

Offline Lecture 1.5
Async functions vs sync functions, real use of callbacks
JS Browser architecture
Promises
Async await

Async functions vs sync functions

What does synchronous mean?

Together, one after the other, sequential

Only one thing is happening at a time

What does asynchronous mean?

Opposite of synchronous

Happens in parts

Multiple things are context switching with each other

Async functions vs sync functions

Lets build some intuition

Human brain and body is single threaded

1. **We can only do one thing at a time**
2. **But we can context switch b/w tasks, or we can delegate tasks to other people**

Async functions vs sync functions

Lets build some intuition

Human brain and body is single threaded

1. We can only do one thing at a time
2. But we can context switch b/w tasks, or we can delegate tasks to other people

You have 4 tasks -

1. Boil water
2. Cut vegetables
3. Cut maggi packet
4. Get ketchup from the shop nearby

How would you do this? Synchronously or Asynchronously?

Async functions vs sync functions

Lets build some intuition

Lets say you have 4 tasks to make food -

1. Boil water
2. Cut vegetables
3. Open maggi packet
4. Get ketchup from the shop nearby

How would you do them?

Async functions vs sync functions

Lets build some intuition



Total time
5 +



5 minutes

Async functions vs sync functions

Lets build some intuition



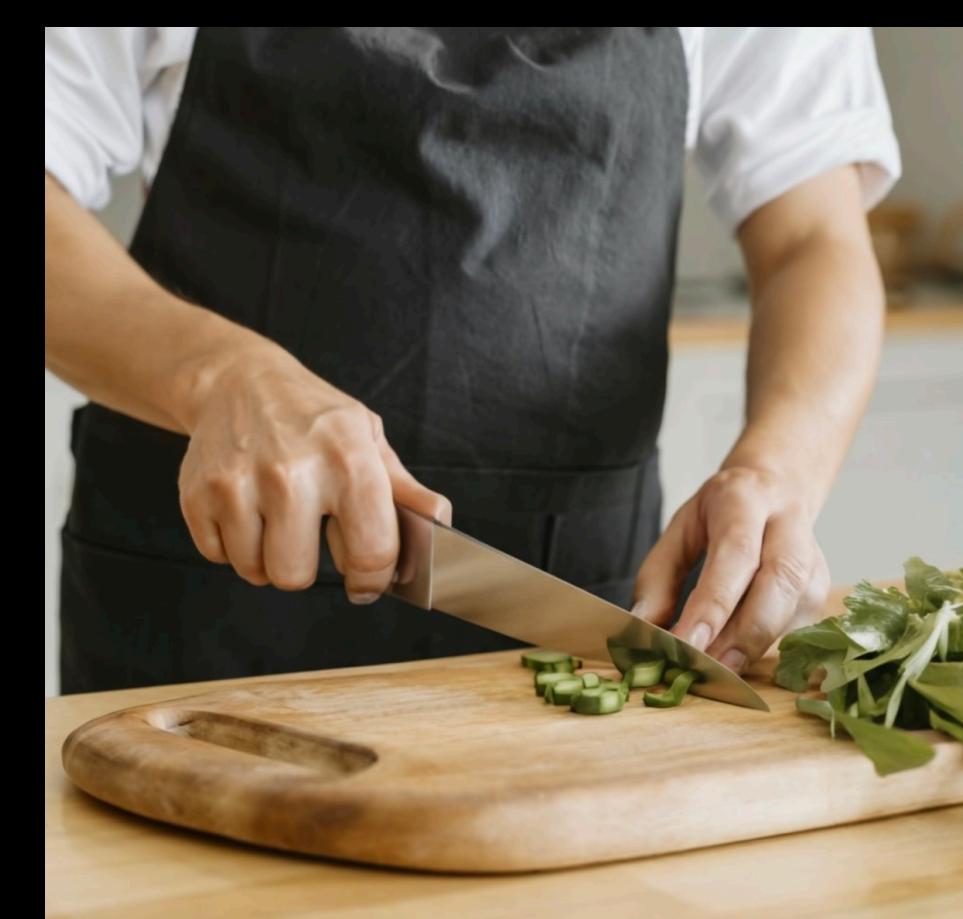
Total time
5 +



2 minutes

Async functions vs sync functions

Lets build some intuition



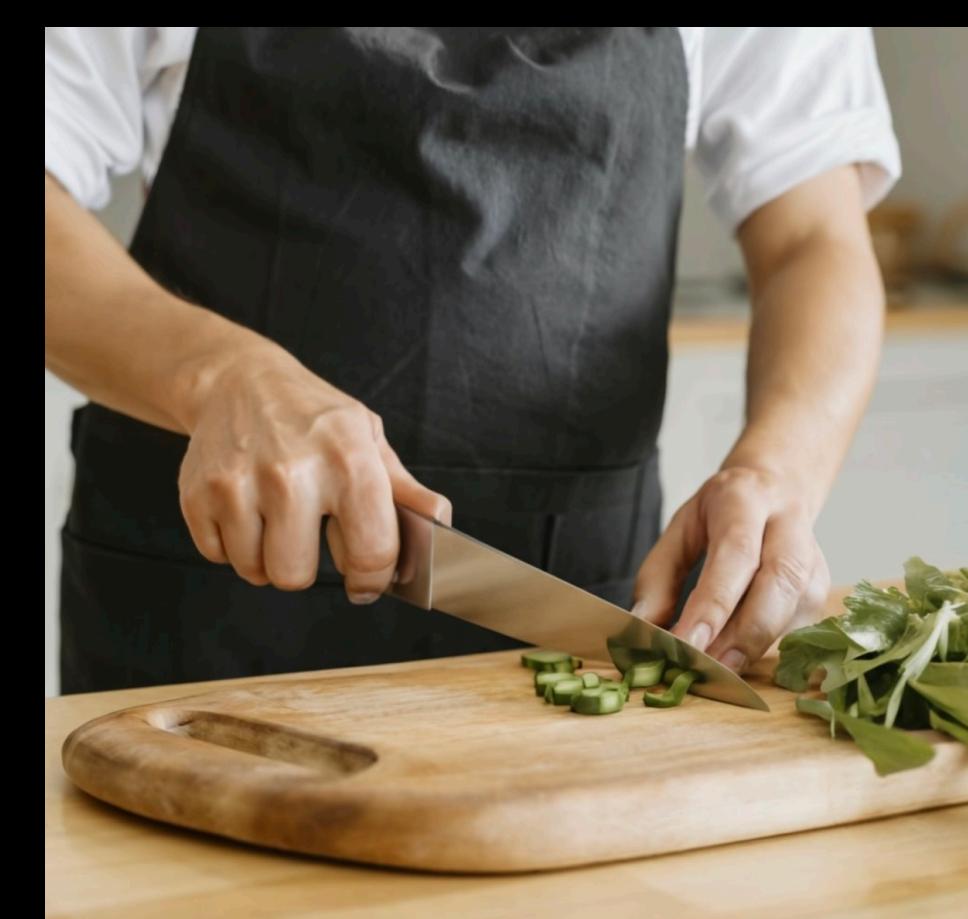
Total time
 $5 + 2$



2 minutes

Async functions vs sync functions

Lets build some intuition



Total time
 $5 + 2 + 10$



10 minutes

Async functions vs sync functions

Lets build some intuition



Total time
 $5 + 2 + 10 + 20$
= 37 mins



20 minutes

Async functions vs sync functions

Lets build some intuition

**Or, you could multi-task
(More technically, context switch and delegate)**

Async functions vs sync functions

Lets build some intuition



Total time



Turn on gas (2 seconds)

Async functions vs sync functions

Lets build some intuition



Total time



Delegate task to house help

Async functions vs sync functions

Lets build some intuition



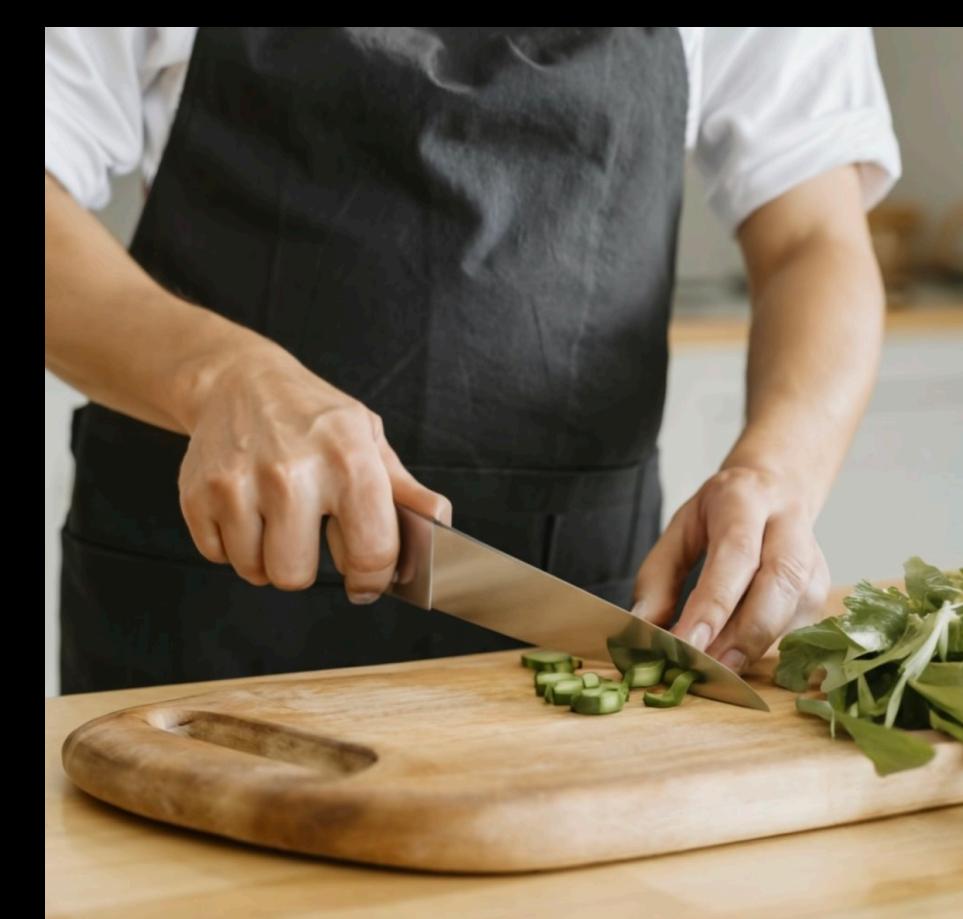
Total time



Delegate task to house help (kamla didi)

