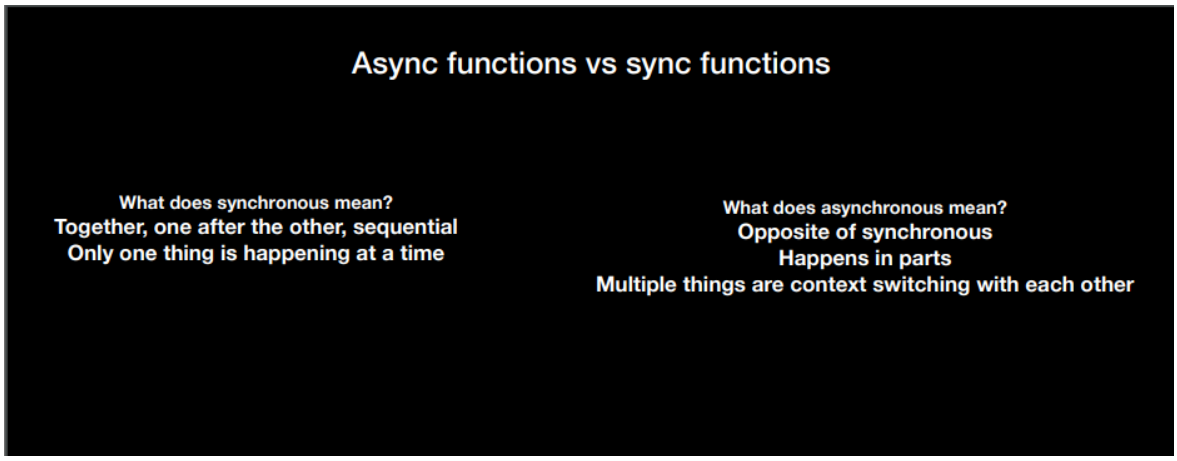
Async functions vs sync functions, real use of callbacks Js
Browser architecture
Promises
Async await

Async functions Vs sync functions



Eg

Human brain and body is single threaded

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

Same is true for Javascript.

Eg

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?

Asynchronous Vs Synchronous function

Synchronous

The lady is making Maggi synchronously it takes

5 min to boil water

2 mins to cut Maggi

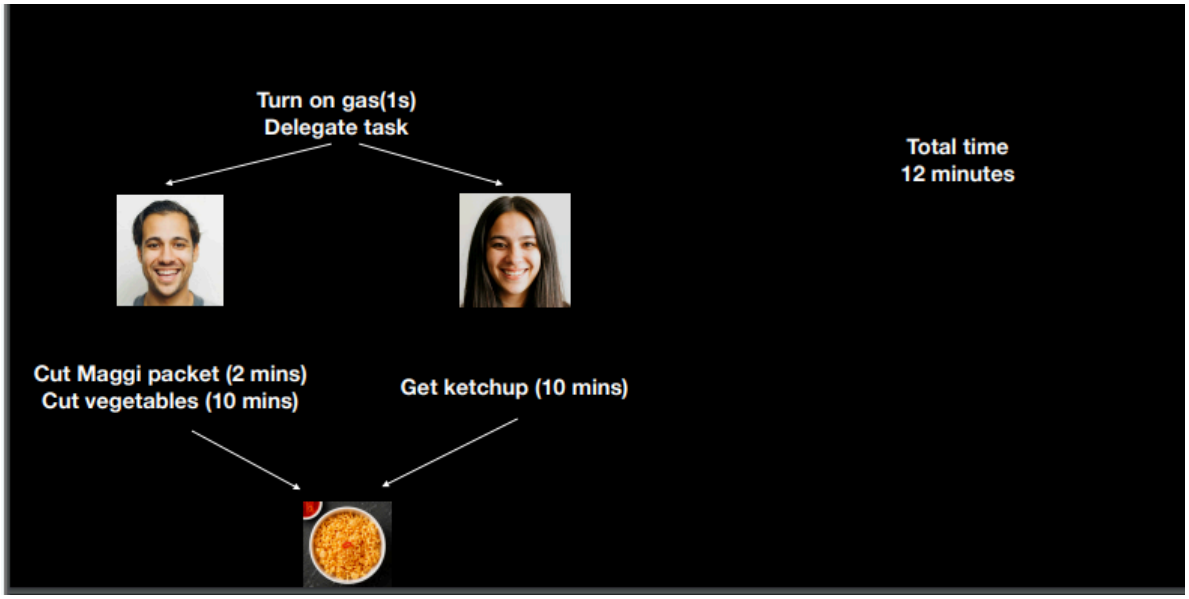
10 minutes to cut vegetables

20 minutes to get ketchup from the shop.

Asynchronous

He is making Maggi in an asynchronous manner

- Put water on the stove and let it boil, then ask help from a friend to bring some ketchup, she is on her way to get ketchup. He context switches between boiling water and opening Maggi packet
- She was back with ketchup in 10 mins
- In those ten minutes water is boiled and he is in the middle of cutting vegetables (8 mins done) she waits for two minutes to finish, cutting vegetables and after this, everything is done in 12 minutes



Eg : Is reading a file from the file system

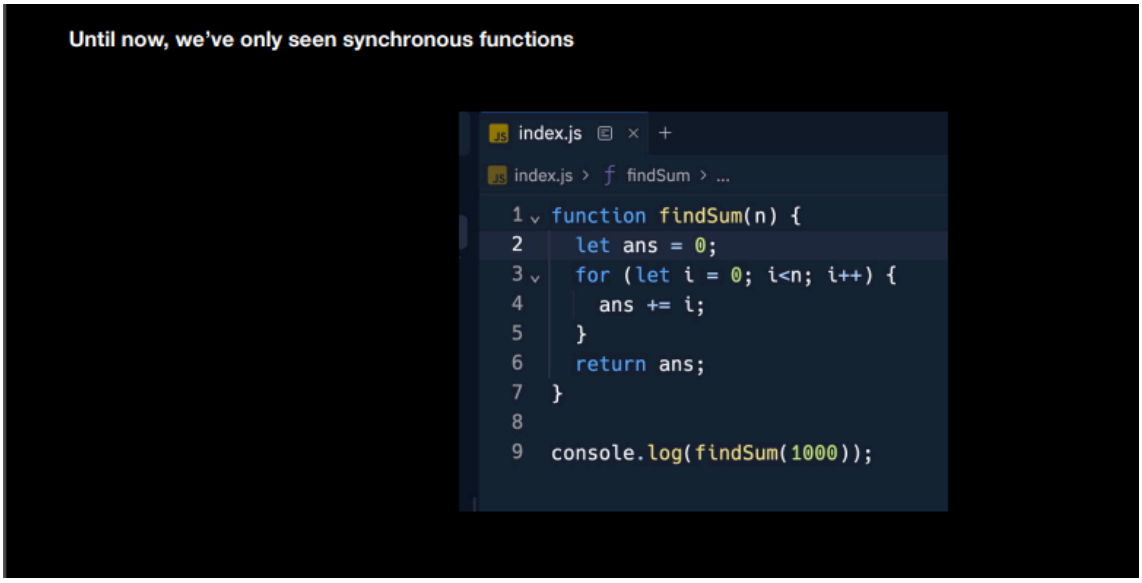
Net amount of time take to do a task can be decreased by doing these two things (delegating and context switching)

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

Yes , by using **asynchronous functions**

Until now we have seen synchronous functions only.

