TASK

Async / Await

Visit our website

Introduction

WELCOME TO THE ASYNC / AWAIT TASK!

Now that you have gone through the concept of promises and how they work asynchronously from the rest of the code, we can now move on to another core concept of asynchronous functions; async/await. This is a much easier way of creating your own promises while keeping the code easily readable and easier to manage and create!

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** to start a chat with a code reviewer, as well as share your thoughts and problems with peers. You can also schedule a call or get support via email.

Our code reviewers are happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

ASYNC AGAIN?

Yes! You may recall from the other task that we briefly discussed the creation of your own promises and how they can be used in the world of programming; however, we didn't go into great detail on creating your own. The reason for this is we can easily create our own asynchronous functions with a lot less effort and a neater and more understandable code base using the *async* keyword!

⚠ **WARNING**

Please note that this task heavily depends on the concepts learnt in L1T32, particularly the portion that goes over what asynchronous functions are. If this term feels unfamiliar to you, please return back to that task and read over the content again.

WHAT IS THE DIFFERENCE?

In the last task you may recall our creation of a promise and the check for whether the return value was accepted or rejected. The way we wrote this was similar to the example below:

```
let myPromise = new Promise(function(resolve, reject) {
   let randNumber = Math.floor(Math.random() * 10);

   if (randNumber >= 5) {
       resolve("Number was greater than or equal to 5 [RESOLVED]");
   }else{
       reject("The number was less than 5 [REJECTED]")
   }
})
console.log(myPromise);
```

Notice how lengthy this is to create. It can also be quite difficult to read and understand without having to read over it a couple of times. What if we told you this could easily be converted into a normal function that performs the exact same thing? That's exactly what async does!

Take a look at the code below:

```javascript
async function myAsyncFunction() {
   let randNumber = Math.floor(Math.random() * 10);

   if (randNumber >= 5) {
       return("Number was greater than or equal to 5 [RESOLVED]");
   }else{
       return("The number was less than 5 [REJECTED]")
   }
}
console.log(myAsyncFunction());
```

Notice how the syntax of this function feels a lot more familiar to you and a lot more like how you've created functions in the past! There is no need to create a new promise object and the concept is a lot easier to read and understand. The only difference you may have noticed is that we have the *async* keyword just before our function.

This is the only thing we need to let JavaScript know that we want to run this function asynchronously from the rest of the code. From there you can write any code you may want and just return the value you wish to return, as usual.

This is typically why async is the preferred choice for new programmers. Note that this does not mean that you should only ever use async, or always choose it over promises. Let's consider how to decide when to use which.

PROMISES vs ASYNC

While these concepts are very similar to one another, there are a couple of things to keep in mind when deciding which one to use.

Take a look at the table below, which will give you a quick overview of the key differences between the two concepts.

	Promise	**Async**
Scope	Only the original promise itself is asynchronous.	The entire function itself is asynchronous.
Logic	Any synchronous work can be completed inside the same callback.	Synchronous work needs to be moved out of the callback.

	Multiple promises make use of the promis.all() method.	Multiple promises can be assigned to variables
Error Handling	A combination of then/catch/finally	A combination of try/catch/finally

Which to use?
It's really up to your personal preference at this point in time. As you start going through different API calls and running asynchronous code you will find specific use cases for each. Nonetheless, here is some general advice.

When to use a promise
Promises are a good option should you want to quickly but also concisely grab results from a promise. Promises are a good choice to avoid writing multiple wrapper functions inside an async when a simple *.then* will suffice.

When to use Async
You'll typically make use of asynchronous functions when you are using complex code that needs to run separately from the rest of the code. The simple syntax of asynchronous functions and the fact that this is more familiar to people, makes it easier to follow the code.

