Github Copilot – JavaScript

Website Development With JavaScript and Copilot

Environment:

Since we are going to work with HTML, CSS and JavaScript, there is no setup to begin with. You just need to decide on a folder to create your files.

You do not need GitHub Copilot Professional to get a lot from this bootcamp but it does help to have it installed. There is a free *limited* version. Here are the instructions for that version: https://docs.github.com/en/copilot/managing-copilot/managing-copilot-as-an-individual-subscriber/getting-started-with-copilot-on-your-personal-account/getting-started-with-a-copilot-plan#accessing-copilot-free

Although the environment that I am using is a JavaScript/web development one, the ideas presented can be adapted to any programming environment.

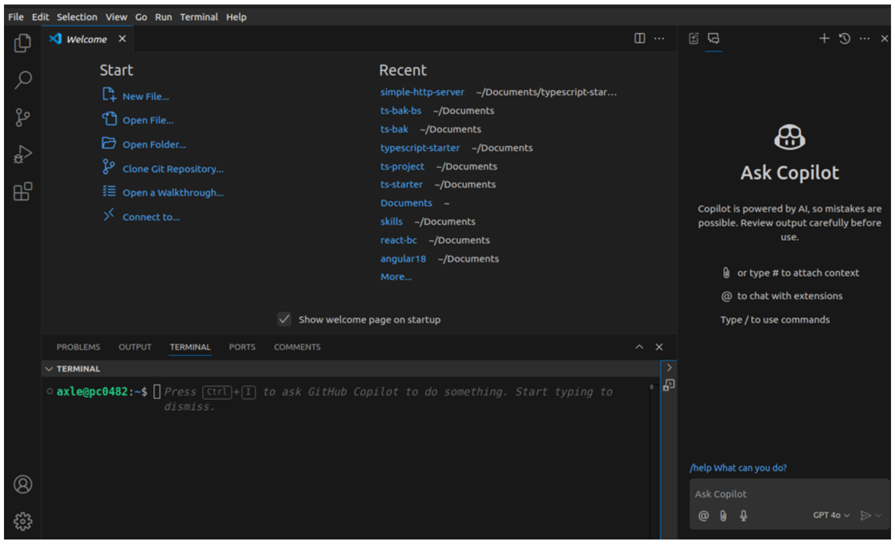
Part 01: Folder Setup

Decide on a folder to start from, e.g. Documents. Open VS Code, you don't have to navigate to a folder yet. Once in VS Code sign in to your GH Copilot account. Type in your first prompt in the Chat window:

```
In the Documents folder/directory create a project folder to support a HTML,CSS, JavaScript project.
```

Once you have a workspace, you can change the folder name to ghc-js or something similar.

Open a terminal window inside the VS Code IDE. In the menu area there should be an option called *Terminal*, just click it and choose *New Terminal*.



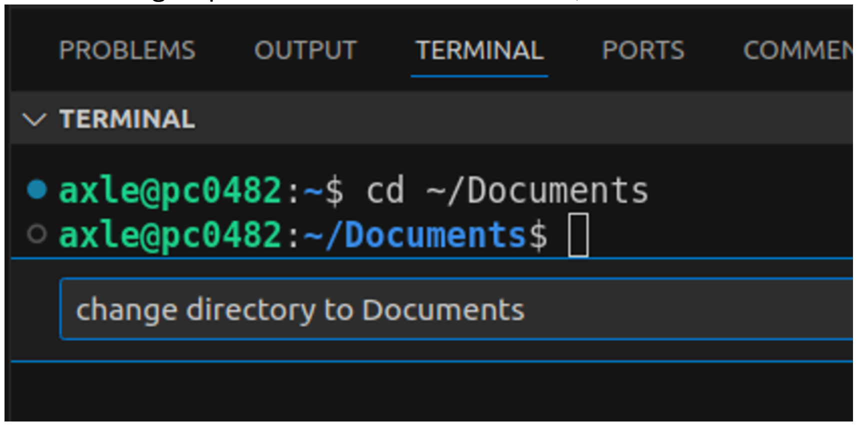
```
axle@pc0482:~$ Press Ctrl+I to ask GitHub Copilot to do something. Start typing to
               dismiss.
```

Copilot is powered by AI, so mistakes are possible. Review output carefully before use.

