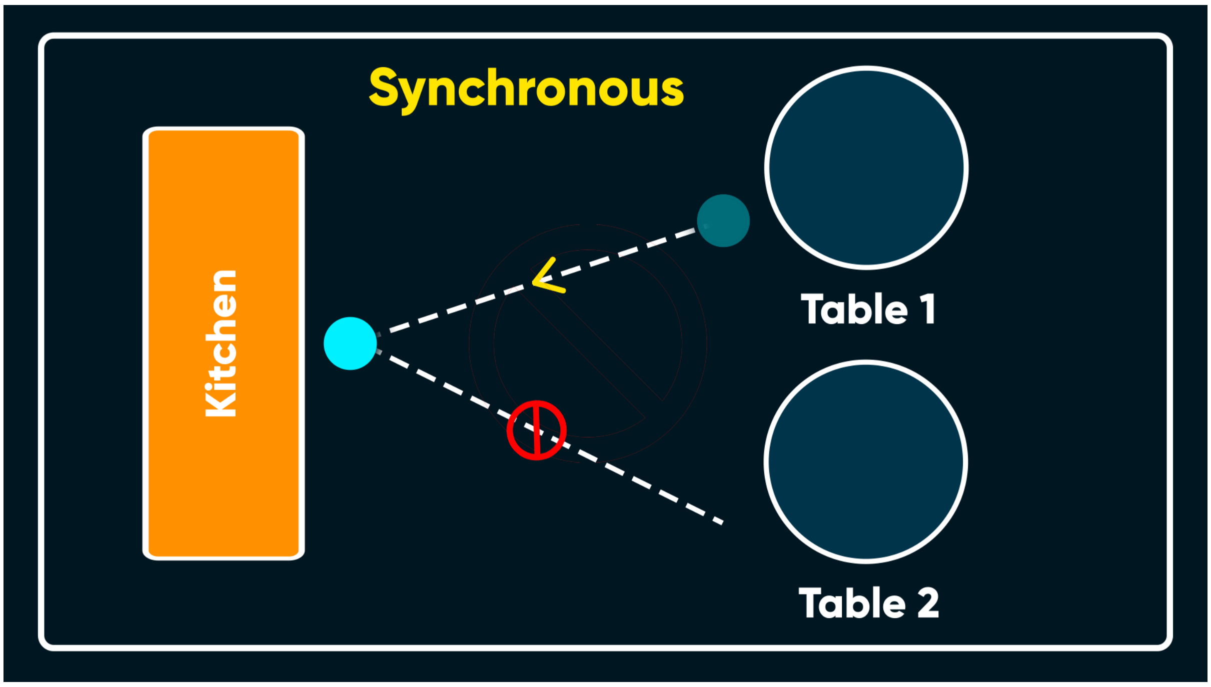
The Asynchronous JavaScript Guide

Before diving into Asynchronous JavaScript Methods, Let’s understand the difference between Synchronous and Asynchronous code. This is the cheatsheet for Asynchronous JavaScript class.