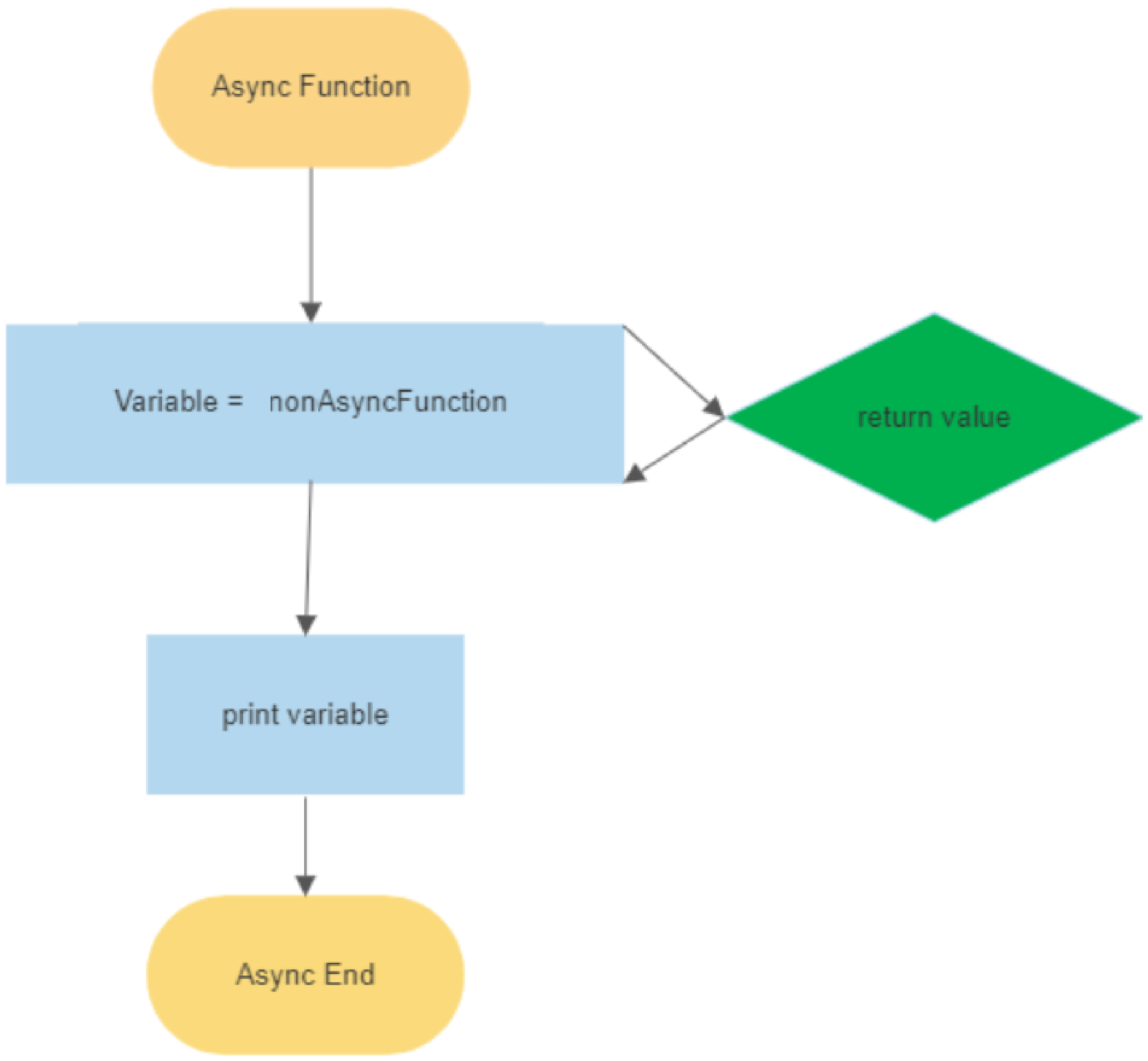
Overall
Use promises when you want to work with simple logic that won't require a lot of processing, and use asynchronous functions for more complex code.

USING AWAIT

Now that you better understand the difference between asynchronous functions and promises, we can introduce the second concept in this task, the await keyword.

The await keyword is rather simple to understand. Because an asynchronous function runs code separately from the rest of our code, there is a chance that we may need to have our code wait for another function to run before we can continue.

Take a look at the image below:



Notice how our variable is assigned to the “nonAsyncFunction” which is a separate function that returns a new value. We want to end up printing that value out into our program using our async function. The issue here is that because of the nature of async functions, it will attempt to print the variable before we have an assigned value for it! This, of course, would cause our code to crash. So how do we fix this? With await!

The await keyword basically tells an async function to wait for the other function’s process to complete before running the rest of the code. This is perfect if you require a set of content to continue the process.

Take a look at the code provided in the table below explaining await using functions and arrow head functions:

	Arrow Function	Normal Function
Await	`const returnName = () => { return 'spinel' }` `const asyncArrowFunction = async () => { let myName = await returnName(); console.log(myName); }` `asyncArrowFunction();`	`function returnNameFunction() { return 'spinel' }` `async function asyncFunction() { let myName = await returnNameFunction(); console.log(myName); }` `asyncFunction();`
Non Await (possible error)	`const returnName = () => { return 'spinel' }` `const asyncArrowFunction = async () => { let myName = returnName(); console.log(myName); }` `asyncArrowFunction();`	`function returnNameFunction() { return 'spinel' }` `async function asyncFunction() { let myName = returnNameFunction(); console.log(myName); }` `asyncFunction();`

The reason that the nonAwait may not always produce an error is because of the fact that some machines/servers process information a lot more quickly than other devices. This is why it's important to always consider different devices and speeds and always account for the usage of await.

FETCHING APIs

As you have learnt, asynchronous functions are a form of promises, and as such, can fetch APIs! Take a look at the below code. It should seem very familiar to you and even more understandable than before!

```
async function apiFunction() {
   let item = await fetch("apiLink")
   console.log(item)
}
```

Notice how it's exactly the same as a promise, just without all the headaches of the *.this* keyword! Where the word *apiLink* is, is where you would place the API URL.

Notice how easily readable it is! Just don't forget the **await** keyword, or else you will run into issues.

Compulsory Task 1

Follow these steps:

- The group that writes all your learning content is fondly known as the "content cats". Because of this, it'd be a no-brainer to have you create a website that will randomly generate GIFs of cats!
    - Use the following API URL to fetch a random gif of a cat: **http://thecatapi.com/api/images/get?format=src&type=gif**
    - **Ensure that you make use of async/await and not a promise when fetching the API.**
    - Please only output the image URL in the console.
    - **YOU DO NOT NEED TO CREATE A FULL FRONTEND FOR THIS PROJECT.**

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.