**JavaScript**

**Daily Exercises**

Version 6.5 ITC

### Introduction

#### This document contains all of the exercises you will complete during the five days you are working through the JavaScript content.

#### Most of the videos you will watch this week are centered on JavaScript syntax. But we want you to have lots of experience creating dynamic web pages too so we have included some "Quick Lessons" in this exercise guide to share that knowledge.

#### *You may decide you want to do the exercises in the JavaScript session in groups of 2 or 3. If so, that is fine! Talk it over in the group, but make sure in most cases everyone types their own solutions.*

#### [Day 1](#_heading=h.s0kpb5af6i50)

#### Tasks:

#### Code-Along: Hello World

#### Exercises: Math Scripts

#### Quick Lesson: Parsing Strings into Numbers

#### Exercise: Conditionals

#### Quick Lesson: JavaScript in the Browser

#### Exercise: Hello World!

#### Quick Lesson: Working with <input> Elements

#### Quick Lesson: Events

#### Exercise: Click for Hello World

#### Exercises: Two Simple Websites

#### (Optional) Mini-Project

#### [Day 2](#_heading=h.2s8eyo1)

#### Tasks:

#### Quick Lesson: Working with Strings (Review)

#### Exercise: Working with Strings

#### Quick Lesson: Anonymous Functions

#### Quick Lesson: Working with Checkboxes

#### Quick Lesson: Working with Radio Buttons

#### Quick Lesson: Using querySelector()

#### Exercises: Two Simple Websites

#### [Day 3](#_heading=h.44sinio)

#### Tasks:

#### Quick Lesson: JavaScript Object Literals

#### Exercise: Object Literals

#### Quick Lesson: Arrays

#### Exercises: Working with Arrays

#### Quick Lesson: Working with <select> Lists

#### Exercise: Loading a Dropdown

#### Quick Lesson: Determining Which Option is Selected

#### Quick Lesson: onchange Event

#### Exercise: Working with Dropdowns

#### Exercise: Cascading Dropdowns

#### [Day 4](#_heading=h.uqbu7qvqyisf)

#### Tasks:

#### Exercise: REST APIs and jsonplaceholder

#### Quick Lesson: Using the VS Code REST Client

#### Quick Lesson: Fetch API GET Request to jsonplaceholder

#### Exercises: Fetch API and GET Requests

#### Quick Lesson: Node.js and Web Applications

#### Code-Along: Cloning and Configuring a Node.js REST API

#### Exercise: Working with Courses API

#### (Optional Challenge) Mini-Project

#### [Day 5](#_heading=h.41mghml)

#### No self-paced videos today!

#### Tasks:

#### Quick Lesson: Sending a Post Request

#### Quick Lesson: Sending a Put Request

#### Quick Lesson: Sending a Delete Request

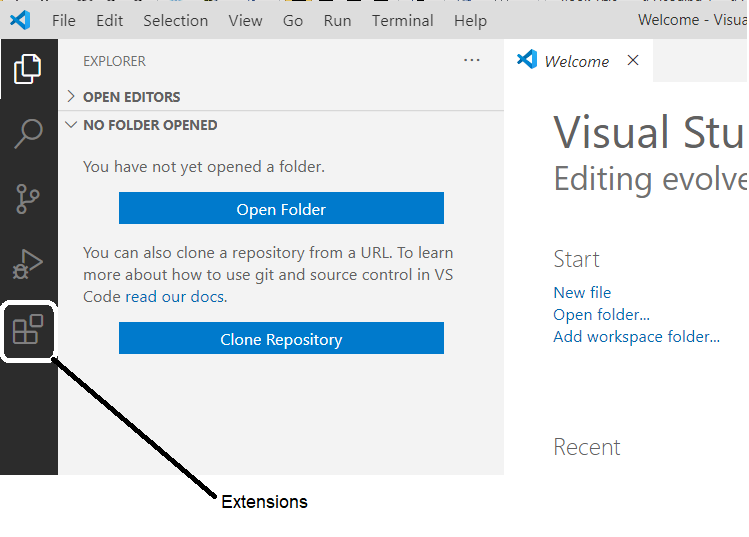
#### Exercises: Rounding out the Courses Website

#### Catch up from anything during the week

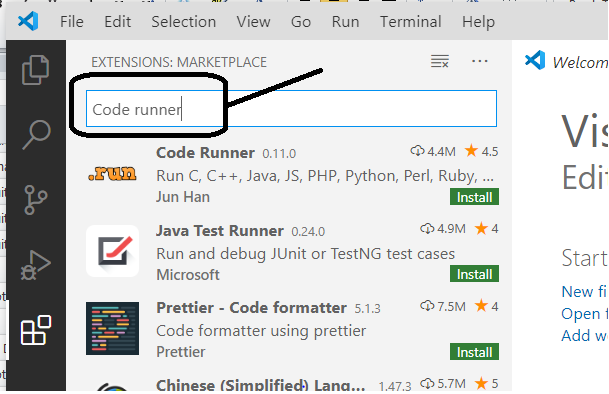
# DAY 1

## Code-Along: Hello World

* **In this code-along, we will get Visual Studio Code configured for JavaScript and then code "Hello World"**
* **Before we begin coding, we need to add a couple of VS Code extensions**
* **Code Runner** allows us to run JavaScript scripts without having to embed them in an HTML page
* **Live Server** creates a local development server that hosts the pages open in Visual Studio Code
* This keeps us from having to deploy a site to a web server to test it
* **To install extensions, click on the Extensions icon on the left bar**



* **In the search pane, type the name of the extension you want to install**



* Then click the Install button
* You may have to close and then re-open Visual Studio Code in order to see your extensions
* **Did you get Code Runner and Live Server installed? If so, it is time to code "Hello World"**
* **To get started, let's create a place to keep our code**
* Create a GitHub repo named js-experiments

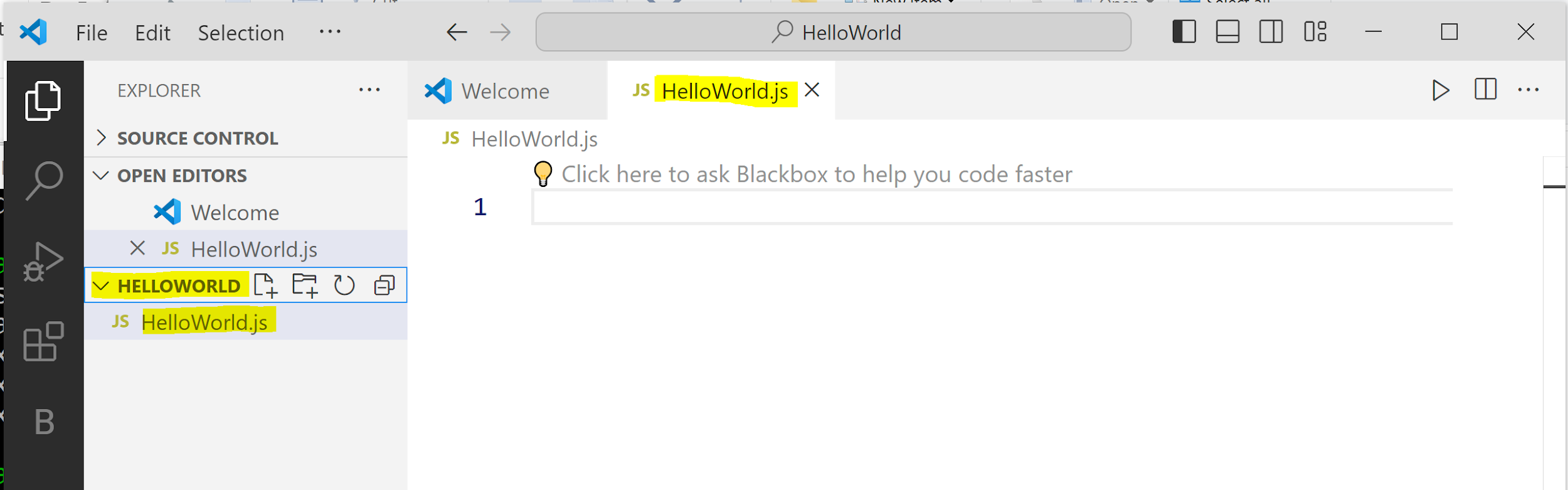
###### This repo will hold JavaScript experiments that are NOT connected to a web page ~ that is, JavaScript code that practices coding concepts

##### Clone it to your local machine

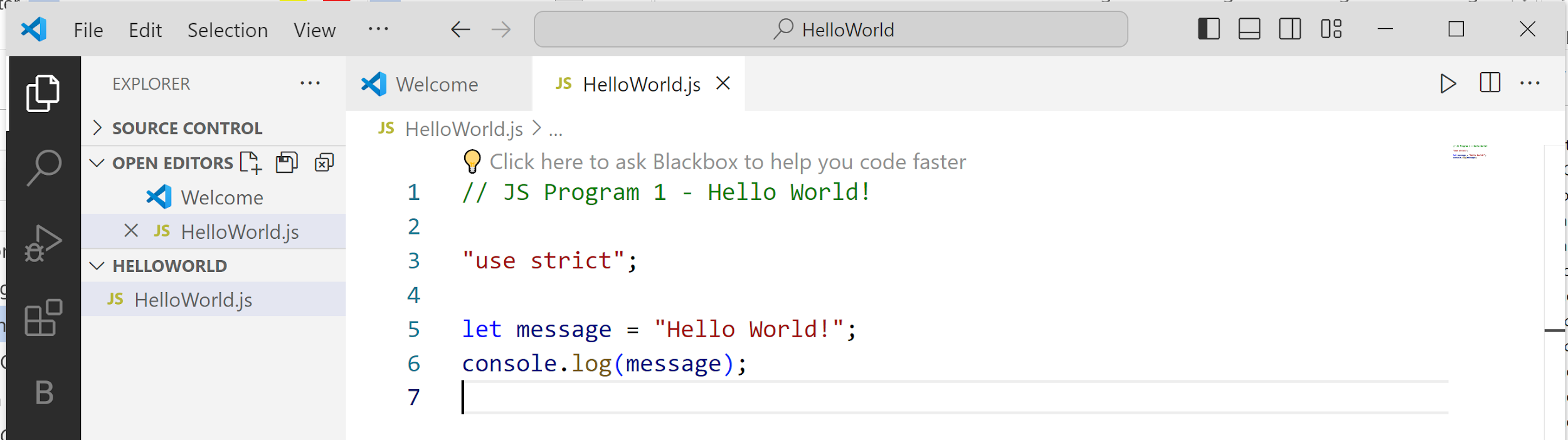
##### Inside of that, create a folder named HelloWorld

##### 

* **In Visual Studio Code, use the Open Folder option to open your HelloWorld folder**
* **Create a new file named HelloWorld.js**

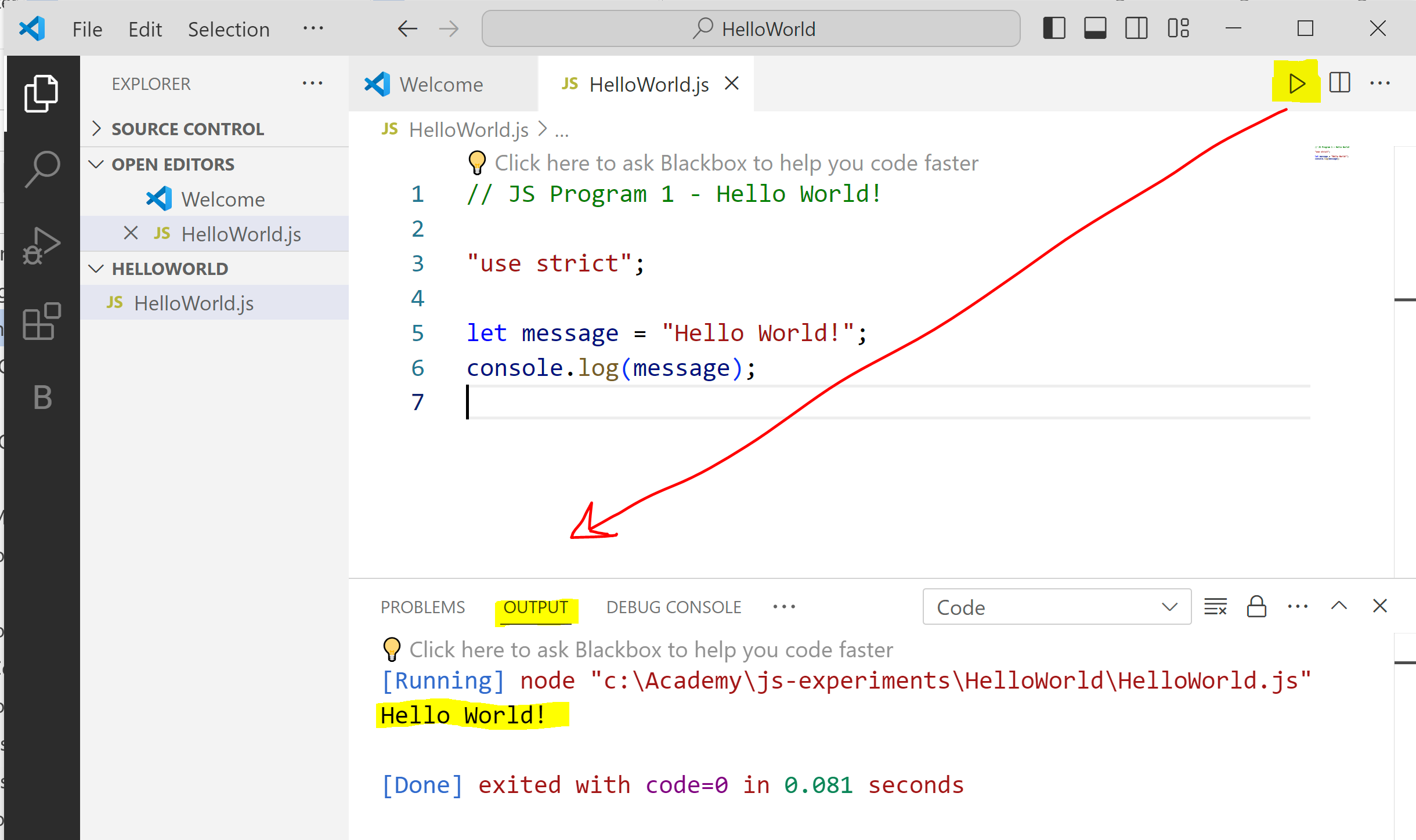


* **Code the following:**



###### NOTE: Updates to VS Code sometimes create small disparities between our screenshots and your installation. Don't let it throw you!

