**How it works**

An async function returns a promise, like in this example:

const doSomethingAsync = () => {

return new Promise(resolve => {

setTimeout(() => resolve('I did something'), 3000)

})

}

When you want to **call** this function you prepend await, and **the calling code will stop until the promise is resolved or rejected**. One caveat: the client function must be defined as async. Here’s an example:

const doSomething = async () => {

console.log(await doSomethingAsync())

}

**A quick example**

This is a simple example of async/await used to run a function asynchronously:

const doSomethingAsync = () => {

return new Promise(resolve => {

setTimeout(() => resolve('I did something'), 3000)

})

}

const doSomething = async () => {

console.log(await doSomethingAsync())

}

console.log('Before')

doSomething()

console.log('After')

The above code will print the following to the browser console:

Before

After

I did something //after 3s

**Promise all the things**

Prepending the async keyword to any function means that the function will return a promise.

Even if it’s not doing so explicitly, it will internally make it return a promise.

This is why this code is valid:

const aFunction = async () => {

return 'test'

}

aFunction().then(alert) // This will alert 'test'

and it’s the same as:

const aFunction = async () => {

return Promise.resolve('test')

}

aFunction().then(alert) // This will alert 'test'

**The code is much simpler to read**

As you can see in the example above, our code looks very simple. Compare it to code using plain promises, with chaining and callback functions.

And this is a very simple example, the major benefits will arise when the code is much more complex.

For example here’s how you would get a JSON resource, and parse it, using promises:

const getFirstUserData = () => {

return fetch('/users.json') // get users list

.then(response => response.json()) // parse JSON

.then(users => users[0]) // pick first user

.then(user => fetch(`/users/${user.name}`)) // get user data

.then(userResponse => userResponse.json()) // parse JSON

}

getFirstUserData()

And here is the same functionality provided using await/async:

const getFirstUserData = async () => {

const response = await fetch('/users.json') // get users list

const users = await response.json() // parse JSON

const user = users[0] // pick first user

const userResponse = await fetch(`/users/${user.name}`) // get user data

const userData = await userResponse.json() // parse JSON

return userData

}

getFirstUserData()

**Multiple async functions in series**

Async functions can be chained very easily, and the syntax is much more readable than with plain promises:

const promiseToDoSomething = () => {

return new Promise(resolve => {

setTimeout(() => resolve('I did something'), 10000)

})

}

const watchOverSomeoneDoingSomething = async () => {

const something = await promiseToDoSomething()

return something + ' and I watched'

}

const watchOverSomeoneWatchingSomeoneDoingSomething = async () => {

const something = await watchOverSomeoneDoingSomething()

return something + ' and I watched as well'

}

watchOverSomeoneWatchingSomeoneDoingSomething().then(res => {

console.log(res)

})

Will print:

I did something and I watched and I watched as well

**Easier debugging**