

# **Async functions vs sync functions, real use of callbacks Js**

## **Browser architecture**

### **Promises**

### **Async await**

#### **Async functions Vs sync functions**

Async functions vs sync functions	
<p>What does synchronous mean? Together, one after the other, sequential Only one thing is happening at a time</p>	<p>What does asynchronous mean? Opposite of synchronous Happens in parts Multiple things are context switching with each other</p>

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

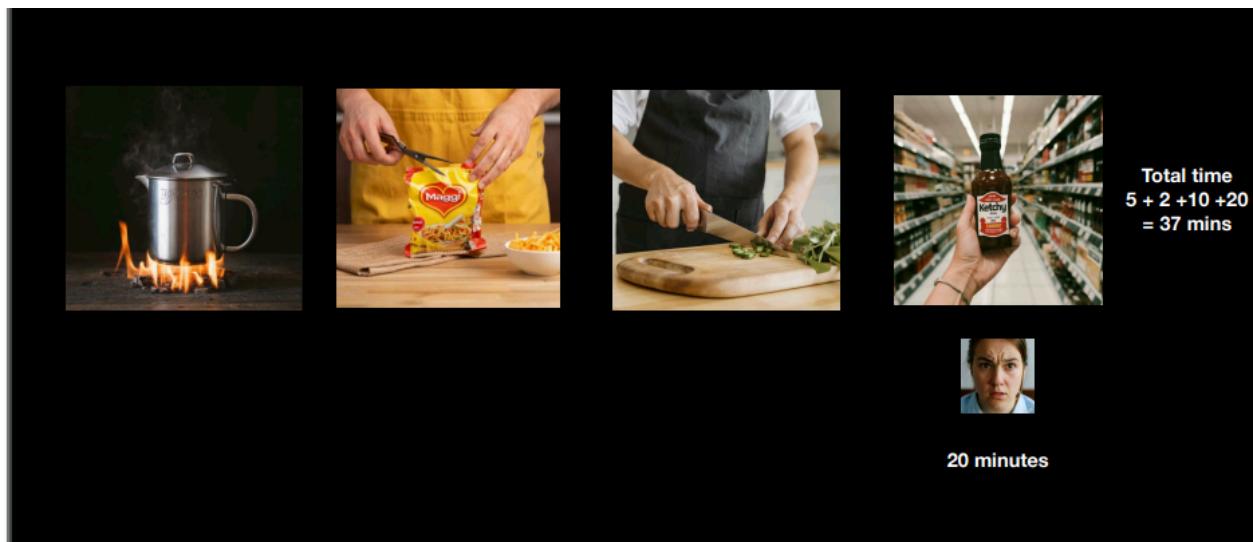
You have 4 tasks:-

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

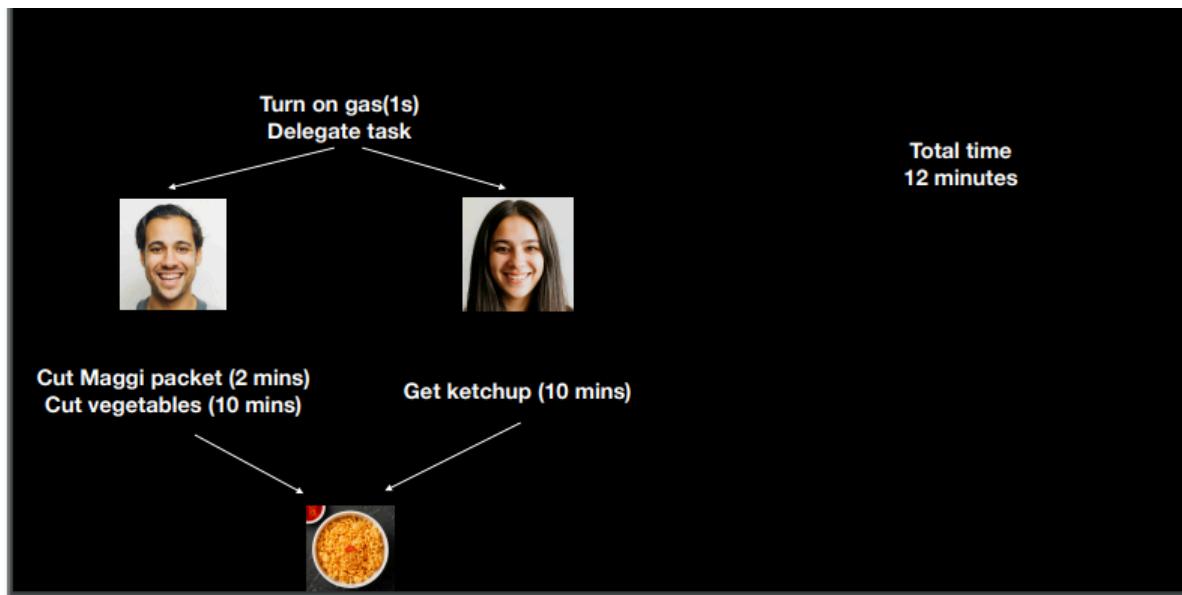
20 minutes to get ketchup from the shop.

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)

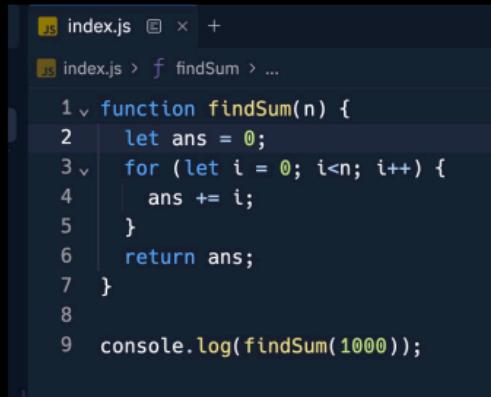
How does JS do the same? **Can Js delegate? Can Js context switch?**