JS thread is busy and does anything else while inside the loop



Asynchronous function

setTimeout() is a global function Js give us, it is used to run a function after some duration. This is an Asynchronous function because when it reaches line no. 13, Js doesn't wait for one second (1000 ms) and goes to line no. 14 . After one minute setTimeout() will let it know after 1 second(Calls the callback).

```
function findSum(n) {
    let ans = 0;
    for (let i = 0; i<n; i++) {
      ans += i;
```

```
    }
    return ans;
  }

  function findSumTill100() {
    console.log( findSum(1000));
    // will log the sum of numbers from 0 to 999 after 2 seconds
  }

  setTimeout(findSumTill100, 2000)
  console.log("hello world");
  //This will be printed first
```

We can also create a synchronous setTimeout() function

Dumb way:

```
function findSum(n) {
    let ans = 0;
    for (let i = 0; i<n; i++) {
      ans += i;
    }
    return ans;
  }

  function findSumTill100() {
    console.log( findSum(1000));
  }
  //busy waiting
  function syncSleep(){
    let a=1;
    for(let i=0;i<1000000000;i++){
      a++;
    }
    // Synchronously sleeping thread
  }
  syncSleep();
  findSumTill100();
//   setTimeout(findSumTill100, 2000)
  console.log("hello world");
  //This will be printed first
```

4950

hello world

Common async function?

setTimeout()

fs.readFile- to read a file from your filesystem

Fetch - to fetch some data from an API endpoint

Eg: To read a file from your file system

```
const fs = require('fs');

fs.readFile("a.txt", "utf-8", function(err, data) {
  console.log(data);
});
console.log("hello world");
```

Console:

“hello world” prints before the

```
PS C:\Users\NTC\Desktop\Dev\Week1> cd c:\Users\NTC\D
esktop\Dev\Week1\Async_Await_Promises
PS C:\Users\NTC\Desktop\Dev\Week1\Async_Await_Promis
es> node "c:\Users\NTC\Desktop\Dev\Week1\Async_Await
_Promises\tempCodeRunnerFile.js"
hi there guyz i am the don of UK
PS C:\Users\NTC\Desktop\Dev\Week1\Async_Await_Promis
es>
ses  ⊗ 0 ⚠ 0  0  JavaScript  Go Live  Prettier
```

Make sure you are in the correct directory structure

```
const fs = require('fs');

fs.readFile("a.txt", "utf-8", function(err, data) {
```

```
  console.log(data);
});
console.log("hello world");

let a=0;
//takes very long time
for(let i=0;i<1000000000;i++){
  a++;
//   thread will not go to the file until this loop is finished, even if the callback is ready(fs.readFile)
}
console.log("hello world 2");
```

Console:

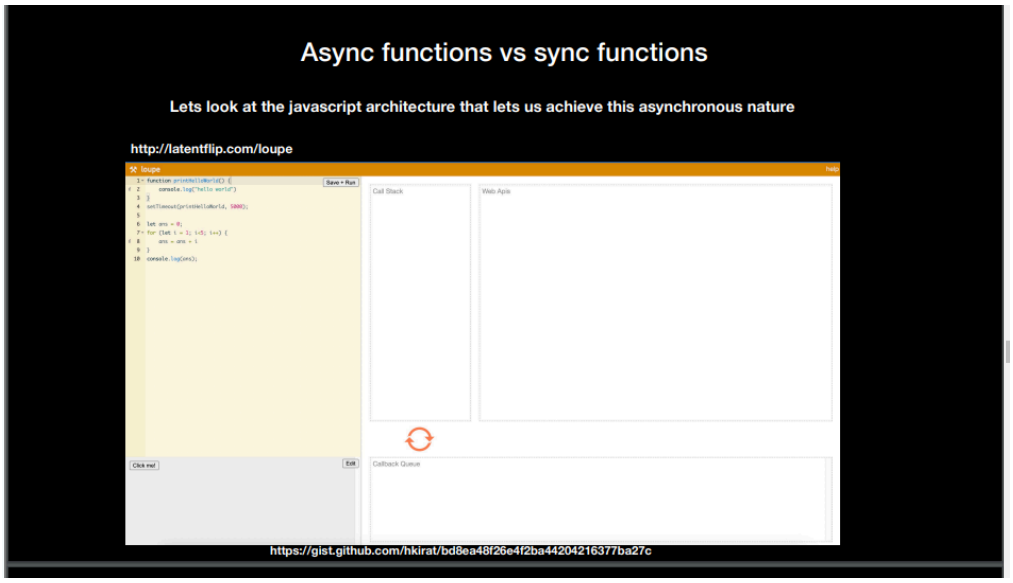
hello world

hello world 2

Hi there guys this is a.txt

Until the thread is free from the loop, control will not go for pending callback (even if it is asynchronous nature)

To visualize



There are four things in high-level in Javascript

Call stack(line of code running)

Web Apis

Callback Queue

Event loop(the job is to check if is there something in Callback Queue after the thread is free and put in Call Stack)

Write different synchronous and asynchronous code and observe call stack and web APIs(delegate a task to, isn't part of js), etc