Async functions vs sync functions

Lets build some intuition



Total time

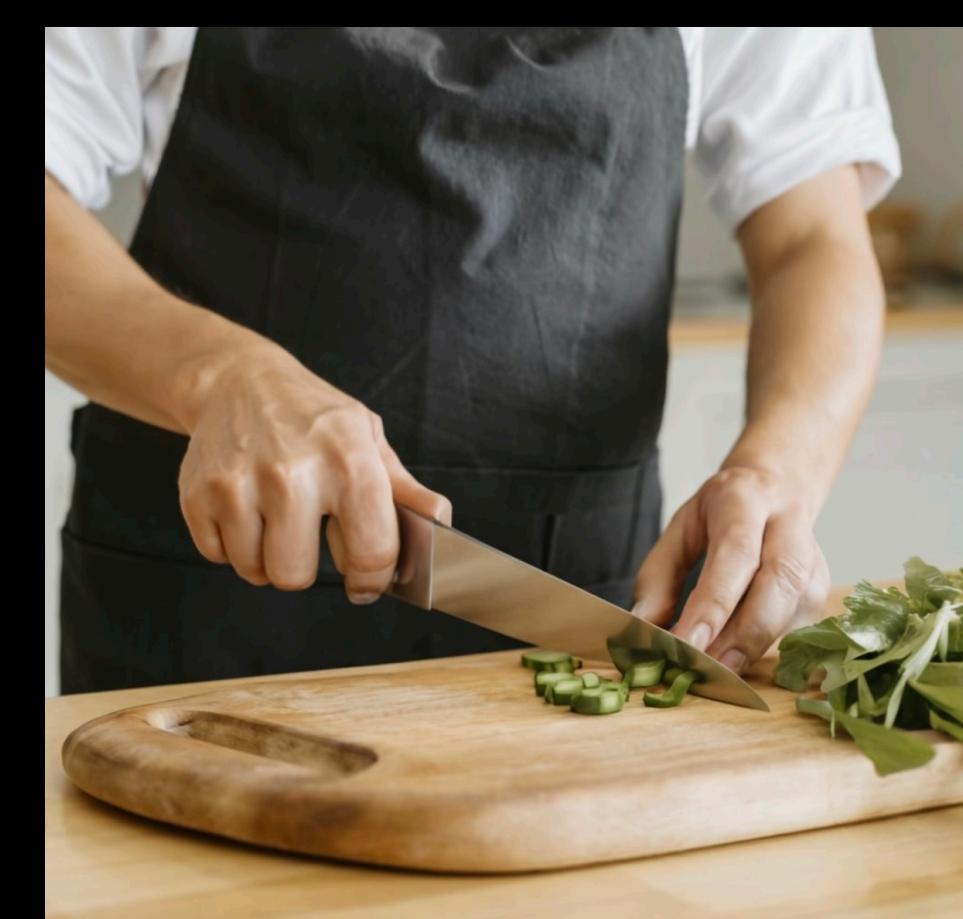


Cut Maggi packet (2 mins)

On the way to get ketchup (2 mins)

Async functions vs sync functions

Lets build some intuition



Total time



Check on water (1s)



On the way to get ketchup (2 mins 1 s)

Async functions vs sync functions

Lets build some intuition



Total time



Cutting vegetables (8 mins)

She's back with ketchup (10 mins)

Async functions vs sync functions

Lets build some intuition



Total time



Cut vegetables (10 mins)

Waits for 2 mins for u to finish

Async functions vs sync functions

Lets build some intuition



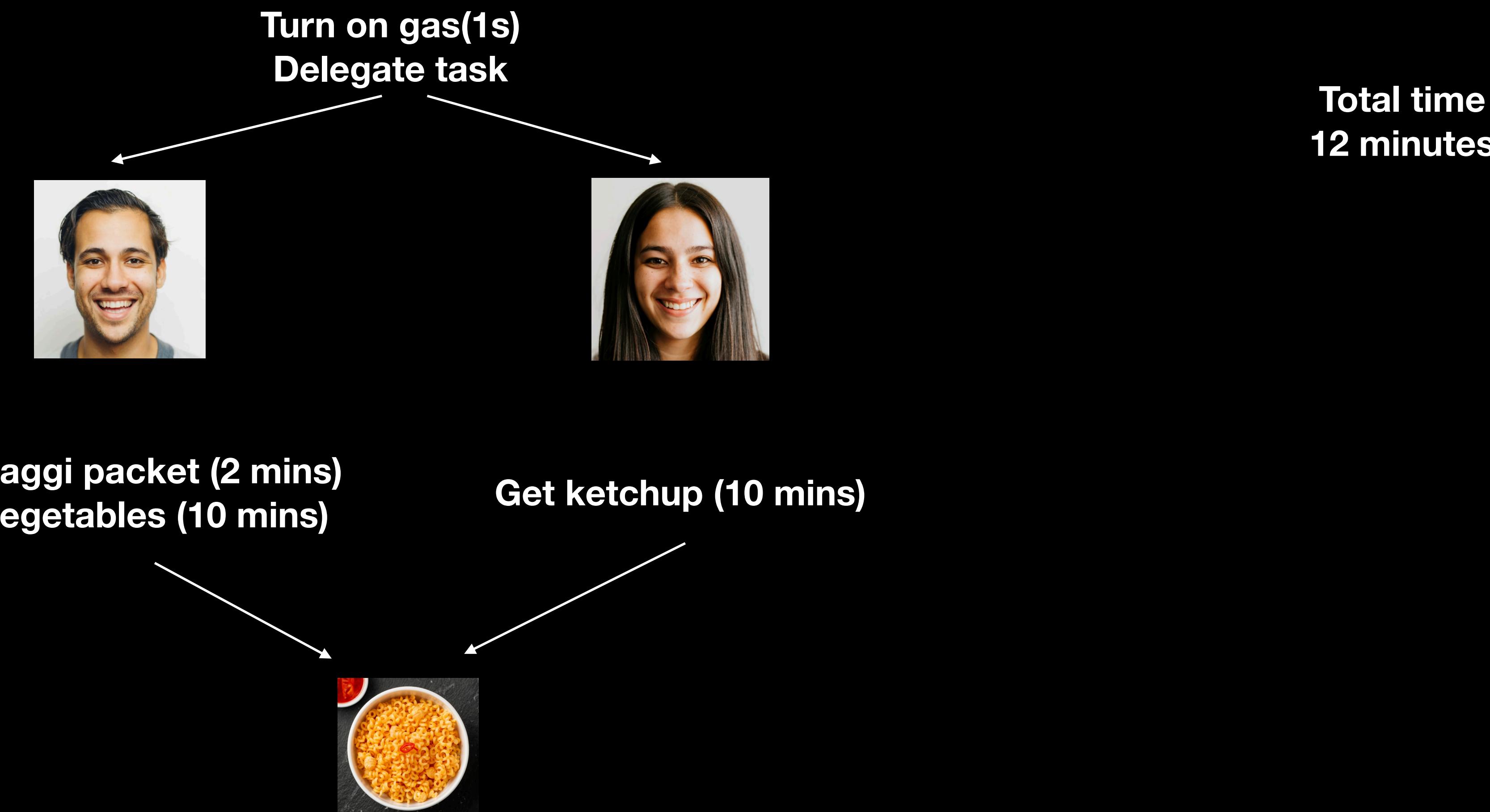
Total time
12 minutes



Everything is done!

Async functions vs sync functions

Lets build some intuition



Async functions vs sync functions

What did we learn?

**Even if you are single threaded (brain can do only one thing at a time), you can do things parallely by Delegating
You can also context switch between tasks if need be (the net time to do both the things would still be the same)**

**Net amount of time take to do a task can be decreased
by doing these two things (delegating and context switching)**

Async functions vs sync functions

How does JS do the same? Can JS delegate? Can JS context switch?