Yes , by using **asynchronous functions**

Until now we have seen synchronous functions only.

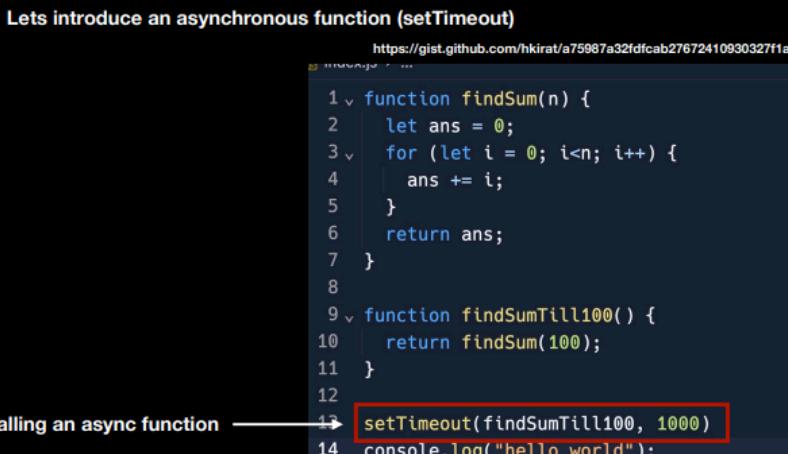
JS thread is busy and does anything else while inside the loop

Until now, we've only seen synchronous functions



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

## Asynchronous function



Lets introduce an asynchronous function (setTimeout)

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

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

function findSumTill100() {
  return findSum(100);
}

Calling an async function → setTimeout(findSumTill100, 1000)
console.log("hello world");
```

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\Desktop\Dev\Week1\Async_Await_Promises
PS C:\Users\NTC\Desktop\Dev\Week1\Async_Await_Promises> 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_Promises>
```

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:

