asynchronousJS.txt

JS_AsynchronousProgramming > asynchronousJS.txt

```
It allows code to intiate operations and continue running other tasks without waiting for
them to finish.

It includes
-> Fetching Data from servers
-> Reading/Writing lines
-> Making API Calls

To handle asynchronous operations, JS provides mechanisms like
-> Callbacks :  Function executed after an asynchronous operation completes.

-> Promises  :  Represent future values and allow chaining multiple asynchronous operations.

-> Async/Await : Provide a more consise and synchronous-like way to write asynchronous code.

It improves
-> Program Responsivness
-> Prevent Blocking

It is mainly used with the Promises.

When a long running task needs to be performed without blocking the main thread of
executation.
-> Network requests.
-> Accesing a Database
That time Async/Await is used.
```

- JS async & await makes function return promise
  - makes function wait Promise
- Keywords
  - for async → async
  - for await → await

JUNE						2020
Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

```
async function myf() {
    return "Hi";
}
```

Same as

```
function myf() {
    return
    Promise.resolve("Hi");
}
```

Tuesday

```
function display (some) {
    document.write (some);
}
async function asyn() {
    return "Hello";
}
asyn().then(
        function(val) {display (val)};
        function(err) {display (err)}
)
</script>
```

await :- Only used inside async function

makes function a) pause exe → wait for

continues ← before ← resolved promise

```
let value = await promise;
```

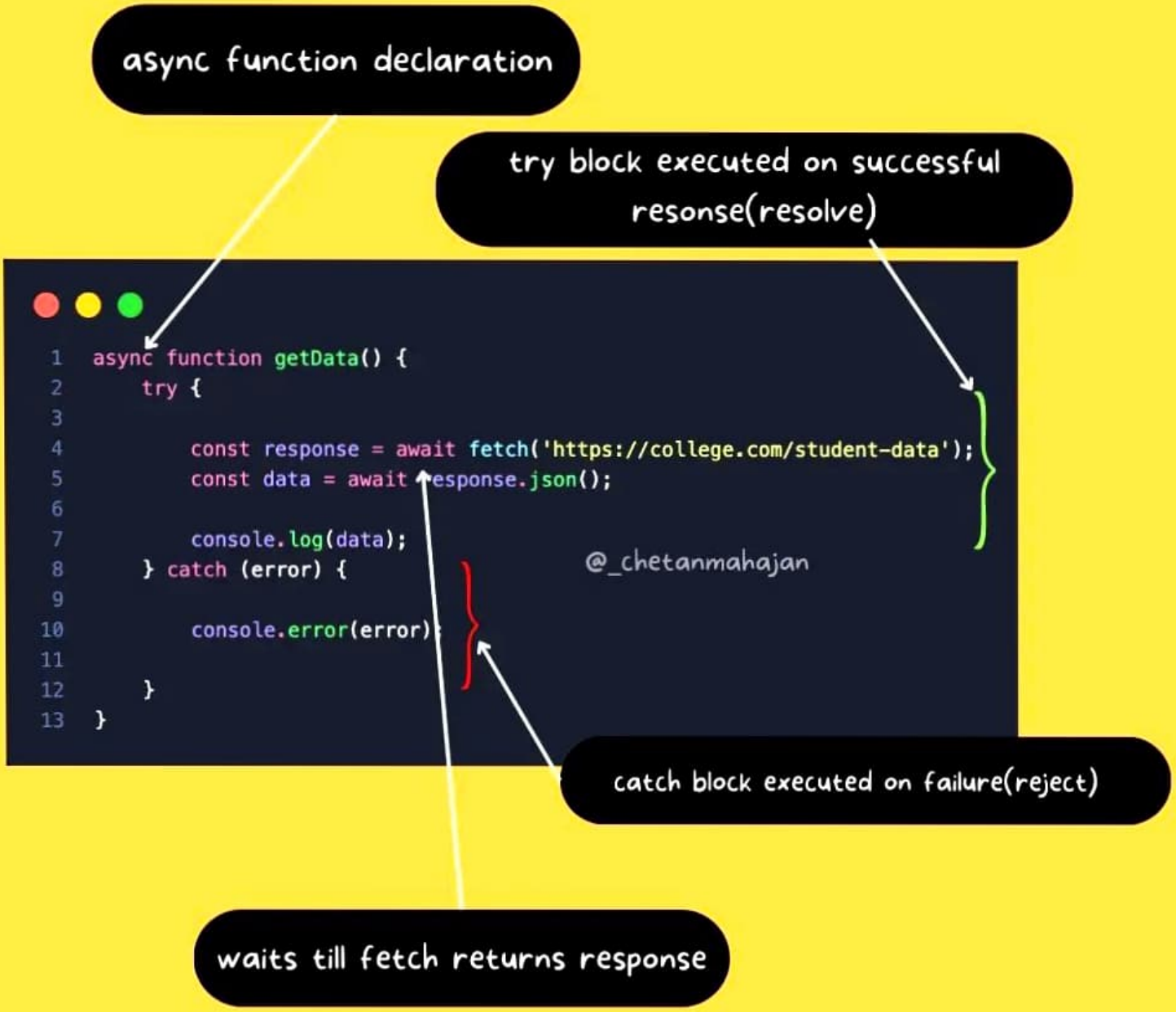
Two Arguments (resolve, reject) → predefined by JS

Not create them but can one of them when executor function ready

Eg.:

```
<body>
    <p id = "get">
    <script>
        async function display() {
        let promy = new Promise(function (resolve, reject) {
            resolve("I Love Maggi");
        });
    document.getElementById("get").innerHTML
        = await promy;
    }
    display();
</script>
</body>
```

For Example,




```
1  async function getData() {
2      try {
3
4          const response = await fetch('https://college.com/student-data');
5          const data = await response.json();
6
7          console.log(data);
8      } catch (error) {
9
10         console.error(error);
11
12     }
13 }
```

As you can see in the example, the async keyword is used before the function, it is used to make the function return a promise, and await keyword is used to pause execution till program resolve/reject the promise.