Eg

```
console.log("hi there")
setTimeout(function(){
    console.log("from inside the asynchronous function")
},20000)
setTimeout(function(){
    console.log("from inside the asynchronous function 2")
},10000)

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

In the Callback Queue second setTimeout() function will be on the top (first)

Output:

hi there

9

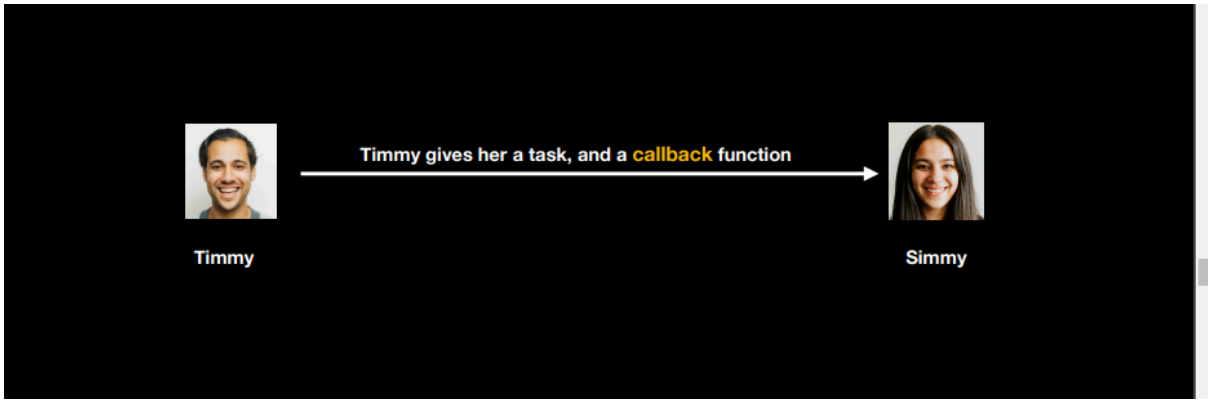
from inside the asynchronous function 2

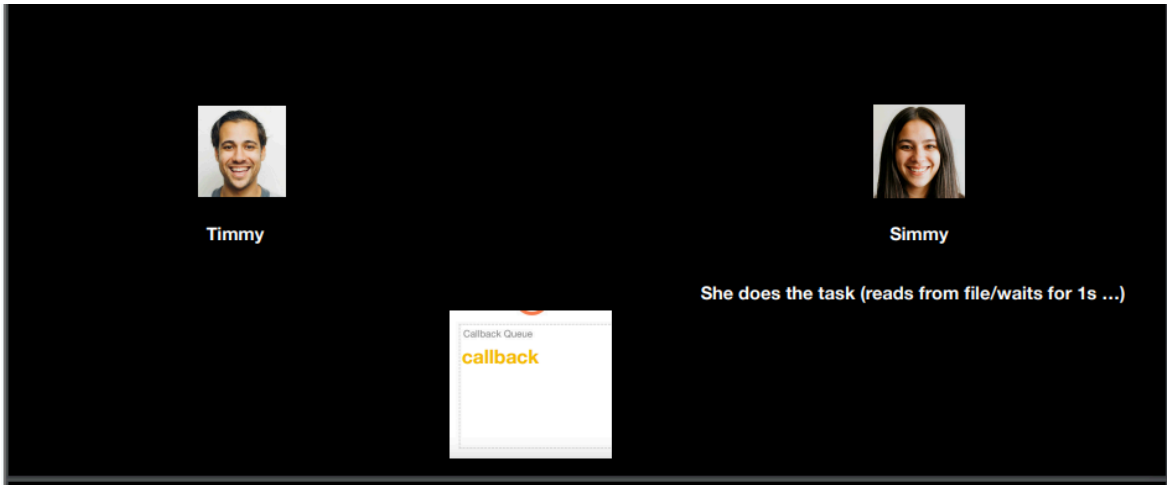
from inside the asynchronous function

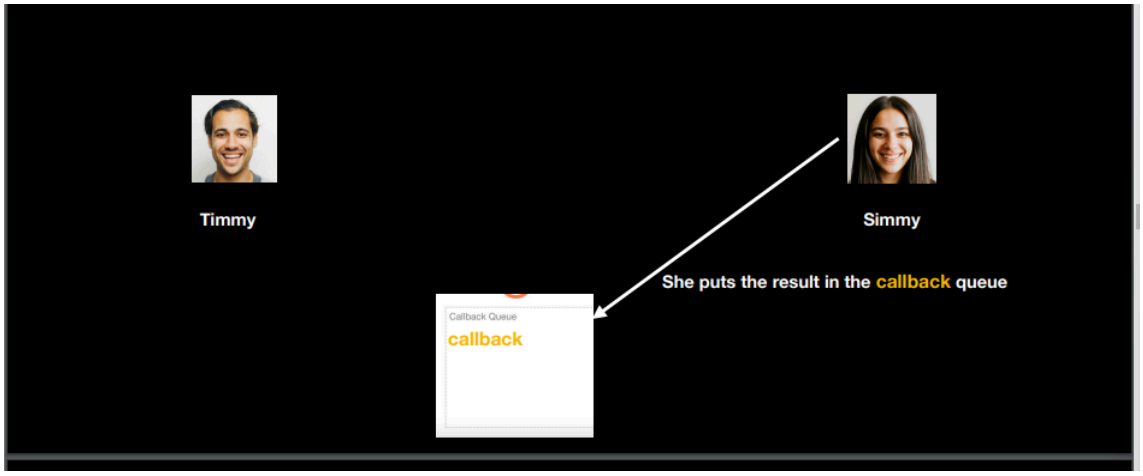
Callback doesn't make much sense in synchronous function

Promises

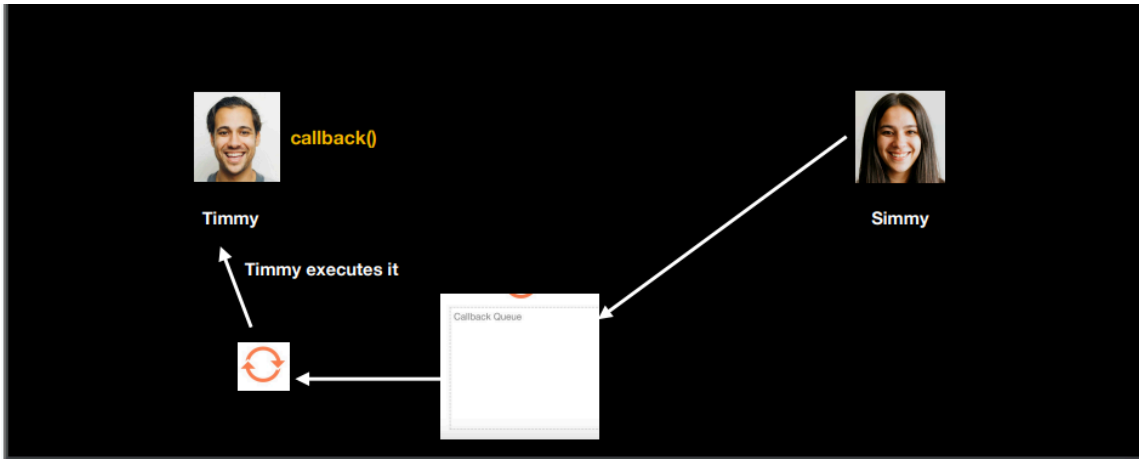
Let see communication between two







When timmy is free



Code will look like this

