Day 38

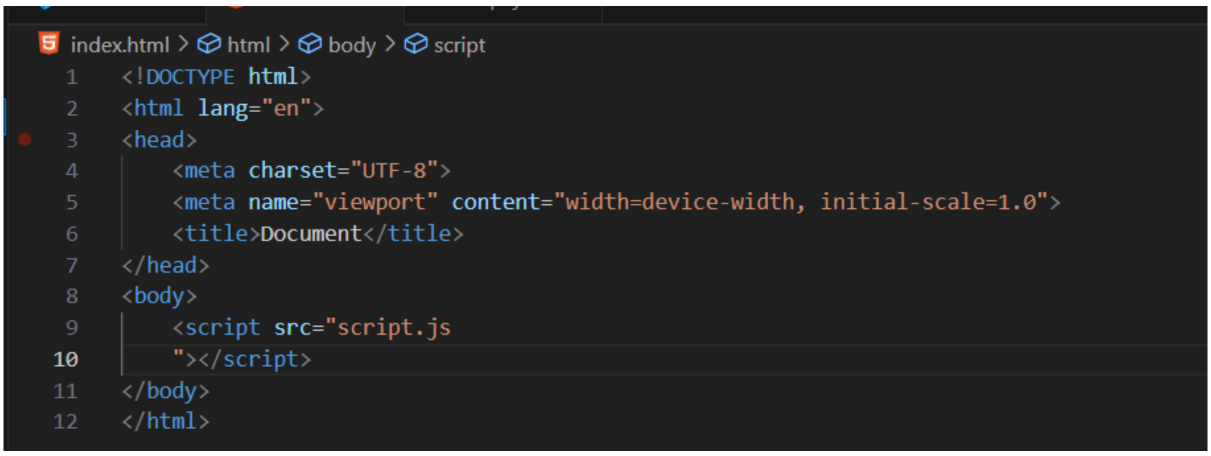
“Web Development + Security”

Async/Await & Fetch API in JavaScript:

What we actually know about promise?

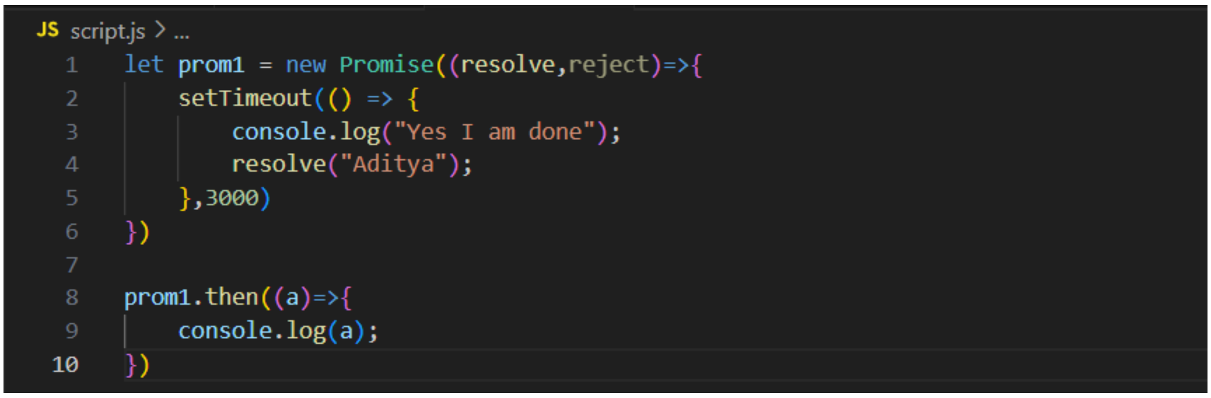
Code:

Index.html:



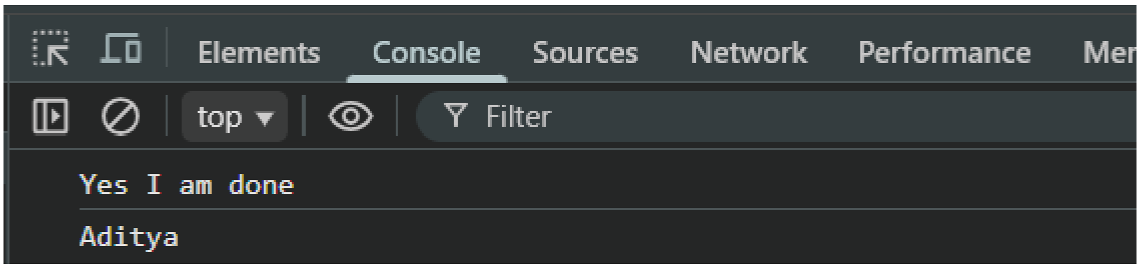
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="script.js
    "></script>
</body>
</html>
```

Script.js: a very basic example of promise.



```javascript
let prom1 = new Promise((resolve,reject)=>{
    setTimeout(() => {
        console.log("Yes I am done");
        resolve("Aditya");
    },3000)
})

