Lesson **Weekend**

Intermediate JavaScript (/intermediate-javascript) / Test-Driven Development and Environments with JavaScript (/intermediate-javascript/test-driven-development-and-environments-with-javascript) / Basic Project Structure

Text

Over the course of this section, we will build an application called **Shape Tracker** based on the Triangle Tracker (https://www.learnhowtoprogram.com/introduction-to-programming/javascript-and-web-browsers/practice-triangle-tracker) project from the Introduction to Programming course. There's a good chance you have a repo with your own triangle project, but even if you don't (or it's not working), don't worry. We will walk through the project step by step.

This course section is not about learning JavaScript itself. While we will learn about some new JavaScript features, our main goal is learning how to use JavaScript libraries and external tools to build our projects. This is what we'll build, step by step:

1. We'll start with building a full-fledged development environment for our application.

2. Then, we'll create business logic using test-driven development. We'll use Jest to write automated tests.
3. Finally, we'll update our user interface logic and learn tools to manage images and other files.

You are welcome to code along with the weekend homework, or just read through the lessons. In class, the very first classwork lesson will prompt you to work through these lessons to build the development environment for the Shape Tracker project.

Do not make any commits or push your code just yet! We will be learning some additional Git best practices in just a few lessons in Git Best Practices and Adding a `.gitignore` File (https://www.learnhowtoprogram.com/intermediate-javascript/test-driven-development-and-environments-with-javascript/git-best-practices-and-adding-a-gitignore-file). In that lesson, we'll prompt you to make your first commit.

Project Structure

First, we'll need some starter code. We'll start with a similar structure to what we've used in the past. Go ahead and create a project with the following structure:

```
shape-tracker/
├── index.html
├── src
│   ├── index.js
│   └── triangle.js
└── css
    └── styles.css
```

We have an `index.html` file, a CSS stylesheet, and two JavaScript logic files, one for the user interface and one for the business logic. However, a few things are different:

- We are calling this `shape-tracker`, not `triangle-tracker`, because we will eventually expand our application to include other shapes.
- We're using a few new naming conventions:
    - Our user interface logic will go in a file called `index.js`. This is a common naming convention for entry point files with webpack. But that's getting ahead of ourselves — we don't need to know about webpack yet! Just know that we are using the naming convention now to help make the transition to webpack go smoothly.
    - We call the directory holding our JavaScript files `src` instead of `js`. Both are fine but `src` is a common naming convention in the industry.

HTML

First let's add the HTML:

index.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <script type="text/javascript" src="src/index.js"></scrip
t>
  <script type="text/javascript" src="src/triangle.js"></sc
ript>
  <link rel="stylesheet" href="css/styles.css">
  <title>Shape Tracker</title>
</head>
<body>
  <h3>Enter three lengths to determine if they can make a t
riangle.</h3>
  <form id="triangle-checker-form">
    <label for="length1">Enter a number:</label>
    <input id="length1" type="number">
    <label for="length2">Enter a number:</label>
    <input id="length2" type="number">
    <label for="length3">Enter a number:</label>
    <input id="length3" type="number">
    <button type="submit">Submit</button>
  </form>
  <div id="response"></div>
</body>
</html>
```

There's not much to see here — a form for getting three length values and a div that will display the response from the application.

One thing worth noting is that we now have two script tags for JavaScript — plus we are linking to a CSS file as well. What if our application was also handling ten other shapes and they each had their own file? Well, that would be ten more script tags. And what if we had multiple stylesheets? Even more links to files. We barely have any code so far so it's easy to see how things can get cluttered very quickly. But that's one of many reasons we are switching to using a development environment — it makes managing many files an easier and more automated process.

JS

Next, we'll add the JS in the `src` directory. Here's the business logic for `triangle.js`:

src/triangle.js

```
function Triangle(side1, side2, side3) {
  this.side1 = side1;
  this.side2 = side2;
  this.side3 = side3;
}

Triangle.prototype.checkType = function() {
  return "I can't answer that yet!";
}
```

We add just enough code to be able to create a `Triangle` object and call the `Triangle.prototype.checkType()` method. For now, the `Triangle.prototype.checkType()` method just returns `"I can't answer that yet!"`. Later we'll update this code when we start exploring test-driven development with Jest.

Here's our user interface logic that we'll add to `index.js`:

src/index.js

