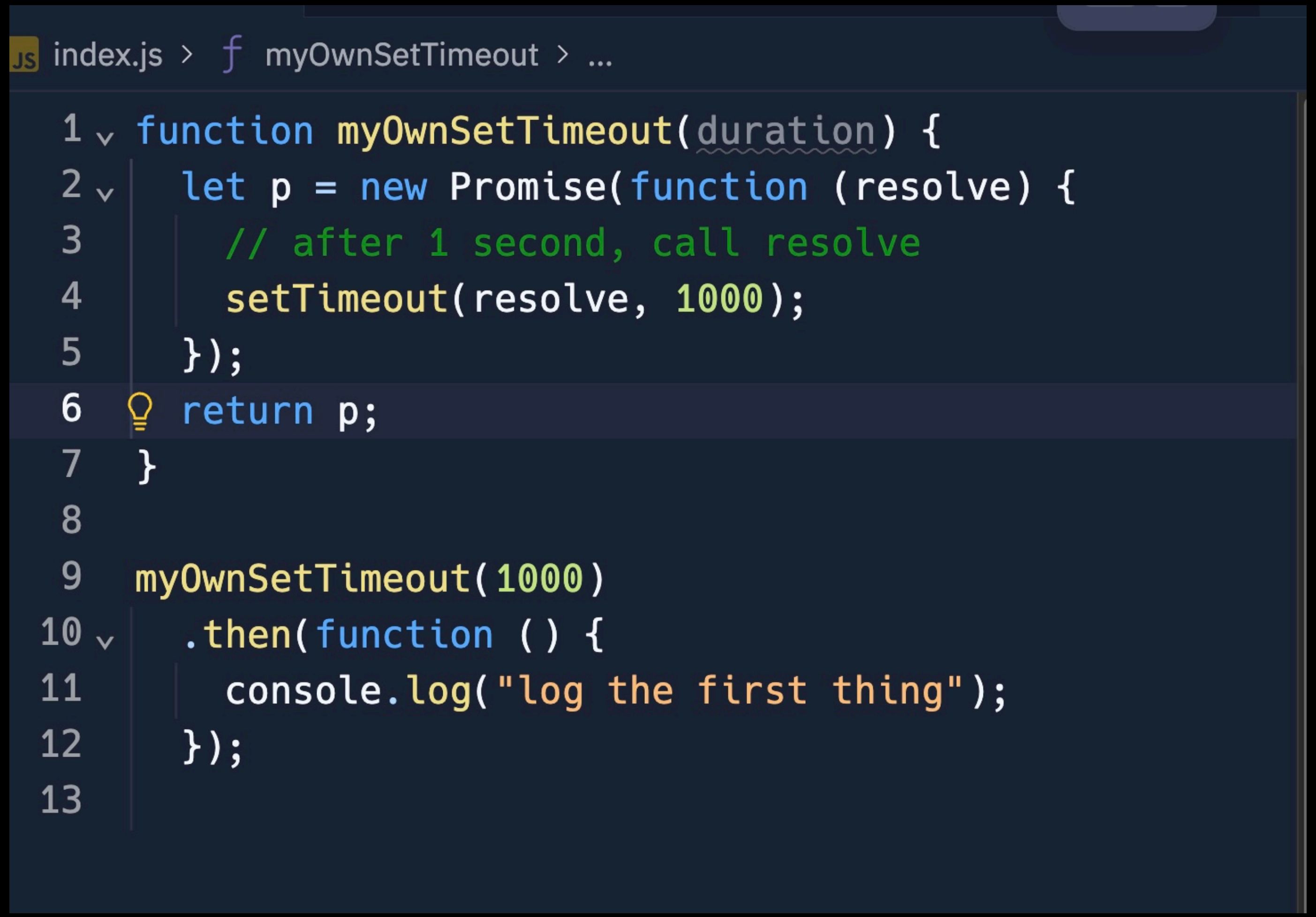
## Approach #1
## (Wrapping another async fn)



index.js > ...

```javascript
function myOwnSetTimeout(fn, duration) {
  setTimeout(fn, duration);
}

myOwnSetTimeout(function() {
  console.log("hi there");
}, 1000)
```