If you see a column with the *DEBUG CONSOLE* label at the top, just click the downward facing arrow and that window will go away.

Depending on your version you may already start seeing Copilot prompts in a dark grey colour in the terminal window. This is called Ghost Text, it is already trying to help you get started.

Here I am asking Copilot via the Terminal window, to move me to the Documents directory in my Linux file system



```
axle@pc0482:~$ cd ~/Documents
axle@pc0482:~/Documents$
```

change directory to Documents

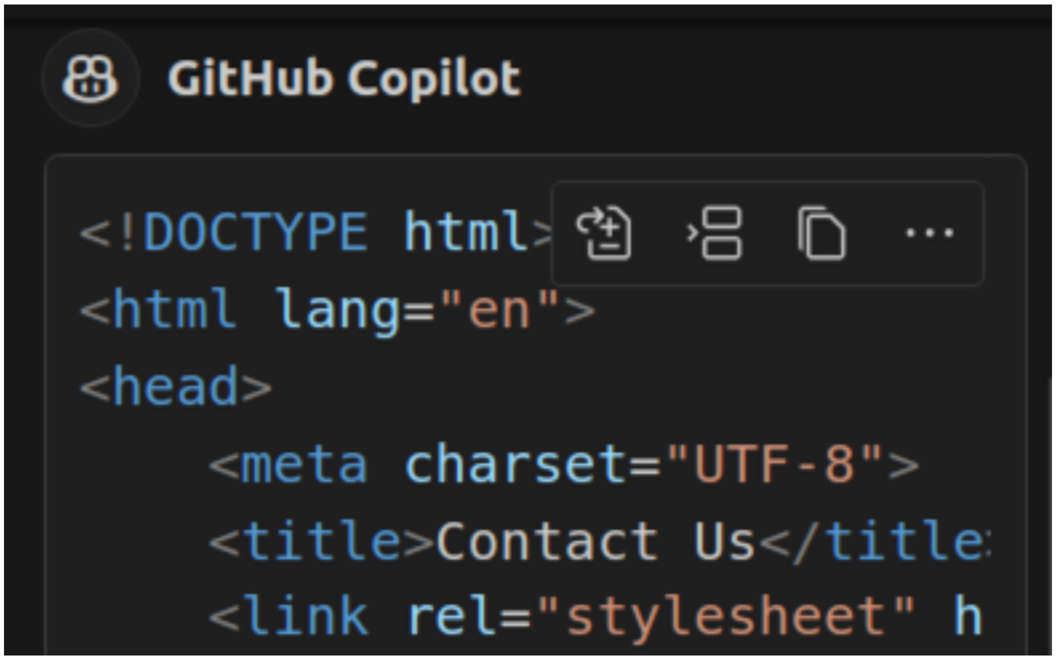
So, now if I create a new project folder, it will be created inside of the Documents directory.

Part 2

As a JS developer you will work with forms a lot, lets see how GH Copilot can help us develop forms. Now some of the work will be in HTML and CSS but we will get to the JS soon. Here is my next prompt:

```
In the ghc-js project directory, create a new html document called contact-
us.html, in it add a simple contact us form
```

Once the response shows in the Chat window, you will have to accept it by clicking on the "Apply in Editor" icon, it's the left-most icon just above the response code:



```
<!DOCTYPE html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Contact Us</title
    <link rel="stylesheet" h
```

You will have to provide a path and a name for the new HTML document. Note, this may have changed by the time you read this.

The next prompt will attempt to setup JS validation for the HTML form. You should have a form already built in the file called contact-us.html.

Here is the next prompt:

`I do not want to post the form but I want to setup validation using the app.js file in the scripts folder`

If this prompt goes well, you will now have a scripts tag in the html file along with changes to the app.js file in the JavaScript folder. Open the contact-us.html file with Live Server and check the validation.

In my case, the validation looks good but the error messages are showing as an alert. We can fix this with GH Copilot. Here is my next prompt:

`Error messages are appearing in an alert pop-up box, i want all user messages to appear on the HTML document itself.`

Notice I did not differentiate between error messages and just informational messages.