```
function handleTriangleForm() {
  event.preventDefault();
  document.querySelector('#response').innerText = null;
  const length2 = parseInt(document.querySelector('#length
2').value);
  const length1 = parseInt(document.querySelector('#length
1').value);
  const length3 = parseInt(document.querySelector('#length
3').value);
  const triangle = new Triangle(length1, length2, length3);
  const response = triangle.checkType();
  const pTag = document.createElement("p");
  pTag.append(response);
  document.querySelector('#response').append(pTag);
}

window.addEventListener("load", function() {
  document.querySelector("#triangle-checker-form").addEvent
Listener("submit", handleTriangleForm);
});
```

You may find it surprising that we can separate our user interface logic and business logic into two separate files and our application still works as expected. For example, how does the `handleTriangleForm()` function know what the `Triangle()` constructor is, if it's not declared in the same function?

Well, when our browser loads our `index.html` file, it loads all of the links and scripts listed in the `<head>` tags.

```
<head>
  <script type="text/javascript" src="src/index.js"></scrip
t>
  <script type="text/javascript" src="src/triangle.js"></sc
ript>
  <link rel="stylesheet" href="css/styles.css">
  <title>Shape Tracker</title>
</head>
```

And the contents of these files ultimately get added to the `window` object, and they are available globally in the project. This is what allows us to call `new Triangle(length1, length2, length3);` in `index.js` even though the `Triangle()` constructor is declared in another file.

Later when we use webpack, we'll see this behavior change because our JavaScript will not be added directly to the global `window` object. But we'll wait to discuss that in depth until we get to the relevant code!

CSS

Finally, here's our CSS file. There's just enough code to let us know that it's connected and working.

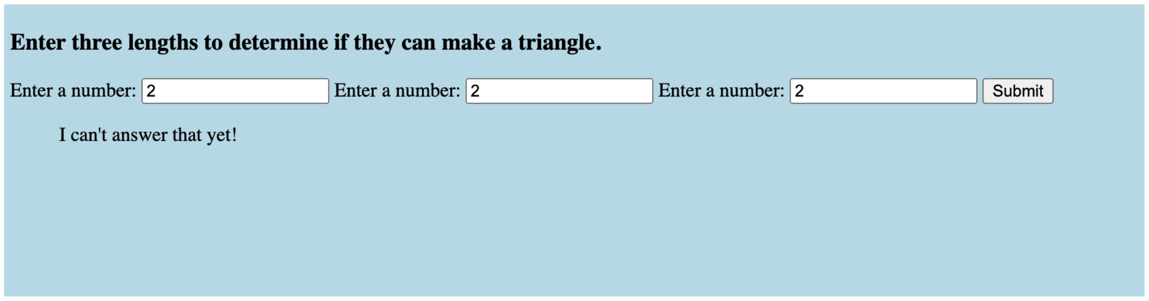
css/styles.css

```
body {
  background-color: lightblue;
}
```

Ahh, light blue. Very relaxing. Don't forget to take care of yourself and take a deep breath during those moments when the problem solving gets tough and frustrating.

We Create Development Environments to Handle Complex Codebases

We can now open our project in the browser and see what we have:



Enter three lengths to determine if they can make a triangle.

Enter a number: 2 Enter a number: 2 Enter a number: 2 Submit

I can't answer that yet!

Well, that's really not much. But we needed four files and nearly fifty lines of code to do next to nothing. Would this way of doing things work with an application like Flickr? No way.

Imagine our little application eventually becoming a full-fledged app that teaches kids about different kinds of shapes — an app that could easily run to thousands of lines of code.

Let's take it a step further. What if it were part of a math application for schools K through 12, ranging from simple addition and shapes to algebra and calculus? Make that application interactive, add logins for students, allow teachers to track student progress and administrators to track teacher progress, and... well, you get the picture. That application would be huge. It simply wouldn't work to use the code structure we've used so far.

In the next lesson, we'll start the process of transitioning our application to one with a fully-functioning development environment — the kind of environment we'll commonly see at tech companies.

Previous (/intermediate-javascript/test-driven-development-and-environments-with-javascript/modern-javascript-development)
Next (/intermediate-javascript/test-driven-development-and-environments-with-javascript/future-project-structure)

Lesson 3 of 49
Last updated more than 3 months ago.

disable dark mode

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