```
function findSum(n) {
    let ans = 0;
    for (let i = 0; i<n; i++) {
      ans += i;
    }
    return ans;
  }
  function findSumTill1000() {
    console.log( findSum(1000));
  }
  setTimeout(findSumTill1000, 1000)
    console.log("hello world");
```

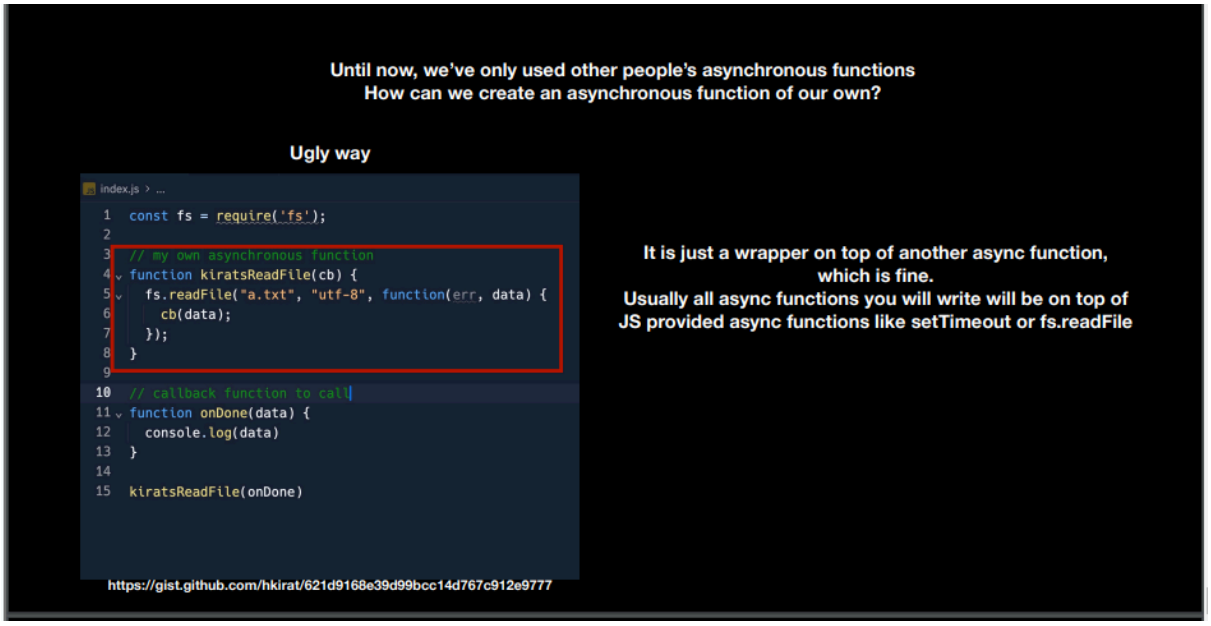
No promises introduced yet

This code is not good looking.

Promises are syntactical sugar that make this code slightly more readable

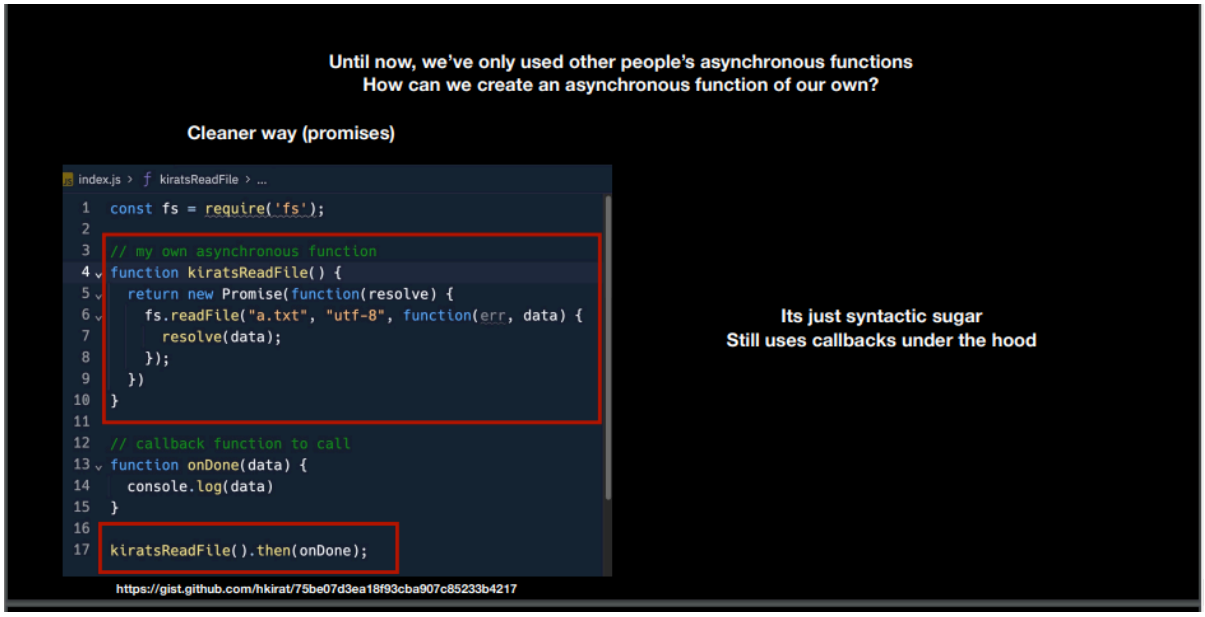
Under the hood it still uses callback, event loop, web apis etc

Until now we have only used other people asynchronous function . How can we create an asynchronous function of our own?



Just creating a wrapper on top of others asynchronous function.

The cleaner way is to use promises



This function at very top doesnt have a callback function . How does simmy know when to call when she does get the ketchup. Purpose of promises is to get rid of callbacks , since callbacks are ugly way to write asynchronous function. (Callback hell)

Basically it means i will read file and pass data to resolve and call it .
Lets see how Simmy and timmy communicate

Simmy synchronously returns promise (not data), that was we were able to log it. But data comes asynchronously.

Simmy says to Timmy please put .then on the promise so we(Simmy) know where to send the data

Timmy does the main logic on the main JS thread

Code:

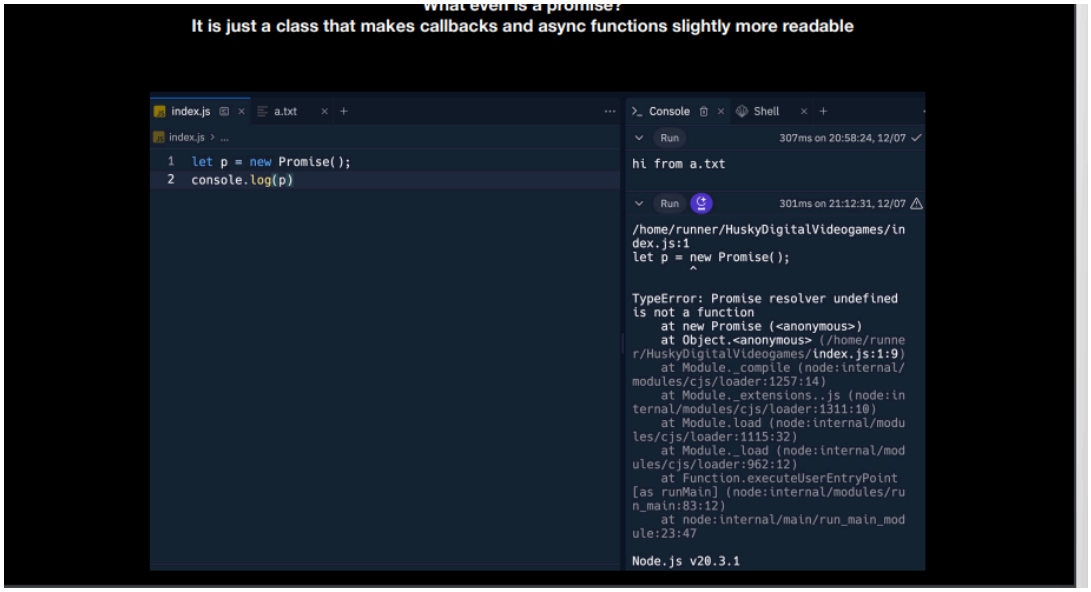
```
const fs = require('fs');