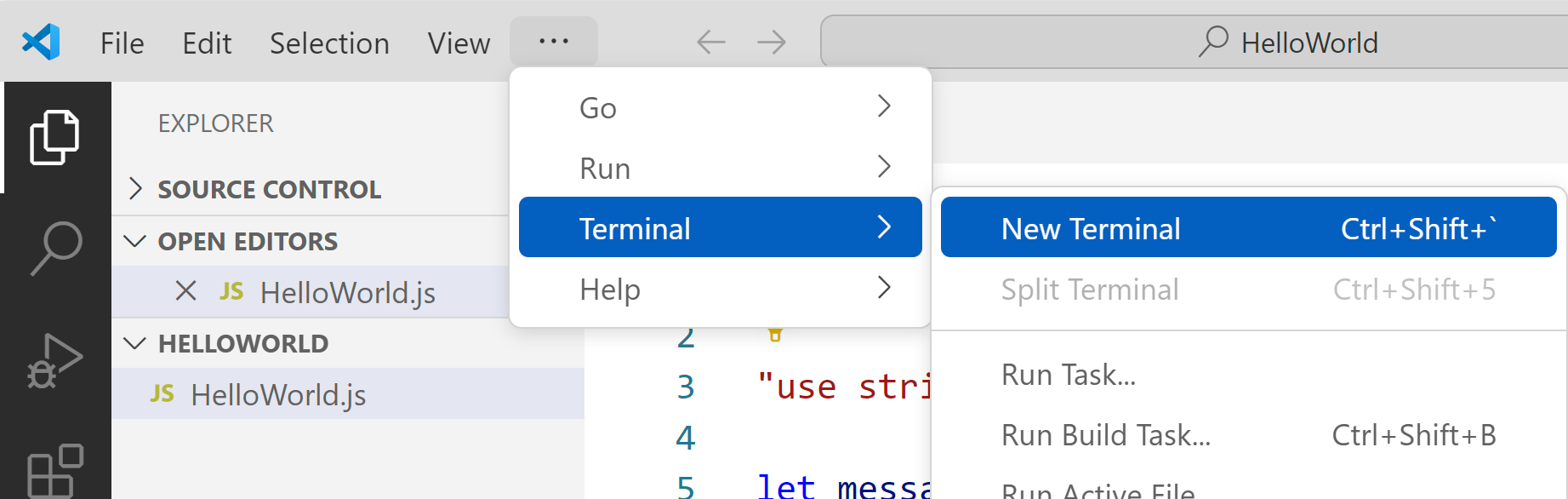
* Save the file
* **To run the script, click the run icon in the top right corner and then look at the Output pane for the results**



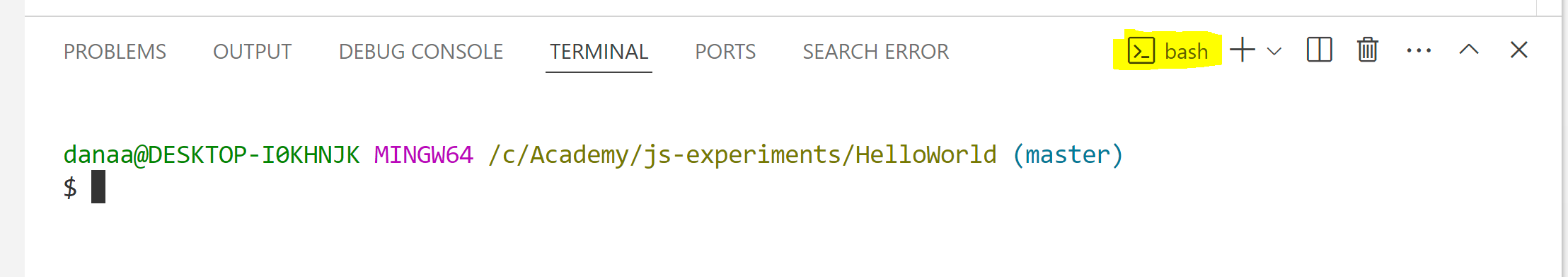
**WARNING**: Sometimes Code Runner generates an error. You can tell it is a Code Runner error and not you if it's not YOUR file (ex: HellowWorld.js) in the error message. Just re-run the script. After trying 2-3 times, sometimes you have to close VS and reopen. Yes - it is frustrating!

#### To commit your changes and push your local repo to GitHub, you can use the Terminal menu

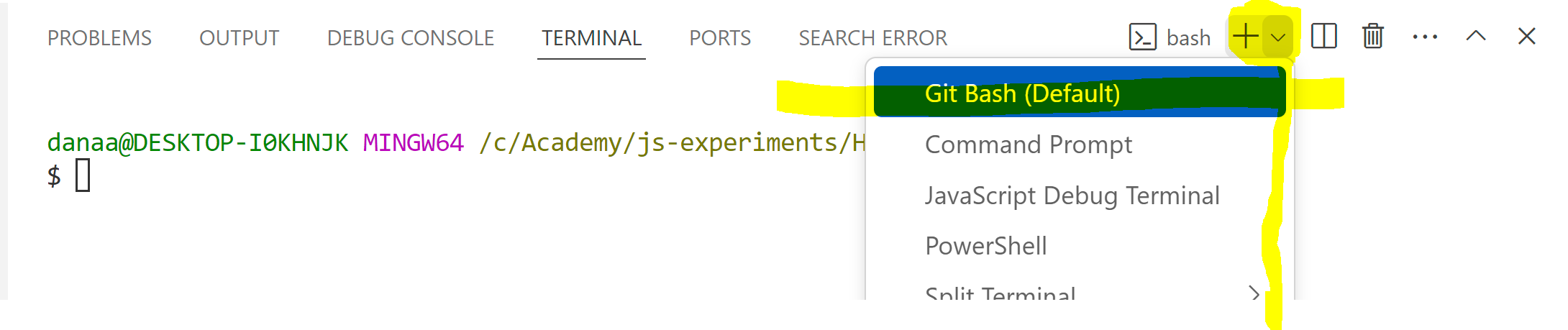
##### NOTE: Using the Terminal menu, select New Terminal to open a command window



##### In this example, you are one folder below the "root" of your repo (you are in HelloWorld) so use " cd .. " to navigate up one folder



##### NOTE: If VS Code doesn't open the Git Bash shell, you can use the dropdown to change it



##### Now, use Git commands to create a commit and push your repo to GitHub

##### 

## Exercises: Math Scripts

**EXERCISE 1**

In this exercise, you will use Visual Studio Code to create and run a script.

PROBLEM: How do you calculate the total due at a restaurant given the food cost, the tax and the tip?

INSTRUCTIONS

Keep working in the js-experiments repo

Create a subfolder named MathScripts and do your work in there.

Create a code file named tips.js.

Now type the following code into the file:

// sample inputs

let foodCost = 79.25;

let tax = 6.54;

let tip = 12.00;

// calculations

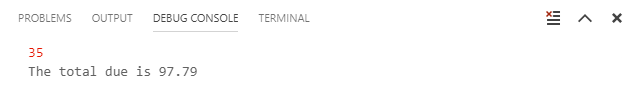
let totalDue = foodCost + tax + tip;

// output

console.log("The total due is " + totalDue);

Save your script.

There are two ways to run your code. You can click the run icon like you did in the last exercise or you can use the Run > Start Debugging menu to run your script. You will see the output in a window at the bottom.



Now change the console.log() statement to the following to build more complicated output.

console.log(

"Food cost is " + foodCost + " and tax is " + tax);

console.log("Tip is " + tip);

console.log("Total Due is " + totalDue);

Save your script changes and then re-run your script.

Do you see how the tip amount doesn't have two digits to the right of the decimal point? Replace the line that displayed the tip amount with the following.

console.log("Tip is " + tip.toFixed(2));

Save your script and re-run it. Does it look better?

**EXERCISE 2**

How do you calculate the area of a circle?

Now, code your script in a file named **area\_of\_circle.js** within the MathScripts folder. We will let you research what code you should write.

Your output should be formatted as follows:

The area of a circle with radius *some-number* is *some-number*

Continue adding these exercises the same repo.

**EXERCISE 3**

How do you convert a Celsius temperature into Farenheit?

Code the script in a file named **c\_to\_f.js**

**(Optional) EXERCISE 4**

How do you find the distance between (x1,y1) and (x2,y2)?

Code the calculate distance in a script named **distance.js.** You may need to read about Math.sqrt()

DON'T FORGET: commit your changes when you finish these exercises.

**Quick Lesson: Parsing Strings   
into Numbers**

* **You can't perform mathematical operations on strings**
* + performs concatenation when the operands are strings

**Example**

let x = "123";

let y = "456";

let z = x + y;

console.log(z); **// displays 123456**

* **JavaScript has functions that can parse a string value and return the numeric equivalent, including**
* parseInt() parses a string and returns an integer
* parseFloat() parses a string and returns a floating point number
* Number() parses a string and returns a number value

###### It doesn't prefer an integer or float conversion

* **You will need these functions quite a bit when you start building web pages and let users enter data in forms**
* Why? Because for fields return strings no matter what type of data the user entered!
* **parseInt() parses a string and returns an integer**
* An integer is a whole number (ex: 7, 1023, or -17)

**Example**

let hrsWorked = "33"; // a String object

hrsWorked = **parseInt(**hrsWorked**)**; // value is 33

* **Leading and trailing spaces are allowed**

**Example**

let hrsWorked = " 33 ";

hrsWorked = **parseInt(**hrsWorked**)**; // value is 33

* **If the first character can't be converted to a number, parseInt() returns NaN**
* You will then have to use an if to detect the issue and respond accordingly

**Example**

let hrsWorked = "Thirty three";

hrsWorked = **parseInt(**hrsWorked**)**; // value is NaN

if (**isNaN(**hrsWorked**)**) {

// tell user about the problem with hrsWorked

}

* **parseFloat() parses a string and returns a floating point number**
* A floating point number can have a decimal point (ex: 7.123, .000505, or -17.2)

**Example**

let payRate = "10.75"; // a String object

payRate = **parseFloat(**payRate**)**; // value is 10.75

* **Leading and trailing spaces are allowed**

**Example**

let payRate = " 10.75 "; // a String object

payRate = **parseFloat(**payRate**)**; // value is 10.75

* **If the first character can't be converted to a number, parseFloat() returns NaN**

**Example**

let a = parseFloat("10.00"); // a is 10

let b = parseFloat(".33"); // b is .33

let c = parseFloat("10.5 years ago"); // c is 10.5

let d = parseFloat("over 10"); // d is NaN

* **Number()converts different object values to their numeric equivalent**
* Strings (containing numbers) are parsed to numeric values
* Booleans are parsed to numbers
* true is 1 and false is 0
* Dates are parsed to a numbers (which is a millisecond value)

**Example**

let a = Number("123"); // a is 123

let b = Number("45.67"); // b is 45.67

let c = Number("40 years"); // c is NaN

let d = Number(true); // d is 1

let e = Number(false); // e is 0

let f = new Date(1990, 5, 20);

let g = Number(f); // g is 1561006800000

**Exercise: Conditionals**

Add this exercise to your js-experiments repo in a folder called Conditionals

**EXERCISE 1**

In a file named pay\_rules.js, create a script to calculate gross pay given the variables payRate and hoursWorked. If the person works more than 40 hours, pay the overtime hours at 1.5 times the rate of regular hours.

When you are finished, review your script with a colleague. Are your algorithms similar? Do you believe each other's code will work?

Run your script several times with different values for payRate and hoursWorked and confirm the output is right.

Examples of good test data might be:

Pay Rate Hours Worked Gross Pay Reason

12.50 20 250.00 Under 40 hours

25.00 40 1000.00 Exactly 40 hours

17.30 45 821.75 Over 40 hours

**Quick Lesson: JavaScript in the Browser**

* **Let's see how to use JavaScript within a browser**
* We will combine HTML/CSS/Bootstrap and JavaScript to create dynamic web sites
* **To include JavaScript in a web page, you must use a <script> element to include it**
* **The <script> element can:**
* contain the script
* point to an external script file (more common)
* **Code in scripts run when the browser's loading process encounters the script**

**Example**

In this example, we would include the JavaScript directly in the page

<html>

<head>

<title>Demo</title>

</head>

<body>

**<div id="messageDiv">**</div>

**<script>**

// TODO: Find the messageDiv element

// TODO: Set the div's contents to a string

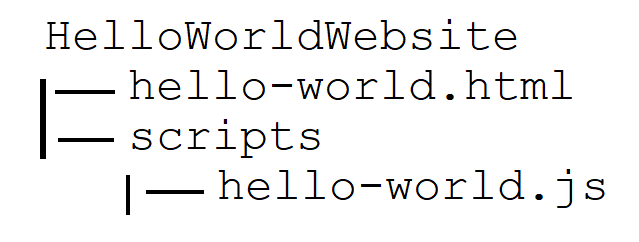
**</script>**

</body>

</html>

* **It is more common to place JavaScript in an external file that is included in the page**

##### Typically, a subfolder named scripts holds the .js files



**Example**

**hello-world.html**

<html>

<head>

<title>Demo</title>

</head>

<body>

<div id="messageDiv"></div>

**<script src="scripts/hello-world.js"></script>**

</body>

</html>

**scripts/hello-world.js**

"use strict";

// A function to show a greeting

**function showGreeting() {**

// TODO: Find the messageDiv element

// TODO: Set the div's contents to a string

**}**

// calls the showGreeting() function when the .js

// file loads and the line below is encountered

**showGreeting();**

* **Now that we can add a script tag, what's next?**
* **To interact with an HTML element using JavaScript, you must have a reference to it**

##### document.getElementById is the most efficient way to get a reference to an HTML element if the element has an id attribute

**Example**

**HTML**

<div id="**messageDiv**"> ... </div>

<input type="text" id="**nameField**" />

<input type="button" id="**clickMeBtn**" value="Click Me" />

**JavaScript**

let **messageDiv** = document.getElementById("**messageDiv**");

let **nameField** = document.getElementById("**nameField**");

let **clickMeBtn** = document.getElementById("**clickMeBtn**");

##### If an element with the specified id doesn't exist on the page, getElementById returns null

* **Let's start by seeing how to update a <p>, <span> or <div>** **element using its innerHTML property**

**Example**

**hello-world.html**

<html>

<head>

<title>Demo</title>

</head>

<body>

<div id="**messageDiv**"></div>

<script src="scripts/hello-world.js"></script>

</body>

</html>

**scripts/hello-world.js**

"use strict";

function showGreeting() {

// Find the messageDiv element

**let messageDiv = document.getElementById("messageDiv");**

// Set its contents to "Hello World!"

**messageDiv.innerHTML = "Hello World!";**

}

// call the showGreeting() function when the .js

// file loads and this line is encountered

**showGreeting();**

**Exercise: Hello World!**

For each of our website exercises, we will create a new repo. In this case, go to GitHub and create a HelloWorldWebsite repo. Then clone it to your local machine.