At this point, you can try to fix the form if it is not centered, and you may add additional validation criteria such as length of name or you may even try sanitizing the text area looking for profanity etc. For example here is a prompt that works for me:

`I want to add basic sanitization of the text in the text area, specifically I want to look for SQL injection.`

Also you can highlight any line you are not sure about and ask Copilot to explain the line.

As a final task in this section, you might want to make sure that the user is human by having them check a box or even a captcha:

```
i want to check if the user is human before clicking the send button
```

Part 3

Note: for this bootcamp June 2025, we will NOT be following these instructions below. The website is currently down. The instructor will have an alternative.

In this part we will attempt to access a public endpoint, consume the exposed data at that site and ask GH Copilot to organize the data into a formal table with some structure.

The public API endpoint is located at 'https://www.freetestapi.com/api/v1/animals'

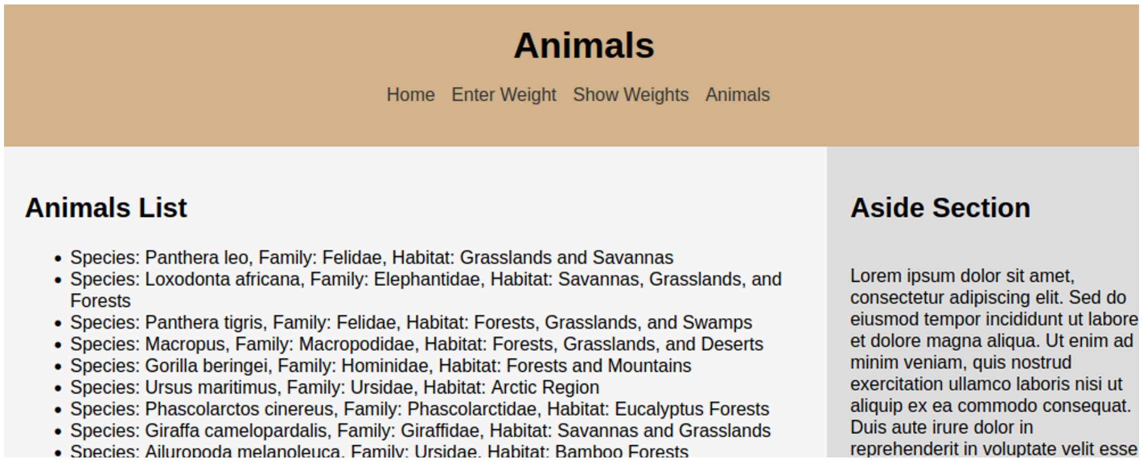
Lets create a new HTML page and use the JS file to make a RESTful request for data from this endpoint. We can use the same website we created in Part 1. Of course what we are about to do does not make sense in the real world, but again, this is NOT about creating web sites or even consuming API data, it is about the GitHub Copilot tool.

```
Create a new web page called animals.html which will make a get request to this API endoint 'https://www.freetestapi.com/api/v1/animals' and display the data from species, family and habitat. Follow the same format, theme and layout as the other three web pages.
```

In my case, the AI tool also wanted me to update the other three html files, since it thinks that all files are now connected in some way. I complied with this suggestion, however the JS function was included in the animals.html file. I wanted it to be in the js folder and the script.js file, so I had to issue another prompt for this:

```
Remove all the JavaScript from animals.html file and place that code in the script.js file which is in the js folder of this project folder. connect the animals.html file to the script.js file in the js folder
```

This worked but the list does not look very nice, we will try to fix this with a few prompts, this is what my page looks like now:


Animals

Home Enter Weight Show Weights Animals

Animals List

- Species: Panthera leo, Family: Felidae, Habitat: Grasslands and Savannas
- Species: Loxodonta africana, Family: Elephantidae, Habitat: Savannas, Grasslands, and Forests
- Species: Panthera tigris, Family: Felidae, Habitat: Forests, Grasslands, and Swamps
- Species: Macropus, Family: Macropodidae, Habitat: Forests, Grasslands, and Deserts
- Species: Gorilla beringei, Family: Hominidae, Habitat: Forests and Mountains
- Species: Ursus maritimus, Family: Ursidae, Habitat: Arctic Region
- Species: Phascolarctos cinereus, Family: Phascolarctidae, Habitat: Eucalyptus Forests
- Species: Giraffa camelopardalis, Family: Giraffidae, Habitat: Savannas and Grasslands
- Species: Ailuropoda melanoleuca, Family: Ursidae, Habitat: Bamboo Forests