// my own asynchronous function
function kiratsReadFile() {
    console.log("inside kiratsReadFile")
  return new Promise(function(resolve) {
```

```
    console.log("inside promise")
    fs.readFile("a.txt", "utf-8", function(err, data) {
        console.log("before resolve")
      resolve(data);
    });
  })
}

// callback function to call
function onDone(data) {
  console.log(data)
}

var a = kiratsReadFile();
console.log(a)
// log instance of promise
a.then(onDone);
```

Output:

```
PS C:\Users\NTC\Desktop\Dev\Week1\Async_Await_Promises> node "c:\Users\NTC\
Desktop\Dev\Week1\Async_Await_Promises\practice.js"
inside kiratsReadFile
inside promise
Promise { <pending> }
before resolve
hi there guyz i am the don of UK
PS C:\Users\NTC\Desktop\Dev\Week1\Async_Await_Promises>
```

What is Promise?



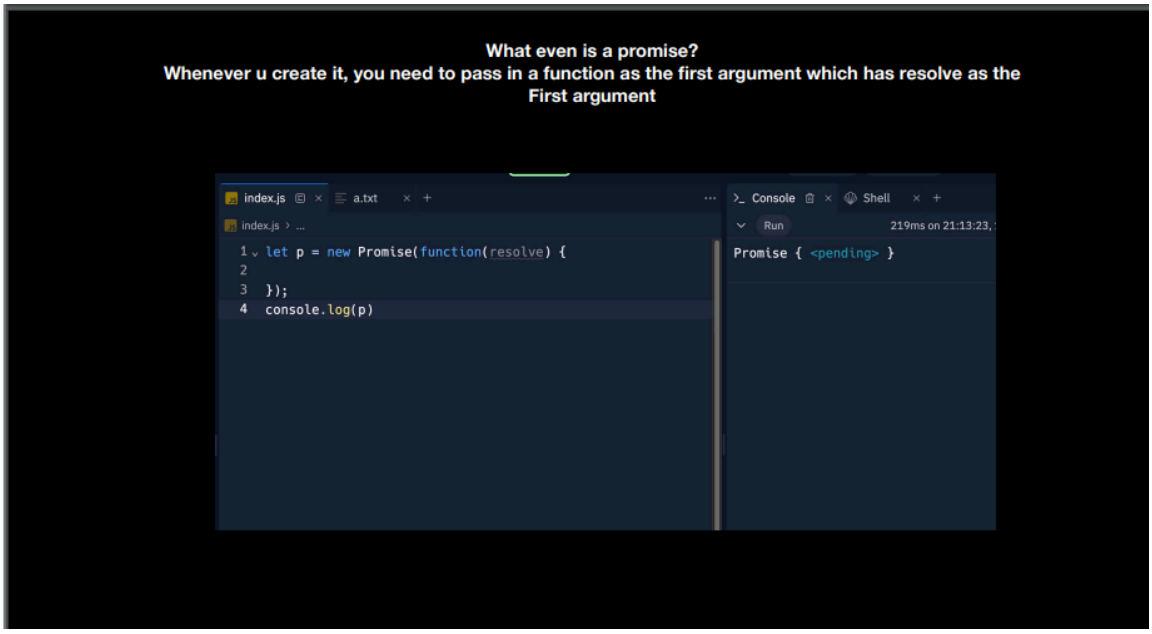
Eg:

```
// A promise at a high level has three state, Pending, resolved, and rejected

// Promise is like just any other class it can also be used outside the function
var d = new Promise(function(resolve){
    // Promise must consist function, and we need the first argument as resolve (we can name it anything)
    setTimeout(function(){
        resolve("Chutney");
    },1000)
})