Take a few minutes and code the exercise above. Follow the steps below:

1. Create a hello-world.html file and add the HTML code just shown to it
2. Create a subfolder named scripts
3. Create a hello-world.js file within scripts and add the JavaScript code just shown to it

To display the page in a browser, right click on the hello-world.html file in the VS Code editor window and choose Run with Live Server

Did it work? Commit and push your changes

**Quick Lesson: Working with <input> Elements**

* **If you want to get or set the value of an <input> element, you must:**
* get a reference to the form field using getElementById
* use the reference to access <input> element's the value property

###### Note: the value property is a string so you may have to convert it to a number to use it in a mathematical equation

**Example**

**HTML snippet**

<p>Name **<input type="text" id="nameField" />**</p>

<p>Age **<input type="text" id="ageField" />**</p>

<p><input type="button" id="showBtn" value="Show" /></p>

<p id="messagePara"></p>

**JavaScript snippet**

**// We want this code to run when the user clicks the Show button**

let nameField = document.getElementById("nameField");

let ageField = document.getElementById("ageField");

**// Get values in the name and age fields**

let name = **nameField.value;**

let age= **ageField.value**; // age is a string here

let message =   
 "Hi " + name + "! I hear you are " + age + " years old!";

const messagePara = document.getElementById("messagePara");

messagePara.innerHTML = message

* **FUN TIP: JavaScript string interpolation can simplify formatting a message**
* Instead of using a long concatenation expression… use backticks!

###### **NOTE:** On my keyboard, the backtick is on the top left key along with ~

* Place backticks around the string and place variables/values you want inserted between **${ *expression-here* }**

**Example**

let message =

**`**Order **${orderNum}**'s status is: **${orderStatus}`;**

**JavaScript snippet**

// We want this code to run when the user clicks the Show button

let nameField = document.getElementById("nameField");

let ageField = document.getElementById("ageField");

let name = nameField.value;

let age = ageField.value; // age is a string here

**let message =   
 `Hi ${name}! I hear you are ${age} years old!`;**

const messagePara = document.getElementById("messagePara");

messagePara.innerHTML = message

* **You can also put a value into the element using value**

**Quick Lesson: Events**

* **In the example we just saw, we wanted the code to run when the user clicked on a button… so we need to learn about events**
* **An event is a notification that something happened to an element on the web page. For example:**
* the user has "clicked" a button
* an input field has lost focus
* the web page has finished loading
* **You can react to events in JavaScript by writing***event handlers*
* An event handler is a JavaScript function that executes when the event occurs
* **HTML elements have attributes that allow you to assign event handlers, including:**
* onclick - runs code when the user clicks an HTML element
* onchange - runs code when an HTML element has been changed
* onfocus - runs code when the HTML element gains focus
* onblur - runs code when the HTML element loses focus
* **A common way to associate an event handler function to an event is to assign it when the window finishes loading**
* The window.onload event handler executes when the web page has completely loaded all content (including images, script files, CSS files, etc.)

**Example**

**index.html**

<html>

<head>

<title>Demo</title>

</head>

<body>

<input **id="helloBtn"** type="button" value="Say Hello" />

<div **id="messageDiv"**></div>

<script src="scripts/index.js"></script>

</body>

</html>

**scripts/index.js**

"use strict";

**// Let the window's onload know there is a function called init**

window.onload = init; // Do not put () after init!

// This isn't calling init now -- it

// saying call the init function when

// the window finishes loading

**// The init function connect the button to the code that**

**// should run when the button is clicked**

function init() {

// Find the helloBtn

const helloBtn = document.getElementById("helloBtn");

// Let helloBtn's onclick know there is a function

// called onHelloBtnClicked that should be called when

// the button is clicked

helloBtn.onclick = onHelloBtnClicked;

}

// This function is called when helloBtn is clicked

function onHelloBtnClicked() {

// Find the messageDiv

let messageDiv = document.getElementById("messageDiv");

// Set the contents of the div

messageDiv.innerHTML = "Hello World!";

}

**Exercise: Click for Hello World**

Create a ClickForHelloWorldWebsite repo. Then clone it to your local machine.

Take a few minutes and code the exercise above. Follow the steps below:

1. Create an index.html file and add the HTML code just shown to it
2. Create a subfolder named scripts
3. Create an index.js file within scripts and add the JavaScript code just shown to it

Right click on the index.html file in the VS Code editor window and choose Run with Live Server

Click the button. Did it work?

EXTEND THE FEATURES

Let's add a second button to the form that clears the message when clicked.

<html>

<head>

<title>Demo</title>

</head>

<body>

<input id="helloBtn" type="button" value="Say Hello" />

**&nbsp; &nbsp;**

**<input id="clearBtn" type="button" value="Clear Message" />**

<div id="messageDiv"></div>

<script src="scripts/index.js"></script>

</body>

</html>

Now connect clearBtn to its event handler.

function init() {

// Find the helloBtn

let helloBtn = document.getElementById("helloBtn");

// Let helloBtn's onclick know there is a function

// called onHelloBtnClicked that should be called when

// the button is clicked

helloBtn.onclick = onHelloBtnClicked;

**// Find the clearBtn**

**let clearBtn = document.getElementById("clearBtn");**

**// Let clearBtn's onclick know there is a function**

**// called onClearBtnClicked that should be called when**

**// the button is clicked**

**clearBtn.onclick = onClearBtnClicked;**

}

Finally, let's write the clearBtn's click event handler.

**// This function is called when clearBtn is clicked**

**function onClearBtnClicked() {**

**// Find the messageDiv**

**let messageDiv = document.getElementById("messageDiv");**

**// Clear the contents of the div**

**messageDiv.innerHTML = "";**

**}**

Right click on the index.html file in the VS Code editor window and choose Run with Live Server

Click the "Say Hello" button. Click the "Clear Message" button. Did it work?

Commit and push your changes

**IMPORTANT NOTES**

#### Because we are handling a button's onclick event in these examples, HTML5 validation attributes like required or maxlength will not work

###### Hint: You must use an HTML form, submit buttons, and handle the form's onsubmit event rather than onclick

#### You may see several ways to interact with HTML elements depending the preference of the person coding.

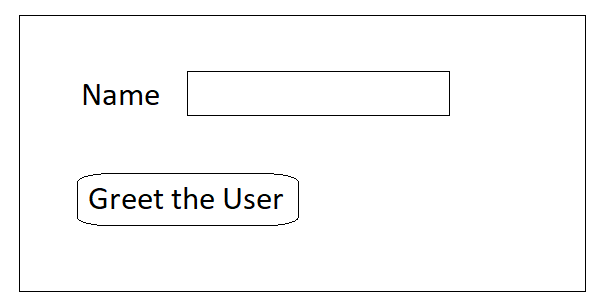
**Exercises: Two Simple Websites**

**EXERCISE 1**

Now that you've typed in code exactly as we’ve shown, let's try writing some simple code by yourselves

Create a repo named GreetByNameWebsite. Then clone it to your local machine.

A web page will allow the user to enter their name. When they click a button, the greeting will be personalized for them.



**Step 1:** Create a web page named index.html. Begin by designing it as shown above. Also add a message div.

<p>Name <input type="text" id="nameField" /></p>

<p><input type="button" id="greetUserBtn"

value="Greet the User" /></p>

<div id="messageDiv"></div>

**Step 2:** Create a scripts subfolder, then add a script named index.js to it.

**Step 3:** In index.js, write code to connect the window.onload event to an event handler named init.

**Step 4:** In the init() function, find the button and connect it to an event handler named onGreetUserBtnClicked

**Step 5:** Code the onGreetUserBtnClicked(). Within it:

* find the name text field using getElementById
* extracts the name from the text field
* use the name to create a string that contains "Hello *userNameHere*"
* displays the message in the message div

**Step 6:** Include index.js in a <script> element at the bottom of the <body> element in index.html

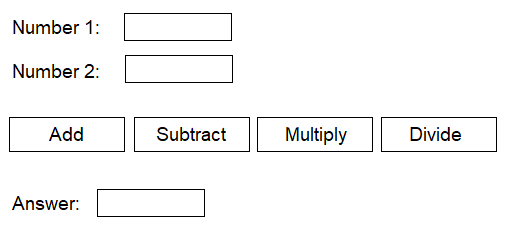
Finally, test your page. What happens when you enter your name in the input field and click the button?

DON'T FORGET: commit your changes when you finish this exercise.

**EXERCISE 2**

Create a repo named CalculatorWebsite. Then clone it to your local machine.

Create an index.html web page that provides a simple calculator to the user. Design the page as shown below. Give your input fields the ids number1Field, number2Field, and answerField. Make answerField read only in HTML.



Create an index.js file and include it in the page. Include all of your JavaScript there.

Connect the window.onload event to an init function.

In the init event handler, find each button and assign its onclick to an event handler. Name the event handlers something like onAddBtnClicked, onSubtractBtnClicked, etc.

Within the click event handler for the Add button:

* extract the value from the number 1 input field and convert it to a number using Number()

let number1Field =

document.getElementById("number1Field")

let number1 = Number(number1Field.value);

* repeats the process for number 2
* add the two numbers together and save it in a variable
* display the results in the answer field

Test your add feature.

Once you get the Add button working, use it as a pattern for the other three buttons.

Commit your changes and push.

**(Optional) Mini-Project**

ONLY ATTEMPT THIS PROJECT IF YOU HAVE EXTRA TIME ON MONDAY. DO NOT WORK ON IT ON TUESDAY MORNING.

Create a new GitHub repo named TemperatureConverterWebsite. Then clone it to your local machine.

In this exercise, you will create three web pages:

* a home page
* a page with a calculator that converts Fahrenheit temperatures to Celsius
* a page with a calculator that converts Celsius temperatures to Fahrenheit

Configure the site structure by:

* adding a nav bar to all three pages that will allow navigation between them
* adding two paragraphs of lipsum text from https://www.lipsum.com to your home page

On the each calculator page, create an HTML form with a place for an input temperature, a convert button, and a place to put the output temperature.

Use the window.onload event on each page to connect the button to an event handler with a good name (ex: onConvertBtnClicked). Then, code the button behavior on each form.

Test your pages. Make sure you see an error message if the input temperature isn't a number.

Now add a reset button to each page. Test to make sure it clears your form

DON'T FORGET TO commit and push.

# DAY 2

**Quick Lesson: Working with Strings (Review)**

* **Do you remember the built-in String methods?**

##### One that is very useful is substring

* **Sometimes, you know exactly which part of a string you need to find**
* US Social Security numbers (tax ID numbers) have the format xxx-xx-xxxx where each x is a digit

**Example**

let ssn = "111-22-3333";

let first3 = ssn.substring(0, 3);

let second2 = ssn.substring(4, 6);

let last4 = ssn.substring(7, 11);

let ssnWithoutDashes = first3 + second2 + last4;

* **When you don't know where the substrings begin and end, you must look for delimiters**

**Example**

let partCode = "27543**-**large";

let partNum, size;

**let dashPos = partCode.indexOf("-");**

partNum = partCode.substring(0, **dashPos**); // 27543

size = partCode.substring(**dashPos + 1**); // large

**Exercise: Working with Strings**

Add this exercise to your js-experiments repo in a folder called Strings. We are back to practicing JavaScript syntax for a bit and will run this exercise using Code Runner

**EXERCISE 1**

Create a new script file and name it parsing.js. Within it, declare a variable that contains a first and last name:

let name = "Brenda Kaye";

Write code to find the space using indexOf() and then extract and display the first and last name. Example output might be:

Name: Brenda Kaye

First name: Brenda

Last name: Kaye

Once it works, move your code into a function called parseAndDisplayName() and call it using the following:

parseAndDisplayName("Brenda Kaye");

parseAndDisplayName("Ian Auston");

parseAndDisplayName("Siddalee Grace");

Commit and push.

**Quick Lesson: Anonymous Functions**

* **JavaScript supports a type of function called an anonymous function**
* It is called that because it isn't given a name
* **A popular use of the anonymous function to assign event handling behavior to window.onload**

**Example**

**HTML**

<html>

<head>

<title>Anonymous Functions</title>

</head>

<body>

<input **id="helloBtn"** type="button" value="Say Hello" />

<script src="scripts/index.js"></script>

</body>

</html>

**JavaScript**

"use strict";

// Do you see that we didn't write the init function. Instead,

// we assigned an **anonymous function** to the window's onload.

// It looks just like a named function but is missing the

// name between the word function and the ()

**window.onload = function() {**

**const btn = document.getElementById("helloBtn");**

**btn.onclick = sayHello;**

**};**

function sayHello() {

alert("Hello!"); // low-tech popup message

}

**Quick Lesson: Working with Checkboxes**

* **Do you remember from HTML that we create checkboxes using <input type="checkbox"> ?**

**Example**

In this rental car example, you can choose 0 to 3 options: 1) enable a toll tag to automatically handle tolls for any toll roads you might drive on, 2) enable GPS in the car, and 3) sign up for roadside assist in case you have car problems

<fieldset>

<legend>Rental Car Options:<legend>

**<input type="checkbox"** id="tollTagChkbox" checked>

Electronic Toll Tag ($3.95/day plus tolls)<br>

**<input type="checkbox"** id="gpsChkbox">

GPS ($4.95/day)<br>

**<input type="checkbox"** id="roadsideChkbox>

Roadside Assistance ($2.95/day)<br>

</fieldset>

* **In JavaScript, you can determine whether a checkboxes is checked using the Boolean checked property**

**Example**

let extraPerDay = 0;

let tollTagChkbox = document.getElementById("tollTagChkbox")**;**

**if (tollTagChkbox.checked)** {

extraPerDay += 3.95;

}

let gpsChkbox = document.getElementById("gpsChkbox")**;**

**if (gpsChkbox.checked)** {

extraPerDay += 4.95;

}

let roadsideChkbox = document.getElementById("roadsideChkbox")**;**

**if (roadsideChkbox.checked)** {

extraPerDay += 2.95;

}

* **A checkbox supports the onclick event**

##### When a checkbox is clicked, you have to use an if statement to see if it is being checked or unchecked

**Example**

In this example, when a user checks the checkbox that says they are a government employee, the roadside assistance checkbox is automatically checked

**HTML**

**<input type="checkbox"** id="govtChkbox">

Are you a US Government employee?<br>

**JavaScript**

window.onload = function() {

// Set the govtChkbox's clicked event handler to the

// onGovtChkboxClicked function

**let govtChkbox = document.getElementByid("govtChkbox");**

**govtChkbox.onclick = onGovtChkboxClicked;**

// other events handlers connected here

}

function onGovtChkboxClicked() {

let govtChkbox = document.getElementById("govtChkbox")**;**

if (govtChkbox.checked) {

// check the roadside assist checkbox

**let roadsideChkBox =**

**document.getElementById("roadsideChkbox");**

**roadsideChkBox.checked = true;**

}

}

**Quick Lesson: Working with Radio Buttons**

* **Radio buttons are similar to checkboxes, except that only one in a group may be checked**
* They must have the name property set in HTML in order to work correctly

**Example**

In this example, we pre-check the auto insurance policy type so we are always sure there will be one radio button selected.