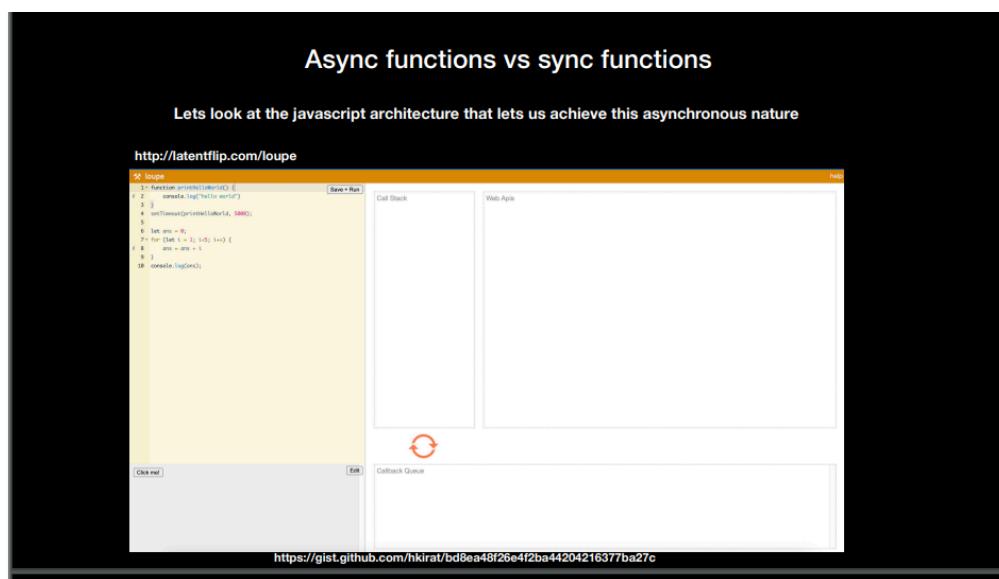
hello world

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

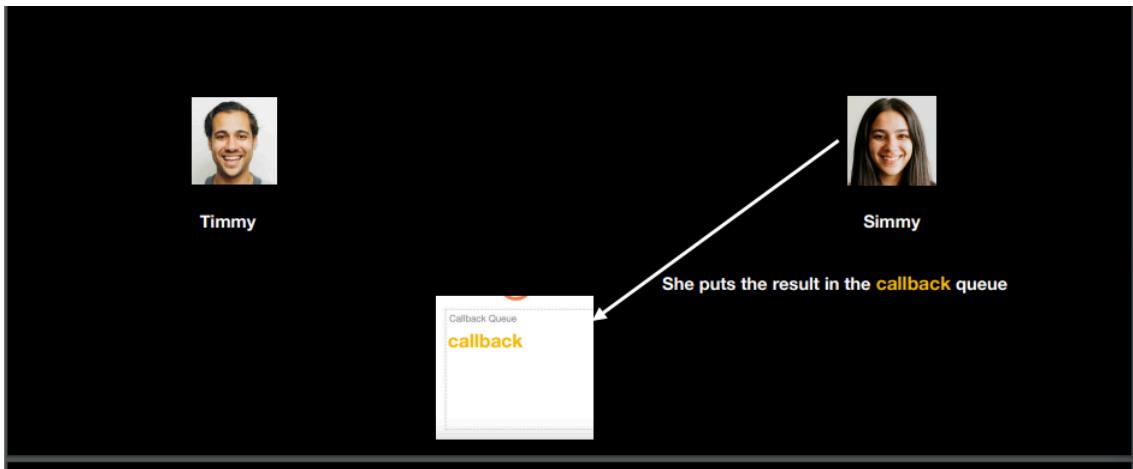