function callback(){
    console.log(d);//Promise state is resolved
}
console.log(d);//Here the promise state is pending
d.then(callback);
```

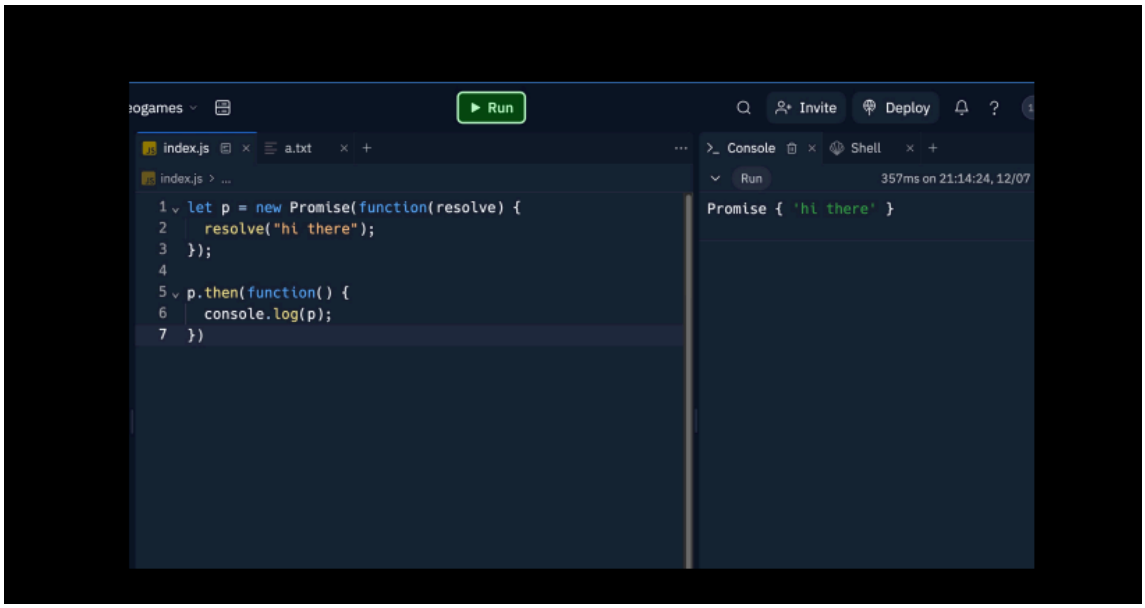
Output:

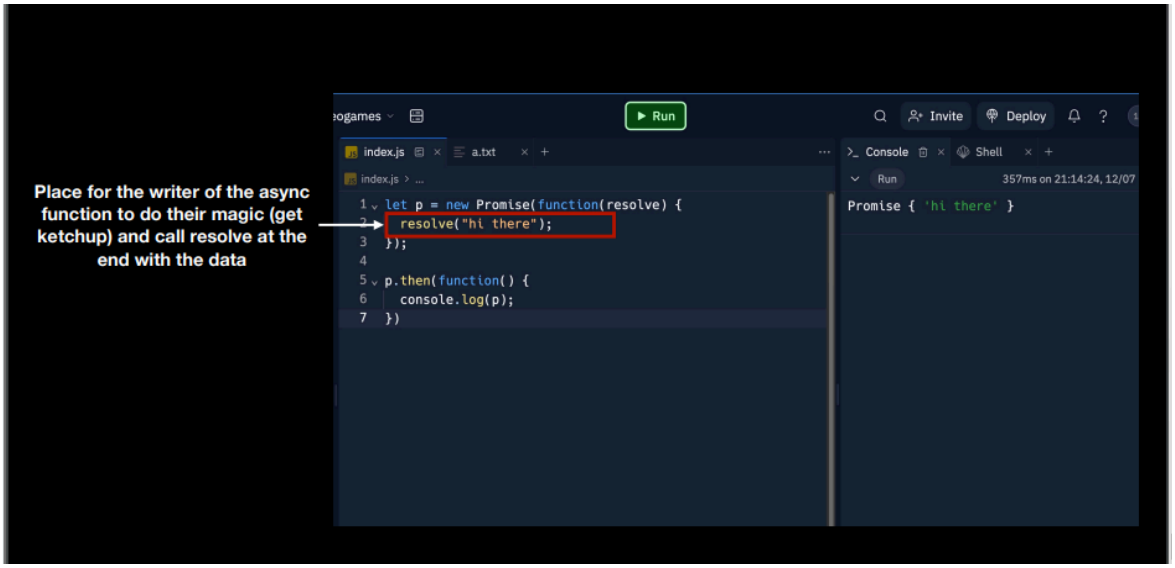
Promise { <pending> }

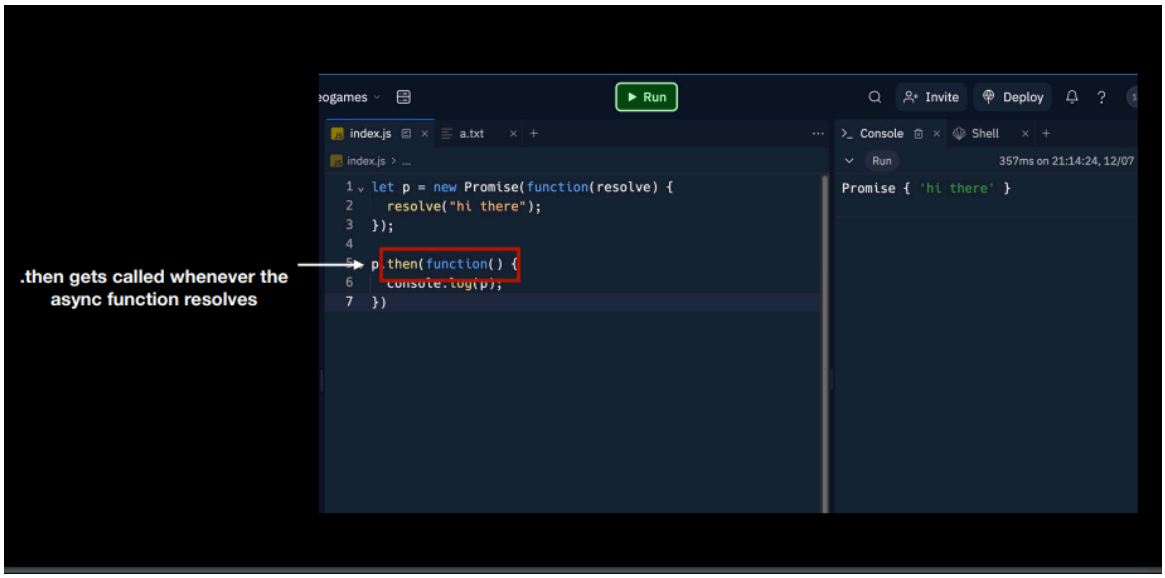
Promise { 'Chutney' }

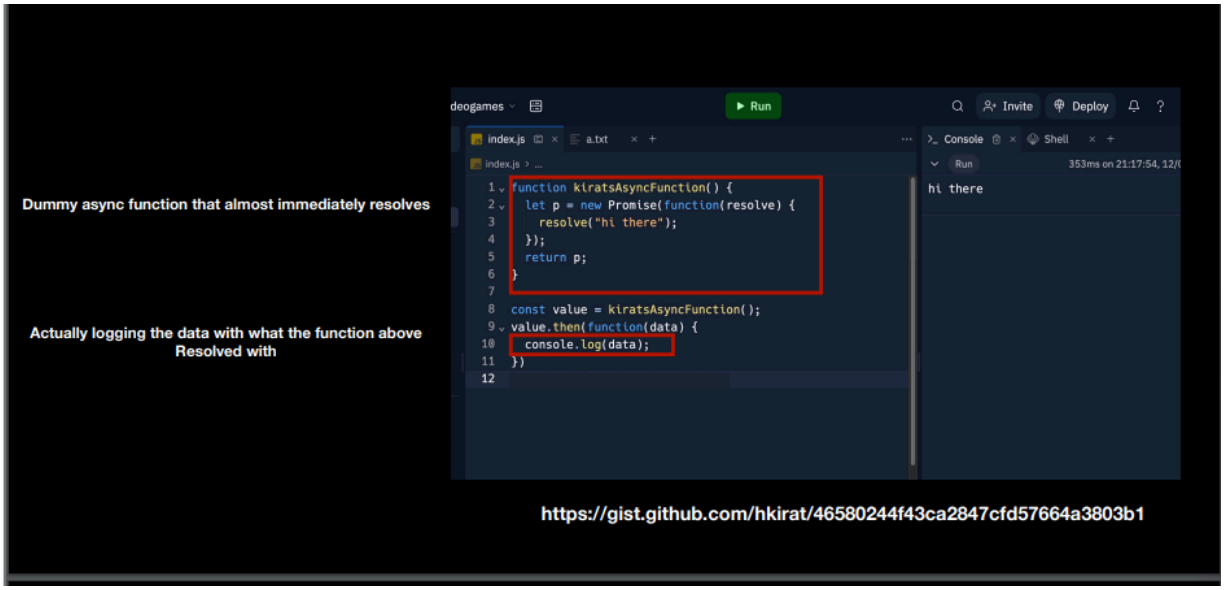


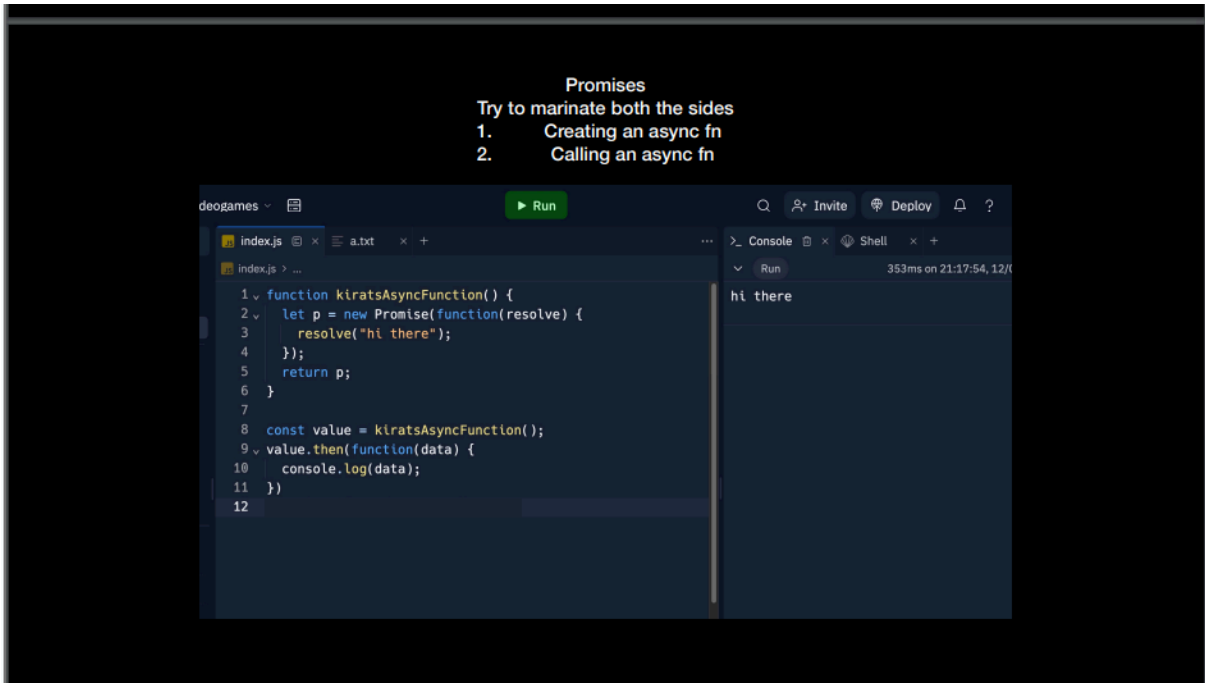
A simple promise that immediately resolve



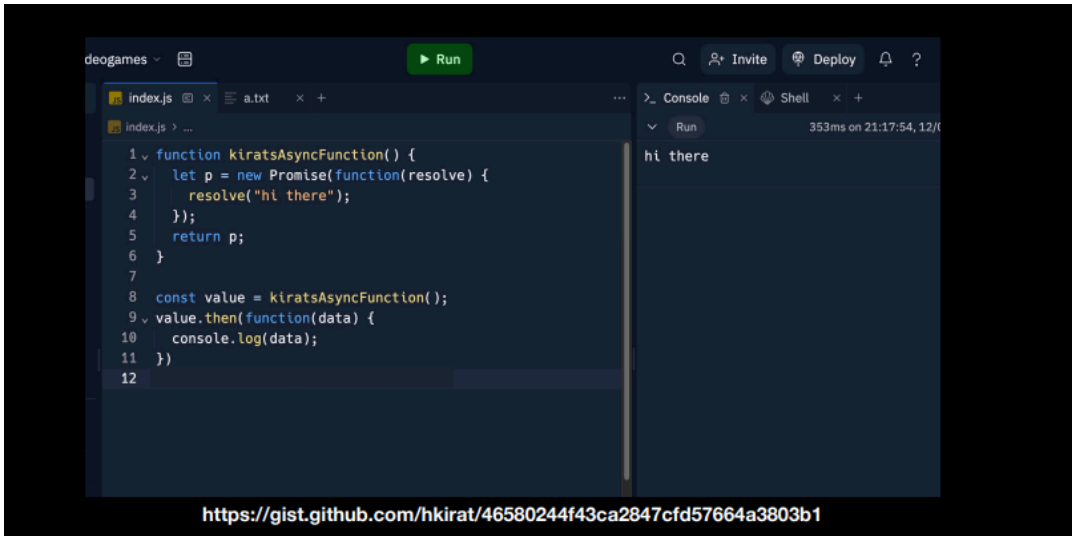

Place for the writer of the async function to do their magic (get ketchup) and call resolve at the end with the data
Run
Invite
Deploy