<fieldset>

<legend>Select the insurance policy type:<legend>

<input type="radio" **name="policy" id="autoRadioBtn"**

value="auto" checked> Auto<br>

<input type="radio" **name="policy" id="homeRadioBtn"**

value="home"> Home<br>

<input type="radio" **name="policy" id="lifeRadioBtn"**

value="life"> Life<br>

</fieldset>

* **Radio buttons also have a Boolean checked property**

**Example**

let autoRadioBtn = **document.getElementById("autoRadioBtn");**

let homeRadioBtn = **document.getElementById("homeRadioBtn");**

let basePremium = 0;

if (autoRadioBtn.checked) {

basePremium = 175.00;

}

else if (homeRadioBtn.checked) {

basePremium = 395.00;

}

else { // it must be life since one is always checked!

basePremium = 225.00;

}

* **Radio buttons also support the onclick event**
* Unlike checkboxes, if a radio button's click event is triggered, it is definitely clicked!

**Quick Lesson: Using querySelector()**

* **We know that you can find an HTML element on a page using getElementById()**
* **You can also find an HTML element using a CSS selector combined with the function querySelector()**

**Example**

querySelector() will select the first input elements with a name attribute of policy that is checked.

**HTML**

<fieldset>

<legend>Select the policy type:<legend>

<input type="radio" **name="policy"** id="autoRadioBtn"

**value="auto"** checked> Auto<br>

<input type="radio" **name="policy"** id="homeRadioBtn"

**value="home"**> Home<br>

<input type="radio" **name="policy"** id="lifeRadioBtn"

**value="life"**> Life<br>

</fieldset>

**JavaScript**

let selectedOption =

document.**querySelector("input[name='policy']:checked");**

let basePremium = 0;

if (**selectedOption.value** == "auto") {

basePremium = 175.00;

}

else if (**selectedOption.value** == "home") {

basePremium = 395.00;

}

else {

basePremium = 225.00;

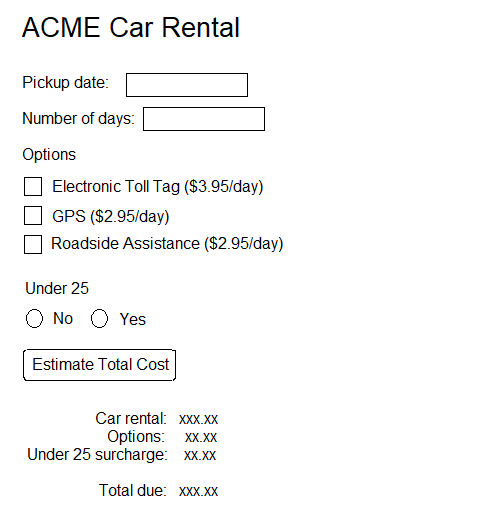
}

**Exercises: Two Simple Websites**

**EXERCISE 1**

Create a GitHub repo named CarRental. Clone it to your local machine.

Create index.html and design it similar to what you see below.



Write code in the window.onload event handler to hook up a click event handler for the button. Rules for determining the rental charges are outlined below:

basic car rental is $29.99 per day

calculate options cost by examining the checkboxes

there is a 30% surcharge on the basic car rental for drivers under 25

(there is no surcharge on options)

all taxes have already been incorporated into the fees shown

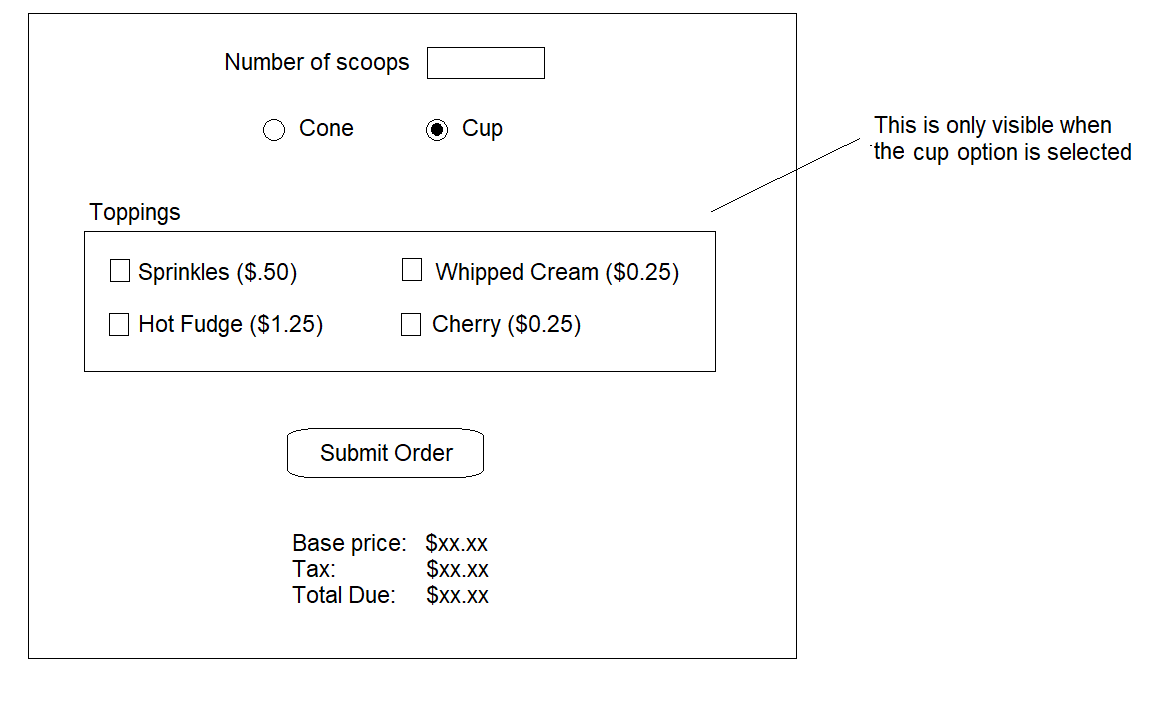
DON'T FORGET TO commit and push!

**(Optional) EXERCISE 2**

Create a GitHub repo named IceCream. Clone it to your local machine folder.

Let's build a web site that allows a user to order an ice cream dessert. Clearly, the site won't submit an order to a worker that will make the ice cream dessert, but it will price out the order.

Your web site should resemble the diagram below, however the behavior of the page is much more important than its appearance.



An ice cream cone or cup with one scoop costs $2.25. Each additional scoop of ice cream is $1.25.

Only cups can have toppings. This means you will need to hide or show the toppings section when the radio buttons for cone and cup are clicked.

Use HTML validation attributes so that the number of scoops is required and must be between 1 and 4.

DON'T FORGET TO commit and push!

**FINISHED EARLY??**

If you finish early, go back and do the mini-project from yesterday

# DAY 3

**Quick Lesson: JavaScript Object Literals**

* **JavaScript allows you to declare objects literals with values for each property**
* Properties are written as name and value pairs that are separated by a colon

**Example**

let emp = **{**

**employeeId:** "1",

**name:** "Ezra",

**jobTitle:** "Theater Teacher",

**payRate:** 38.46

**}**;

* Spacing, indentation, and line breaks are a matter of preference
* **You can access object properties using:**

**objectName.propertyName**

**Example**

**let emp1** = {

employeeId: "1",

name: "Ezra",

jobTitle: "Theater Teacher",

payRate: 38.46

};

**let emp2** = {

employeeId: "2",

name: "Elisha",

jobTitle: "Game Programmer",

payRate: 43.27

};

console.log("Employee 1: " + **emp1.name**);

console.log("Employee 2: " + emp2.name);

**Exercise: Object Literals**

Add this exercise to your js-experiments repo under a folder named ObjectScripts. The exercise lets you practice some JavaScript and will be run using Code Runner.

**EXERCISE 1**

Create a script that named label\_maker.js. In it, define a JavaScript object literal with the following properties with sample values of your choice:

name

address

city

state

zip

Then pass the object to a function named printContact().

You can invoke it using:

let myInfo = {

name: "Pursalane Faye",

address: "121 Cardinal Street",

/\* other properties not shown \*/

};

printContact(myInfo);

Inside the printContact() function, call console.log() to print each property formatted like a mailing label.

For example:

Pursalane Faye

121 Cardinal Street

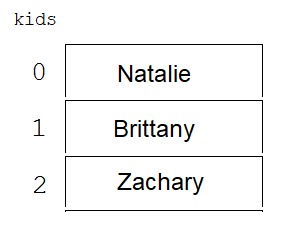
Benbrook, Texas 76126

**Quick Lesson: Arrays**

* **A JavaScript array is used to store multiple values in a single variable**

**Example**

let kids = **[**"Natalie", "Brittany", "Zachary"**]**;



* **To access an element in an array, you use a subscript representing the item's position in the array**
* Subscripts in JavaScript are **0-based**

**Example**

let kids = ["Natalie", "Brittany", "Zachary"];

let youngest = kids**[2];**

**Example**

let kids = ["Natalie", "Brittany", "Zachary"];

let numKids = kids.length;

for(let i = 0; i < numKids; i++) {

console.log(kids[i]);

}

* **JavaScript arrays can store any type of data**

**Example**

**// an array that stores all numbers**

let mileAgeLog = [313, 328, 349, 287, 301];

**// an array that stores objects**

let menu = [

{item: "Hamburger", price: 6.95},

{item: "Cheeseburger", price: 7.95},

{item: "Hot dog", price: 4.95}

];

* **When you loop through an array of objects, you must use the subscript after the array name and then the property name after the subscript**

**Example**

let myOrder = [

{item: "Chicken Tacos", price: 8.95},

{item: "Guacamole", price: 2.85},

{item: "Sweet Tea", price: 2.75}

];

let yourOrder = [

{item: "Chicken Tandoor", price: 16.95},

{item: "Chai Tea", price: 2.75},

{item: "Apple Pie", price: 4.95}

];

function getMealCost(orders) {

let sum = 0;

let numOrders = orders.length;

for(let i = 0; i < numOrders; i++) {

sum += **orders[i].price**;

}

return sum;

}

**let mealCost = getMealCost(myOrder);**

let totalWithTip = mealCost \* 1.2;

console.log("My meal costs " + totalWithTip.toFixed(2));

**mealCost = getMealCost(yourOrder);**

totalWithTip = mealCost \* 1.2;

console.log("Your meal costs " + totalWithTip.toFixed(2));

**Exercises: Working with Arrays**

Add these exercises to your js-experiments repo under a subfolder named Arrays.

**EXERCISE 1**

Write a script named avg\_scores.js that declares two arrays of exam scores.

let myScores = [92, 98, 84, 76, 89, 99, 100];

let yourScores = [82, 98, 94, 88, 92, 100, 100];

Now, create a function named getAverage()to find the average score in that array. (To find an average, loop through and add up all the numbers in the array and then divide by the length of the array) Return the average.

Call your getAverage() function and pass it myScores. Catch the return value and display it as my average. Repeat with your scores.

**(Optional) EXERCISE 2**

Write a script named foods.js that declares an array that contains objects you ordered the last time you ate out. For example,

let lunch = [

{item: "Chicken Fajitas", price: 19.95},

{item: "Chips and Guacamole", price: 5.25},

{item: "Sweet Tea", price: 2.79}

];

Write code to loop through the array and add up the price of everything you ate and print it out as a subtotal.

Also display the tax on that total (assume 8%), the tip on that total (assume 18%), and the total due.

**Quick Lesson: Searching an Array**

* **JavaScript has the indexOf() function to search an array for an element and return its position**
* It returns -1 if the item is not found

**Example**

This code searches the list from the beginning

let teams = ["Red Sox", "Rangers", "Blue Jays",

"Astros", "White Sox", "Rangers"];

let index = **teams.indexOf("Rangers")**; **// returns 1**

if (index == -1)

console.log("Item not found");

else

console.log("Item at position: " + index);

* **ES6 introduced the find() method for arrays that searches the array for the first value that matches a specified condition**
* **To specify the condition, you pass find() a function object**
* find() iterates over the array and calls that function for each element in the array
* The function determines whether the array element passed "matches" the search condition
* If the function returns true, find() stops and returns that element
* If the function returns false, find() continues to iterate over the array
* If the function never finds a value that matches the condition, find() returns [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined)

**Example**

**// This is the function that will be passed in ONE element in**

**// the array and returns true if it is the element you are**

**// searching for (any value over 60 in this case)**

**function isOver60(**arrayValue**)** **{**

**if (arrayValue > 60)** {

return true;

}

else {

return false;

}

**}**

// Here is array of numbers

let numbers = [7, 9, **64**, 60, 12, 13, 65, 62];

**// The find() function here calls the isOver60 function**

**// over and over with each value from the array. If the**

**// function returns true, its stops and returns the value**

**// from the array where isOver60 returned true**

let firstValOver60 = numbers.**find(**isOver60**)**; // returns 64

**// If no value in the array was found, firstValOver60 will**

**// be undefined**

if (firstValOver60 != undefined) {

console.log(firstValOver60);

}

else {

console.log("No values over 60");

}

* **ES6 also contains a findIndex() method that behaves similar to find() except that it returns the index of the first element that passes the test function**
* If no values match, [-1](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) is returned

**Example**

**function isOver60(**arrayValue**)** **{**

if (arrayValue > 60) {

return true;

}

else {

return false;

}

**}**

let numbers = [7, 9, **64,** 60, 12, 13, 65, 62];

let firstValOver60Position = numbers.**findIndex(**isOver60**)**;

// returns the index 2

if (firstValOver60Position != -1) {

console.log("First value over 60 found at position " +

firstValOver60Position);

}

else {

console.log("No values over 60");

}

* **ES6 also introduced the filter() method for arrays that returns an array of all values that pass a test function**
* If no values match, an empty array (length == 0) is returned

**Example**

**function isOver60(**arrayValue**)** **{**

**if (arrayValue > 60)** {

return true;

}

else {

return false;

}

**}**

let numbers = [7, 9, **64**, 60, 12, 13, **67, 66**];

let allOver60 = numbers.**filter(**isOver60**)**;

// returns an array containing [64, 67, 66]

if (allOver60.length > 0) {

console.log(allOver60); // displays the array

}

else {

console.log("No values over 60");

}

**Exercises: Searching Arrays**

Add these exercises to your js-experiments repo in the same Arrays subfolder as the last exercises.

**EXERCISE 1**

Create a script named es6\_course\_search.js. Add a courses array to it that resembles the following. (You should be able to copy the text from this PDF.)

let courses = [

    {

        CourseId: "PROG100",

        Title: "Introduction to HTML/CSS/Git",

        Location: "Classroom 7",

        StartDate: "09/08/24",

        Fee: "100.00",

    },

    {

        CourseId: "PROG200",

        Title: "Introduction to JavaScript",

        Location: "Classroom 9",

        StartDate: "11/22/24",

        Fee: "350.00",

    },

    {

        CourseId: "PROG300",

        Title: "Introduction to Java",

        Location: "Classroom 1",

        StartDate: "01/09/24",

        Fee: "50.00",

    },

    {

        CourseId: "PROG400",

        Title: "Introduction to SQL and Databases",

        Location: "Classroom 7",

        StartDate: "03/16/24",

        Fee: "50.00",

    },

    {

        CourseId: "PROJ500",

        Title: "Introduction to Angular",

        Location: "Classroom 1",

        StartDate: "04/25/24",

        Fee: "50.00",

    }

];

Write code that searches the courses array and display the results using the find() or filter() functions to determine:

//  When does the PROG200 course start?