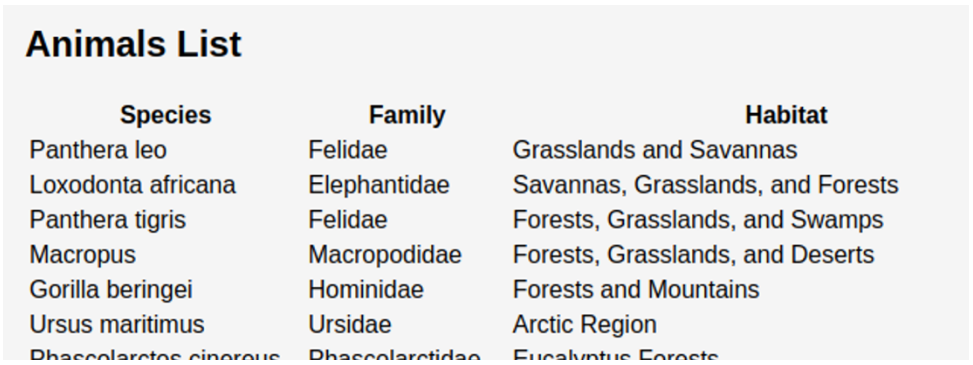
Aside Section

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse

Here is my first prompt to try to get this data organized, it is rather vague by design:

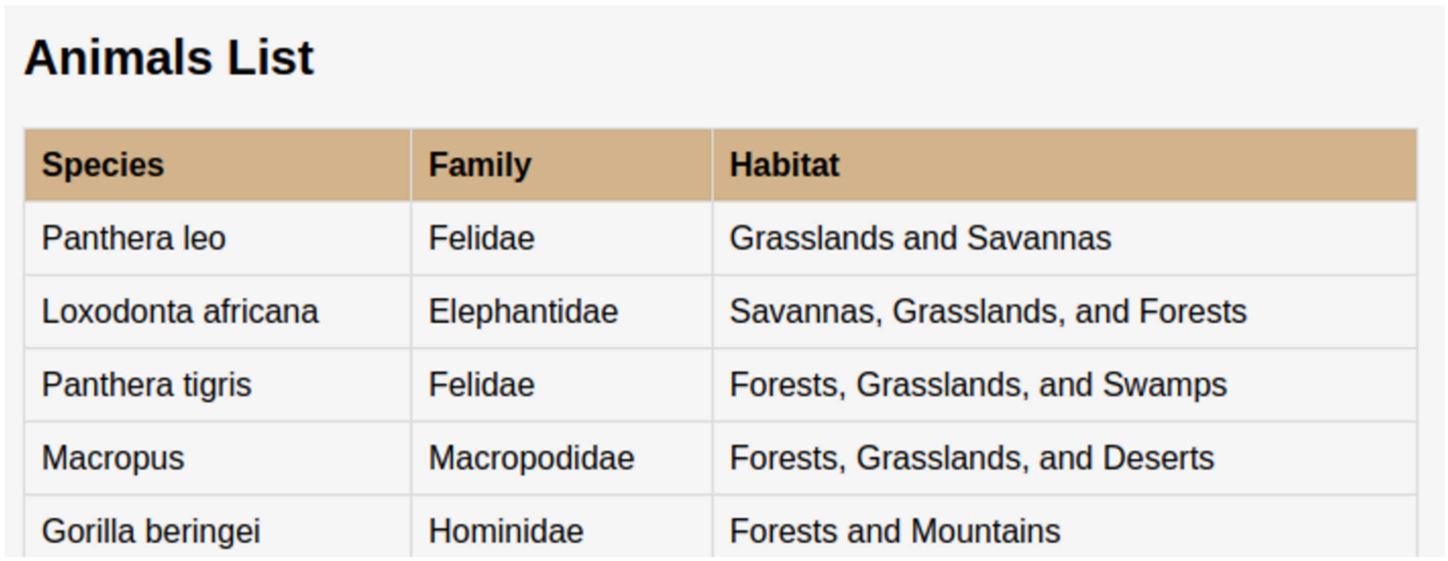
Organize the data returned from the fetch request into a table with an appropriate heading.