index.js
a.txt
Console
Shell
Run
357ms on 21:14:24, 12/07
Promise { 'hi there' }
let p = new Promise(function(resolve) {
resolve("hi there");
});
p.then(function() {
console.log(p);
})


.then gets called whenever the async function resolves
Run
Invite
Deploy
index.js
a.txt
Console
Shell
Run
357ms on 21:14:24, 12/07
Promise { 'hi there' }
let p = new Promise(function(resolve) {
resolve("hi there");
});
p.then(function() {
console.log(p);
})





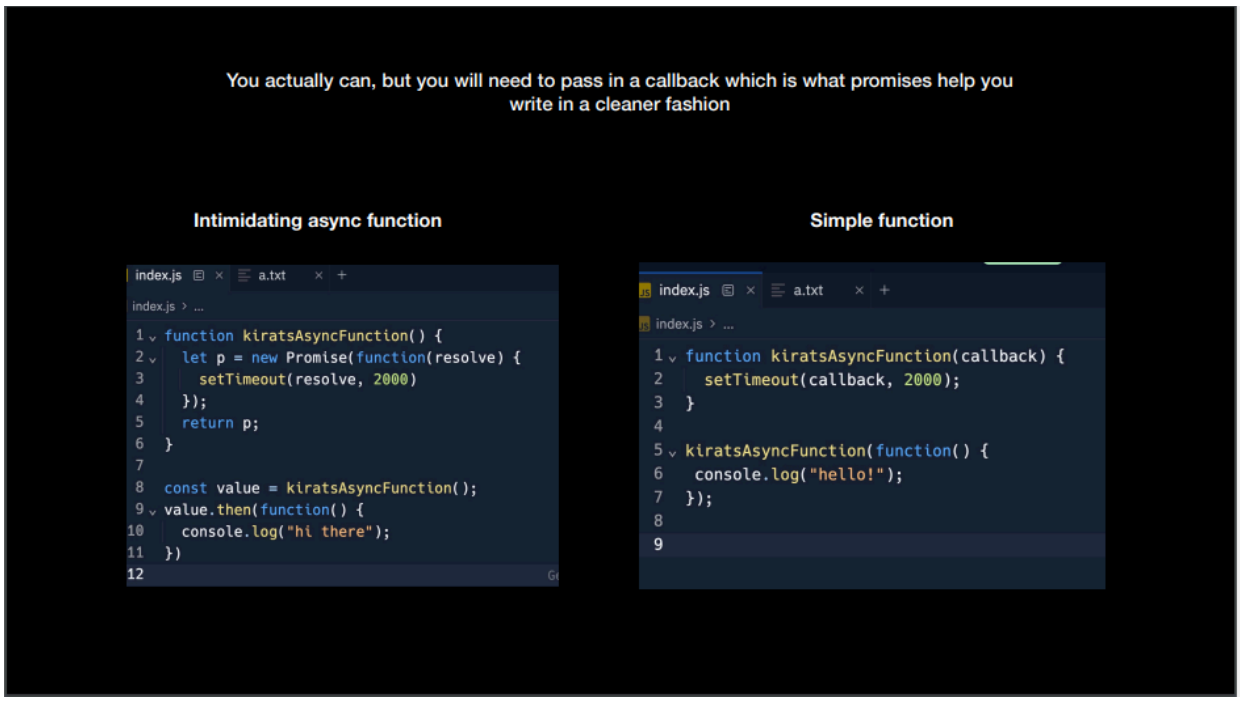
Why even make it async it seems like you are just doing something that can be done with a normal function



It can be simple done by