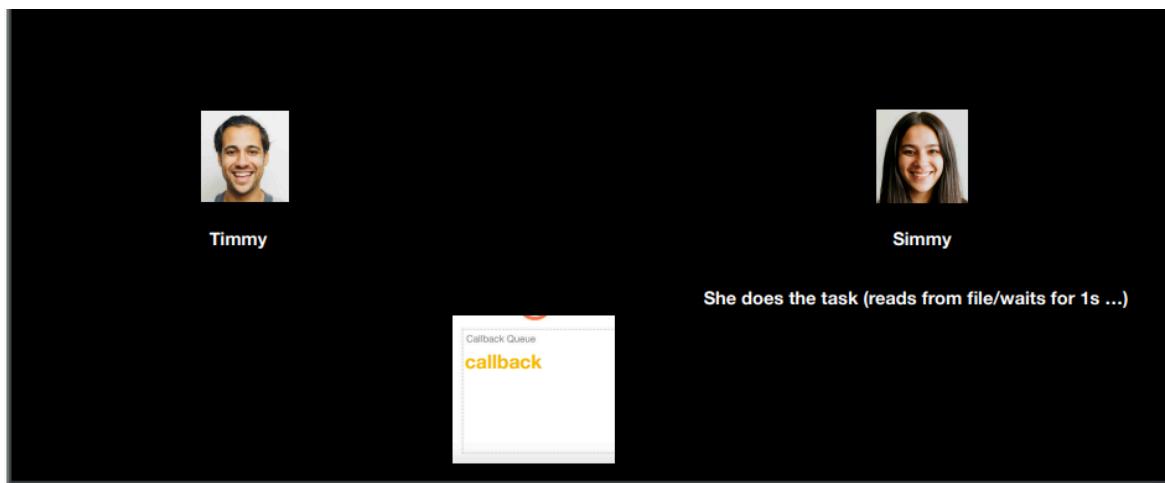
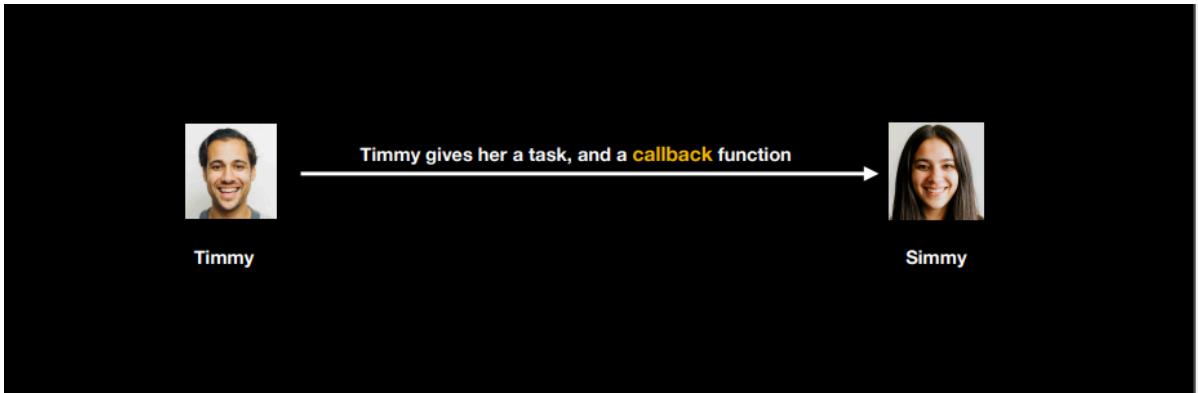
from inside the asynchronous function 2