//  What is the title of the PROJ500 course?

//  What are the titles of the courses that cost $50 or less?

//  What  classes meet in "Classroom 1"?

**(Optional) EXERCISE 2**

Create a script named actors.js that defines an array called famousActors. It should contain the following items:

let famousActors = [

    {

memID: 101,

name: "Bob Brown",

films: ["Bob I", "Bob II", "Bob III", "Bob IV"]

},

    {

memID: 142,

name: "Sallie Smith",

films: ["A Good Day", "A Better Day"]

},

    {

memID: 187,

name: "Fred Flanders",

films: ["Who is Fred?", "Where is Fred?",

"What is Fred?", "Why Fred?"]

 },

    {

memID: 203,

name: "Bobbie Boots",

films: ["Walking Boots", "Hiking Boots",

"Cowboy Boots"]

},

];

Write code that searches the array to find:

// Who is the actor whose ID is 187?

// Who has have been in at least 3 films?

// Who has a name that starts with "Bob"?

// HARDER: Which actors have been in a film

// that starts with "A"

DON'T FORGET TO commit and push your repo.

**Quick Lesson: Working with   
<select> Lists**

* **The <select> element in HTML creates a list**
* It is a dropdown list if a size attribute isn't specified

**Example**

<select id="statesList" name="states">

<option value="CO">Colorado</option>

<option value="ME">Maine</option>

<option value="TX">Texas</option>

<option value="WA">Washington</option>

</select>

* **You can load a <select> from an array when the window's onload event fires**

##### Once you create the option element in JavaScript, you use appendChild()to add it "inside of" the <select> element

**Example**

**HTML**

<select id="statesList">

</select>

**JavaScript**

window.onload = function() {

**// load the dropdown list (see function below)**

**initStatesDropdown();**

// other stuff

...

};

function initStatesDropdown() {

**// load the dropdown list**

let states = ["Colorado", "Maine", "Texas", "Washington"];

let abbrev = ["CO", "ME", "TX", "WA"];

**const statesList = document.getElementById("statesList");**

let length = states.length;

for (let i = 0; i < length; i++) {

// create the option element and set the text and

// value at the same time

**let theOption = new Option(states[i], abbrev[i]);**

// append the option as a child of (inside) the

// select element

**statesList.appendChild(theOption);**

}

}

**Exercise: Loading a Dropdown**

Create a GitHub repo named AmericanFootballWebsite. Clone it to your local machine.

**EXERCISE 1**

Create an index.html file that contains one <select> element. The <select> dropdown will contain the American Football teams found in the array below.

let teams = [

{ teamcode:"DAL",

name:"Dallas Cowboys",

locatedIn:"Arlington, TX"

},

{ teamcode:"DEN",

name:"Denver Broncos",

locatedIn:"Denver, CO"

},

{ teamcode:"HOU",

name:"Houston Texans",

locatedIn:"Houston, TX"

},

{ teamcode:"KAN",

name:"Kansas City Chiefs",

locatedIn:"Kansas City, MO"

}

];

Take a few minutes to write the HTML for index.html.

Now let's code index.js. Define the array at the top just below "use strict".

Write code to handle the window's onload event. Within it, call a function to initialize the football select element.

In the initialize function, loop through the array and load the football teams into the select. Use the team's name for the option text and the team's code for the option value. For example:

text: Dallas Cowboys value: DAL

Test the page. When you pull down the dropdown, are the teams there?

If so, commit and continue.

We need to learn how to determine which option the user selects in a dropdown before we can do anything else interesting!

**Quick Lesson: Determining Which  
Option is Selected**

#### To determine which option is selected, use the value property of the selected option

* It returns null if nothing is selected in the dropdown

**Example**

We are going back to the example that had a dropdown of state names with the values being state abbreviations.

**HTML**  
<select id="statesList" name="states">

<option value="CO">Colorado</option>

<option value="ME">Maine</option>

<option value="TX">Texas</option>

<option value="WA">Washington</option>

</select>

**JavaScript**

const statesList = document.getElementById("statesList");

let selectedValue = statesList.**value**;

if (**selectedValue** == null) {

alert("No state was selected"); // ugly popup!

return; // exit the event handler

}

// otherwise, selectedValue would be TX if Texas was selected

* **You can also use the selectedIndex property to determine the index number of the selected option**
* It is -1 if nothing is selected

**Example**

const statesList = document.getElementById("statesList");

if (**statesList.selectedIndex** >= 0) {

alert("You selected # " + **statesList.selectedIndex**);

}

* **You can programmatically select an option in the dropdown by setting the value property to the one you want selected**

**Example**

**HTML**  
<select id="statesList" name="states">

<option value="CO">Colorado</option>

<option value="ME">Maine</option>

<option value="TX">Texas</option>

<option value="WA">Washington</option>

</select>

**JavaScript**

window.onload = function() {

const statesList = document.getElementById("statesList");

**statesList.value = "TX"; // selects Texas**

}

**Quick Lesson: onchange Event**

* **The onchange event occurs when the value of a <select> element has been changed (aka, they have picked a different option!)**
* You can handle it to provide immediate action upon the change

**Example**

**window.onload = function() {**

**// When the user selects an option in the dropdown, the**

**// onStatesSelectionChanged event will be called**

const statesList = document.getElementById("statesList");

statesList.onchange = onStatesSelectionChanged;

// other things

...

**};**

**function onStatesSelectionChanged() {**

**// find the value of the option selected**

const statesList = document.getElementById("statesList");

let selectedValue = statesList.**value**;

**// now do something with selectedValue**

**}**

**Exercises: Working with Dropdowns**

**EXERCISE 1**

Continue working in the Football project.

Add a button that the use should click AFTER they select an option in the list. Write code to handle the button's click event. In the click event handler, determine which team the user selected and place a message in a paragraph. For example, if they picked Dallas Cowboys the message could say:

You selected the Dallas Cowboys (DAL) who play in Arlington, TX

Test your page. If it works, commit and push.

**EXERCISE 2**

Continue working in the Football project.

Write code to add an option programmatically to the team dropdown with the text "Select a team" and the value "" BEFORE you load the actual teams.

Before you code new logic, test to make sure the option appears before the teams in the dropdown

You will need to write some new code in the button's click event handler. If the user selects "Select a team", clear the message area.

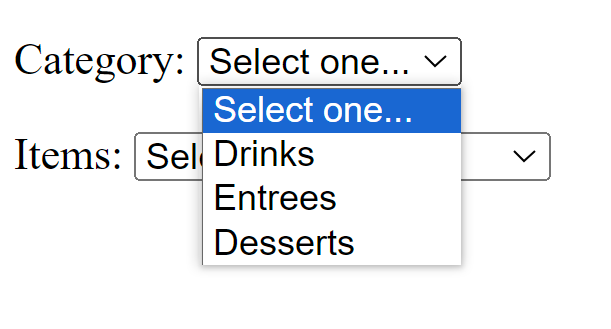
You will need to make some changes to your event handler -- if the user selects the "Denver Broncos" and clicks the button, then goes back and selects "Select a team", you will need to clear the text in the paragraph

Test your page. If it works, commit and push.

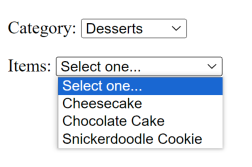
**Exercise: Cascading Dropdowns**

Create a GitHub repo named FoodMenu. Clone it to your local machine.

Design a web page that allows a user to select a category of menu items from a dropdown. Once a category is selected, another dropdown displays items in that category.



Once Desserts is chosen, you see this:



Here is an example of the data structure you might use:

let menu = {

   drinks : [

"Water", "Tea", "Sweet Tea",

"Coke", "Dr. Pepper", "Sprite"

],

   entrees : [

      "Hamburger w/ Fries",

"Grilled Cheese w/ Tater Tots",

      "Grilled Chicken w/ Veggies",

"Chicken Fried Steak w/ Mashed Potatoes",

      "Fried Shrimp w/ Coleslaw",

"Veggie Plate"

   ],

   desserts: [

"Cheesecake", "Chocolate Cake", "Snickerdoodle Cookie"

]

};

NOTE: There will be NO button in this example. The second dropdown is updated with the items that match the selected category when the change event on the category dropdown occurs.

Write JavaScript to handle the change event for the category dropdown list. In the event handler, clear any items in the items dropdown and then re-load it with the matching items from the selected category.

Test your page. Don't forget to commit and push

# DAY 4

**Exercise: REST APIs and jsonplaceholder**

* [**https://jsonplaceholder.typicode.com/**](https://jsonplaceholder.typicode.com/) **is a REST API that is often used as a learning tool**
* **It comes with a set of resources:**
* /posts 100 posts
* /users 10 users
* /comments 500 comments
* /albums 100 albums
* /photos 5000 photos
* /todos 200 todos
* **You can send an HTTP GET request such as:**

https://jsonplaceholder.typicode.com/todos/1

* **It will return a JSON object**

Example

{

"userId": 1,

"id": 1,

"title": "delectus aut autem",

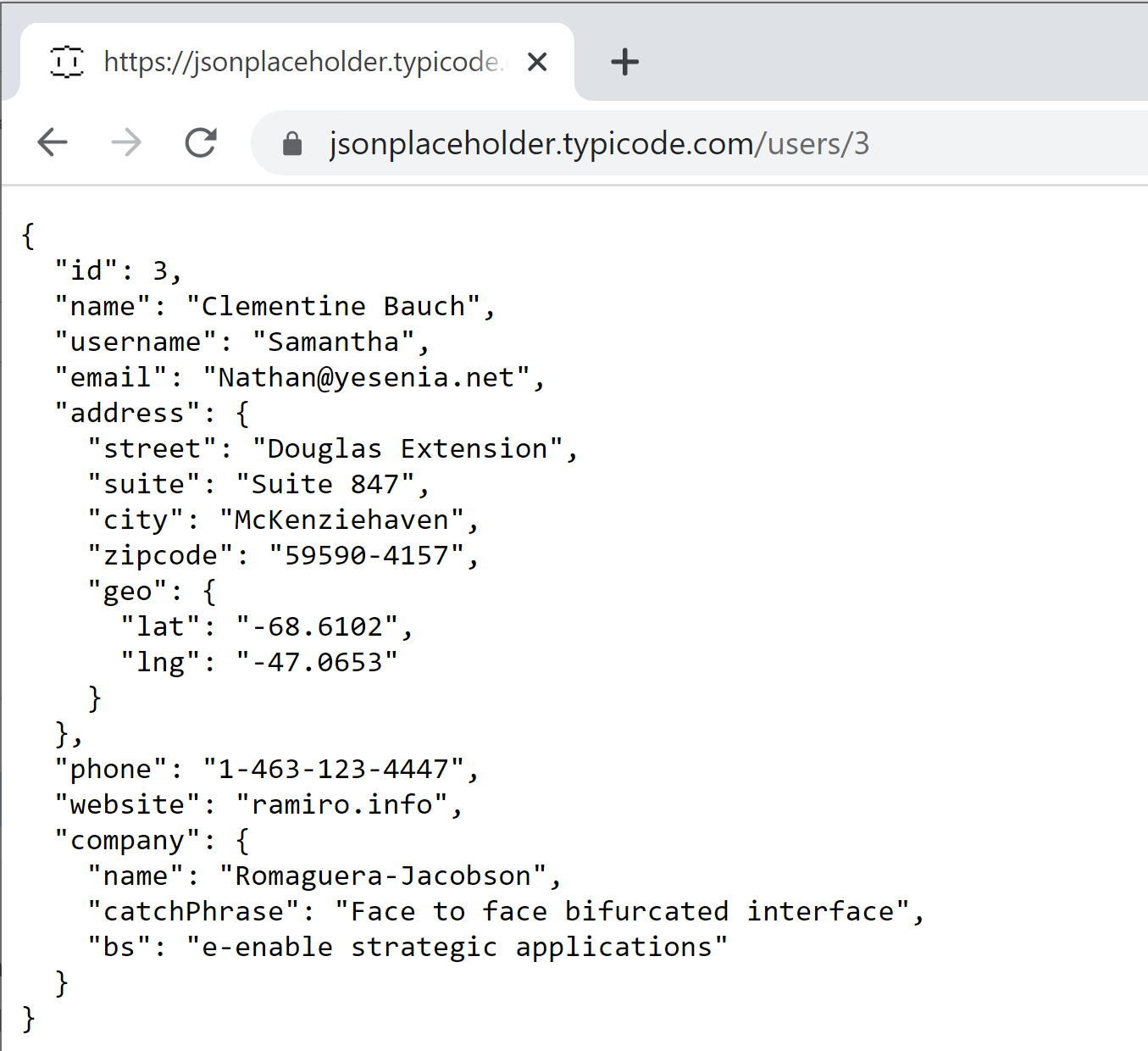
"completed": false

}

#### Check the browser URL…



#### Check the browser URL…



**EXERCISE 1**

Take a few minutes to look at the website <https://jsonplaceholder.typicode.com/>

Test out a few URLs in your browser to see if you understand what they are requesting and what resource(s) it returns.