Here is the result, everything is off so we will need more prompts to get the content aligned properly:


Animals List

Species	Family	Habitat
Panthera leo	Felidae	Grasslands and Savannas
Loxodonta africana	Elephantidae	Savannas, Grasslands, and Forests
Panthera tigris	Felidae	Forests, Grasslands, and Swamps
Macropus	Macropodidae	Forests, Grasslands, and Deserts
Gorilla beringei	Hominidae	Forests and Mountains
Ursus maritimus	Ursidae	Arctic Region
Phascolarctos cinereus	Phascolarctidae	Eucalyptus Forests

center the table and align the text in the columns to match the column heading

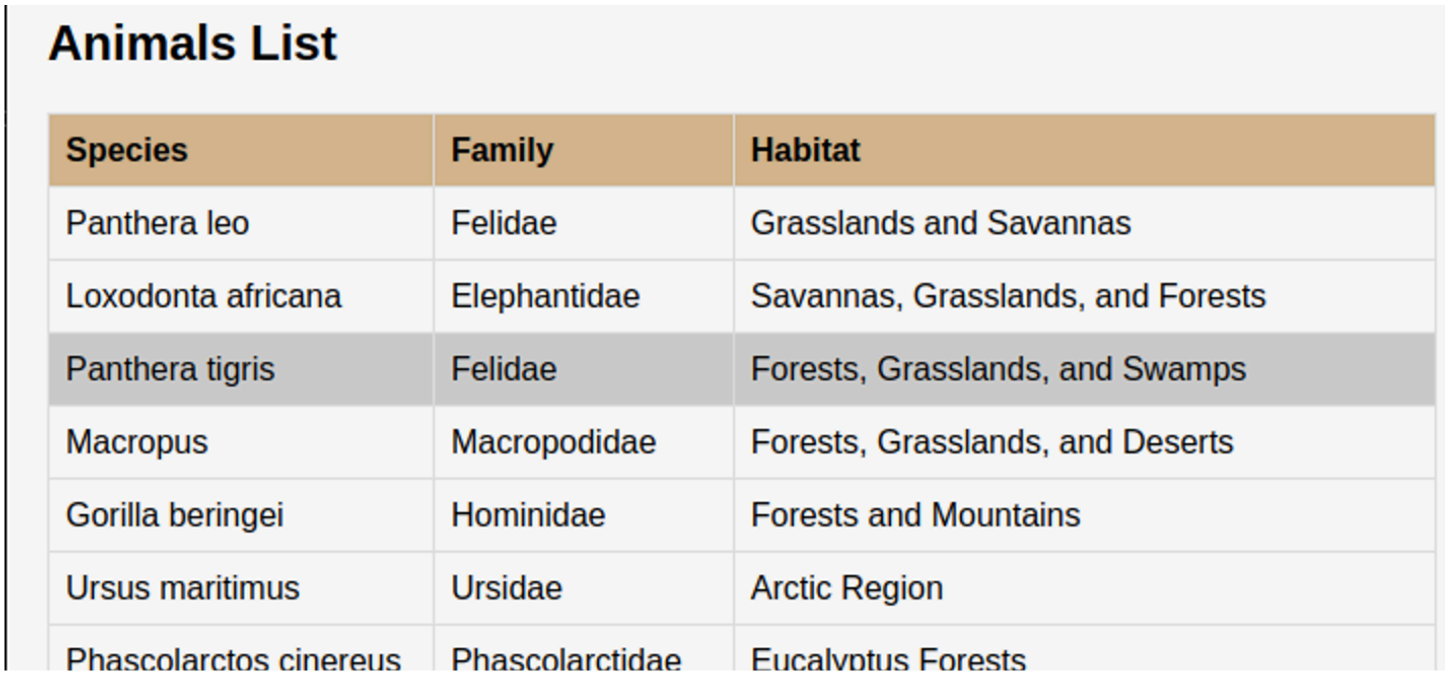


Animals List

Species	Family	Habitat
Panthera leo	Felidae	Grasslands and Savannas
Loxodonta africana	Elephantidae	Savannas, Grasslands, and Forests
Panthera tigris	Felidae	Forests, Grasslands, and Swamps
Macropus	Macropodidae	Forests, Grasslands, and Deserts
Gorilla beringei	Hominidae	Forests and Mountains

```
add a hover effect as the mouse moves over the rows of the table
```

```
the hover effect is too light, make it slightly darker
```