```
function kiratAsyncFunction(){
    return "hi there";
}
const data = kiratAsyncFunction();
console.log(data);
```

Let's see another example



Async / Await

Still uses callback/Promises under the hood.

Normal Syntax

```
function kiratsAsyncFunction() {
    let p = new Promise(function(resolve) {
      //Do some async logic here,instead of immediate resolving
      resolve("hi there!")
    });
    return p;
  }

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

  main();
```

Async/Await Syntax:

```
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);
    //We get rid of .then() syntax
  }

  main();
```

It logs

hi there!

async: this function will be now asynchronous

If we console.log(kiratAsyncFunction())

It will log

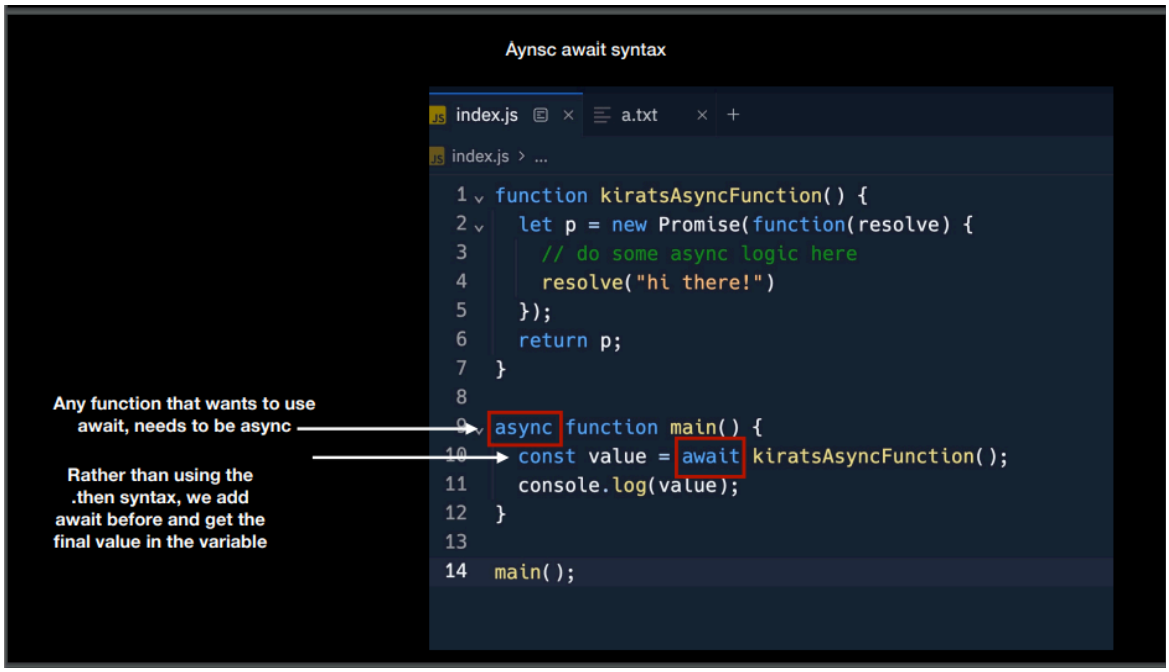
Promise {‘hi there’}

And if there is an asynchronous logic inside promise then it will log

Promise {<pending>}

Async/Await Syntax:

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.



Every await must be wrapped inside an async function.

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

Callback syntax

index.js > main > ...

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

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

Promise (then) syntax

index.js a.txt

index.js > ...

```
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  function main() {
10   kiratsAsyncFunction().then(function(value) {
11       console.log(value);
12   });
13 }
14
15 main();
```

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

Async/await syntax

index.js a.txt

index.js > ...

```
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();
```

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