Specifically, test out the following and try to understand the response:  
  
https://jsonplaceholder.typicode.com/todos/1

https://jsonplaceholder.typicode.com/todos

https://jsonplaceholder.typicode.com/albums

https://jsonplaceholder.typicode.com/users/2

https://jsonplaceholder.typicode.com/comments/3

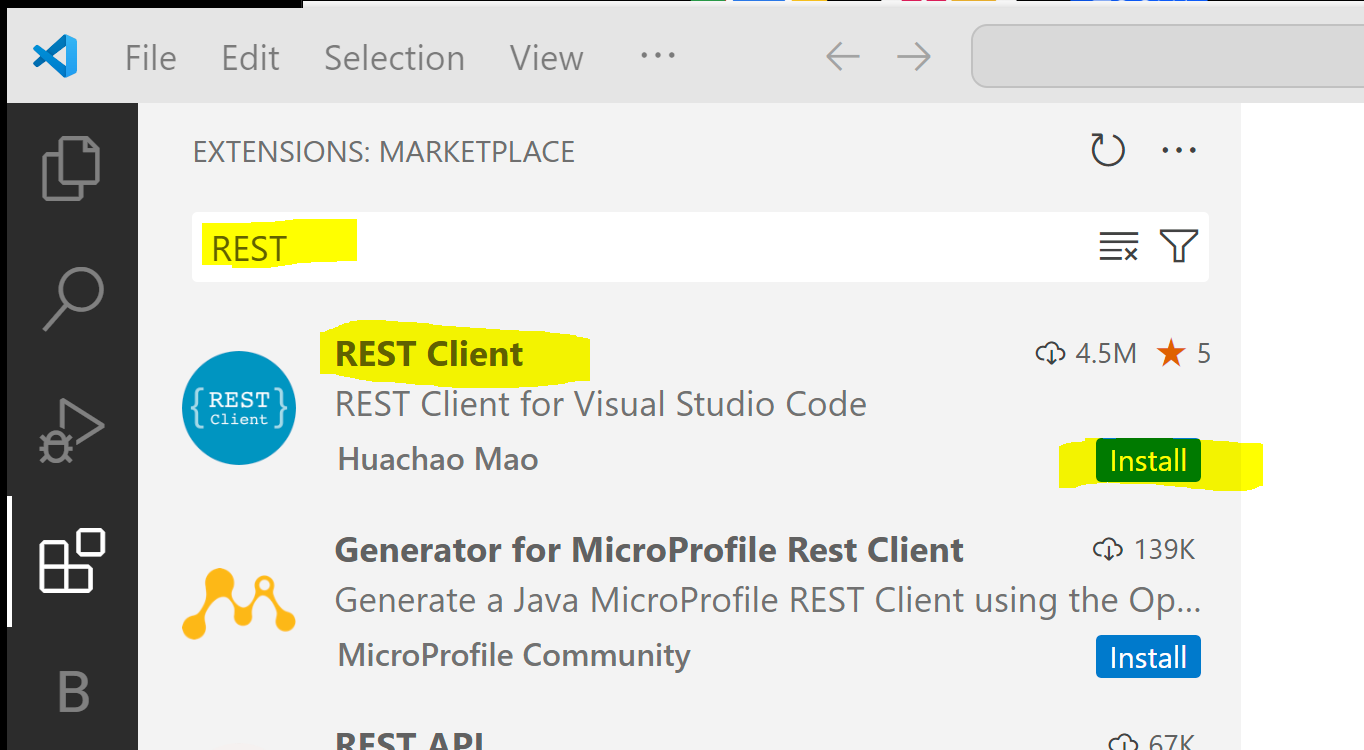
Test out a few others on your own to make sure you understand this API.

NOTE: Remember that the browser is sending an HTTP GET request, which is why the REST API is returning data.

### Quick Lesson: Using the VS Code REST Client Extension

#### Browsers can only be used to send simple GET requests to REST APIs

#### The VS Code Rest Client extension gives you much more flexibility when exploring REST APIs



#### To learn how to use it, start by reading: <https://www.makeuseof.com/make-rest-api-calls-from-vscode/>

##### Then Google more on the web if you need additional examples

#### NOTE: Don't get distracted now by going to read it. We will save that until Friday

### Quick Lesson: Fetch API GET Request to jsonplaceholder

#### We want to make sure you can equate a GET request in the browser to a similar GET request using the Fetch API

#### Do the following examples make sense? Talk it over with a classmate!

**Example**

**// fetches user 1 and displays their email in a div**

let element = document.querySelector("#messageArea");  
**fetch(**"http://jsonplaceholder.typicode.com/users/1"**)**  
 **.then(**response **=>** response.json()**)**  
 **.then(**data **=>** {

let message = "User email: " + data.email;

element.innerHTML = message

}**)**;

**Example**

**// fetches all users and places their names and emails**

**// in a table**

let table = document.getElementById("userTable");  
**fetch(**"http://jsonplaceholder.typicode.com/users"**)**  
 **.then(**response **=>** response.json()**)**  
 **.then(**data **=>** {

for(let i=0; i<data.length; i++) {

let row = table.insertRow(-1);

let cell1 = row.insertCell(0);

let cell2 = row.insertCell(1);

cell1.innerHTML = data[i].name;

cell2.innerHTML = data[i].email;

}

}**)**;

### Exercises: Fetch API and GET Requests

Create a GitHub repo named jsonplaceholderClient and then clone it to your local machine.

**EXERCISE 1**

You will be creating a page that displays TODO data returned from https://jsonplaceholder.typicode.com/

REMINDER: If you send an HTTP GET request to the URL:

https://jsonplaceholder.typicode.com/todos/1

It will return a JSON object that resembles:

{

"userId": 1,

"id": 1,

"title": "delectus aut autem",

"completed": false

}

If want TODO #2, send a GET request to:

https://jsonplaceholder.typicode.com/todos/2

Design a TODOs page that contains an input box where the user can enter an id and a button that they can then click.

Call the REST API above and modify the number at the URL to match the id the user entered. Display the results in a div below the button.

Commit and push your changes.

**EXERCISE 2**

Add a Users page to the project above. In it, you will make an HTTP GET call to the URL below:

https://jsonplaceholder.typicode.com/users

It returns an array of user objects. Before you write any code, use your browser to send the GET request to jsonplaceholder so you understand the data returned.

Pick 6 fields to display in an HTML table of each returned user. If you examine the Quick Lesson just before this exercise again, you may see code that gives you a fast start!

### Quick Lesson: Node.js and Web Applications

#### Node.js is a open-source runtime that allows you to run JavaScript code outside of a browser

##### We used it early this week to run the JavaScript syntax labs using Code Runner

#### Node.js supports frameworks like Express that make building web applications easy

##### Pre-built ***packages*** do many of the common tasks for you

* **NPM is the official package manager for Node.js**
* It manages one of the largest open-source libraries in the world

#### Node.js projects have a packages.json file that lists the packages that NPM installed into the project

##### The packages themselves are stored in a node\_modules folder

#### The node\_modules folder is typically huge

##### It could easily require > 3 MB of disk space

##### Because of this, the node\_modules folder is not pushed to a remote GitHub repository

#### When you clone a Node.js repo, you reinstall the packages locally using NPM

#### In this lesson, we will download and configure a Node.js web application that will contain both a REST API and a set of web pages that interact with it

##### In this front-end course you will NOT build the REST API, however, in the Back-end Academy, you will spend time studying Node.js in detail

### Code-Along: Cloning and Configuring a Node.js REST API

#### In this code-along demo, we will clone a repository that contains a Node.js REST API and configure it to run locally

##### The REST API within it manages course data

#### Let's begin by navigating to the parent folder of where you want the repository to live and enter the following Git command

**Example**

**git clone** *url*

Where the URL is:  
<https://github.com/DevelopIntelligenceBoulder/courses2-api-express-server>

#### Now, change directories into this project

**Example**

cd courses2-api-express-server

#### You must use NPM to re-install the packages by running the command below

##### Take a quick glance in the folder. Do you see there is no node\_modules folder?

**Example**

npm install

##### It determines what packages to install by reading the packages.json file and then downloads and installs them

##### After running the command, do you see the node\_modules folder?

* NOTE: Files shown at the root of CoursesWebSite are not shown in this diagram

#### To start the local server, use the start command

**Example**

**Command to start the server**

npm start

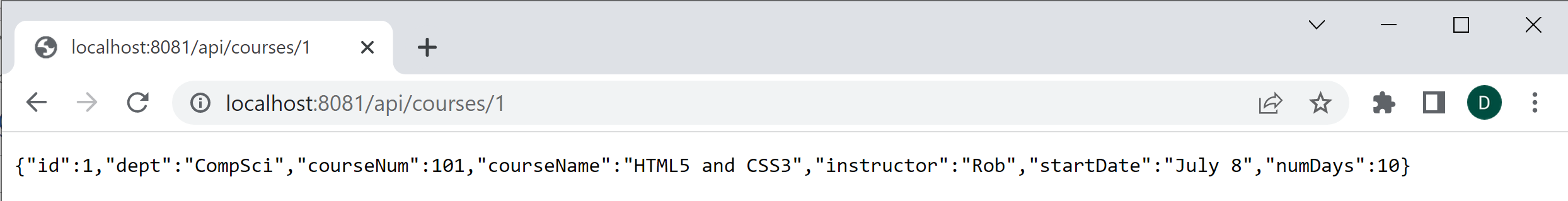
**Expected Output**

App listening at port 8081

#### Verify that it is running by using a browser to call the REST API and "get" courses

##### The server is configured to use a URL like: <http://localhost:8081/api/courses> (for all courses) or <http://localhost:8081/api/courses/3> (for course 3)





#### Congratulations! You have installed the server and are ready to create website!

#### Note: You can stop the server by pressing Ctrl-C from the same command line window that you used to start it

### Exercise: Working with Courses REST API

**EXERCISE 1**

Now that you have a new REST API that is running locally, we will create an application that makes HTTP calls to it.

Don't be confused about this API… the only difference now that it is installed and running is that the URL looks different.

Instead of:

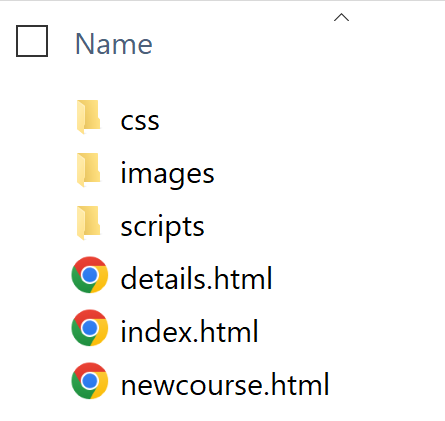
[**http://www.some-distant-server**/api/courses](http://www.some-distant-server/api/courses)

You will use:

[**http://localhost:8081**/api/courses](http://localhost:8081/api/courses)

Create a new GitHub repo named CoursesWebsite. Clone it to your local machine.

###### Create the folders and HTML files shown below.



Let's start by having the index.html page display a list of every course in a table. HOWEVER, only display the department, course number and title!

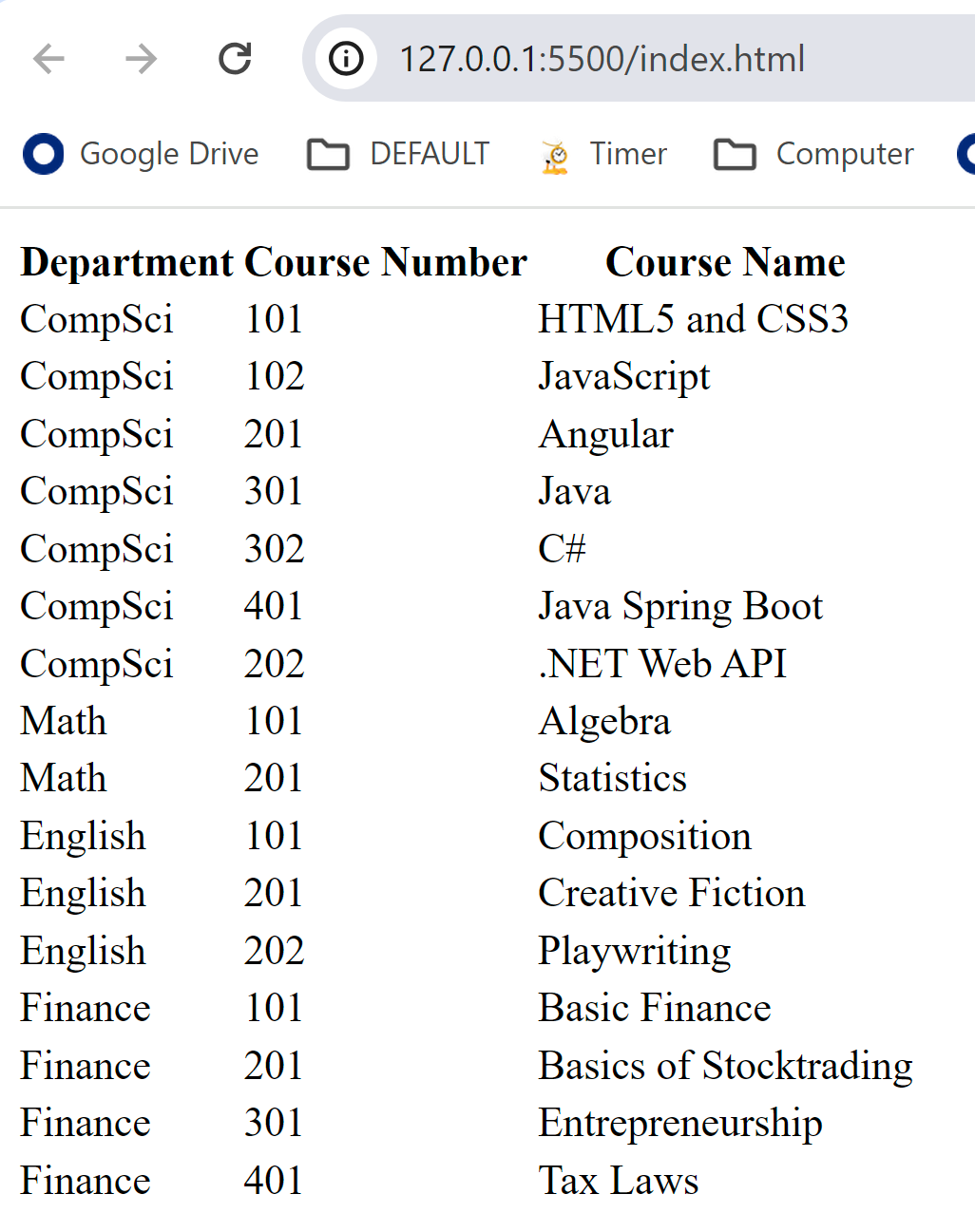
Hints:

\* start by looking at courses.json to make sure you understand the data that will be returned from the api/courses endpoint

\* design a simple HTML page with a table placeholder

\* design a script that fetches all courses and displays the department, course number and course name in the table

Run it using Live Server. Note: It doesn't have to be pretty!



When this works commit and push.

**EXERCISE 2**

Continue working in the same project.

Now that you have the "show all courses" page working, let's add a hyperlink next to each course on the index page that links to the details.html page. The text of the link could be "Show details", or you might turn the course name into a hyperlink.

The details page will show all information about that specific course, including the instructor, when it starts, and how many days it lasts.