from inside the asynchronous function

Callback doesn't make much sense in synchronous function

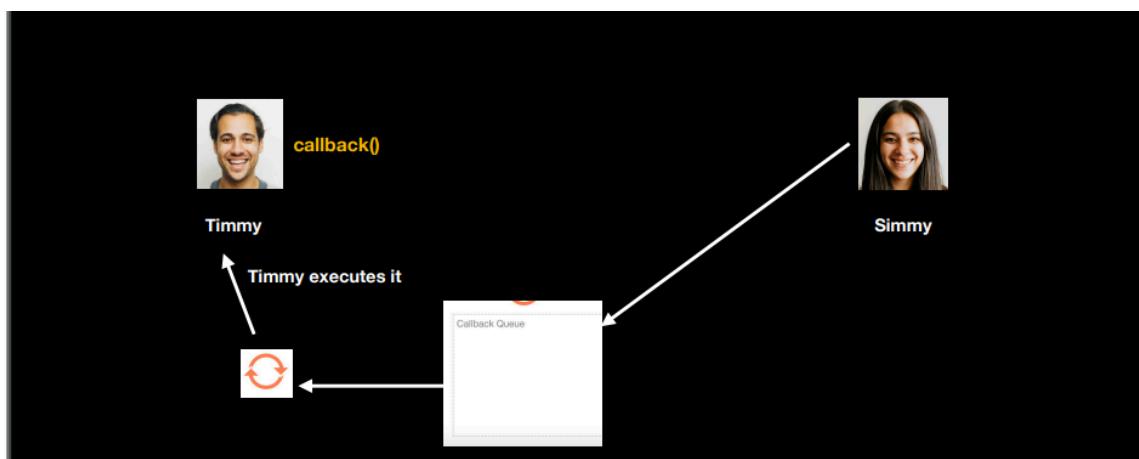
## Promises

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?

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

Ugly way

```
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)
```

https://gist.github.com/hkirat/621d9168e39d99bcc14d767c912e9777

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

Just creating a wrapper on top of others asynchronous function.

The cleaner way is to use 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)

```
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);
```

<https://gist.github.com/kirat/75be07d3ea18f93cba907c85233b4217>

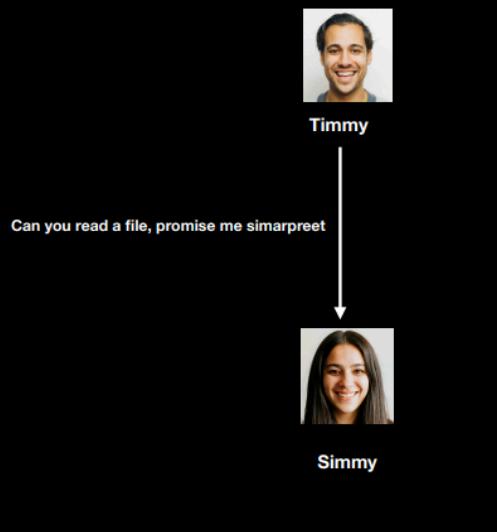
Its just syntactic sugar  
Still uses callbacks under the hood

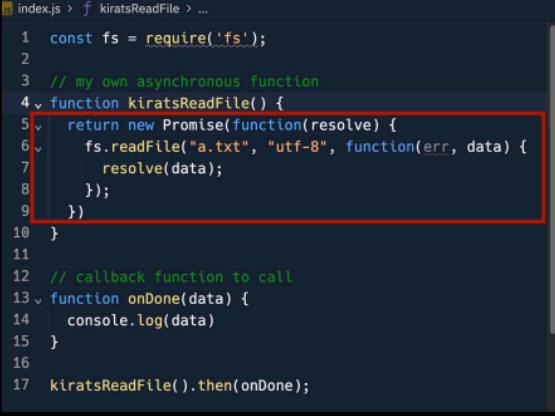
This function at very top doesn't have a callback function . How does Simmy know when to call when she does get the ketchup. Purpose of promises is to get rid of callbacks , since callbacks are ugly way to write asynchronous function.  
(Callback hell)

Basically it means i will read file and pass data to resolve and call it .

