Lesson Monday

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Adding JS to a Project

Text Cheat sheet

So far we've written all of our JavaScript in the DevTools console. This works for simple practice exercises, but it quickly becomes unwieldy as we want to write longer scripts (JavaScript). In this lesson, we'll learn how to create a project with JavaScript as we continue to practice writing functions and using `window` methods.

You are welcome to read along with this lesson or code along with it. In an upcoming lesson, we'll refactor the code in this lesson. Refactor just means to rewrite. Then in an upcoming practice lesson, you'll create this project (with starter code) and ask you to extend the functionality of the program (meaning, add new functionality to it).

Project Setup

We'll call our project `calculator`, and we'll set it up how we usually do by adding a CSS folder and file, an HTML file, and a README. Your project structure should look like this:

```
calculator/
|_ css/
  |_ styles.css
|_ index.html
|_ README.md
```

We won't worry about adding anything to our CSS right now. In our index file, we'll add basic HTML:

index.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <link rel="stylesheet" href="css/styles.css" type="tex
t/css">
    <title>Calculator</title>
  </head>
  <body>
    <h1>Calculator</h1>
  </body>
</html>
```

Next, we'll add a folder called `js` and an empty file called `scripts.js` for our JavaScript. `.js` is the extension that we give to any file that will contain JavaScript. We could name our file something other than `scripts.js`, but this is a pretty standard name for a project with a single JS file. Your project structure should now look like this:

```
calculator/
|_ css/
  |_ styles.css
|_ js/
  |_ scripts.js
|_ index.html
|_ README.md
```

To connect our JS file to our HTML, we need to add a `<script>` tag to the `<head>` of our HTML. You may remember using a `<script>` tag with Bootstrap in the last course section. The `<script>` tag lets us embed JS directly in our HTML or link to an external JS file. We don't embed JS directly into HTML in this program, though you will see use cases for it on the internet. This is what adding embedded JS in HTML looks like (don't add this to index.html):

```
<!-- We're not actually going to add this to our index.html
-->
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <link rel="stylesheet" href="css/styles.css" type="tex
t/css">
    <title>Calculator</title>
  </head>
  <body>
    <h1>Calculator</h1>
    <script>
      window.alert("Woo-hoo! We've embedded a script into o
ur HTML.");
    </script>
  </body>
</html>
```

At Epicodus, the convention is to always keep our JS separate from our HTML. This helps to create more organized code. Let's continue now and update our `index.html` with the new `<script>` tag in the `<head>` of the HTML.

index.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <link rel="stylesheet" href="css/styles.css" type="tex
t/css">
    <!-- the new line is below this HTML comment --->
    <script src="js/scripts.js"></script>
    <title>Calculator</title>
  </head>
  <body>
    <h1>Calculator</h1>
  </body>
</html>
```

The `src` attribute lets us connect the JavaScript in `scripts.js` to our HTML, so that JS is loaded and run when our webpage is opened. We need to make sure to include the path to our JS file as the value of the `src` attribute.

And with `<script src="js/scripts.js"></script>` added to the `<head>` of our HTML, we are ready to start writing some JavaScript!

Writing JavaScript

Let's add some of the JavaScript functions we wrote in the "Writing Functions" lesson.

js/scripts.js

```
function saySomething(whatToSay) {
  window.alert(whatToSay);
}

function add(number1, number2) {
  return number1 + number2;
}
```

We always write our functions on multiple lines. This follows proper indentation and spacing. Doing this is essential when you begin writing more complex functions.

Alright! We have our code — let's try running it. To run our scripts, we simply need to open the project in the browser. I like to drag and drop the `index.html` file from VS Code into my browser. When I'm working with a pair remotely, I use Live Server to serve the project.

When we do this... nothing happens. What's going on? This is because we have simply defined two functions and made them available for future use. We have not yet **called** the functions, so the code inside of them has not yet been executed. Now add this line after the function declarations in `scripts.js`:

```
saySomething("hi");
```

Let's refresh our browser window opened to `index.html`. You should see an alert pop up with the message "hi". When we click "ok", the dialogue box will disappear and we'll see our HTML. Any script that's tied to our HTML will get executed immediately when we access our website.