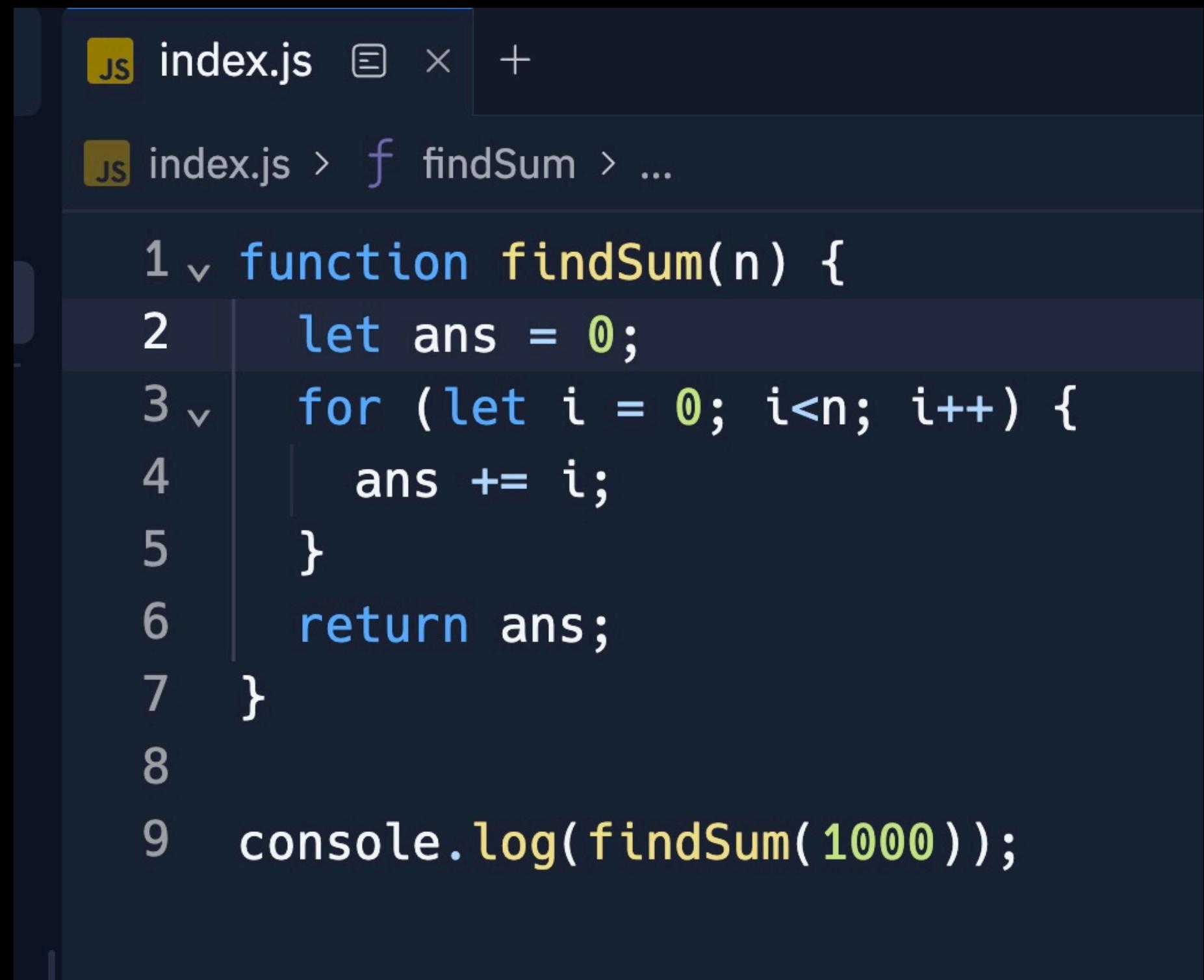
Async functions vs sync functions

How does JS do the same? Can JS delegate? Can JS context switch?

Yes! Using asynchronous functions

Async functions vs sync functions

Until now, we've only seen synchronous functions



The image shows a screenshot of a code editor with a dark theme. A file named 'index.js' is open, containing the following code:

```
1 function findSum(n) {  
2     let ans = 0;  
3     for (let i = 0; i<n; i++) {  
4         ans += i;  
5     }  
6     return ans;  
7 }  
8  
9 console.log(findSum(1000));
```

The code defines a synchronous function 'findSum' that calculates the sum of all integers from 0 to n-1. It uses a for loop to iterate through the numbers and adds them to a variable 'ans'. Finally, it returns the total sum.

Async functions vs sync functions

Lets introduce an asynchronous function (`setTimeout`)

<https://gist.github.com/hkirat/a75987a32fdfcab27672410930327f1a>

```
index.js ...  
1 function findSum(n) {  
2     let ans = 0;  
3     for (let i = 0; i<n; i++) {  
4         ans += i;  
5     }  
6     return ans;  
7 }  
8  
9 function findSumTill100() {  
10    return findSum(100);  
11 }  
12  
13 → setTimeout(findSumTill100, 1000)  
14 console.log("hello world");
```

Calling an async function



Async functions vs sync functions

Lets introduce an asynchronous function (`setTimeout`)

<https://gist.github.com/hkirat/a75987a32fdfcab27672410930327f1a>

```
 1 function findSum(n) {  
 2     let ans = 0;  
 3     for (let i = 0; i<n; i++) {  
 4         ans += i;  
 5     }  
 6     return ans;  
 7 }  
 8  
 9 function findSumTill100() {  
10     return findSum(100);  
11 }  
12  
13 setTimeout(findSumTill100, 1000)  
14 console.log("hello world");
```



Calling an async function



After 1 second,
remind me to run a function

Async functions vs sync functions

Lets introduce an asynchronous function (`setTimeout`)

<https://gist.github.com/hkirat/a75987a32fdfcab27672410930327f1a>

```
macx.js
```

```
1 function findSum(n) {
2     let ans = 0;
3     for (let i = 0; i<n; i++) {
4         ans += i;
5     }
6     return ans;
7 }
8
9 function findSumTill100() {
10    return findSum(100);
11 }
12
13 setTimeout(findSumTill100, 1000)
14 console.log("hello world");
```



Proceeds to his own thing

14 →



Maintains a clock for 1s

Async functions vs sync functions

Lets introduce an asynchronous function (`setTimeout`)

<https://gist.github.com/hkirat/a75987a32fdfcab27672410930327f1a>

```
macx.js
```

```
1 function findSum(n) {
2     let ans = 0;
3     for (let i = 0; i<n; i++) {
4         ans += i;
5     }
6     return ans;
7 }
8
9 function findSumTill100() {
10    return findSum(100);
11 }
12
13 setTimeout(findSumTill100, 1000)
14 console.log("hello world");
```



Done with his own thing, sitting idle



Still running 1s clock

Async functions vs sync functions

Lets introduce an asynchronous function (`setTimeout`)

<https://gist.github.com/hkirat/a75987a32fdfcab27672410930327f1a>



```
 1 function findSum(n) {  
 2     let ans = 0;  
 3     for (let i = 0; i<n; i++) {  
 4         ans += i;  
 5     }  
 6     return ans;  
7 }  
8  
9 function findSumTill100() {  
10    return findSum(100);  
11 }  
12  
13 setTimeout(findSumTill100, 1000)  
14 console.log("hello world");
```



Calls the callback

Async functions vs sync functions

Lets introduce an asynchronous function (`setTimeout`)

<https://gist.github.com/hkirat/a75987a32fdfcab27672410930327f1a>

Starts executing
the callback



```
1 function findSum(n) {  
2     let ans = 0;  
3     for (let i = 0; i<n; i++) {  
4         ans += i;  
5     }  
6     return ans;  
7 }  
8  
9 function findSumTill100() {  
10    return findSum(100);  
11 }  
12  
13 setTimeout(findSumTill100, 1000)  
14 console.log("hello world");
```



Done with their task!

Async functions vs sync functions

What are common async functions?

