

Lesson

Monday

Introduction to Programming

(/introduction-to-programming)

/ JavaScript and Web Browsers

(/introduction-to-programming/javascript-and-web-browsers)

/ Calculator UI and Business Logic

Text

Cheat sheet

Now that we've been introduced to the difference between **business logic** and **user interface** logic, let's revisit our calculator project and re-write some of the code. In an upcoming practice lesson, you'll add subtraction, multiplication, and division to this project.

Remember, business logic is all the computation that users can't see. For instance, when we write functions that add numbers together, that is business logic. A computation that happens behind the scenes.

User interface logic, on the other hand, is the logic that allows a user to interact with a page. An example is a function to hide or show an image, or to gather user input.

Defining an Addition Function

We've already written an `add()` function, so let's use that as a jumping off point. If you've been coding along with the previous lesson that set up our calculator project, we're going to remove the `saySomething()` function and simplify our code:

js/scripts.js

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

// user interface logic
window.alert(add(10,5));
```

Notice, too, that we've added comments to distinguish between user interface and business logic in our `scripts.js` file. While you won't see comments like these in production code, they are helpful to include when you are learning and beginning to practice separation of logic.

In the user interface (UI) section, we've simplified our code to use an alert that displays the results of calling the `add()` function with two arguments. The line `window.alert(add(10,5));` calls our `add()` function, passing it two arguments. The return value from `add()` is then immediately passed to the `window.alert()` method. If it's easier to understand, we could rewrite this code on multiple lines as follows:

```
const additionResult = add(10,5);
window.alert(additionResult);
```

At this point, when we run our project in the browser (opening the `index.html`), our scripts should pop up a dialog box with the value that the `add` function returns when passed those two arguments.

Keep in mind that we're using JavaScript's data type coercion: we're letting our JS implicitly change the number `additionResult` into a string, which is the data type that the `window.alert()` method expects as an argument.

Gathering User Input

Our simple (addition-only) calculator isn't very useful because you have to go in and modify the code every time you want to do a new calculation. Let's add in the ability to collect input from the user.

Remember that we can collect input through the `window.prompt()` method, which takes a string as an argument and returns a string containing the user's input.

js/scripts.js

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

// user interface logic
const number1 = prompt("Enter a number:");
const number2 = prompt("Enter another number:");

window.alert(add(number1, number2));
```

This is an exciting update! Here we've collected two numbers from the user with `window.prompt()`, then we've called `add()` with the two user inputted numbers as arguments, and then returned the result in an alert. Now our simple calculator can respond to user input. Let's try this in the browser. Either open `index.html` in the browser, or reload the page that's already opened to it.

Parsing Integers

Oops. We have the problem we previously encountered where the `window.prompt()` method returns strings, so the `+` operator we use inside of our `add()` function concatenates the two strings together rather than doing what we want. Because our `add()` function expects numbers as arguments rather than strings, we must convert the inputs into numbers before calling our `add()` function.

js/scripts.js

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

// user interface logic
const number1 = parseInt(prompt("Enter a number:"));
const number2 = parseInt(prompt("Enter another number:"));

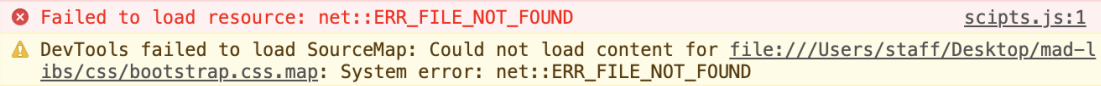
window.alert(add(number1, number2));
```

Now we're getting somewhere! In the next exercise, you'll add in support for subtraction, multiplication and division.

Reminder: Work with your DevTools Console Open and Look for Console Errors

Whenever we are running JavaScript in the browser and something isn't working correctly, the first step is *always* to open the DevTools console. In the console, any glaring errors will show up in red with an X to the left of the message. Meanwhile, warnings will show up in yellow. Messages in the console that are red must be addressed immediately.

Let's review a very common error: Failed to load resource:
net::ERR_FILE_NOT_FOUND scripts.js:1:



The screenshot shows the DevTools console with two messages. The first is a red error message: "Failed to load resource: net::ERR_FILE_NOT_FOUND" with the file path "scripts.js:1". The second is a yellow warning message: "DevTools failed to load SourceMap: Could not load content for file:///Users/staff/Desktop/mad-l.../css/bootstrap.css.map: System error: net::ERR_FILE_NOT_FOUND".

At the time this lesson was written, there was an error *and* a warning. The warning appears to be related to a bug in the latest Chrome update — it doesn't affect our code and it's something we can ignore.

The error, however, is code breaking. It should be a familiar error at this point because we addressed the exact same thing in the Debugging HTML and CSS (<https://www.learnhowtoprogram.com/introduction-to-programming/git-html-and-css/debugging-html-and-css>) lesson. In that lesson, we stated that there are three reasons this error will occur:

1. The file doesn't exist
2. The file is in a different directory than the one we specified
3. The file is in the correct place but the name doesn't match the name specified

In our example case above, the issue is the third one. We've left the `r` out of `scripts.js` in our HTML file: `<script src="js/scripts.js">` `</script>`.

Misspelling a filename is a *very* common error — and one that's not limited to beginners. It's easy to flub a few keystrokes or mistype a filename. Fortunately, debugging makes it equally easy to find and fix this problem.

[Previous \(/introduction-to-programming/javascript-and-web-browsers/business-and-user-interface-logic\)](#)

[Next \(/introduction-to-programming/javascript-and-web-browsers/practice-calculator-ui-and-business-logic\)](#)

Lesson 38 of 75

Last updated March 24, 2023

[disable dark mode](#)



© 2023 Epicodus (<http://www.epicodus.com/>), Inc.