Introduction to Async/Await

In the realm of web development, managing **asynchronous operations** can quickly become a headache.
Thankfully, JavaScript offers us **async** and **await**, two revolutionary keywords that transform the way we write and comprehend asynchronous code.

Say **goodbye** to **complex promise chains** and **callback** hell! With async/await, your asynchronous code becomes as **simple** and straightforward as synchronous code.

Why Use Async/Await

Async/await provides an elegant alternative to promises and callbacks for handling asynchronous tasks.

This approach allows for writing **clear** and **linear** code without getting entangled in .then() or nested callbacks.

The main reasons to embrace async/await are its **syntactic simplicity**, improved code **readability**, and intuitive **error handling** through try/catch blocks.

Imagine simplifying the logic of your API requests, file read/write operations, or even animations, all with a streamlined syntax.

```
function fetchUserData() {
  return fetch('https://api.example.com/user')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error("An error occurred:", error));
}

fetchUserData();
```

The 'async' Keyword

Every adventure with async/await begins with the async keyword, transforming an ordinary function into an asynchronous function.

Once declared async, your function can contain one or more **await** expressions and **implicitly returns a promise**.

The magic of async lies in setting the stage to use await inside, allowing your code to **patiently wait** for promise resolution **without blocking** the execution of the rest of your script.

```
async function loadContent() {
  // This function is now a promise
}
```

The 'await' Keyword

At the heart of our topic, the await keyword shines in its full glory.

Used within an async function, **await pauses** the function's **execution** until a promise is resolved. Remarkably, await unlocks synchronous-like behavior in our asynchronous code without freezing the browser.

It allows us **to wait for data from an API,** the result of a database operation, or any asynchronous task as if we were writing a simple variable assignment.

Error Handling with Try/Catch

One of the great strengths of async/await is its ability to **use try/catch** blocks for error handling.

This familiar method to many developers offers a clear and structured way to deal with errors that may occur during asynchronous operations.

Instead of juggling **.catch()** at the end of each promise, you can **encapsulate** your await calls in a try block and catch potential errors in a corresponding catch block.

Concrete Usage Example

Let's look at a concrete example where async/await shines for its utility.

Suppose we need to fetch user data from an API. Without async/await, we'd be lost in a sea of promises.

With async/await, the task becomes a piece of cake. We simply wait for the fetch request to resolve, then for the response to convert to JSON, all in a linear and easy-to-understand control flow.

```
async function getUser() {
  try {
    const response = await fetch('https://api.example.com/user');
    const user = await response.json();
    console.log(user);
  } catch (error) {
    console.error('Error fetching data', error);
  }
}
```

Synchronous Js

You may have heard that " JavaScript is a single-threaded ,synchronous language " at some point in your programming career

---> Single-threaded , Js runs one line of code at a time

---> Sync Js , Js runs line after line regardless of how much time it takes

Code Example : 2


```javascript
console.log("hanzala");
syncJs();
console.log("ikrama");

function syncJs() {
  for (let i = 0; i < 1000; i++) {
    console.log("wosqa");
  }
  console.log("done with the task");
}

// hanzala
// wosqa (1000)
// done with the task
// ikrama
```

In this case, console.log (ikrama) will not run until the function (syncJs) is completed

As a result our application will become slow

Async & await

Async/await is a feature in JavaScript that allows you to work with asynchronous code in a more synchronous-like manner

The async/await syntax is a special syntax created to help you work with promise objects

Await keyword

The await keyword makes JavaScript ‘wait’ until the promise is resolved or rejected

The await keyword is placed before the call to a function or variable that returns a promise. It makes JavaScript wait for the promise object to settle before running the code in the next line

Await syntax


```
// then
fetch("https://jsonplaceholder.typicode.com/todos/1")
  .then((response) => response.json())
  .then((json) => console.log(json))
  .catch((error) => console.log(error));

// await
const response = await fetch("https://jsonplaceholder.typicode.com/todos/1");
const json = await response.json();
console.log(json);
```

The async keyword

To create an asynchronous function, you need to add the async keyword before your function name.

```
const intro= async()=>{
  const response = await fetch("jsonplaceholder");
  const data = response.json()
  console.log(data);
}

intro()
```

Here, we created an asynchronous function called intro() and put the code that uses the await keyword inside it. We can then run the asynchronous function by calling it, just like a regular function.

Without setTimeout


```javascript
console.log("hanzala");
syncJs();
console.log("ikrama");

function syncJs() {
  for (let i = 0; i < 1000; i++) {
    console.log("wosqa");
  }
  console.log("done with the task");
}

// hanzala
// wosqa (1000)
// done with the task
// ikrama
```

With setTimeout


```javascript
console.log("hanzala");
asyncJs(1000);
console.log("ikrama");

function asyncJs(time) {
  setTimeout(loopOver, time);
}

function loopOver() {
  for (let i = 0; i < 1000; i++) {
    console.log("wosqa");
  }
}

// hanzala
// ikrama
// 1000 wosqa
```