`setTimeout`

`fs.readFile` - to read a file from your filesystem

`Fetch` - to fetch some data from an API endpoint

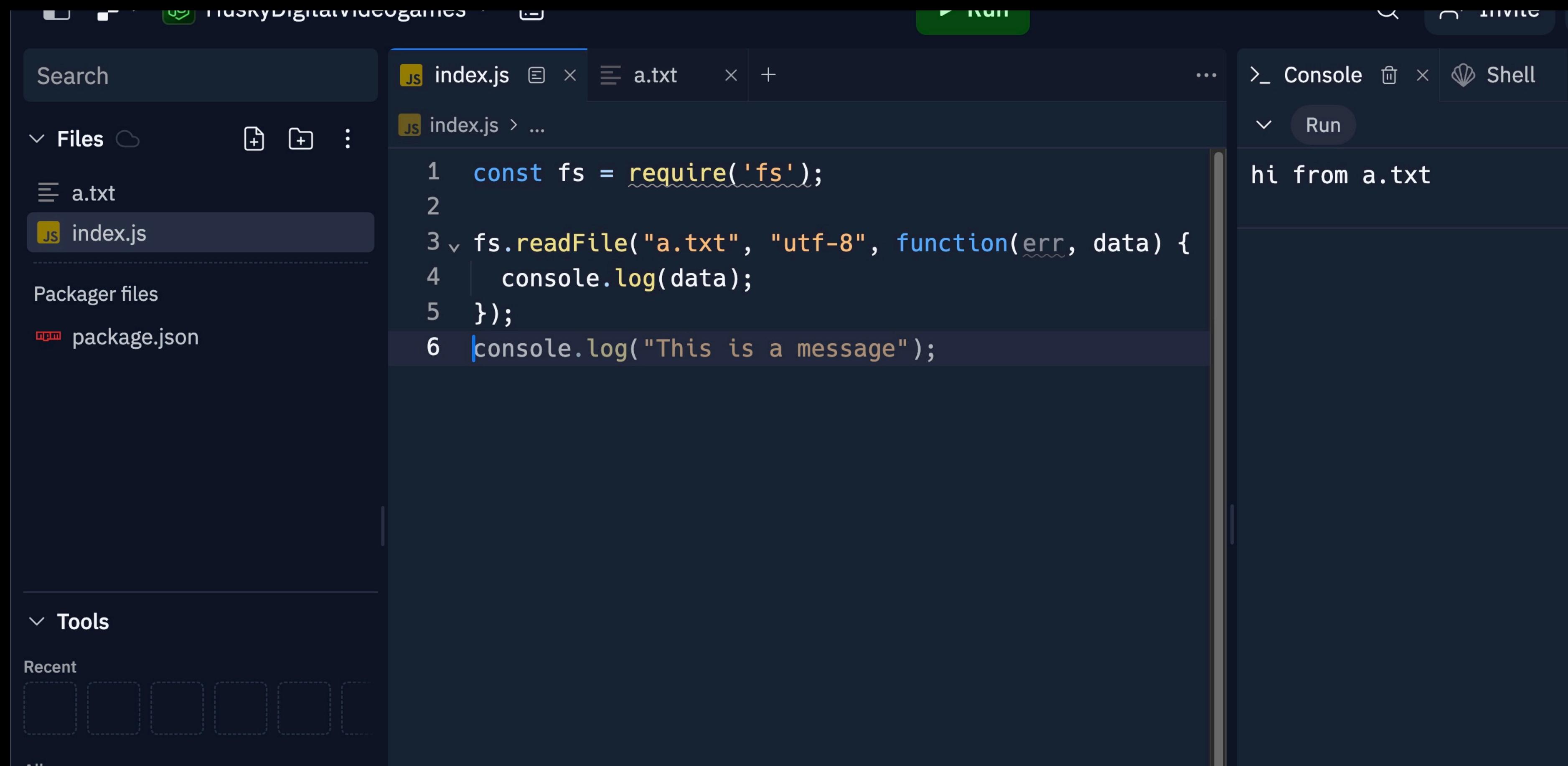


Things that are delegated

Async functions vs sync functions

Lets try `fs` to read a file

<https://gist.github.com/hkirat/8e85f1bc53207ea878801c9f18aab016>



The screenshot shows a code editor interface with a dark theme. On the left, the file tree (Files) shows an `a.txt` file and an `index.js` file. The `index.js` file is open in the main editor area. Its code is as follows:

```
1 const fs = require('fs');
2
3 fs.readFile("a.txt", "utf-8", function(err, data) {
4   console.log(data);
5 });
6 console.log("This is a message");
```

To the right of the editor is a `Console` tab which displays the output of the script's execution: `hi from a.txt`. Below the editor, there is a `Tools` section with a `Recent` subsection containing several small, dashed square icons.



Reads from the filesystem

Async functions vs sync functions

Lets look at the javascript architecture that lets us achieve this asynchronous nature

<http://latentflip.com/loupe>

The screenshot shows the loupe.js developer tool interface. On the left, there is a code editor window titled "loupe" containing the following JavaScript code:

```
1 function printHelloWorld() {
2     console.log("hello world")
3 }
4 setTimeout(printHelloWorld, 5000);
5
6 let ans = 0;
7 for (let i = 1; i<5; i++) {
8     ans = ans + i
9 }
10 console.log(ans);
```

Below the code editor are two buttons: "Save + Run" and "Edit". To the right of the code editor are three performance monitoring panels:

- Call Stack:** A tall, narrow panel showing a stack of numerous frames, indicating a long-running or deeply nested function call.
- Web APIs:** An empty panel.
- Callback Queue:** A tall, narrow panel showing a queue of numerous items, represented by small orange arrows pointing upwards, indicating a large backlog of pending callbacks.

At the bottom center of the interface is a large orange circular arrow icon, likely a refresh or reload button. On the far left, below the code editor, is a small button labeled "Click me!".

Good checkpoint

Async functions vs sync functions, real use of callbacks ✓

JS Browser architecture ✓

Promises

Async await

Promises

Lets see the communication b/w these 2



Timmy



Simmy

Promises

Lets see the communication b/w these 2



Timmy



Simmy

Timmy gives her a task, and a **callback function**



Promises

Lets see the communication b/w these 2

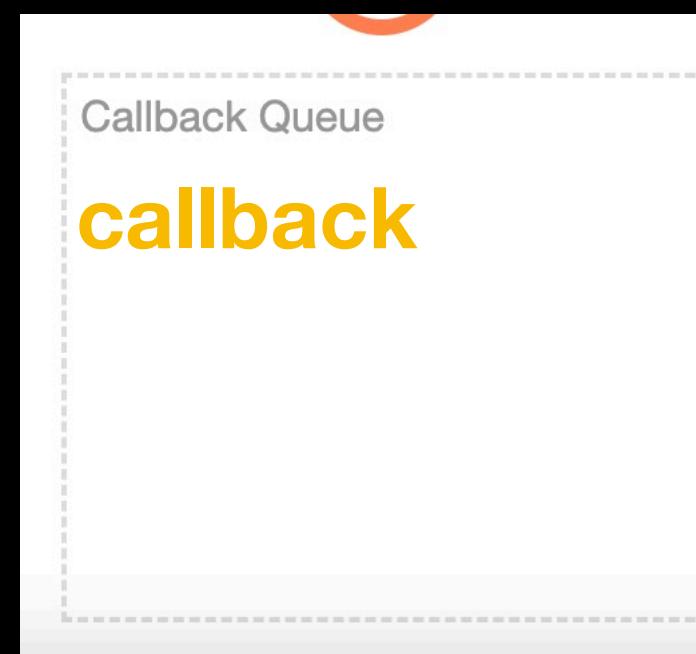


Timmy



Simmy

She does the task (reads from file/waits for 1s ...)