Animals List

Species	Family	Habitat
Panthera leo	Felidae	Grasslands and Savannas
Loxodonta africana	Elephantidae	Savannas, Grasslands, and Forests
Panthera tigris	Felidae	Forests, Grasslands, and Swamps
Macropus	Macropodidae	Forests, Grasslands, and Deserts
Gorilla beringei	Hominidae	Forests and Mountains
Ursus maritimus	Ursidae	Arctic Region
Phascolarctos cinereus	Phascolarctidae	Eucalyptus Forests

Lets call this the end of Part 3

Part 4 – Closures

In this part we will look at the concept of a closure in JS. A closure is a function that remembers its outer scope even when the outer function completes.

```
"use strict";
function Student(id, name, age, major) {
  this.id = id;
  this.name = name;
  let _age = age;
  let _major = major;
  this.printStudent = function() {
    console.log(
      ` ID: ${this.id},
      Name: ${this.name},
      Age: ${_age},
      Major: ${_major}`
    );
  };
};
//
let student1 = new Student(
  1, "Alice", 20, "Computer Science"
);
//
console.log("Student 1:", student1.id, student1.name, student1._age);
```

In the code above, a closure is formed around the printStudent() function that will include the _age and _major variables.
The variables _age and _major do not belong to the object being created by the function constructor, they become part of the closure of any nested function, printStudent in this case.

When the above code is executed, we get the student id and student name but _age is undefined.

Lets ask Copilot why this is:

```
In the index.js file, line 21 prints the id and name but I get undefined for _age
```

Copilot should respond with possible reasons as well as a fix for the undefined. It may or may not be correct in the explanation.

If it does not focus in on the topic of closures, try to log the entire student1 object and then rephrase the prompt. For example the log statement should be this:

```
console.log("Student 1:", student1);
```

In my case I had to dump the entire code into the Chat window.

Note if you do not get a good explanation from Chat, just call the printStudent() function on the student object and all four properties will print.

Part 5 – JS Dates

In part 5 we will work with dates in JS.

1. Create a new JS file called index.js in the scripts folder and connect your index.html to that file via a <script> tag

   ```
       <footer>
         <p>&copy; 2025 Web Application</p>
       </footer>
       <script src="scripts/index.js"></script>
     </body>
   ```

2. Here is the first prompt:
   `use css to push the footer down towards the bottom of the visible web page`

3. For the next prompt I want a panel to display dates, so try this prompt:
   `add a secondary div tag below the header to display important information`

4. Now we can continue with prompts to setup dates and work with date arithmetic. Try these prompts:
   `in the div you just added, make today's date appear at teh very right end of that div tag. Use JavaScript to get todays date.`

5. `I want the date to be in the format mm-dd-yy.`
   Note, if you got the initial date in this format from Copilot, just try a different format.

6. Depending on what happened, you may want to move the date. Here is my next prompt:
   `i want the number of days opened to show inside of the div tag on line 13, the div tag that has a class of important-info`

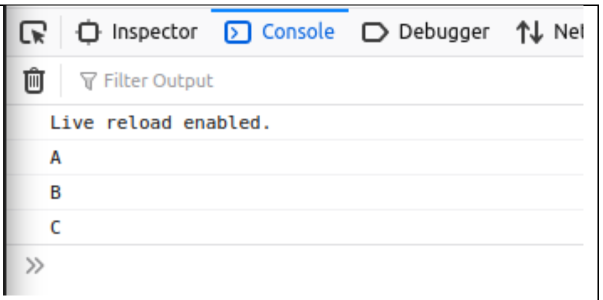
Part 6 – Asynchronous JavaScript

In part 6 we will work with asynchronous JavaScript.