To make this happen, the details page will need to be sent a query string parameter specifying the course id. For example:

courses.html**?cid=3**

To create a hyperlink that includes the course id of the course the user clicks, use the code below. Our query string parameter will be "cid".

const detailsCell = row.insertCell();

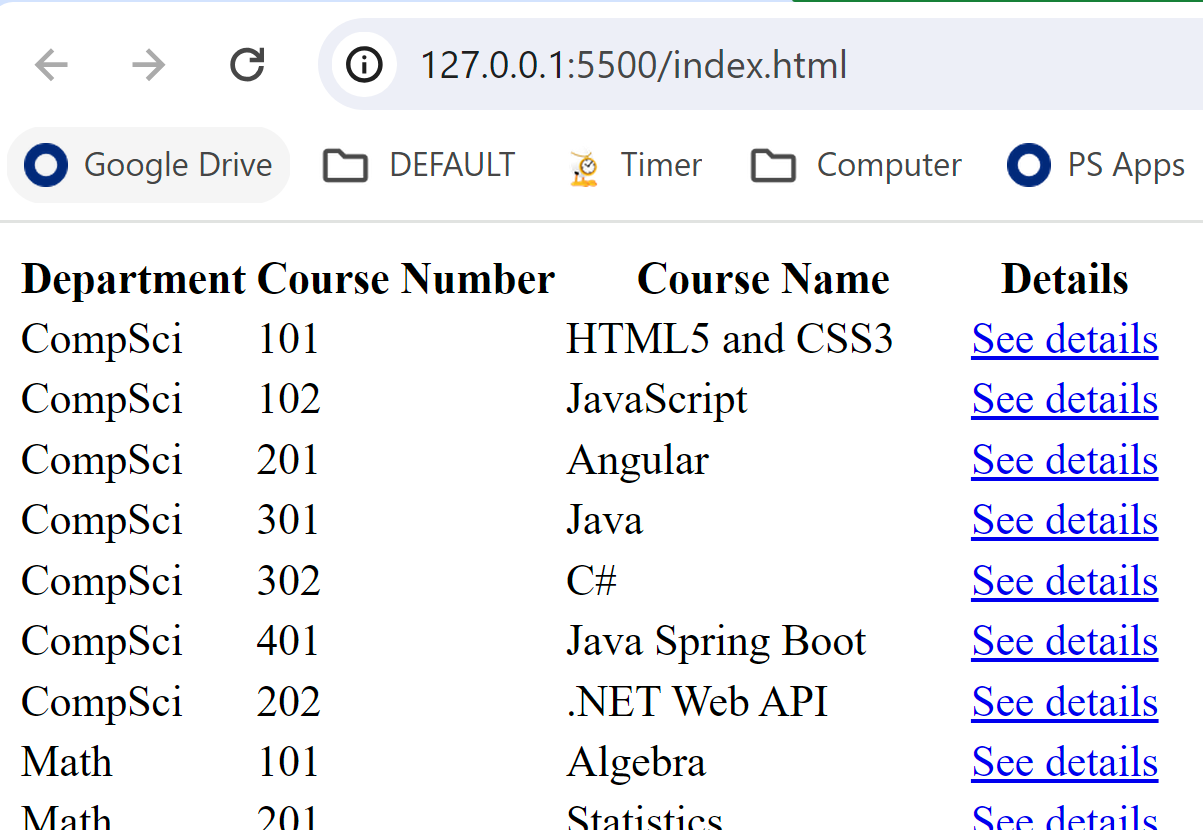
let anchor = document.createElement("a");

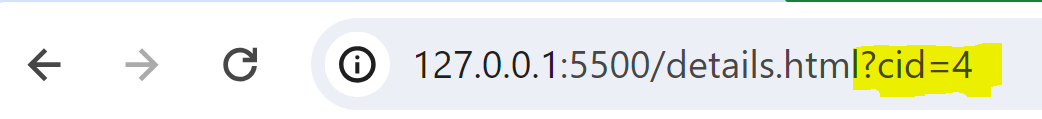
anchor.href = details.html?cid=${courses[i].id}`;

anchor.text = "See details";

detailsCell.appendChild(anchor);

You can test at this point to see if the hyperlink appears and, when you click on it, the details page appears in the browser and the query string parameter is visible in the address bar.





When this works commit and push.

**EXERCISE 3**

Continue working in the same project.

Now let's design the details page. What will the details page do? It will show the details of whatever course is passed in the query string.

You will write code to make this happen during onload.

How do read the query string? In onload, write the following code.

const urlParams = **new URLSearchParams(**location.search**)**;

// location.search returns the query string part of the URL

Once you have the query string, you need to get the value of a ***specific*** query string parameter. To do this, use get() to look inside urlParams. Make sure the query string parameter is there first!

In the code below, cid is the name we used for our query string parameter!

const urlParams = new URLSearchParams(location.search);

// location.search returns the query string part of the URL

let cid = -1;

if (urlParams.has("cid") === true)

{

cid = urlParams.get("cid")

// call a method that fetches this course

getCourse(cid);

}

Note: Every modern browser except Opera Mini and Baidu supports URLSearchParams.

So what would getCourse() look like?

function **getCourse(cid)** {

  fetch(**'http://localhost:8081/api/courses/' + cid**)

   .then(response => response.json())

   .then(course => {

**// this returns a single course!**

      const container =

document.getElementById('courseContainerDiv');

**// display one course property in a <p>**

      const deptP = document.createElement('p');

      deptP.textContent = `Name: ${**course.dept**}`;

      container.appendChild(deptP);

**// repeat for each field you want to display**

    })

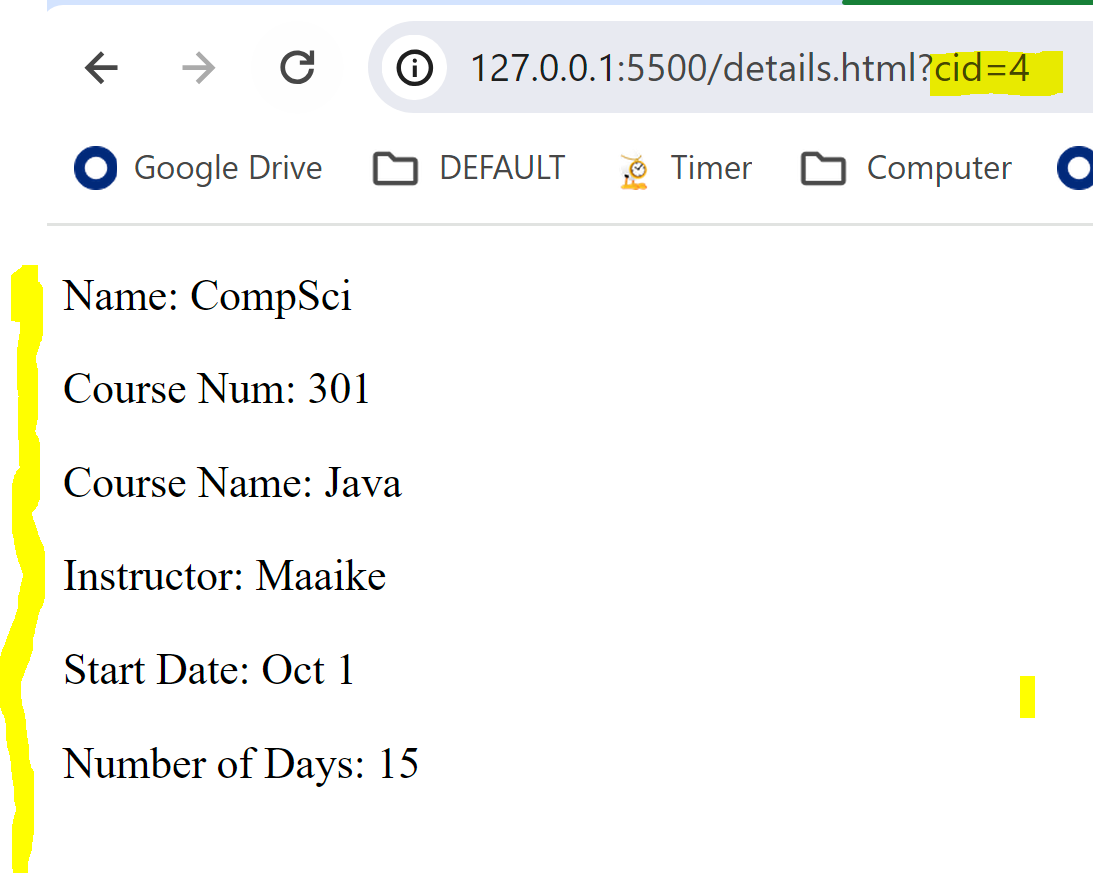
   .catch(error => {

      // handle errors that occurred during the fetch request

    });

}

When you get everything coded, test your page.



**(Optional) EXERCISE 4**

Continue working in the same project.

Instead of displaying course data in details.html in paragraphs placed in a div, see if you can create a Bootstrap card dynamically to display it!

Do not attempt this if you are running behind!

### (Optional Challenge!) Mini-Project

Create a GitHub repo named Weather and clone it to your local machine. It will display the weather forecast at a select number of cities using the US government's REST API at https://api.weather.gov/

**Step 1.** Your page will need a dropdown list containing cities for which you might want to know weather information. Start by creating an array of city objects that contains latitude and longitude information as shown below:

let cities = [

{ name: "Benbrook, TX",

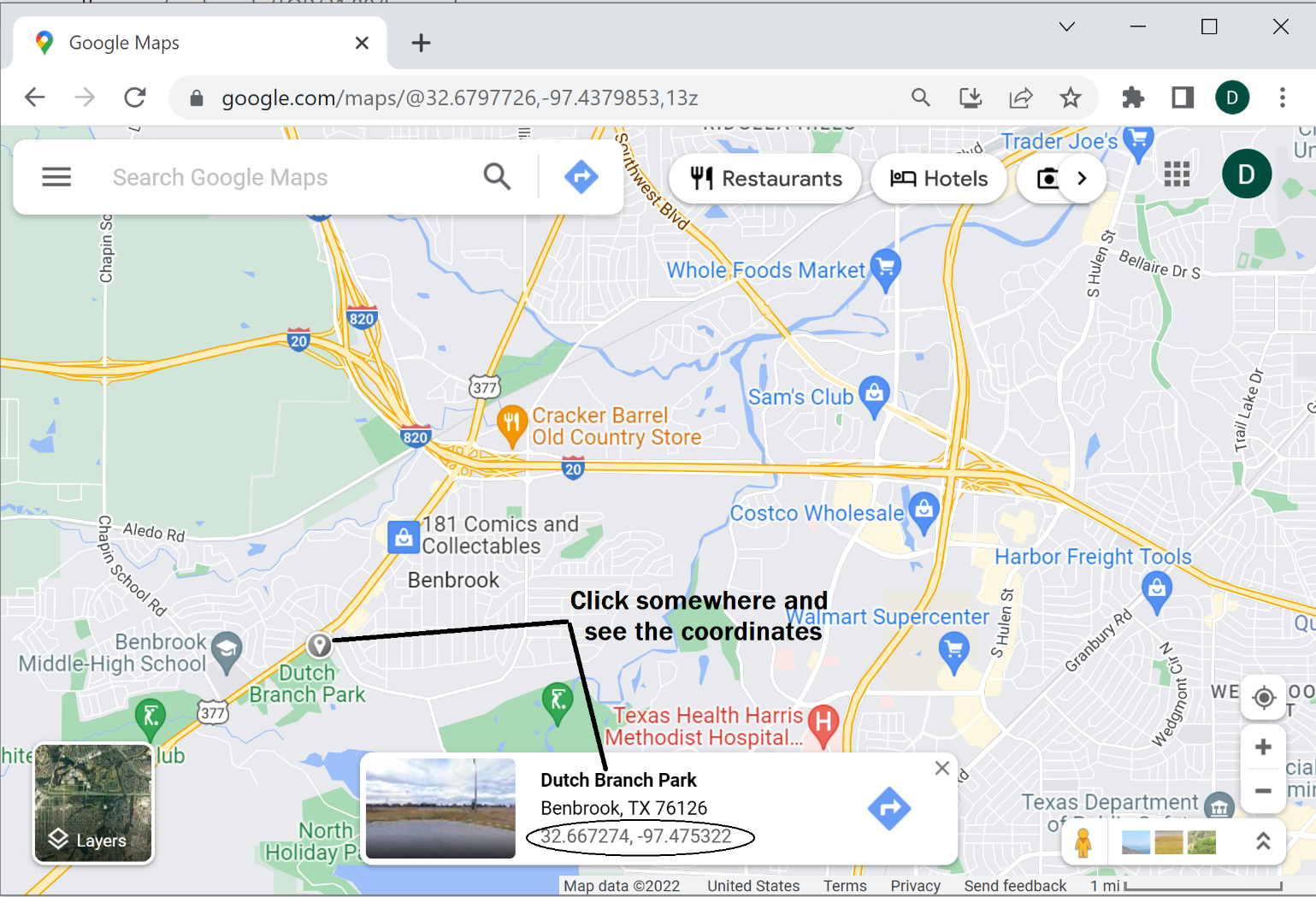
latitude: 32.6732,

longitude: -97.4606 },

...

];

You can use Google Maps to get the latitude and longitude of any location.



Next, when the page loads, populate a dropdown with the names if the cities in the cities array. Place an empty table element below it to hold the (eventual) results. This is a great time to test the page to see if the dropdown loads.

**Step 2.** Let's understand the REST API. Take a moment to read <https://www.weather.gov/documentation/services-web-api>. It tells us that we can use a longitude and latitude to get information about where to go to get the weather.

For example, use your browser to submit the GET request to <https://api.weather.gov/points/32.6791,-97.4641>. Examine the JSON returned. What do you see? A snippet of that JSON is:  
  
{

"@context": [

"https://geojson.org/geojson-ld/geojson-context.jsonld",

{

"@version": "1.1",

"wx": "https://api.weather.gov/ontology#",

"s": "https://schema.org/",

"geo": "http://www.opengis.net/ont/geosparql#",

...

],

"id": "https://api.weather.gov/points/32.6791,-97.4641",

"type": "Feature",

...

**"properties": {**

"@id": "https://api.weather.gov/points/32.6791,-97.4641",

...

"gridId": "FWD",

"gridX": 63,

"gridY": 99,

**"forecast": "https://api.weather.gov/gridpoints/FWD/63,99/forecast",**

"forecastHourly": "https://api.weather.gov/gridpoints/FWD/63,99/forecast/hourly",

...

},

...

}

}

The returned JSON shows us the URL to use to get the weather forecast. For the latitude and longitude we started with, the URL is: <https://api.weather.gov/gridpoints/TOP/31,80/forecast>

Use your browser to submit the GET request to URL above and you will see something that resembles the snippet below. (Only significant pieces for this lab exercise are shown.)

{

"@context": [

...

],

"type": "Feature",

"geometry": {

...

},

**"properties": {**

"updated": "2022-11-04T08:25:17+00:00",

"units": "us",

"forecastGenerator": "BaselineForecastGenerator",

"generatedAt": "2022-11-04T08:59:00+00:00",

"updateTime": "2022-11-04T08:25:17+00:00",

"validTimes": "2022-11-04T02:00:00+00:00/P7DT23H",

"elevation": {

"unitCode": "wmoUnit:m",

"value": 441.95999999999998

},

**"periods": [**

{

**"number": 1,**

**"name": "Overnight",**

"startTime": "2022-11-04T03:00:00-05:00",

"endTime": "2022-11-04T06:00:00-05:00",

"isDaytime": false,

**"temperature": 40,**

**"temperatureUnit": "F",**

"temperatureTrend": null,

**"windSpeed": "15 to 20 mph",**

**"windDirection": "N",**

"icon": "https://api.weather.gov/icons/land/night/tsra,70?size=medium",

**"shortForecast": "Showers And Thunderstorms Likely",**

"detailedForecast": "Showers and thunderstorms likely before 4am, then showers and thunderstorms likely. Cloudy, with a low around 40. North wind 15 to 20 mph, with gusts as high as 30 mph. Chance of precipitation is 70%."