Promises

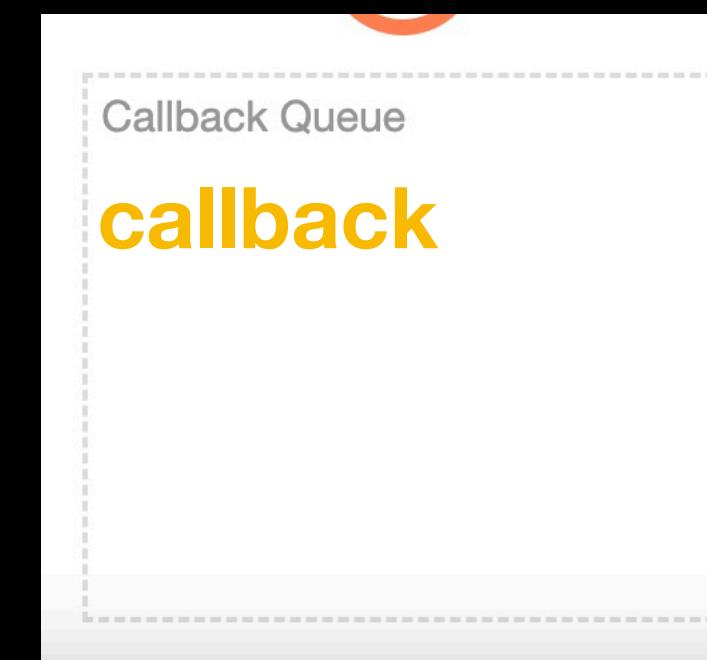
Lets see the communication b/w these 2



Timmy



Simmy



She puts the result in the **callback** queue

Promises

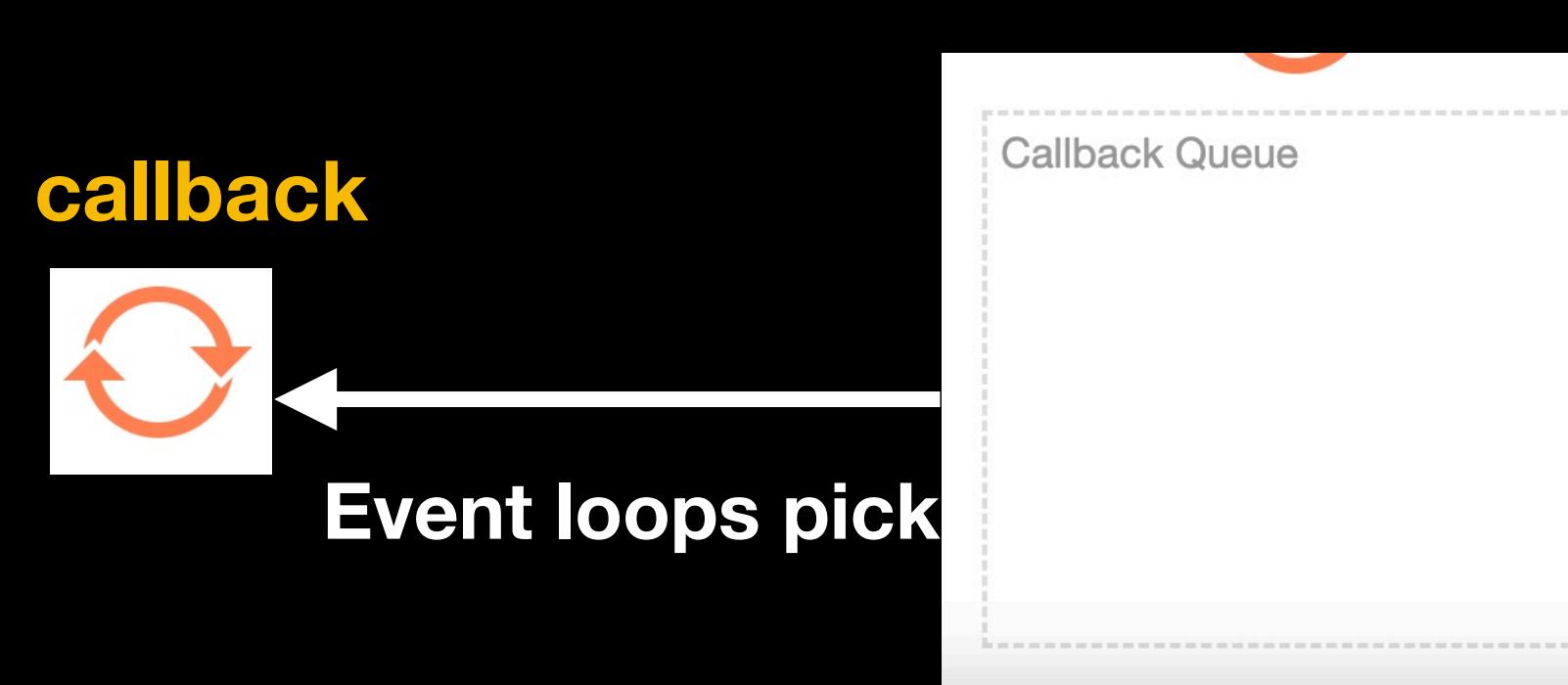
Lets see the communication b/w these 2



Timmy



Simmy



Promises

Lets see the communication b/w these 2



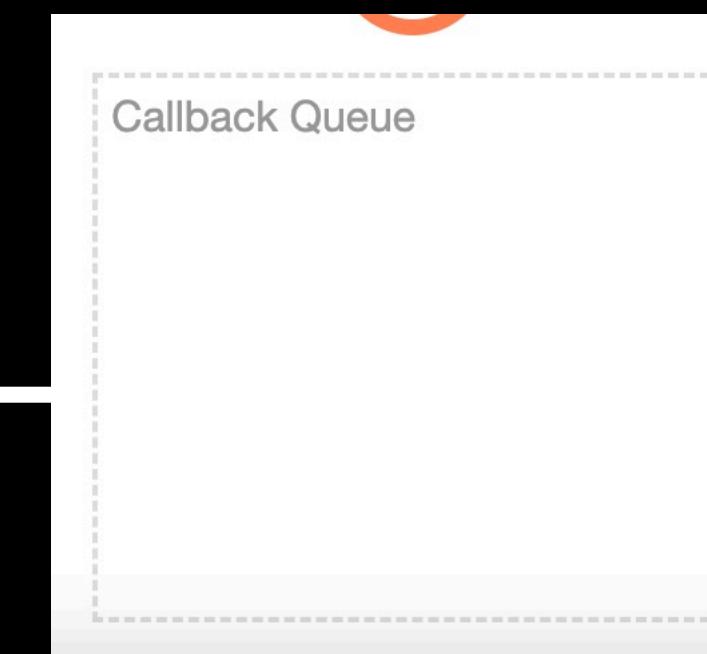
callback()



Timmy

Simmy

Timmy executes it



Promises

What does the code look like?

```
index.js - ...
1 function findSum(n) {
2     let ans = 0;
3     for (let i = 0; i<n; i++) {
4         ans += i;
5     }
6     return ans;
7 }
8
9 function findSumTill100() {
10    return findSum(100);
11 }
12
13 setTimeout(findSumTill100, 1000)
14 console.log("hello world");
```

Promises

This code is ugly

Promises are syntactical sugar that make this code slightly more readable

```
index.js - ...
1 function findSum(n) {
2     let ans = 0;
3     for (let i = 0; i<n; i++) {
4         ans += i;
5     }
6     return ans;
7 }
8
9 function findSumTill100() {
10    return findSum(100);
11 }
12
13 setTimeout(findSumTill100, 1000)
14 console.log("hello world");
```

Promises

Until now, we've only used other people's asynchronous functions

How can we create an asynchronous function of our own?

Promises

Until now, we've only used other people's asynchronous functions
How can we create an asynchronous function of our own?

Ugly way

```
JS index.js > ...
1 const fs = require('fs');
2
3 // my own asynchronous function
4 function kiratsReadFile(cb) {
5   fs.readFile("a.txt", "utf-8", function(err, data) {
6     cb(data);
7   });
8 }
9
10 // callback function to call
11 function onDone(data) {
12   console.log(data)
13 }
14
15 kiratsReadFile(onDone)
```

It is just a wrapper on top of another async function,
which is fine.

Usually all async functions you will write will be on top of
JS provided async functions like setTimeout or fs.readFile

Promises

Until now, we've only used other people's asynchronous functions

How can we create an asynchronous function of our own?

- 1. Breathe before the next slide**
- 2. It's supposed to be overwhelming**
- 3. You only need to understand the syntax for now**
- 4. Don't worry about how it actually works under the hood**

Promises

Until now, we've only used other people's asynchronous functions
How can we create an asynchronous function of our own?

Cleaner way (promises)

```
JS index.js > f kiratsReadFile > ...  
1 const fs = require('fs');  
2  
3 // my own asynchronous function  
4 function kiratsReadFile() {  
5   return new Promise(function(resolve) {  
6     fs.readFile("a.txt", "utf-8", function(err, data) {  
7       resolve(data);  
8     });  
9   })  
10 }  
11  
12 // callback function to call  
13 function onDone(data) {  
14   console.log(data)  
15 }  
16  
17 kiratsReadFile().then(onDone);
```

Its just syntactic sugar
Still uses callbacks under the hood

Lets see how Timmy and Simmy talk

```
JS index.js > f kiratsReadFile > ...
1 const fs = require('fs');
2
3 // my own asynchronous function
4 function kiratsReadFile() {
5   return new Promise(function(resolve) {
6     fs.readFile("a.txt", "utf-8", function(err, data) {
7       resolve(data);
8     });
9   })
10 }
11
12 // callback function to call
13 function onDone(data) {
14   console.log(data)
15 }
16
17 kiratsReadFile().then(onDone);
```



Timmy

Can you read a file, promise me simarpreet



Simmy

Lets see how Timmy and Simmy talk

```
JS index.js > f kiratsReadFile > ...
1 const fs = require('fs');
2
3 // my own asynchronous function
4 v function kiratsReadFile() {
5 v   return new Promise(function(resolve) {
6 v     fs.readFile("a.txt", "utf-8", function(err, data) {
7 v       resolve(data);
8 v     });
9 v   })
10 }
11
12 // callback function to call
13 v function onDone(data) {
14   console.log(data)
15 }
16
17 kiratsReadFile().then(onDone);
```



Timmy

Sure, here's a **promise**,
I may or may not resolve this promise
Returns it immediately



Simmy

(Notice, no callback anywhere)

Lets see how Timmy and Simmy talk

```
JS index.js > f kiratsReadFile > ...
1 const fs = require('fs');
2
3 // my own asynchronous function
4 v function kiratsReadFile() {
5 v   return new Promise(function(resolve) {
6 v     fs.readFile("a.txt", "utf-8", function(err, data) {
7 v       resolve(data);
8 v     });
9 }
10 }
11
12 // callback function to call
13 v function onDone(data) {
14   console.log(data)
15 }
16
17 kiratsReadFile().then(onDone);
```



Timmy

Does her thing, reads the file



Simmy

Lets see how Timmy and Simmy talk

```
JS index.js > f kiratsReadFile > ...
1 const fs = require('fs');
2
3 // my own asynchronous function
4 v function kiratsReadFile() {
5 v   return new Promise(function(resolve) {
6 v     fs.readFile("a.txt", "utf-8", function(err, data) {
7 v       resolve(data);
8     });
9   })
10 }
11
12 // callback function to call
13 v function onDone(data) {
14   console.log(data)
15 }
16
17 kiratsReadFile().then(onDone);
```