1. Create a new HTML page called async.js and add a button to call a JS function. You could ask Copilot to do this for you, here is a prompt:
   `Create a new html file called async.html and copy the same theme as in index.js, add a single button to the middle of the page that will call a JavaScript function named show()`

2. Now add the code below into the async.js file:

```
function A() {
  console.log("A")
}
function B() {
  console.log("B")
}
function C() {
  console.log("C")
}
A();
B();
C();
```



```
Live reload enabled.
A
B
C
```

Note: to remove the favicon error from some browsers just add this line in the <head> section of your html file:
<link rel="shortcut icon" href="#">

3. Now add a setTimeout() method to function B so that it does not execute until 2 seconds after the other two functions:

```
        ...
        }
        function B() {
          setTimeout(() => {
            console.log("B")
          }, 2000);
        }
function C() {
...
```

Refresh the browser, you see A, then C but have to wait for B, and the order is not correct.

4. Lets see if Copilot can help us with this problem. Make sure your async.js and async.html files are in context:
   `when this code executes, i see A then C but have to wait for B. I want to print all three output in the order of A, B, C`

Note, you may or may not get a Promise-based option, depending on what option they provide, just ask for another option, just to see how it works. For example in my case I did get a promise-base option, so I sent this prompt: `Is there another option other than a Promise-based one?`

In my case I got a callback-based option.

5. Go back to the Promise-based option and depending on how the handler is coded, just ask if there is a different handler. For example in my case I got an async-await handler, so I asked this prompt:
   `Is there a different handler method instead of async/await`

   I then got the .then() solution.

6. Lets try a more difficult example, remove the previous code in the async.js file and add this code instead:

```
const lcNames = ["jones", "rosales", "rahm"];
const properName = lcName => {

        let ucName = lcName.charAt(0).toUpperCase()
          + lcName.slice(1);
        return ucName;

};
lcNames.forEach(
        upperName => console.log(properName(upperName))
);
```

This should print out in the console window, the three names in the lcNames array, but with the first letter in uppercase, so Jones, Roseles, Rahm

7. Lets now add a set timeout to the ucName() function:

```
const properName = lcName => {
 setTimeout(() => {
  let ucName = lcName.charAt(0).toUpperCase()
   + lcName.slice(1);
  return ucName;
 }, 1000);
};
```

You should now see three undefined in the console window.

8. Lets first ask why we are getting three undefined ouput in the console window:
`how can i fix this code so that i do not get undefined three times`

In my case copilot simply moved the log statement to be the last statement in the setTimeout() method, not an elegant solution but it works:

This is the solution I got:

```
          const properName = lcName => {
           setTimeout(() => {
            let ucName = lcName.charAt(0).toUpperCase()
             + lcName.slice(1);
            console.log(ucName);
           }, 2000);
          };
          //
          lcNames.forEach(
           upperName => properName(upperName)
 );
```

Missing the log statement, it want to the setTimeout() method

So, now I can ask for a more elegant solution, so ask about the promise-based solution with async and await.

```
const lcNames = ["jones", "rosales", "rahm"];
const properName = lcName => {
 return new Promise(resolve => {
  setTimeout(() => {
   let ucName = lcName.charAt(0).toUpperCase() + lcName.slice(1);
   resolve(ucName);
  }, 2000);
 });
};
//
Promise.all(lcNames.map(properName)).then(results => {
 results.forEach(name => console.log(name));
});
```

From Copilot:

The code is now promise-based: `properName` returns a Promise, and `Promise.all` is used to log all capitalized names after the delay. This ensures all results are handled asynchronously and cleanly. Let me know if you need further adjustments!

Part 6 – JavaScript Queues

In part 6 we will work with asynchronous JavaScript.

1. Create a new HTML page called queue.html and a JS file called queue.js. Add the JS file using the <script> tag to the HTML file.

2. Here is my first prompt:
   `Use the queue.js file to create a function called enqueue that will add values into a queue`

3. Once you have a function, you can go on to make the queue better, for example you might want to add more elements into the array:
   `add three more fruits into the queue`

4. Continue working on the queue:
   `Now I want to view what is in the first position in the queue`

5. How about reversing the queue:
   `Add a function to reverse the queue elements`

6. In the HTML file, we can add an interface to interact with the queue:
   `In the queue.html file, add an interface for me to be able to call all the functions in the queue.js file`

7. Finally:
   `Change all the code in the queue.js to be class based`