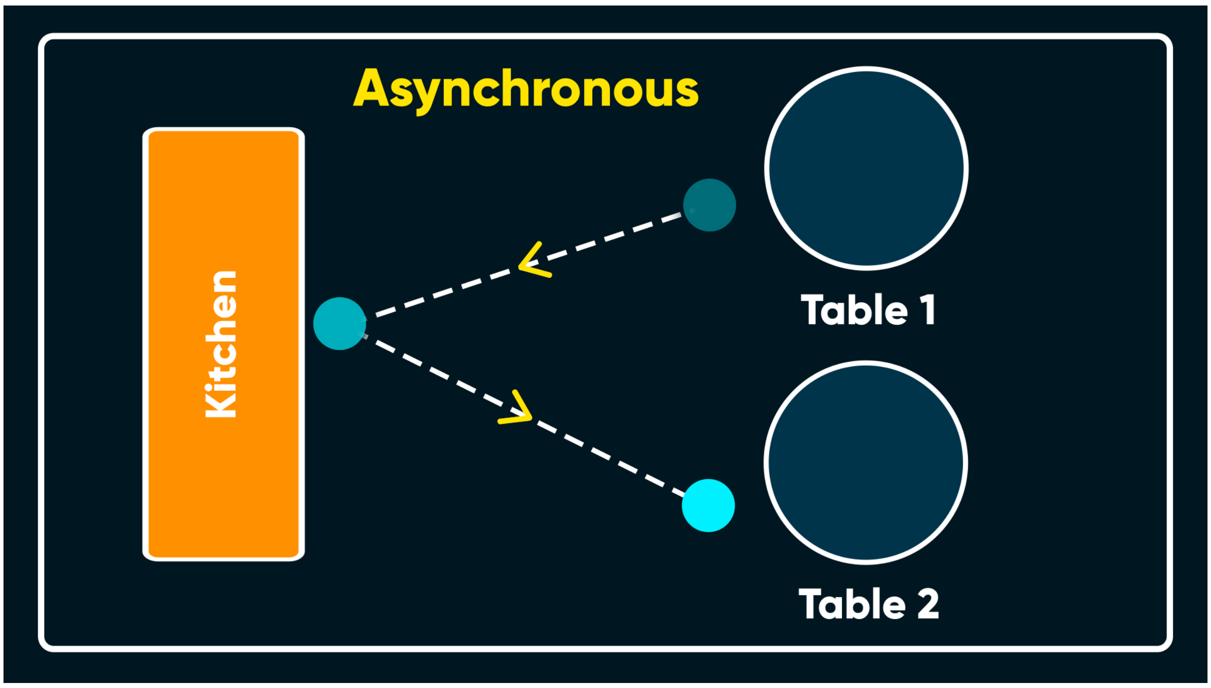
📌 Synchronous vs Asynchronous

Synchronous means our code works line by line. which means first of all first line will execute and the second line has to wait until this second line finishes its task.

Remember our Restaurant example? In one restaurant, a waiter comes to Table 1 and takes orders from them. Then the waiter gives this order to the kitchen and waits for the food to be prepared. He/she is not taking orders from other tables and that's why the restaurant system is being blocked. So that’s the example of blocking or synchronous code.



Now there is another restaurant, a waiter comes to Table 1 and takes orders from them. Then the waiter gives this order to the kitchen and at the place of waiting, he/she goes to another table and takes the order from them. So in this case, the restaurant system is working well and that’s an example of non-blocking or asynchronous code.



So to make our code works like asynchronous code, we have 3 methods.

- Callback
- Promise
- Async/await

🔥 Callback

A callback is a function that is executed after another function has finished execution. In other words, Any function that we passed as an argument to another function, so we call that function a callback.

```
// Callback Example
getStudent(1, (student) => {
    console.log("Student data", student)
})
```

```
function getStudent(id, callback) {
    setTimeout(() => {
        console.log("Getting data from Database");
        callback({ name: "Bob", id: id });
    }, 2000);
}
```

So this callback function will execute after we get data from the database and then simply this function prints student data.

Now when we have more callback functions in another callback then we face the issue of callback hell which is the nested structure of callbacks and it makes our code hard to manage and understand.

```
console.log("First-line");
getStudent(1, (student) => {
    getSubjects(student.id, (subjects) => {
        getMarks(subjects[0], (mark) => {
            console.log(mark);
        });
    });
});
console.log("Last-line");

function getStudent(id, callback) {
    setTimeout(() => {
        console.log("Getting data from Database");
        callback({ name: "Bob", id: id });
    }, 2000);
}

function getSubjects(id, callback) {
    setTimeout(() => {
        console.log("Getting subject of student", id);
        callback(["Maths", "Science", "Computer"]);
    }, 2000);
}

function getMarks(subject, callback) {
    setTimeout(() => {
        console.log("Getting marks of", subject);
        callback(80);
    }, 2000);
}
```

So for solving this callback hell problem, we turn our anonymous functions into named functions. Basically we separately define this functions. Refer this code ⬇️

```
console.log("First-line");
getStudent(1, displayStudent);
console.log("Last-line");

// Defining Callback functions
function displayStudent(student) {
    console.log("Student", student);
    getSubjects(student.id, displaySubjects);
}

function displaySubjects(subjects) {
    console.log(subjects);
    getMarks(subjects[0], displayMarks);
}

function displayMarks(mark) {
    console.log("Mark", mark);
}

// Our Main functions
function getStudent(id, callback) {
    setTimeout(() => {
        console.log("Getting data from Database");
        callback({ name: "Bob", id: id });
    }, 2000);
}

function getSubjects(id, callback) {
    setTimeout(() => {
        console.log("Getting subject of student", id);
        callback(["Maths", "Science", "Computer"]);
    }, 2000);
}

function getMarks(subject, callback) {
    setTimeout(() => {
        console.log("Getting marks of", subject);
        callback(80);
    }, 2000);
}
```

Now this is not the idol solution, but it is better than callback hell.