prom1.then((a)=>{
    console.log(a);
})
```

Console:

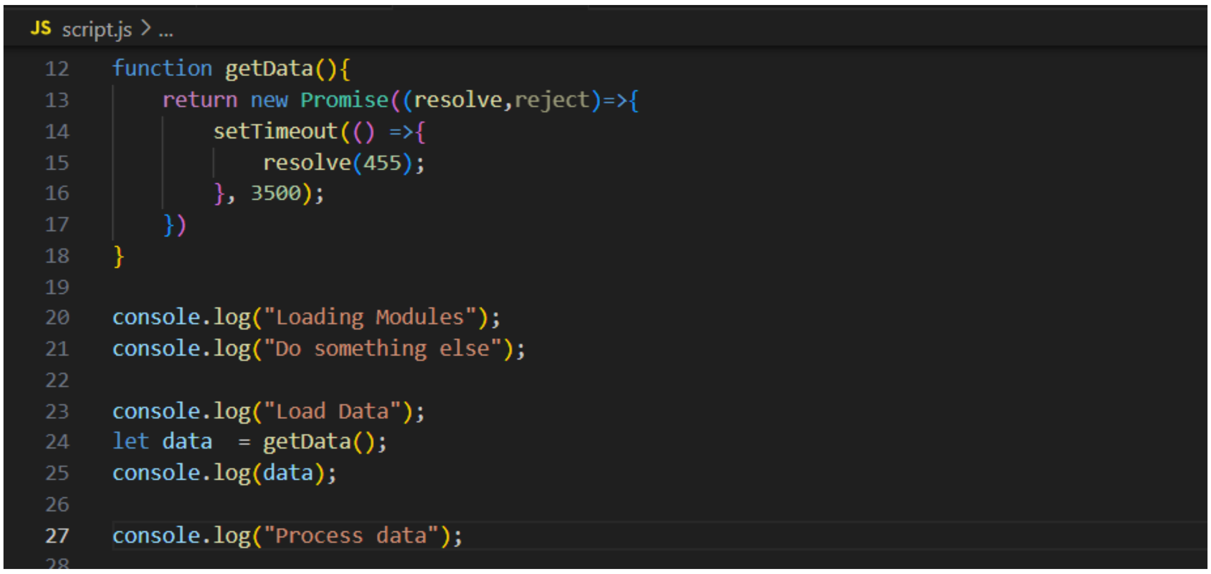


```
Yes I am done
Aditya
```

Seeing the return of a promise for a case in the console:
Code:

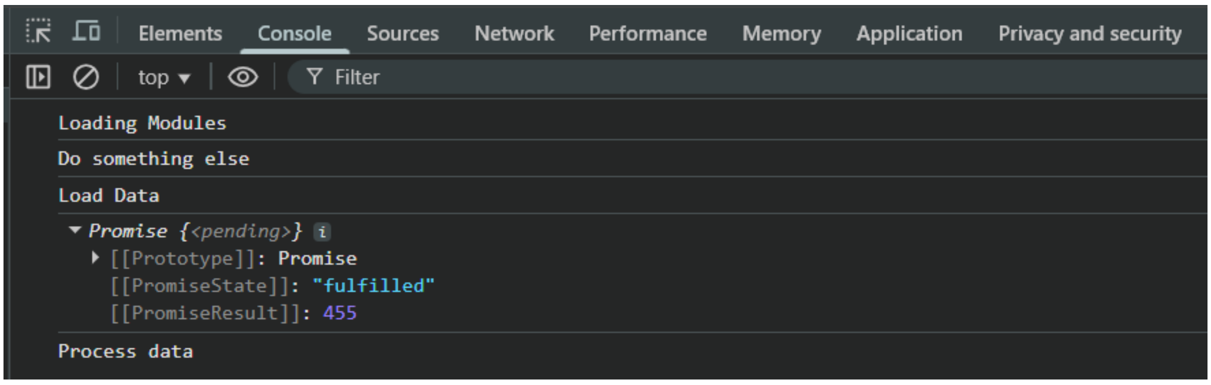
Index.html: as above

Script.js:



```
JS script.js > ...
function getData(){
    return new Promise((resolve,reject)=>{
        setTimeout(() =>{
            resolve(455);
        }, 3500);
    })
}

console.log("Loading Modules");
console.log("Do something else");

console.log("Load Data");
let data  = getData();
console.log(data);

console.log("Process data");
```

Console:



```
Loading Modules
Do something else
Load Data
▼ Promise {<pending>}
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: 455
Process data
```

Explanation:

getData() immediately returns a Promise object, but the actual value (455) isn't available yet—it will be provided after 3.5 seconds when resolve(455) runs. That's why console.log(data) shows the Promise itself, not the value. Meanwhile, JavaScript continues executing the rest of the code (Loading Modules, Do something else, Process data) without waiting, demonstrating its asynchronous nature.

Now, suppose I don't want that anything written below the

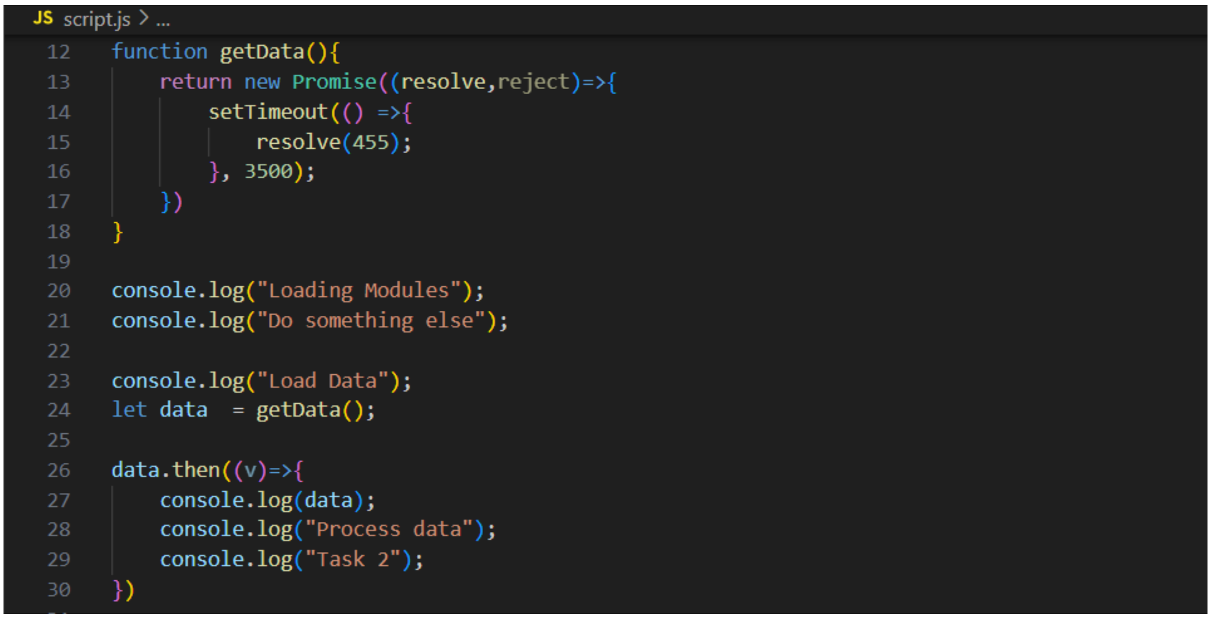
let data = getData();

console.log(data);

should run unless the data don't comes, then we need to have control on the asynchronous nature of JS.

For this, we are first going with the call-based approach:

Script.js: we will use .then() and arrow function

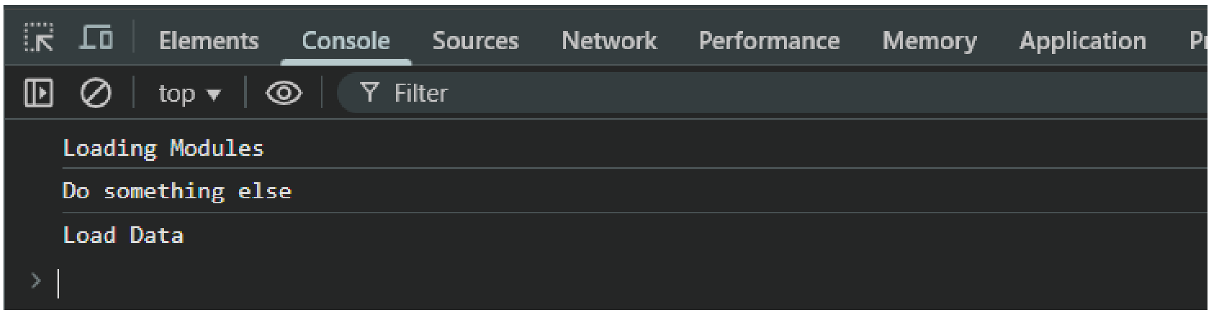


script.js > ...

```javascript
function getData(){
    return new Promise((resolve,reject)=>{
        setTimeout(() =>{
            resolve(455);
        }, 3500);
    })
}

console.log("Loading Modules");
console.log("Do something else");

console.log("Load Data");
let data  = getData();

data.then((v)=>{
    console.log(data);
    console.log("Process data");
    console.log("Task 2");
})
```

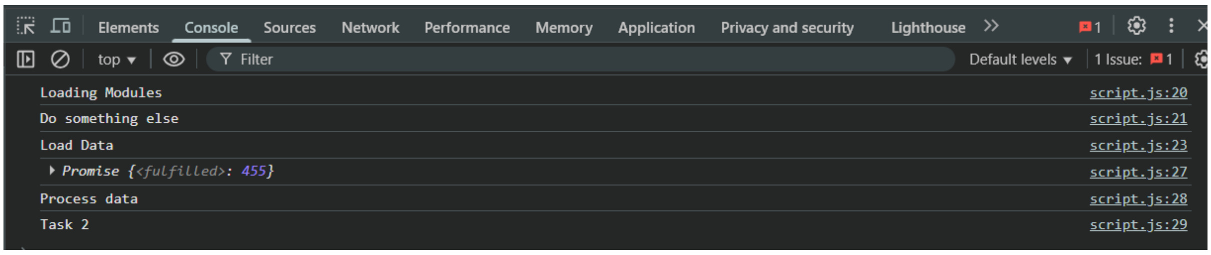
Console:

At start:



```
Loading Modules
Do something else
Load Data
```

After 3.5 seconds: we can see that promise is fulfilled.



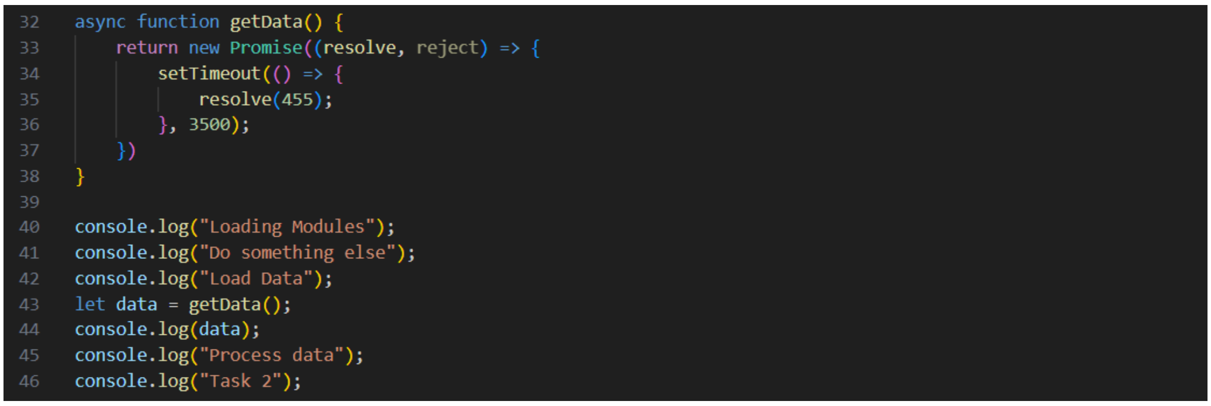
```
Loading Modules                                      script.js:20
Do something else                                    script.js:21
Load Data                                            script.js:23
▸ Promise {<fulfilled>: 455}                         script.js:27
Process data                                         script.js:28
Task 2                                               script.js:29
```

Also, to take control of the asynchronous nature of the JS we can do the 2nd approach as well. For which we need to us async function.

What is an async Function?

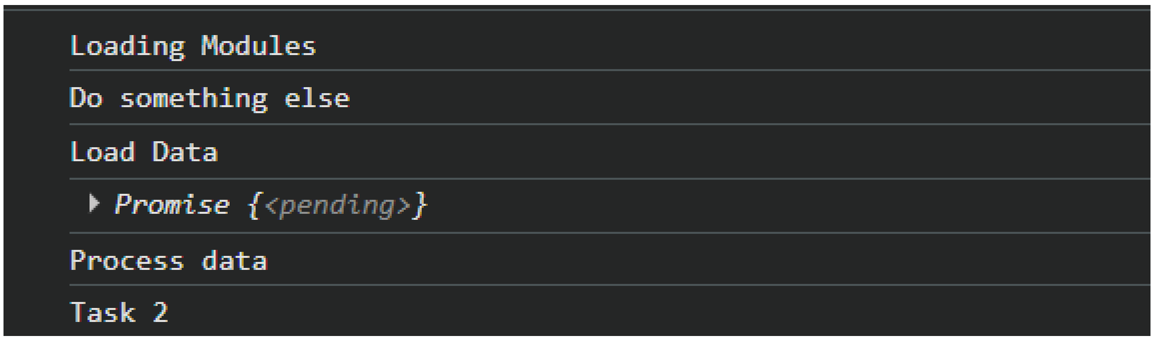
An async function is a function that always returns a Promise, even if you return a simple value. It allows you to write asynchronous code in a synchronous style using await.

Script.js:



```
async function getData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve(455);
        }, 3500);
    })
}

console.log("Loading Modules");
console.log("Do something else");
console.log("Load Data");
let data = getData();
console.log(data);
console.log("Process data");
console.log("Task 2");
```

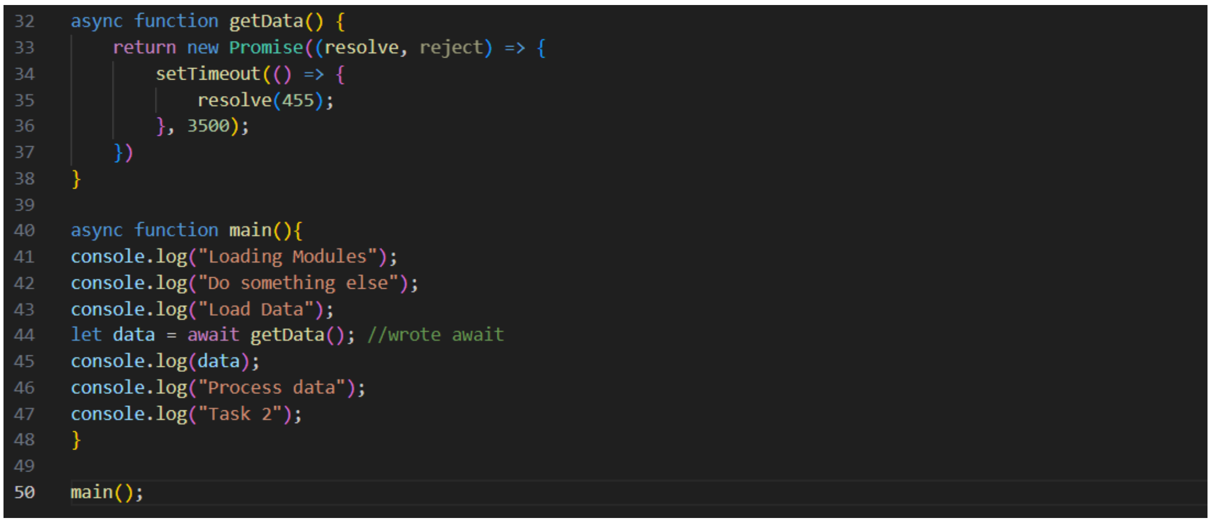
Console:



```
Loading Modules
Do something else
Load Data
▸ Promise {<pending>}
Process data
Task 2
```

Now, suppose I wish to wait for this async function, for that I will use awat, and for doing so I have to create another function wrap, and then run that function.

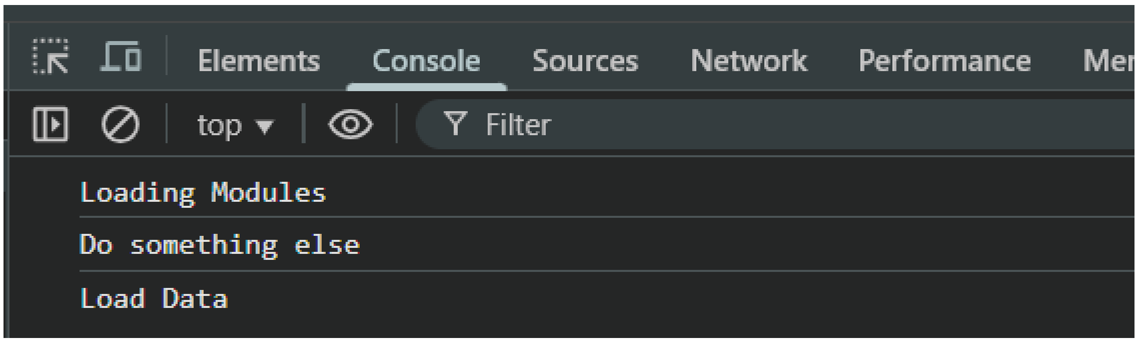
Script.js:



```
async function getData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve(455);
        }, 3500);
    })
}

async function main(){
console.log("Loading Modules");
console.log("Do something else");
console.log("Load Data");
let data = await getData(); //wrote await
console.log(data);
console.log("Process data");
console.log("Task 2");
}

main();
```

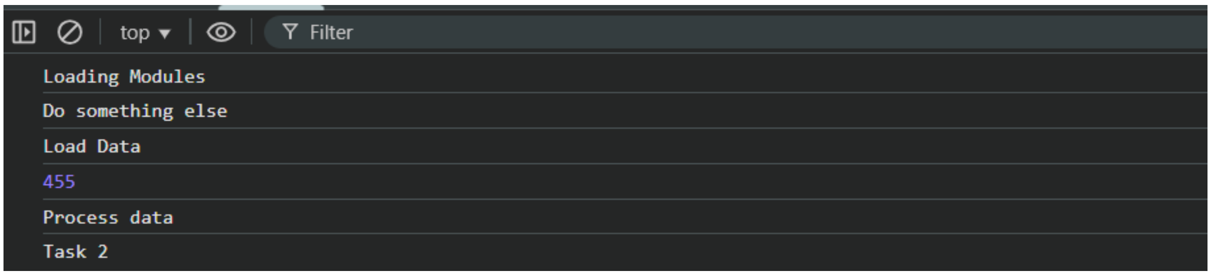
Console:

At start:



```
Loading Modules
Do something else
Load Data
```

After 3.5s:



```
Loading Modules
Do something else
Load Data
455
Process data
Task 2
```

In this code, getData() is an async function that returns a Promise resolving with 455 after 3.5 seconds. Inside the main() async function, the first three console.log statements (Loading Modules, Do something else, Load Data) execute immediately. When let data = await getData() runs, execution pauses inside main until the Promise resolves. After 3.5 seconds, data becomes 455 and the remaining logs (455, Process data, Task 2) execute in order. Using await makes the asynchronous Promise behave like synchronous code within the async function, so the code after it waits for the result.

The await Keyword

- Can only be used inside an async function.
- Pauses execution until the Promise resolves.
- Makes asynchronous code look synchronous and easier to read

Key Points About async/await

1. async → declares a function as asynchronous, always returns a Promise.
2. await → waits for a Promise to resolve, pauses execution inside the async function.
3. Makes asynchronous code more readable than callbacks or chained .then().
4. Works perfectly with Promises, including chaining multiple async tasks.
5. Error handling is clean with try...catch.

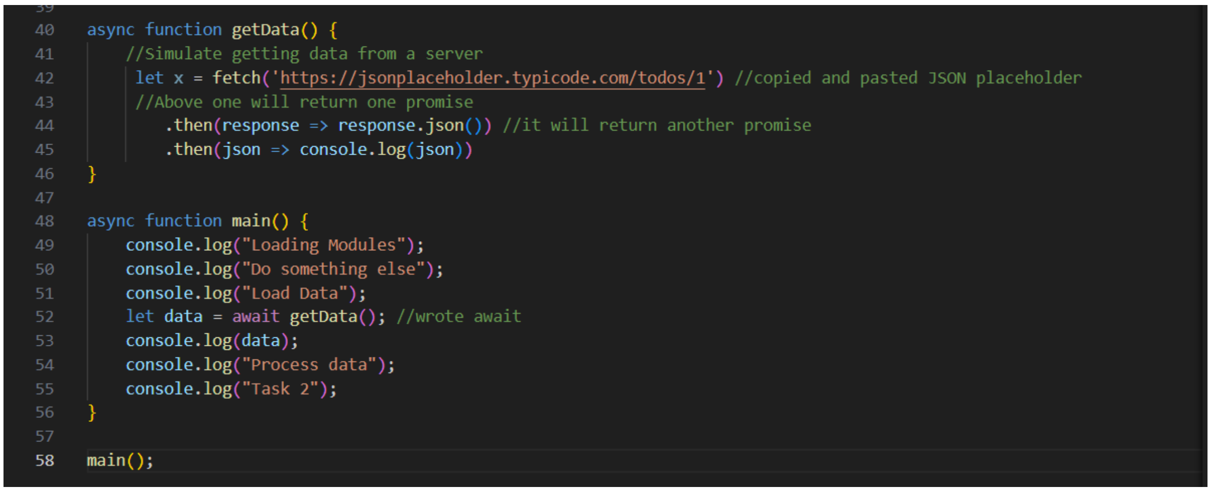
What is Fetch API?

The Fetch API is a modern JavaScript interface used to make network requests (like HTTP GET, POST) to servers and fetch resources such as JSON, text, or files.

- It replaces older XMLHttpRequest (XHR) methods.
- It is Promise-based, so it works perfectly with async/await.

In simple words: Fetch API “fetches” data from a server and gives it to you asynchronously.

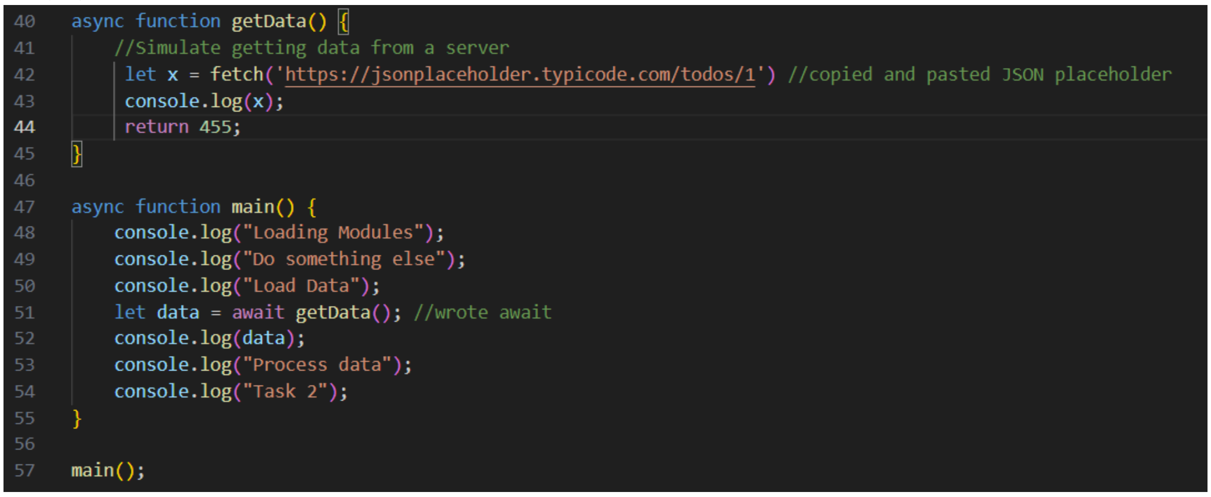
Now, to understand it, we will have to understand that the JSON placeholder given below returns promise, so we will understand each line of the getData one by one.