Timmy



Calls **resolve** with the final data

Simmy

Lets see how Timmy and Simmy talk

```
JS index.js > f kiratsReadFile > ...
1 const fs = require('fs');
2
3 // my own asynchronous function
4 function kiratsReadFile() {
5   return new Promise(function(resolve) {
6     fs.readFile("a.txt", "utf-8", function(err, data) {
7       resolve(data);
8     });
9   })
10 }
11
12 // callback function to call
13 function onDone(data) {
14   console.log(data)
15 }
16
17 kiratsReadFile().then(onDone);
```

onDone gets called on Timmy's side



Timmy



Simmy

Lets see how Timmy and Simmy talk

Ugly code

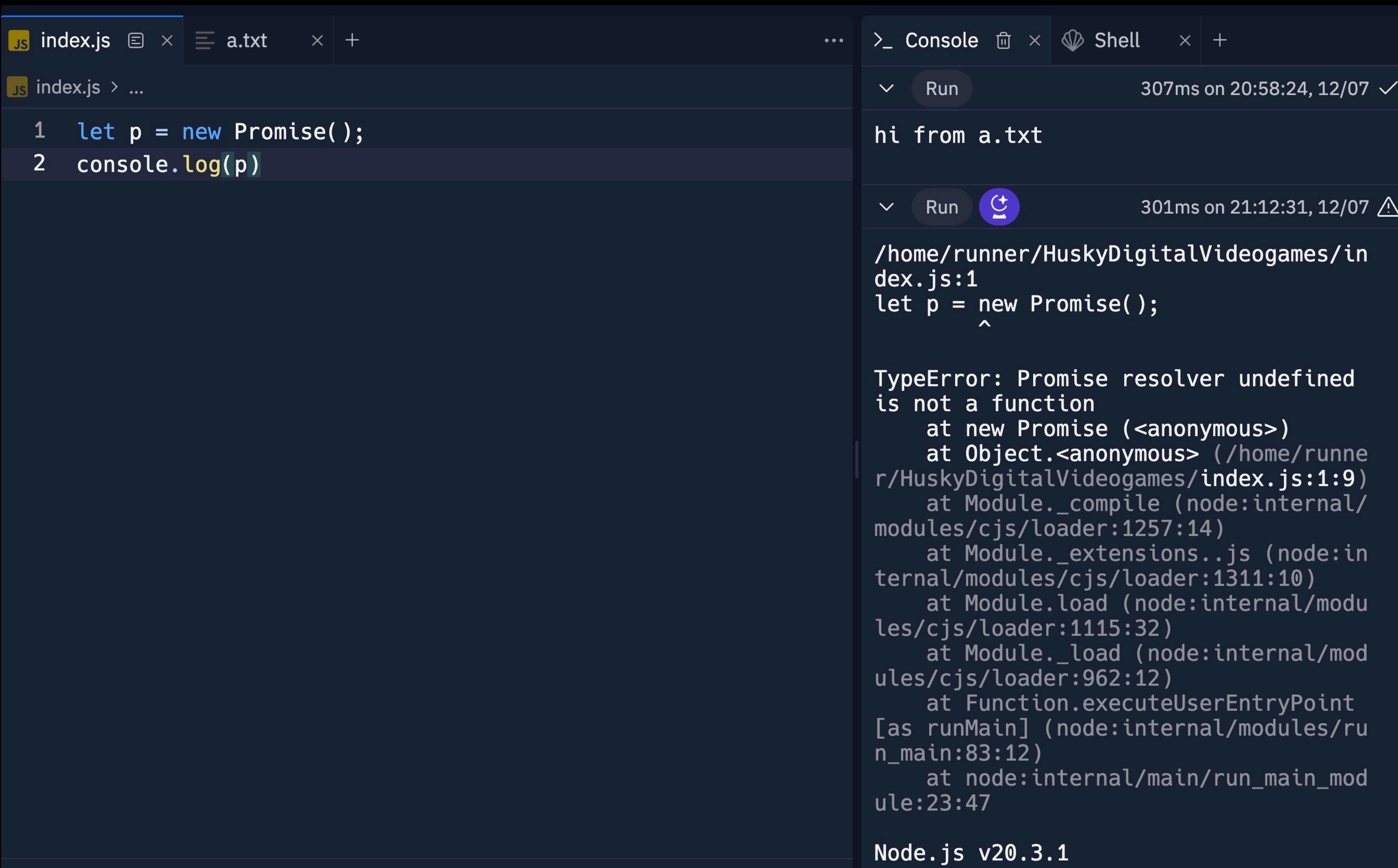
```
JS index.js > ...
1 const fs = require('fs');
2
3 // my own asynchronous function
4 function kiratsReadFile(cb) {
5   fs.readFile("a.txt", "utf-8", function(err, data) {
6     cb(data);
7   });
8 }
9
10 // callback function to call
11 function onDone(data) {
12   console.log(data)
13 }
14
15 kiratsReadFile(onDone)
```

Pretty code

```
JS index.js > f kiratsReadFile > ...
1 const fs = require('fs');
2
3 // my own asynchronous function
4 function kiratsReadFile() {
5   return new Promise(function(resolve) {
6     fs.readFile("a.txt", "utf-8", function(err, data) {
7       resolve(data);
8     });
9   })
10 }
11
12 // callback function to call
13 function onDone(data) {
14   console.log(data)
15 }
16
17 kiratsReadFile().then(onDone);
```

What even is a promise?

It is just a class that makes callbacks and async functions slightly more readable



```
index.js  x  a.txt  x  + ...  
index.js > ...  
1 let p = new Promise();  
2 console.log(p)
```

hi from a.txt

```
Run 307ms on 20:58:24, 12/07 ✓  
Run ⚡ 301ms on 21:12:31, 12/07 !  
/home/runner/HuskyDigitalVideogames/index.js:1  
let p = new Promise();  
^
```

TypeError: Promise resolver undefined
is not a function
at new Promise (<anonymous>)
at Object.<anonymous> (/home/runner/HuskyDigitalVideogames/index.js:1:9)
at Module._compile (node:internal/modules/cjs/loader:1257:14)
at Module._extensions..js (node:internal/modules/cjs/loader:1311:10)
at Module.load (node:internal/modules/cjs/loader:1115:32)
at Module._load (node:internal/modules/cjs/loader:962:12)
at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:83:12)
at node:internal/main/run_main_module:23:47

Node.js v20.3.1

What even is a promise?

Whenever u create it, you need to pass in a function as the first argument which has resolve as the First argument

The screenshot shows a dark-themed code editor interface with two main panels: a code editor on the left and a terminal or console on the right.

Code Editor (index.js):

```
1 v let p = new Promise(function(resolve) {  
2  
3   });  
4 console.log(p)
```

Console:

promise { <pending> }

Here's a simple promise that immediately resolves

The screenshot shows a developer environment with a dark theme. On the left, there is a file tree labeled "eogames" with a "Run" button. Below it, there are two tabs: "index.js" (selected) and "a.txt". The "index.js" tab contains the following code:

```
1 v let p = new Promise(function(resolve) {  
2 |   resolve("hi there");  
3 |});  
4  
5 v p.then(function() {  
6 |   console.log(p);  
7 |})
```

On the right, there is a "Console" tab with a "Run" button and a timestamp "357ms on 21:14:24, 12/07". The console output shows:

```
Promise { 'hi there' }
```

Promises

Place for the writer of the **async function** to do their magic (get ketchup) and call **resolve** at the end with the data

The screenshot shows a developer environment with a dark theme. On the left, there is a file tree labeled 'eogames' with files 'index.js' and 'a.txt'. A green 'Run' button is located above the code editor. The code editor contains the following JavaScript code:

```
1 v let p = new Promise(function(resolve) {  
2 →   resolve("hi there");  
3 );  
4  
5 v p.then(function() {  
6   console.log(p);  
7 })
```

A red box highlights the line '2 → resolve("hi there");'. An arrow points from the text 'Place for the writer of the async function to do their magic (get ketchup) and call resolve at the end with the data' to this highlighted line.

To the right of the code editor is a 'Console' tab showing the output of the run:

```
>_ Console 357ms on 21:14:24, 12/07  
Promise { 'hi there' }
```

Promises

.then gets called whenever the
async function resolves

The screenshot shows a code editor interface with a dark theme. On the left, there are two tabs: 'index.js' (selected) and 'a.txt'. On the right, there are two panes: 'Console' and 'Shell'. The 'Run' button is located at the top center. The code in 'index.js' is:

```
1 v let p = new Promise(function(resolve) {  
2 |   resolve("hi there");  
3 |});  
4  
5 → p.then(function() {  
6 |   console.log(p);  
7 })
```

A red box highlights the 'p.then(function() {' line. An arrow points from the text above to this highlighted line. The 'Console' pane shows the output: 'Promise { \'hi there\' }'. The 'Run' button is green.

Promises

Dummy async function that almost immediately resolves

Actually logging the data with what the function above
Resolved with

The screenshot shows a code editor interface with a dark theme. At the top, there's a header with a user icon, a search bar, and buttons for 'Invite', 'Deploy', and help. Below the header, there are two tabs: 'index.js' and 'a.txt'. The 'index.js' tab is active and contains the following code:

```
1 function kiratsAsyncFunction() {
2   let p = new Promise(function(resolve) {
3     resolve("hi there");
4   });
5   return p;
6 }
7
8 const value = kiratsAsyncFunction();
9 value.then(function(data) {
10   console.log(data);
11 })
12
```

Two specific lines of code are highlighted with red boxes: the resolve statement at line 3 and the console.log statement at line 10. To the right of the editor is a 'Console' panel showing the output: 'hi there'. The status bar at the bottom right indicates a run time of '353ms on 21:17:54, 12/0'.