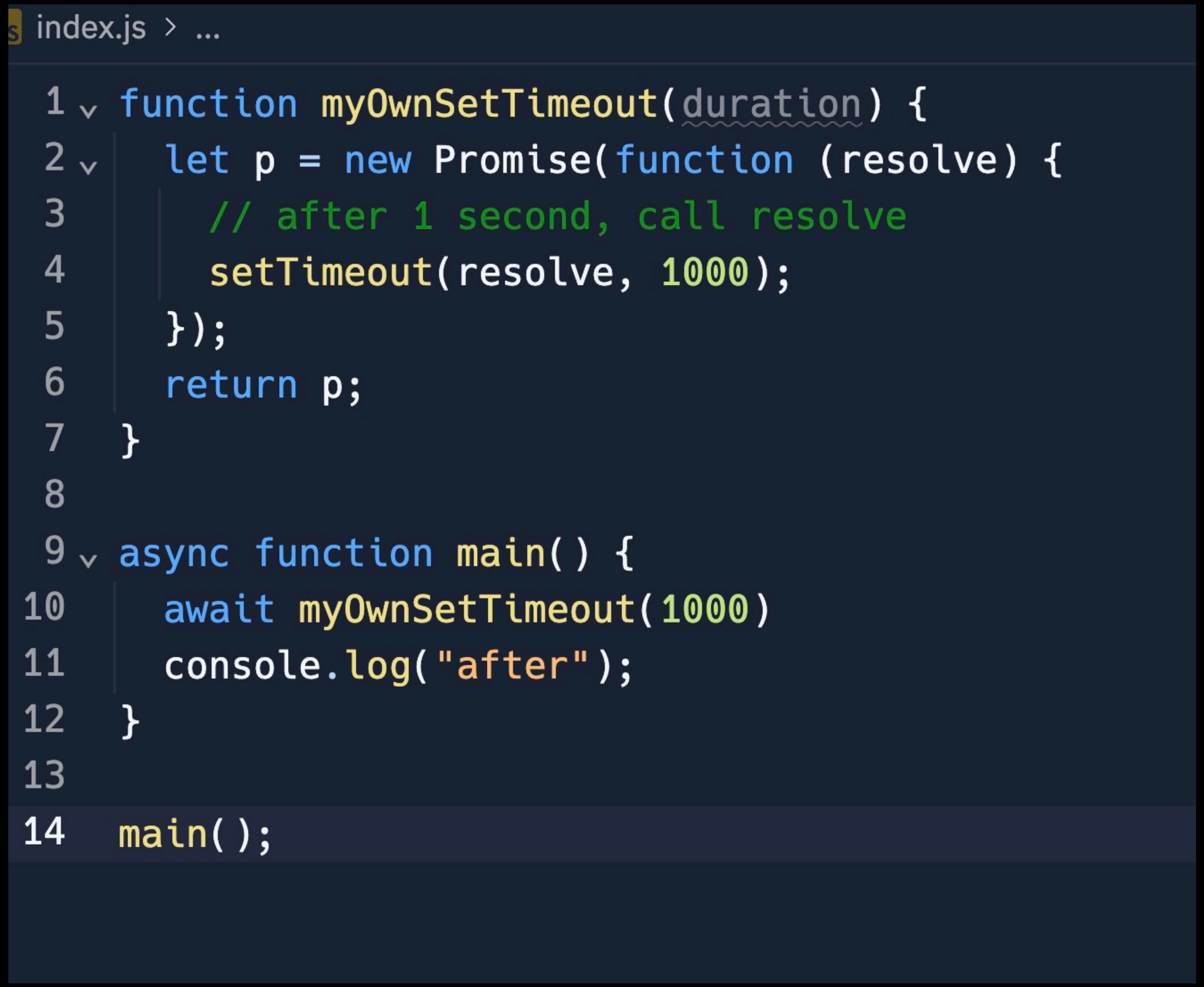
## Approach #2
## (Using promises)



index.js > ƒ myOwnSetTimeout > ...

```javascript
function myOwnSetTimeout(duration) {
  let p = new Promise(function (resolve) {
    // after 1 second, call resolve
    setTimeout(resolve, 1000);
  });
  return p;
}

myOwnSetTimeout(1000)
  .then(function () {
    console.log("log the first thing");
  });

```

## Approach #3
## (Async await)



index.js > ...

```javascript
function myOwnSetTimeout(duration) {
  let p = new Promise(function (resolve) {
    // after 1 second, call resolve
    setTimeout(resolve, 1000);
  });
  return p;
}

async function main() {
  await myOwnSetTimeout(1000)
  console.log("after");
}

main();
```

https://gist.github.com/hkirat/9a7a0ef9ad6788f645497a2cd2b92106

1. Callbacks,
2. Async functions
3. Promises
4. JS functions
5. Assignment

## map

**Question - Convert an array of numbers into a new array with every element Multiplied by 2**

## map

**Question - Convert an array of numbers into a new array with every element Multiplied by 2**

**Do it yourself**

## Filter

**Question - Convert an array of numbers into a new array with only the even elements**

**Do it yourself**

1. Callbacks,
2. Async functions
3. Promises
4. JS functions
5. Assignment

## Assignments