Guillaume Perrin Follow
May 22 · 3 min read

Async/await: a beginner's guide

All there is to know about the new keyword

Writing asynchronous code has become necessary to work with APIs and website automation. Those using JavaScript and AJAX have long relied on callback functions to chain their actions one after the other. This made it hard for beginners to get started and even caused frequent pain to more experienced programmers.

The introduction of async/await has made writing and reading asynchronous code a lot easier.

In this post, we'll use two simple examples so anyone with basic JavaScript knowledge can start coding with async/await and start 'controlling time'

We'll start with the general concept then use a real-life example of scraping the front page of Hacker News.

Asynchronous?

Most of the code we write is synchronous: if we have two lines, L1 followed by L2, then L2 cannot be executed before L1 has finished.

In asynchronous code, L2 might not wait until L1 is fully executed. If L1 takes time or schedules a task to be run in the future, then L2 might run before that task completes.

This is typically what happens when we use an API to get data from an outside source.

In the below example, `test` would print `undefined`, since the async call to get the outside data will not have had the time to return anything.

```
function asyncExample() {
   var test = getAsyncData()
   console.log(test) // test will be undefined
}
```

Async/await basic concepts

The idea is to control the execution of the code so that we use the result of the asynchronous function right when we need it.

Consider the following function. `resolveAfterTwoSeconds()`, which will finish executing after 2 seconds and hold the value `x`. This 2-second timeout tells us the function is asynchronous.

(If you are not familiar with Promises, simply think the below is what helps 'store' the value `x` for future use. It is indeed a 'promise' to let us use ('resolve') this value at a later time)

```
function resolveAfterTwoSeconds(x) {
    return new Promise(resolve => {
        setTimeout(() => { resolve(x) }, 2000)
    })
}
```

How can we easily chain actions so that we only execute a new line of code once the above function is fully executed?
This is where async/await comes into play.

- `async` lets us declare that the function will contain asynchronous methods, and will return a Promise
- `await` lets us wait until the following block is fully executed before doing anything else

- `.then()` let us chain instructions after the Promise is resolved. In this case, it just prints the value it contains.

In the below lines, what do you think will happen?

```
async function addInFourSeconds(x) {
    var y = await resolveAfterTwoSeconds(10);
    var z = await resolveAfterTwoSeconds(20);
    return x + y + z;
}

addInFourSeconds(5).then(value => { console.log(value); });
```

The console will print 35 after four seconds.

Let’s comment the code and add two console.logs to help to see what happened. It will:

- wait 2 seconds and print 10
- wait 2 more seconds and print 20 and 35

```
// Declare the async function
async function addInFourSeconds(x) {
    // In 2 seconds, this Promise will be resolved with the
    // Nothing else will be executed until these 2 seconds
    var y = await resolveAfterTwoSeconds(10);
    // print 10, since the above Promise has been resolved
    console.log(y);
    // In 2 seconds, this Promise will be resolved with the
    // Nothing else will be executed until these 2 seconds
    var z = await resolveAfterTwoSeconds(20);
    // print 20, since the above Promise has been resolved
    console.log(z);

    // At this point, y=10, z=10, and both are ready to be
    using .then()
```

Async/await with a real API

NickJS is a great library to write async/await code with. In the below example, we scrape the front page of Hacker News.

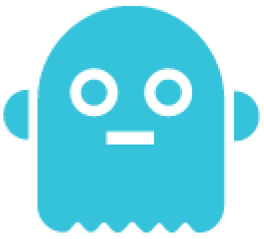
You can see how we easily chain asynchronous operations and keep the code simple to read.

```
nick.newTab().then(async (tab) => {
        await tab.open("news.ycombinator.com")
        await tab.untilVisible("#hnmain") // We make sure w
        await tab.inject("../injectables/jquery-3.0.0.min.j
        const hackerNewsLinks = await tab.evaluate((arg, ca
                const data = []
                $(".athing").each((index, element) => {
                        data.push({
                                title: $(element).find(".st
                                url: $(element).find(".stor
                        })
                })
                callback(null, data)
        })
        console.log(JSON.stringify(hackerNewsLinks, null, 3
})
.then(() => {
```

Are you interested in more coding using async/await? NickJS is open-source, fork it, or run some code directly on the homepage to see what it returns.

Would you rather scrape data and get results straightaway? Create your free account on Phantombuster and enjoy our early-adopter trial.

Happy coding!



PS—Although you don’t need it to get started, the MDN doc is here: async, await, Promise.