Now add another line to `scripts.js`:

```
add(3, 5);
```

Refresh the browser window opened to `index.html` one more time. The alert will still pop up but the new line `add(3, 5);` doesn't appear to do anything. This is because our `add()` method **returns** the result, but we don't do anything with that return value. When we were running this code directly in the DevTools console, the console automatically displayed any return value. But in "real life" programming if you want to display something, you have to tell the computer to do that. We have a variety of options available to us. We could, for instance, assign the return value to a variable and then display it by calling the `window.alert()` method.

Update this line of code:

```
add(3, 5);
```

To this code:

```
const result = add(3, 5);
window.alert(result);
```

Refresh your browser window again and we should get an alert with the result, `8`. If you aren't seeing the alert, make sure that you didn't delete any of the previous code.

Also notice that we're taking advantage of JavaScript's implicit data type coercion: `window.alert()` takes a string as an argument, but `result` is a number. JavaScript doesn't mind this, and will handle converting the result into a string for us.

Combining `add()` with `saySomething()`

Let's update our little program to make use of both functions. Here we have the entire `scripts.js` file, but with two new updated lines at the bottom of the file:

js/scripts.js

```
function saySomething(whatToSay) {
  window.alert(whatToSay);
}

function add(number1, number2) {
  return number1 + number2;
}

const result = add(3, 5);
const outputText = "The sum is " + result + ".";    // this
line is new
saySomething(outputText);                           // this
line is new
```

We're going to break down the last 3 lines of code in our scripts.

Let's start with the first line (of the last 3):

```
const result = add(3, 5);
```

This line calls the `add()` function, passing in two arguments — the number `3` and the number `5`. Our `add()` function assigns the first argument (`3`) to the parameter `number1` and the second argument (`5`) to the parameter `number2`. Parameters are a special kind of variable that hold arguments. So when the function returns `number1 + number2`, it's returning the value of our two arguments: `3 + 5`.

The `add()` function then returns the number `8`, which our code assigns to the variable `result`. Hence the `result` variable is now the number `8`.

Let's look at the next line:

```
const outputText = "The sum is " + result + ".";`
```

This code concatenates three pieces of code together into a new string, which it then assigns to a new variable called `outputText`. Note that `result` above is not in quotes. If you wrote `const outputText = "The sum is " + "result" + ".";` then `outputText` would be the string `The sum is result.` But because we did not put `result` in quotes, the computer understands the `result` variable's value (`8`) should be substituted here. Hence the `outputText` variable is assigned the string `"The sum is 8."`.

You may notice that we're able to concatenate strings and numbers together in this line: `const outputText = "The sum is " + result + ".";`. The `result` variable contains a number, while `"The sum is "` and `"."` are both strings. Well, this is also JavaScript's data type coercion at work: we don't need to explicitly turn the number `result` into a string to concatenate it to other strings with the `+` operator, we can let JavaScript implicitly handle this conversion.

Let's look at the last line:

```
saySomething(outputText);
```

This code calls our `saySomething()` function, passing in the variable `outputText` as the argument. `outputText` has a string value of `"The sum is 8."`. Our `saySomething()` function then assigns that argument (`outputText`) to the value of the `whatToSay` parameter. It then calls the `window.alert(whatToSay)` method in order to pop up a dialog box with the string `"The sum is 8."`.

A Refactor on our `saySomething()` Function Call

If desired, you could refactor those last three lines into one line:

```
function saySomething(whatToSay) {
  window.alert(whatToSay);
}

function add(number1, number2) {
  return number1 + number2;
}

saySomething("The sum is " + add(3,5) + ".");
```

That refactored line calls the `saySomething()` function and passes it a string, which is concatenated together from 3 parts. The first part is just the string `"The sum is "` and the last part is just the string `"."`. The second part is the number `8` because that is the return value of the `add()` function when passed the arguments `3` and `5`. Hence the concatenated string: `"The sum is 8."`

In terms of order of operations in the line `saySomething("The sum is " + add(3,5) + ".");`:

- `add(3,5)` is called first, and we get the value of `8` returned.
- Then the concatenation of `"The sum is "`, `8` and `"."` happens, letting JavaScript implicitly convert the number `8` into a string.
- Then the function call of `saySomething("The sum is 8.")` happens.

Previous (/introduction-to-programming/javascript-and-web-browsers/practice-interactivity-with-window-methods)
Next (/introduction-to-programming/javascript-and-web-browsers/business-and-user-interface-logic)
Lesson 36 of 75
Last updated March 24, 2023

disable dark mode

Epicodus (http://www.epicodus.com)