Lets see how Simmy and Timmy communicate

```
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);
```





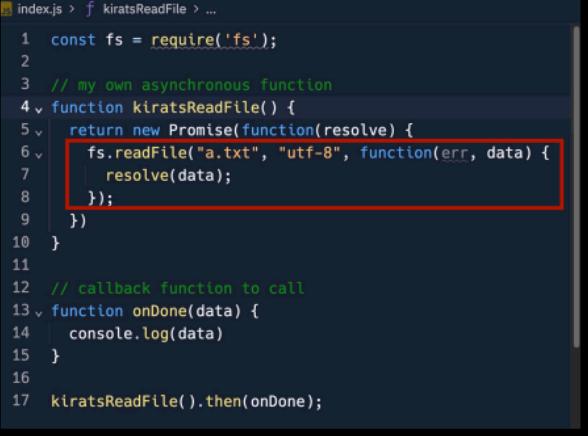
Timmy

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

(Notice, no callback anywhere)

Simmy

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



Timmy

Does her thing, reads the file

Simmy

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

```
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);
```

Calls `resolve` with the final data



Timmy



Simmy

```
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

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?**

What even is a promise?  
It is just a class that makes callbacks and async functions slightly more readable

The screenshot shows a terminal window with two tabs: 'index.js' and 'a.txt'. The 'index.js' tab contains the code: 'let p = new Promise(); console.log(p)'. The 'a.txt' tab contains the text 'hi from a.txt'. Below the tabs is a command-line interface with two 'Run' buttons. The first 'Run' button has a success status (307ms on 20:58:24, 12/07) and the output 'hi from a.txt'. The second 'Run' button has a warning status (301ms on 21:12:31, 12/07) and the output '/home/runner/HuskyDigitalVideogames/index.js:1:9'. A detailed stack trace follows:

```
let p = new Promise();
^
TypeError: Promise resolver undefined
is not a function
    at new Promise (<anonymous>
        at Object.<anonymous> (/home/runne
r/HuskyDigitalVideogames/index.js:1:9)
        at Module._compile (node:internal/
modules/cjs/loader:1257:14)
        at Module._extensions..js (node:in
ternal/modules/cjs/loader:1311:10)
        at Module.load (node:internal/modu
les/cjs/loader:1115:32)
        at Module._load (node:internal/mod
ules/cjs/loader:962:12)
        at Function.executeUserEntryPoint
[as runMain] (node:internal/modules/r
un_main:83:12)
        at node:internal/main/run_main_mod
ule:23:47
Node.js v20.3.1
```

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' }

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

A screenshot of a terminal window. On the left, there is a code editor tab for 'index.js'. The code inside is:1 v let p = new Promise(function(resolve) {  
2  
3 });  
4 console.log(p)The code editor shows line numbers 1 through 4. On the right, there is a 'Console' tab. It shows the output of the 'console.log(p)' command, which is 'Promise { <pending> }'. There is also a timestamp '219ms on 21:13:23,' and a 'Run' button.

A simple promise that immediately resolve

A screenshot of a terminal window. On the left, there is a code editor tab for 'index.js'. The code inside is:1 v let p = new Promise(function(resolve) {  
2 resolve("hi there");  
3 });  
4  
5 v p.then(function() {  
6 console.log(p);  
7 })The code editor shows line numbers 1 through 7. On the right, there is a 'Console' tab. It shows the output of the 'console.log(p)' command, which is 'Promise { "hi there" }'. There is also a timestamp '357ms on 21:14:24, 12/07,' and a 'Run' button.

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

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

Console output: Promise { 'hi there' }

.then gets called whenever the async function resolves

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

Console output: Promise { 'hi there' }

Dummy async function that almost immediately resolves

```
function KiratsAsyncFunction() {
  let p = new Promise(function(resolve) {
    resolve("hi there");
  });
  return p;
}

const value = kiratsAsyncFunction();
value.then(function(data) {
  console.log(data);
})
```

Actually logging the data with what the function above Resolved with

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

Promises

Try to marinate both the sides

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

```
function KiratsAsyncFunction() {
  let p = new Promise(function(resolve) {
    resolve("hi there");
  });
  return p;
}

const value = kiratsAsyncFunction();
value.then(function(data) {
  console.log(data);
})
```

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

A screenshot of a code editor interface. On the left, the file `index.js` 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
```

On the right, the `Console` tab shows the output: `hi there`. Below the editor is a URL: <https://gist.github.com/hkirat/46580244f43ca2847cf57664a3803b1>

It can be simple done by

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

Let's see another example

The image shows two code editor panes side-by-side.