💡 Promise

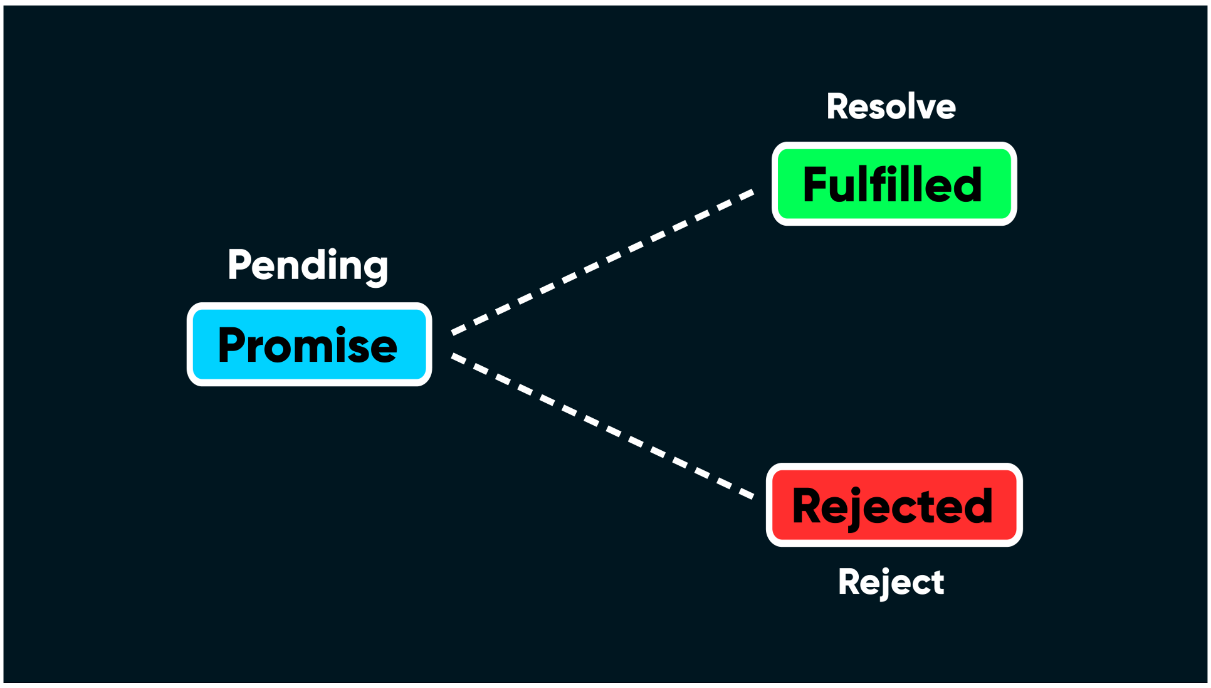
A promise is an object which is able to hold the result of an asynchronous operation. In other words, Promise is promise you - to give you the result of the asynchronous operation or give you an error.

```
// Syntax of Promise
const promise = new Promise((resolve, reject) => {
    // Asynchronous Code
    setTimeout(() => {
        // Getting data from database
        const student = { id: 1, name: "Bob" };
        resolve(student); // Resolve for Fulfill
        // reject(new Error("Student not found")) // Reject for Error
    }, 2000);
});

// Consuming the promise
promise
    .then((result) => console.log(result))
    .catch((error) => console.log(error));
```

Now when we create a `new promise`, it is in a pending state (performing the asynchronous task). After that If that asynchronous task completed successfully then promise in fulfilled state and if there is any error occurs, then the promise is in the rejected state.

So we can get resolve value in then method and reject error in catch method.



```
// Promise
const promise = getStudent(1);
```

```
promise
    .then((student) => getSubjects(student.id))
    .then((subjects) => getMarks(subjects[0]))
    .then((mark) => console.log(mark))
    .catch((error) => console.log(error));

function getStudent(id) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("Getting data from Database");
            resolve({ name: "Bob", id: id });
        }, 2000);
    });
}

function getSubjects(id) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("Getting subject of student", id);
            resolve(["Maths", "Science", "Computer"]);
        }, 2000);
    });
}

function getMarks(subject) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("Getting marks of", subject);
            resolve(80);
        }, 2000);
    });
}
```

This is the same code in which we have a callback hell problem. You can see how easily we can handle asynchronous code with promises.

Now here, we have to write then method again and again. We can get rid of that by using Async/await. (Almost 80% of developers use Async/await)

🚀 Async/await

Async/await helps us to write asynchronous code that looks like synchronous code. When we call the function (which is returning a promise), we can add await operator before that function.

But to use `await` we have to move our code in a function and add an `async` operator before that function. That's it. We don't need anything else.

```
// Async await
async function displayData() {
    const student = await getStudent(1);
    const subjects = await getSubjects(student.id);
    const mark = await getMarks(subjects[0]);
    console.log("Mark", mark);
}

displayData();

function getStudent(id) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("Getting data from Database");
            resolve({ name: "Bob", id: id });
        }, 2000);
    });
}

function getSubjects(id) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("Getting subject of student", id);
            resolve(["Maths", "Science", "Computer"]);
        }, 2000);
    });
}

function getMarks(subject) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("Getting marks of", subject);
            resolve(80);
        }, 2000);
    });
}
```

You can see how simple and easy our code becomes because of `async/await`. So async/await makes our code looks like synchronous code but under the hood, it works like asynchronous code.

Now to handle errors with async/await method, we have to use `try and catch block.` So move all code in `try` block and if any error occurs in the try block then the `catch` block will run.

```
async function displayData() {
    try {
        const student = await getStudent(1);
        const subjects = await getSubjects(student.id);
        const mark = await getMarks(subjects[0]);
        console.log("Mark", mark);
    } catch (error) {
        console.log(error);
    }
}
```

```
displayData();

...
```

I hope you learn a lot from this cheatsheet. This will help you when you want to learn NodeJS. You can check more tutorials on My YOUTUBE channel ~ Code Bless You 💟