<https://gist.github.com/hkirat/46580244f43ca2847cf57664a3803b1>

Promises

Try to marinate both the sides

1. Creating an async fn
2. Calling an async fn

The screenshot shows a code editor interface with a dark theme. On the left, there is a file tree with a single file named 'index.js'. The main workspace contains the following code:

```
1 v function kiratsAsyncFunction() {  
2 v   let p = new Promise(function(resolve) {  
3 |     resolve("hi there");  
4 |   });  
5 |   return p;  
6 }  
7  
8 const value = kiratsAsyncFunction();  
9 v value.then(function(data) {  
10 |   console.log(data);  
11 })  
12
```

A green 'Run' button is located at the top center of the editor. To the right of the editor is a sidebar with tabs for 'Console' and 'Shell'. The 'Console' tab is active, showing the output: "hi there". Above the console, there are buttons for 'Invite' and 'Deploy'.

Why even make it async Harkirat seems like you are just doing something that can be done with a normal function

The screenshot shows a dark-themed interface for a cloud-based development environment. At the top, there's a header with a project name "deogames", a "Run" button, and various navigation icons. Below the header, the left pane displays a code editor with an "index.js" file open. The code contains the following asynchronous function:

```
1 v function kiratsAsyncFunction() {  
2 v   let p = new Promise(function(resolve) {  
3 |     resolve("hi there");  
4 |   });  
5 |   return p;  
6 }  
7  
8 const value = kiratsAsyncFunction();  
9 v value.then(function(data) {  
10 |   console.log(data);  
11 })  
12
```

The right pane shows the execution results. It includes a "Console" tab with the output "hi there" and a "Run" status message "353ms on 21:17:54, 12/0".

<https://gist.github.com/hkirat/46580244f43ca2847cf57664a3803b1>

Why even make it async Harkirat seems like you are just doing something that can be done with a normal function

Intimidating async function

```
index.js > ...
1 v function kiratsAsyncFunction() {
2 v   let p = new Promise(function(resolve) {
3 |     resolve("hi there");
4 );
5   return p;
6 }
7
8 const value = kiratsAsyncFunction();
9 v value.then(function(data) {
10 |   console.log(data);
11 })
12
```

Simple function

```
index.js > ...
1 v function kiratsAsyncFunction( ) {
2 |   return "hi there"
3 }
4
5 const data = kiratsAsyncFunction( );
6 console.log(data);
```

You actually can, but you will need to pass in a callback which is what promises help you write in a cleaner fashion

Intimidating async function

```
index.js × a.txt × +  
index.js > ...  
1 v function kiratsAsyncFunction() {  
2 v   let p = new Promise(function(resolve) {  
3 |     setTimeout(resolve, 2000)  
4 |   });  
5 |   return p;  
6 }  
7  
8 const value = kiratsAsyncFunction();  
9 v value.then(function() {  
10 |   console.log("hi there");  
11 })  
12
```

Simple function

```
index.js × a.txt × +  
JS index.js > ...  
1 v function kiratsAsyncFunction(callback) {  
2 |   setTimeout(callback, 2000);  
3 | }  
4  
5 v kiratsAsyncFunction(function() {  
6 |   console.log("hello!");  
7 |});  
8  
9
```

Async functions vs sync functions, real use of callbacks
JS Browser architecture
Promises
Async await

Async await

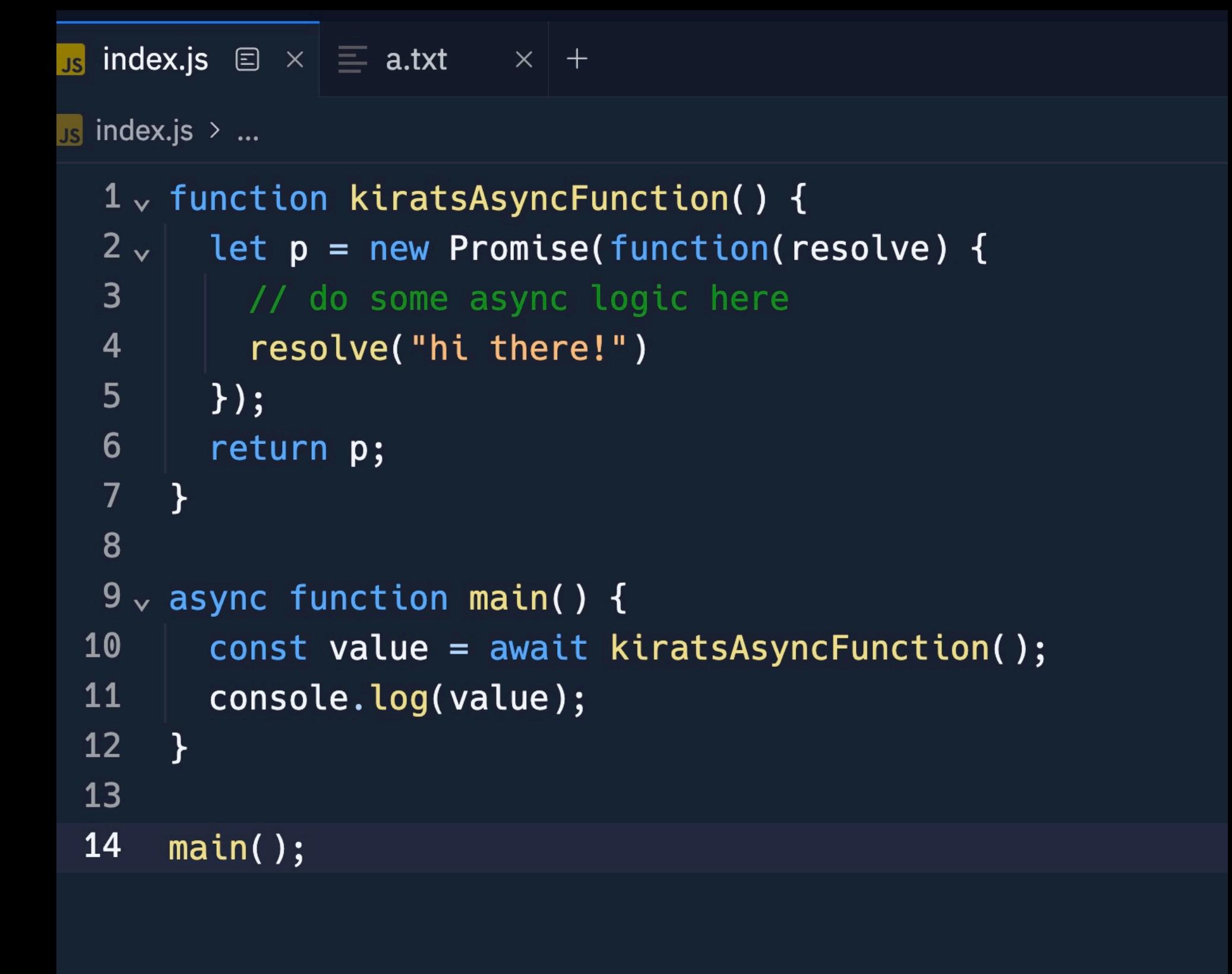
Aynsc await syntax (Again, just syntactic sugar. Still uses callbacks/Promises under the hood)

Normal syntax



```
JS index.js x a.txt x +
JS index.js > ...
1 v function kiratsAsyncFunction() {
2 v   let p = new Promise(function(resolve) {
3   // do some async logic here
4   resolve("hi there!")
5   });
6   return p;
7 }
8
9 v function main() {
10 v   kiratsAsyncFunction().then(function(value) {
11   console.log(value);
12   });
13 }
14
15 main();|
```

Async/await syntax



```
JS index.js x a.txt x +
JS index.js > ...
1 v function kiratsAsyncFunction() {
2 v   let p = new Promise(function(resolve) {
3   // do some async logic here
4   resolve("hi there!")
5   });
6   return p;
7 }
8
9 v async function main() {
10   const value = await kiratsAsyncFunction();
11   console.log(value);
12 }
13
14 main();|
```