**Intimidating async function:**

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

**Simple function:**

```
1 function kiratsAsyncFunction(callback) {
2     setTimeout(callback, 2000);
3 }
4
5 kiratsAsyncFunction(function() {
6     console.log("hello!");
7 });
8
9
```

## 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

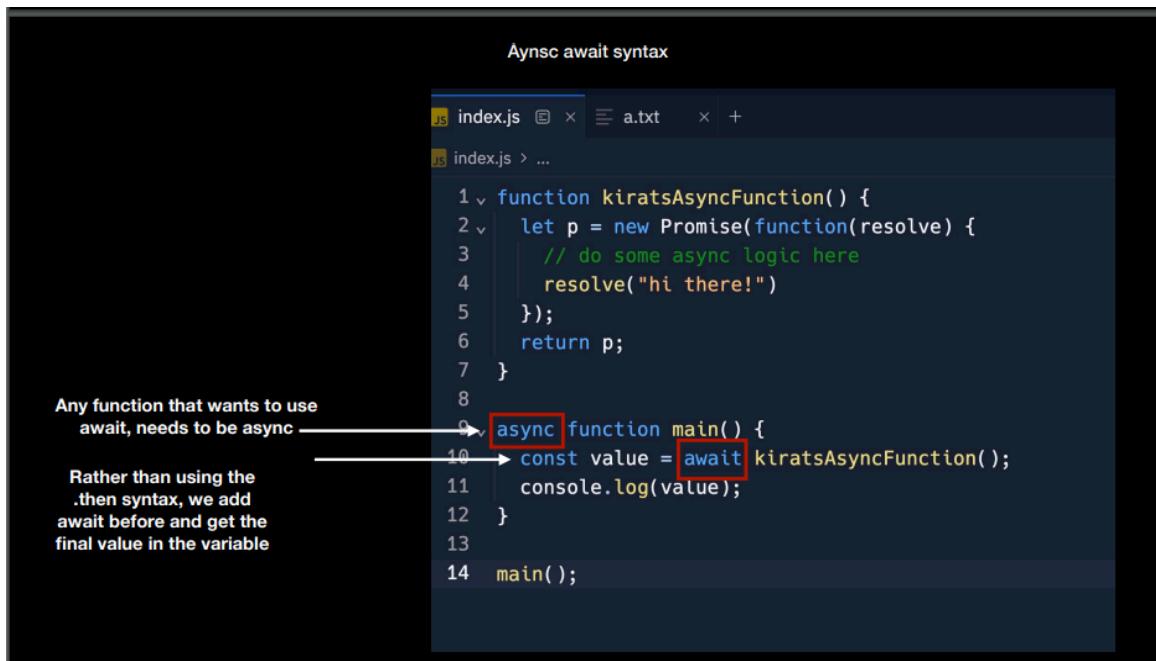
Promise {"hi there"}

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.



```
index.js
1 v function kiratsAsyncFunction() {
2 v   let p = new Promise(function(resolve) {
3 v     // do some async logic here
4 v     resolve("hi there!")
5 v   });
6 v   return p;
7 v }

8
9 v async function main() {
10 v   const value = await kiratsAsyncFunction();
11 v   console.log(value);
12 v }
13
14 main();
```

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

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

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

#### Promise (then) syntax

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

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

#### Async/await syntax

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

<https://gist.github.com/hkirat/9e00de2335d1ead90436aef68c9b2f9>