},

{

**"number": 2,**

**"name": "Friday",**

"startTime": "2022-11-04T06:00:00-05:00",

"endTime": "2022-11-04T18:00:00-05:00",

"isDaytime": true,

**"temperature": 43,**

**"temperatureUnit": "F",**

"temperatureTrend": null,

**"windSpeed": "15 mph",**

**"windDirection": "N",**

"icon": "https://api.weather.gov/icons/land/day/tsra,80/tsra,70?size=medium",

**"shortForecast": "Showers And Thunderstorms",**

"detailedForecast": "Showers and thunderstorms. Cloudy, with a high near 43. North wind around 15 mph, with gusts as high as 30 mph. Chance of precipitation is 80%."

},

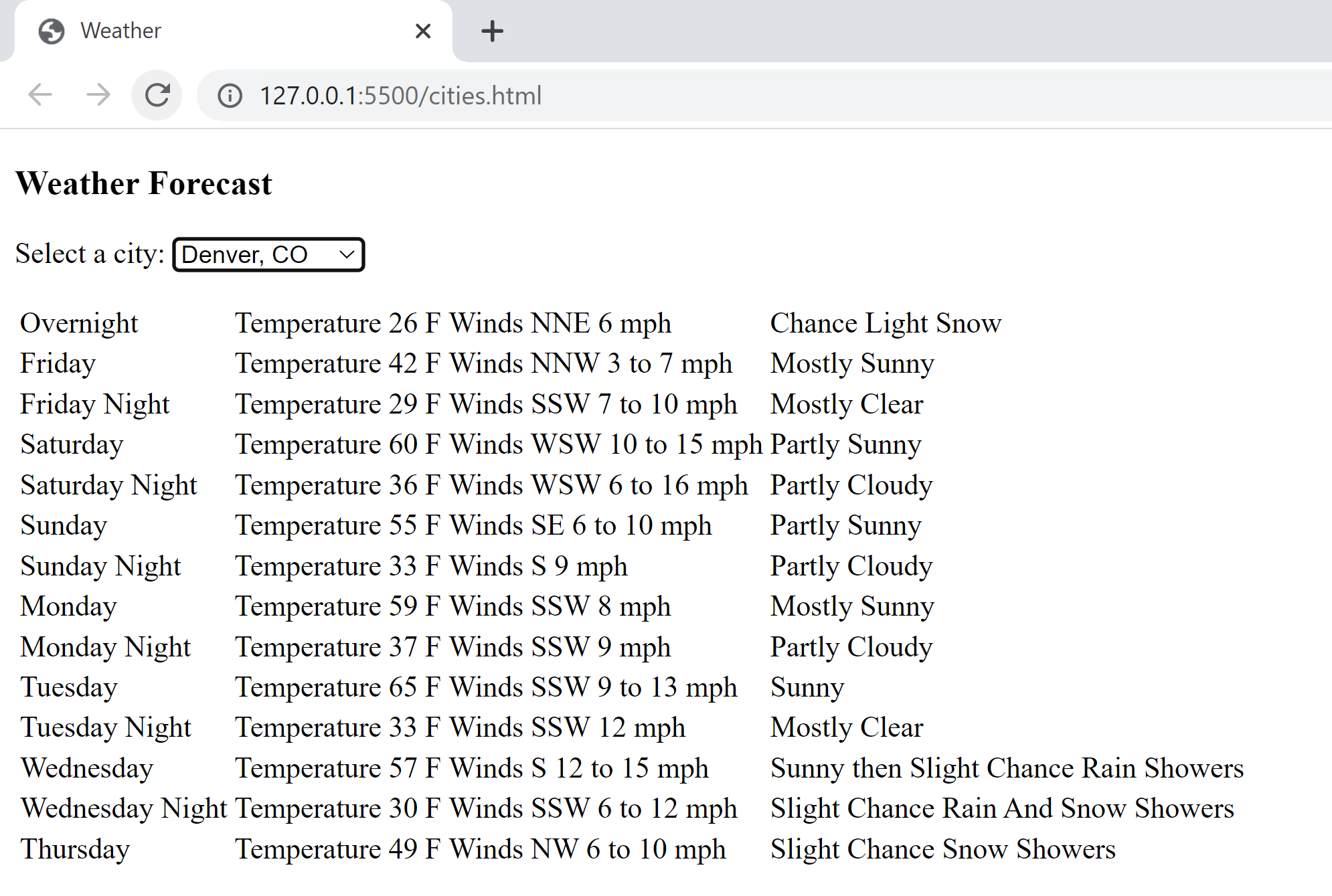
...

]

}

}

You can use the object's properties.periods array to display the location's weather forecast. An unstyled solution to this lab might resemble:



What will make this exercise challenging is that when one fetch() call returns its response, you will have to use that data to initiate a second fetch() call.

**Step 3.** Now that you understand the program, perform that tasks needed to:

1. Load the dropdown list with your cities when the page loads. Also connect the dropdown's change event to a handler.
2. When the user selects a city from the dropdown, find the longitude and latitude of the city selected by searching your cities array. Use it to build the weather station lookup URL

let stationLookupUrl = `https://api.weather.gov/points/${selectedCity.latitude},${selectedCity.longitude}`;

1. Use fetch() to send your GET request to the weather station lookup URL
2. When the response is returned, find the forecast URL and call a helper function to get the actual forecast.

let weatherUrl = data.properties.forecast;

getWeather(weatherUrl);

1. In your helper function getWeather(), use fetch to send a GET request to the forecast URL.
2. When the response is returned, find the periods array containing the weather forecast data and call a helper function to display it.

let forecastArray = data.properties.periods;

displayWeather(forecastArray);

1. Finally, in the displayWeather() helper function, iterate through the array and display the following for each element. Note - the code below is not what you type! It shows the fields to display!

forecastArray[i].name;

"Temperature " + forecastArray[i].temperature + " " +

forecastArray[i].temperatureUnit;

"Winds " + forecastArray[i].windDirection + " " +

forecastArray[i].windSpeed;

forecastArray[i].shortForecast;

# DAY 5

### Quick Lesson: Sending a POST Request

#### A POST request is used to specify that data (a resource) be added to the data store

##### The id can be the last segment of the URL or in the body, depending on how it is designed

#### For an HTTP POST request, you typically specify:

##### "POST" for the method

##### a string of JSON or form-data for the body

##### a content-type HTTP header specifying the content-type

###### There are, of course, many other HTTP headers you can specify

**Syntax**

**fetch**(*endpointUrl,* {  
 **method**: "POST",  
 **body**: *bodyData*,  
 **headers**: {"Content-type":   
 "application/json; charset=UTF-8"}  
 })  
 .**then**(response => response.json())   
 .**then**(json => {  
 // If the POST finishes successfully, ...  
 });  
 **.catch**(err => {  
 // If the POST returns an error, ...  
 });

#### In the following example, we are sending a JSON object with populated data from an HTML form that contains:

###### id, name, and classification

**Example**

// Create JSON object to include in the request body

let bodyData = {  
 id: document.getElementById("stuIdField").value,  
 name: document.getElementById("stuNameField").value,  
 classification:

document.getElementById("stuClassField").value,  
}  
  
// Send the request

**fetch**("http://www.some-url.com/api/students", {  
 **method**: "POST",  
 **body**: JSON.stringify(bodyData),  
 **headers**: {"Content-type":   
 "application/json; charset=UTF-8"}  
 })  
 .**then**(response => response.json())   
 .**then**(json => {  
 **// If the POST finishes successfully, display a message** let confirmationMessage =

document.getElementById(confirmationMessage);  
 confirmationMessage.innerHTML = "New student added";

});  
 **.catch**(err => {  
 // If the POST returns an error, display a message  
 let confirmationMessage =

document.getElementById(confirmationMessage);  
 confirmationMessage.innerHTML = "Unexpected error";  
 });

#### In this next example, we show receiving the new id in the response

##### Sometimes, the id is assigned by the REST API instead of passed in the request and the updated JSON object with newly assigned id is in the response

**Example**

**// Create JSON object to include in the request body**

let bodyData = {

**id: "",** name: document.getElementById("stuNameField").value,  
 classification:

document.getElementById("stuClassField").value,  
}  
  
**// Send the request**

fetch("http://www.some-url.com/api/students", {  
 method: "POST",  
 body: JSON.stringify(bodyData),  
 headers: {"Content-type":   
 "application/json; charset=UTF-8"}  
 })  
 .then(response => response.json())   
 .**then**(json => {  
 **// If the POST finishes successfully, display a message**

**// with the newly assigned id**

let message = "Student " + json.id + " added";  
 let confirmationMessage =

document.getElementById(confirmationMessage);  
 confirmationMessage.innerHTML = message;  
 });  
 .catch(err => {  
 // If the POST returns an error, display a message  
 let confirmationMessage =

document.getElementById(confirmationMessage);  
 confirmationMessage.innerHTML = "Unexpected error";  
 });

### Quick Lesson: Sending a PUT Request

#### A PUT request is used to specify that a resource in the data store be edited

##### But of course, you must also fetch the original data so that it can be edited!

#### For an HTTP PUT request, you typically specify:

##### "PUT" for the method

##### a string of JSON or form-data for the body

##### an HTTP header specify the content-type

**Syntax**

**fetch**(*endpointUrl,* {  
 **method**: "PUT",  
 **body**: *bodyData*,  
 **headers**: {"Content-type":   
 "application/json; charset=UTF-8"}  
 })  
 .**then**(response => response.json())   
 .**then**(json => {  
 // If the PUT finishes successfully, ...  
 });  
 **.catch**(err => {  
 // If the PUT returns an error, ...  
 });

#### This example fetches student 1 data and load it into a form when btnGet is clicked

#### It then send the updated data when btnUpdate is clicked

**Example**

**function btnGetClicked() {**

**// fetch student #1 (hard-coded) to be updated**

fetch("http://www.some-url.com/api/student/1")  
 .then(response => response.json())  
 **.then(data => {**

**// and put returned data in form for editing** let stuIdField =

document.getElementById("stuIdField");

stuIdField.value = data.name;  
 let stuNameField =

document.getElementById("stuNameField");

stuNameField.value = data.id;  
 let stuClassField =

document.getElementById("stuClassField");

stuClassField.value = data.classification;  
 **});**

**}  
  
function btnUpdateClicked() {**

**// send PUT (update) request w/ edited data**

let bodyData = {  
 id: document.getElementById("stuIdField").value,  
 name: document.getElementById("stuNameField").value,  
 classification:

document.getElementById("stuClassField").value,  
 }

fetch("http://www.some-url.com/api/students", {  
 method: "PUT",  
 body: JSON.stringify(bodyData),  
 headers: {"Content-type":

"application/json; charset=UTF-8"}  
 })  
 .then(response => response.json())   
 .then(json => {

**// If the PUT is successful, display a message** let confirmationMessage =

document.getElementById(confirmationMessage);  
 confirmationMessage.innerHTML = "Student updated";  
 });  
 .catch(err => {

**// If the PUT returns an error, display a message** let confirmationMessage =

document.getElementById(confirmationMessage);  
 confirmationMessage.innerHTML = "Unexpected error";  
 });  
}

### Quick Lesson: Sending a DELETE Request

#### A typical DELETE workflow is to fetch data to let the user visually confirm it is the data to delete

##### Then, when the user clicks a Delete button, the DELETE request is sent

#### With a DELETE request, a data body is not typically required

##### Instead, the ID is in the URL

##### Ex: http://www.some-url.com/api/students/**1**

**Syntax**

**fetch**("*endpointUrl*", {  
 **method: "DELETE"**  
 })  
 .then(response => response.json())   
 **.then(json => {** // If the DELETE finishes successfully, ...  
 });  
 **.catch(err => {** // If the DELETE returns an error, ...  
 });

* **Sometimes, DELETE requests don't actually delete data, but just mark it as inactive (like a former customer) or completed (like a claim)**
* In this case, if the item isn't found, it might return a 404

**Example**

**// The user should enter the id in stuIdField before   
// clicking GET button**

let id = "";

**function btnGetClicked() {**

**// fetch student to decide if it should**

**// be deleted**

id = document.getElementById("stuIdField").value;

**fetch**("http://www.some-url.com/api/student**/" + id**)  
 .then(response => response.json())  
 **.then(data => {**

**// and display returned data in a div** let stuDetails = "ID: " + data.id +   
 " Name: " + data.name +

" Classification: " + data.classification;

let studentDetailsDiv =

document.getElementById(studentDetailsDiv);

studentDetailsDiv.innerHTML = stuDetails;  
 **});**

**}**  
**function btnDeleteClicked() {**

**// send DELETE request w/ id as part of URL**

fetch("http://www.some-url.com/api/students**/" + id**, {  
 method: "DELETE"  
 })  
 .then(response => response.json())   
 .then(json => {

**// If the DELETE is successful, display a message** let confirmationMessage =

document.getElementById(confirmationMessage);  
 confirmationMessage.innerHTML = "Student deleted";  
 });  
 .catch(err => {

**// If the DELETE returns an error, display a message** let confirmationMessage =

document.getElementById(confirmationMessage);  
 confirmationMessage.innerHTML = "Unexpected error.";  
 });

}

### Exercises: Rounding out the Courses Website

In these exercises, you will continue working with the Courses website. Recall that it used the Courses REST API to fetch and display a list of courses and course details. Make sure your server is before continuing.

Let's work with POST and DELETE requests for courses.   
  
The HTTP methods and URLs to POST a new course or to DELETE a course are:

POST <http://localhost:8081/api/courses>

DELETE <http://localhost:8081/api/courses>

**EXERCISE 1**

In this exercise, add a "New Course" button in the list all courses page. When the user clicks it, take them to new-course.html.

Provide an HTML form on this page that lets the user enter all of the new data of a course (except the id). When the user clicks the ADD button, use fetch() to POST this data to your Courses API.

Then, re-direct back to the list all courses page when the promise on fetch() resolves in your then().

Test! Is your new course in the list?

NOTE: Our server doesn't do a lot of data validation, so validate that the user has entered data in all fields before calling fetch().

If it works, commit and push!

**EXERCISE 2**

In this exercise, edit the all courses table and add a "Delete Course" hyperlink to EACH course to right of its listing. When the user clicks it, take them to confirm-delete.html page.

The URL would need to be something like:

confirm-delete.html?cid=xxx

where xxx is the id of the course.

When the confirm-delete.html page loads, read the value of the id from the query string fetch that course from the API. Then, display it on the page.

If the user clicks a "Confirm Delete" button located just below the course information, use fetch() to delete the course. Then redirect back to the list all courses page.

Hopefully, the course is gone!

Commit and push!

**WOW! Did you learn a lot this week?**