```
40  async function getData() {
41      //Simulate getting data from a server
42       let x = fetch('https://jsonplaceholder.typicode.com/todos/1') //copied and pasted JSON placeholder
43       //Above one will return one promise
44          .then(response => response.json()) //it will return another promise
45          .then(json => console.log(json))
46  }
47  
48  async function main() {
49      console.log("Loading Modules");
50      console.log("Do something else");
51      console.log("Load Data");
52      let data = await getData(); //wrote await
53      console.log(data);
54      console.log("Process data");
55      console.log("Task 2");
56  }
57  
58  main();
```

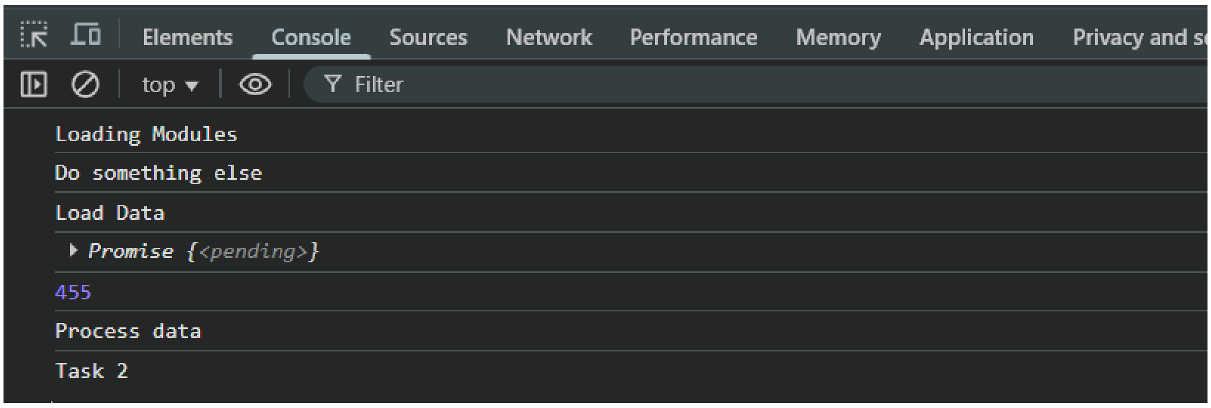
Now, let's prove that line 42 returns a promise:

Script.js:



```
40  async function getData() {
41      //Simulate getting data from a server
42       let x = fetch('https://jsonplaceholder.typicode.com/todos/1') //copied and pasted JSON placeholder
43       console.log(x);
44       return 455;
45  }
46  
47  async function main() {
48      console.log("Loading Modules");
49      console.log("Do something else");
50      console.log("Load Data");
51      let data = await getData(); //wrote await
52      console.log(data);
53      console.log("Process data");
54      console.log("Task 2");
55  }
56  
57  main();
```

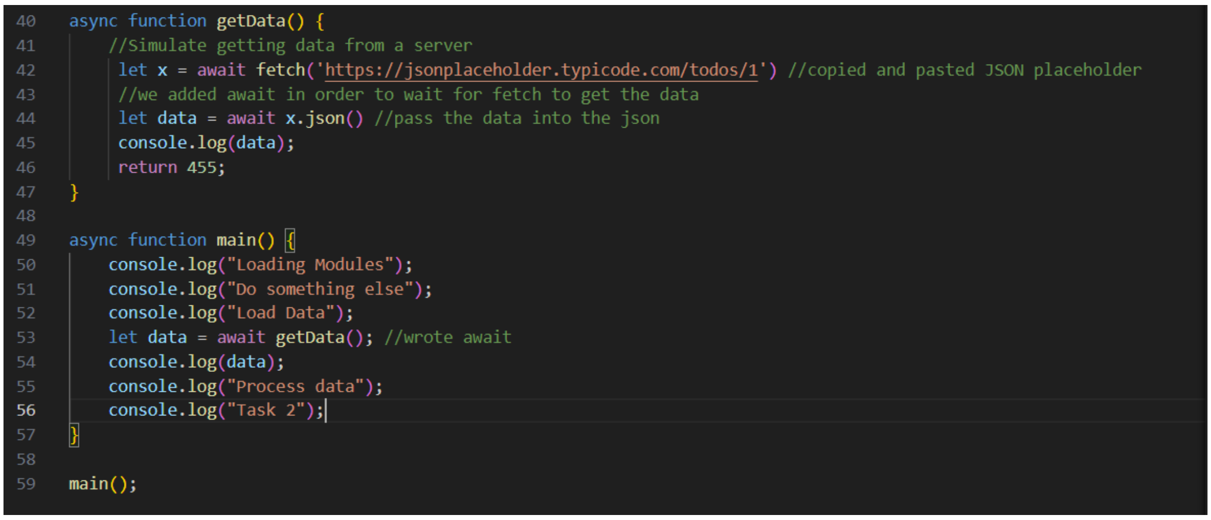
Console:



```
Loading Modules
Do something else
Load Data
▸ Promise {<pending>}
455
Process data
Task 2
```

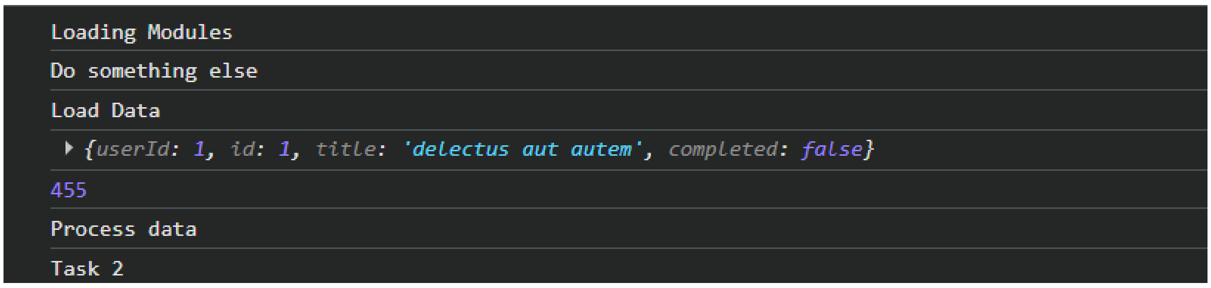
Now, fetching it using the await and fetch:

Script.js: basically we have to write two await as we have two promises form the fetch API. First promise to bring the data and 2nd promise to parse the data.



```
async function getData() {
    //Simulate getting data from a server
     let x = await fetch('https://jsonplaceholder.typicode.com/todos/1') //copied and pasted JSON placeholder
     //we added await in order to wait for fetch to get the data
     let data = await x.json() //pass the data into the json
     console.log(data);
     return 455;
}

async function main() {
    console.log("Loading Modules");
    console.log("Do something else");
    console.log("Load Data");
    let data = await getData(); //wrote await
    console.log(data);
    console.log("Process data");
    console.log("Task 2");
}

main();
```

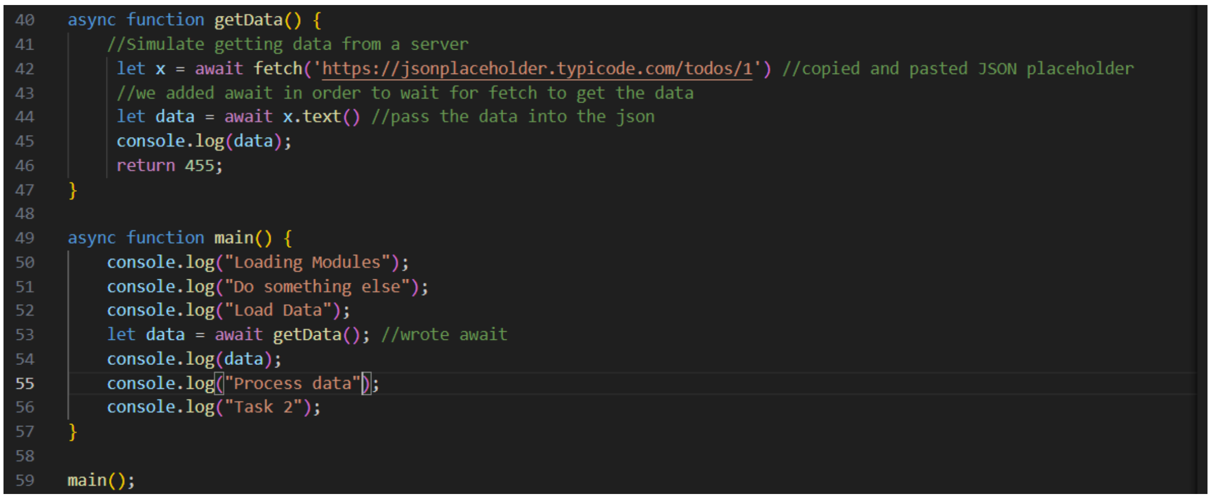
Console:



```
Loading Modules
Do something else
Load Data
▸ {userId: 1, id: 1, title: 'delectus aut autem', completed: false}
455
Process data
Task 2
```

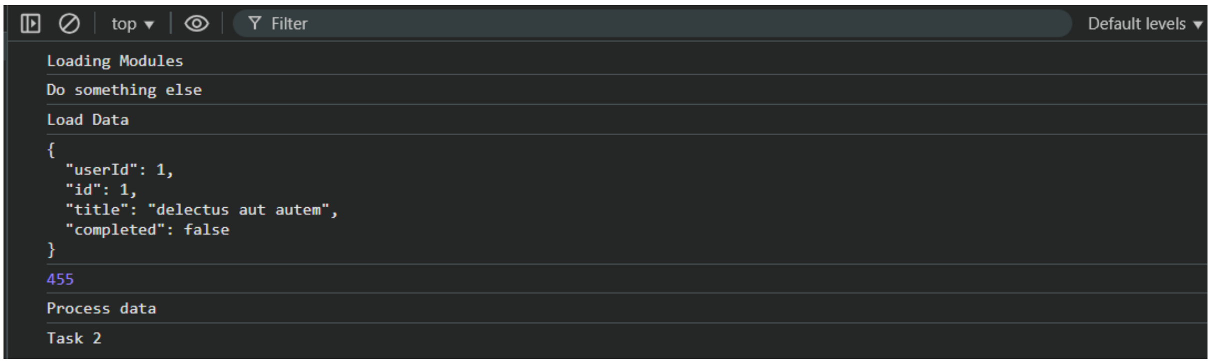
In case we want the paring to happen in the text: we will use .text() instead of .json() as shown below:

Script.js:



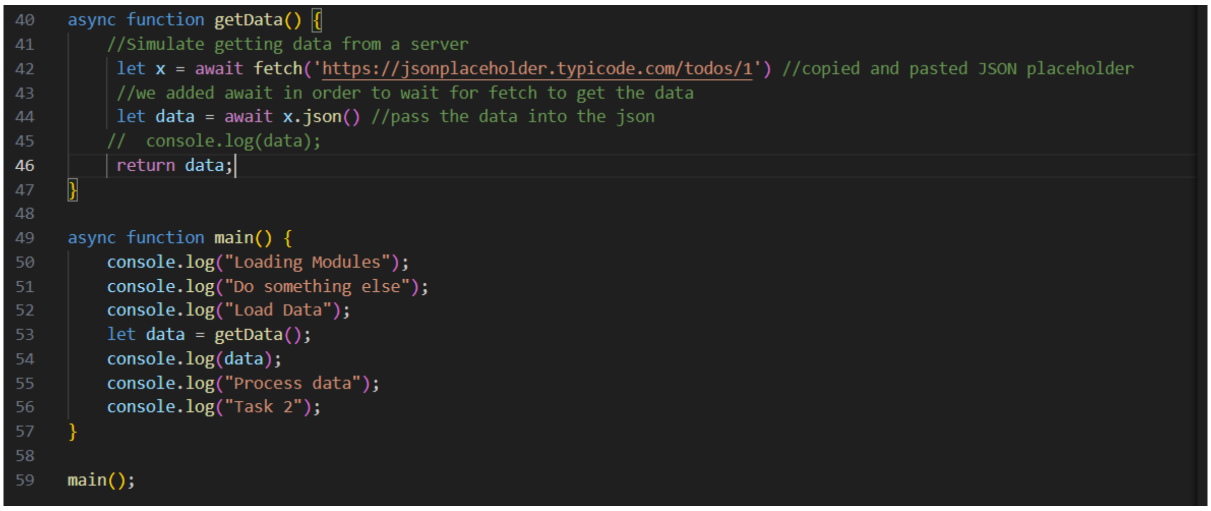
```
async function getData() {
    //Simulate getting data from a server
     let x = await fetch('https://jsonplaceholder.typicode.com/todos/1') //copied and pasted JSON placeholder
     //we added await in order to wait for fetch to get the data
     let data = await x.text() //pass the data into the json
     console.log(data);
     return 455;
}

async function main() {
    console.log("Loading Modules");
    console.log("Do something else");
    console.log("Load Data");
    let data = await getData(); //wrote await
    console.log(data);
    console.log("Process data");
    console.log("Task 2");
}

main();
```

Console:



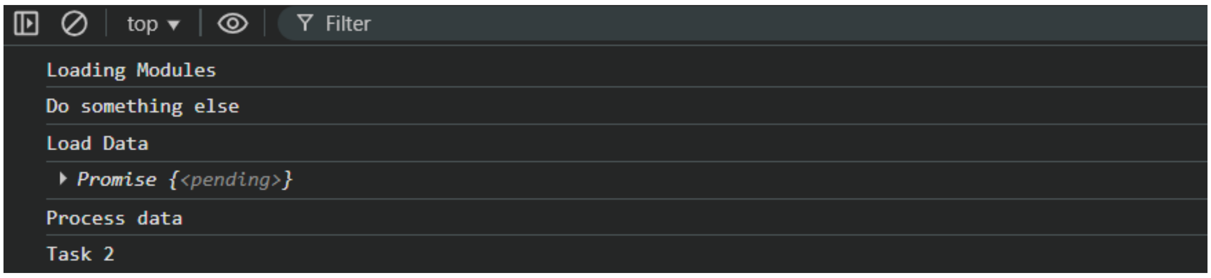
```
Loading Modules
Do something else
Load Data
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
455
Process data
Task 2
```

In case we don't want to wait for promise to settle, we will remove the await keyword from the main():
Script.js:



```
async function getData() {
    //Simulate getting data from a server
     let x = await fetch('https://jsonplaceholder.typicode.com/todos/1') //copied and pasted JSON placeholder
     //we added await in order to wait for fetch to get the data
     let data = await x.json() //pass the data into the json
    //  console.log(data);
     return data;
}

async function main() {
    console.log("Loading Modules");
    console.log("Do something else");
    console.log("Load Data");
    let data = getData();
    console.log(data);
    console.log("Process data");
    console.log("Task 2");
}

main();
```

Console:



```
Loading Modules
Do something else
Load Data
▸ Promise {<pending>}
Process data
Task 2
```

Note:

1. Settle means resolve or reject
2. Resolve means promise has settled successfully
3. Reject means promise has settled successfully

--The End--