Aynsc await syntax

Again, it is just syntactic sugar. Still uses callbacks/Promises under the hood

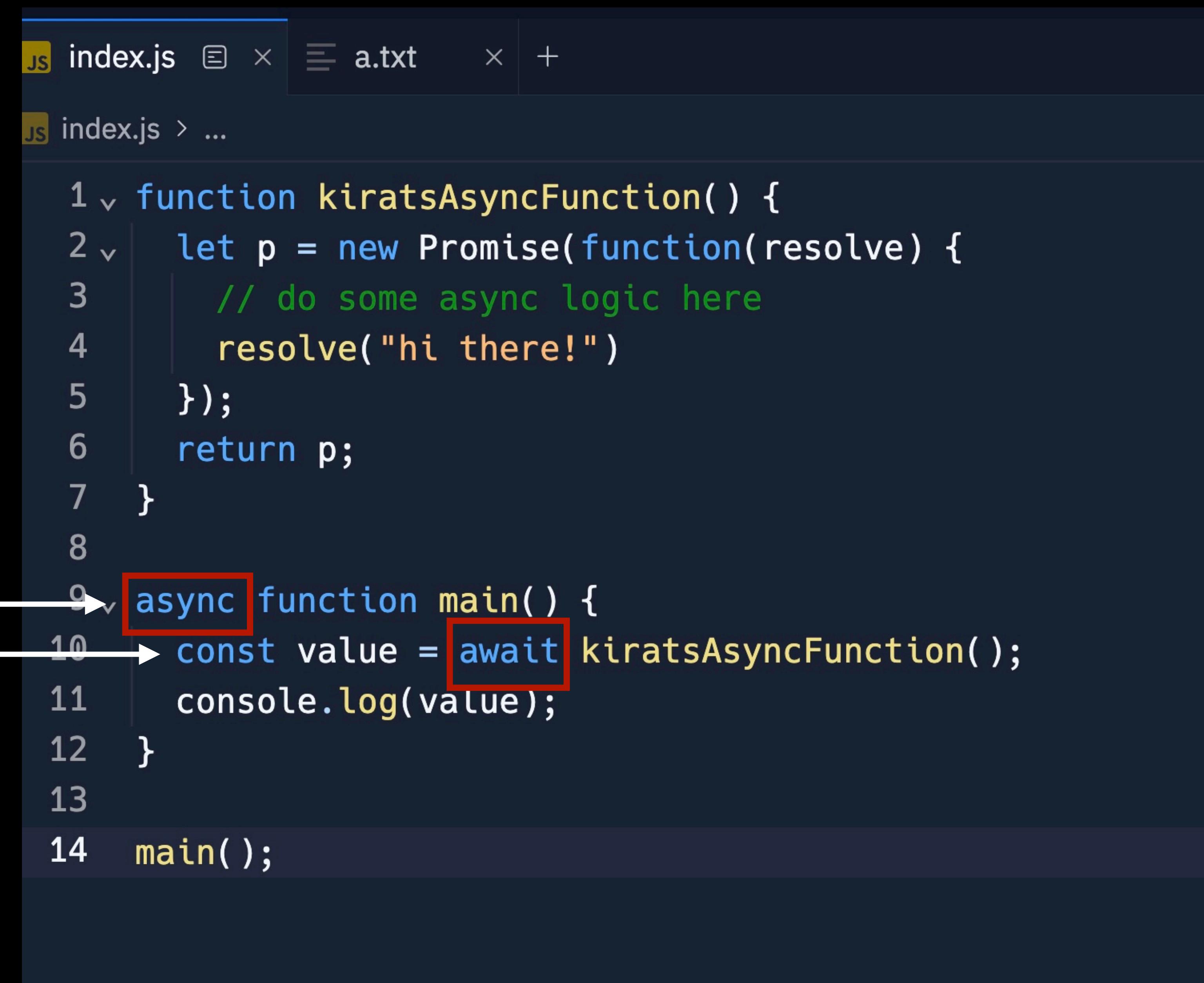
Makes code much more readable than callbacks/Promises

Usually used on the caller side, not on the side where you define an async function

Aynsc await syntax

Any function that wants to use
await, needs to be async

Rather than using the
.then syntax, we add
await before and get the
final value in the variable



The screenshot shows a code editor with two tabs: 'index.js' and 'a.txt'. The 'index.js' tab contains the following code:

```
1 function kiratsAsyncFunction() {  
2   let p = new Promise(function(resolve) {  
3     // do some async logic here  
4     resolve("hi there!")  
5   });  
6   return p;  
7 }  
8  
9 async function main() {  
10   const value = await kiratsAsyncFunction();  
11   console.log(value);  
12 }  
13  
14 main();
```

Annotations highlight the 'async' keyword at line 9 and the 'await' keyword at line 10. Both annotations have red boxes around them and arrows pointing to the corresponding words in the code.

Aynsc await syntax

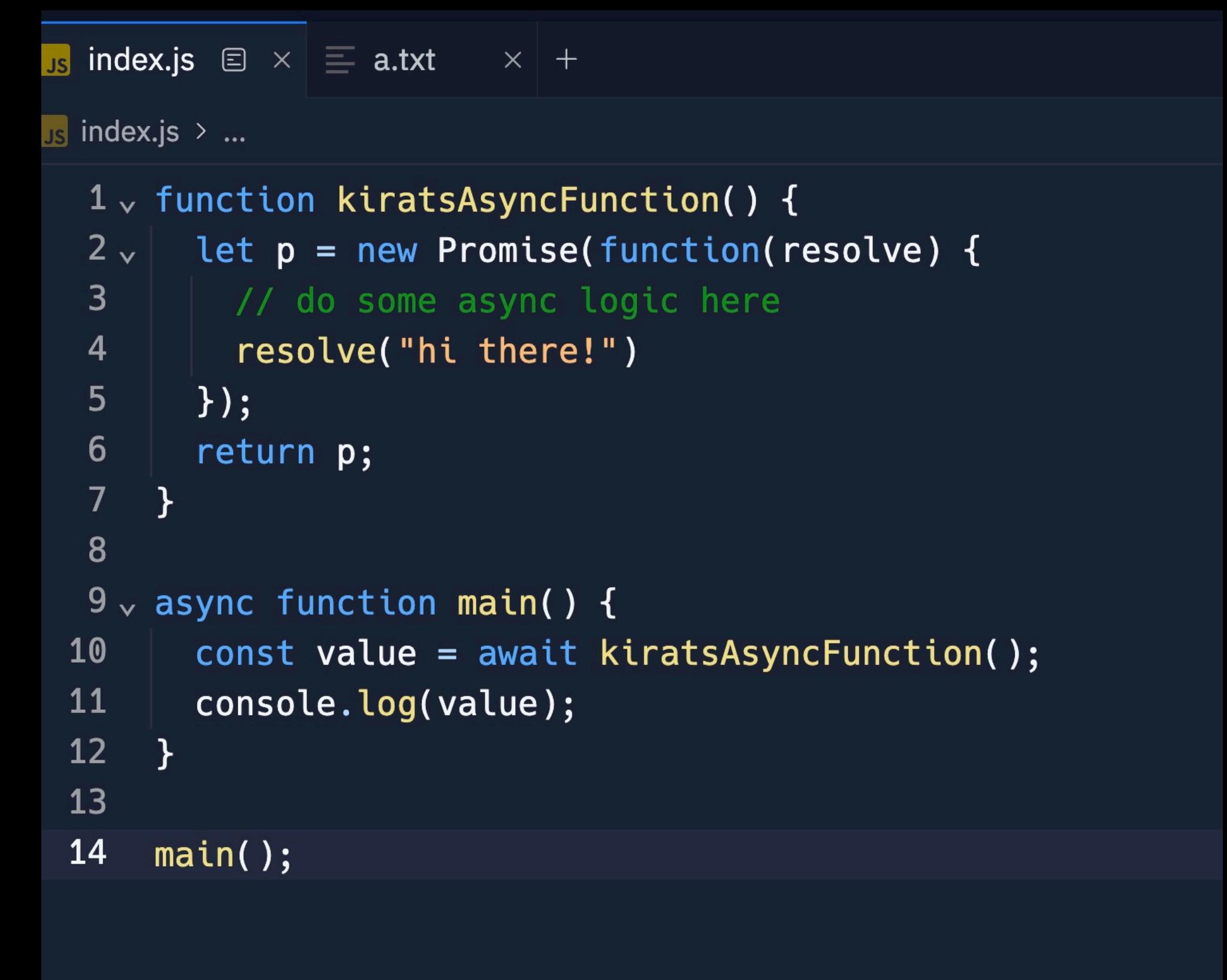
Again, both do the same thing

Normal syntax



```
JS index.js × a.txt × +
JS index.js > ...
1 v function kiratsAsyncFunction() {
2 v   let p = new Promise(function(resolve) {
3   // do some async logic here
4   resolve("hi there!")
5   });
6   return p;
7 }
8
9 v function main() {
10 v   kiratsAsyncFunction().then(function(value) {
11   console.log(value);
12   });
13 }
14
15 main();|
```

Async/await syntax



```
JS index.js × a.txt × +
JS index.js > ...
1 v function kiratsAsyncFunction() {
2 v   let p = new Promise(function(resolve) {
3   // do some async logic here
4   resolve("hi there!")
5   });
6   return p;
7 }
8
9 v async function main() {
10   const value = await kiratsAsyncFunction();
11   console.log(value);
12 }
13
14 main();|
```

Aynsc await syntax

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

Callback syntax

```
index.js > f main > ...

1 v function kiratsAsyncFunction(callback) {
2   // do some async logic here
3   callback("hi there!")
4 }
5
6 v async function main() {
7   kiratsAsyncFunction(function(value) {
8     console.log(value);
9   });
10}
11
12 main();
```

Promise (then) syntax

```
index.js > ...
index.js > ...

1 v function kiratsAsyncFunction() {
2   let p = new Promise(function(resolve) {
3     // do some async logic here
4     resolve("hi there!")
5   });
6   return p;
7 }
8
9 v function main() {
10 v   kiratsAsyncFunction().then(function(value) {
11   console.log(value);
12 });
13 }
14
15 main();
```

Async/await syntax

```
index.js > ...
index.js > ...
index.js > ...

1 v function kiratsAsyncFunction() {
2 v   let p = new Promise(function(resolve) {
3   // do some async logic here
4   resolve("hi there!")
5   });
6   return p;
7 }
8
9 v async function main() {
10 v   const value = await kiratsAsyncFunction();
11   console.log(value);
12 }
13
14 main();
```

<https://gist.github.com/hkirat/9843537dfcaf48ea69df1ec4c4d9091d>

<https://gist.github.com/hkirat/ceac2c9198f1411ba8f5aea3ada769f3>

<https://gist.github.com/hkirat/9e00de2335d1ead90436adef68c9b2f9>