Tooling

Modern JavaScript is almost exclusively done inside of a project. At the root of every project is a file called package.json. You can create this file manually but there's an easier way. Use npm init.

Creating the project itself

1. Open a terminal window and run this on this from the command prompt:

```
npm init
```

2. Accept all defaults.
3. Now take a look at the package.json that you just created.
4. Copy index.html, index.js, and utilities.js from the starters folder to this folder.
5. Edit index.html to add your own greeting. Put your own name in it or something.

In order to properly serve a web project, we need a web server like IIS, Apache, Tomcat, or nginx. But since we're dealing with JavaScript, let's use Node. There's a cool library that can help Node become a super simple web server. Let's install it and run it through npm.

6. Run this from a command line:

```
npm install --save-dev http-server
```

7. Note that there's now a node_modules folder. Also note that package.json has changed.
8. Run this:

```
node ./node_modules/http-server/bin/http-server
```

   It should say serving on localhost on a particular port, probably 8080.
9. Go ahead and navigate to that URL. You should see your greeting because http-server will look for index.html by default.
10. Hit Control-C to kill the server. Refresh the browser just to prove that the server has stopped.
11. That's a lot to type and easy to get wrong. Let's see if we can shorten that with npm scripts.
12. From the terminal, run

```
npm run start
```

It should not work because we don't have an npm script called "start". Let's make one.

13. Add this to the scripts section of package.json:

```
"start": "http-server",
```

14. Try to run from the terminal again. It should serve just fine and when you browse to that URL, you should see your greeting again.

webpack

Remember that webpack is for bundling and minifying your code. Any self-respecting web project should be bundling and minifying. Let's set those up for our fledgling project using webpack.

15. Edit index.html and change the script tag to this:

```
<script src="dist/bundle.js"></script>
```

That bundle.js file will eventually be our bundled and minified JavaScript, JSON, CSS, and HTML files and it is traditionally put in a folder called "dist" or "build". We need to let webpack know to create that file when we're ready to deploy.

16. We'll start by installing webpack:

```
npm install --save-dev webpack webpack-cli
```

17. Add a new npm script to package.json:

```
"build": "webpack",
```

18. Save and go
```
npm run build
```
19. Failure, right? It's because webpack can't find its configuration information. Create a file called webpack.config.js:
```
module.exports = {
  entry: './index.js',
  output: {
      filename: 'bundle.js',
      path: __dirname + '/dist'
  },
  mode: 'production',
};
```
20. Save that file and try to npm run build again. It should work this time.
21. Hey! There's a new folder called "dist". What's in that folder? A bundle.js file? Crack that bad boy open and see what's in there. It is all of your JavaScript files in one big file and minified.
22. Now if you run npm run start and navigate to localhost:8080, your script should be back up and running again.

Transpiling

There are many JavaScript transpilers out there but the two dominant are Babel (thanks to React) and TypeScript (thanks to Angular). Many developers like TypeScript because it adds static typing to JavaScript. Why do we need that? Let's explore static and dynamic typing.

In your index.js, you're doing this:
```
console.log(calculator.add(1, 2, 3, 4));
```
When you've run it, you'll see 10 in the console because 1 + 2 + 3 + 4. Right?

23. Now change it to this:
```
console.log(calculator.add(1, 2, "3", 4));
```
24. Save, build, and run in the browser. It's no longer 10, it is now 334. Why? Because JavaScript is dynamic and will happily let you add strings to numbers. Let's say you wanted to force all other developers to only pass numbers into the add method.
25. Edit utilities.js and find where it says:
```
const add = (...nums) => nums.reduce((prev, curr) => prev + curr, 0);
```
26. Change that to look like this:
```
const add = (...nums:number[]) =>
                          nums.reduce((prev, curr) => prev + curr, 0);
```

Ignore the error for now. That ":number[]" is a little bit of TypeScript that says the arguments must all be numbers. Of course we need to install TypeScript and then tell webpack to transpile the file before bundling and minifying.

27. First install TypeScript and webpack's TypeScript loader
```
npm install --save-dev typescript ts-loader
```
28. Next add these lines inside webpack.config.js. Add them to the JavaScript object that is being exported. (Hint: inside the last closing curly brace.)
```
module: {
  rules: [
    {
      test: /\.ts$/,
      loader: 'ts-loader',
      exclude: /node_modules/,
    },
  ]
},
```

```
resolve: {
  extensions: [".ts", ".js"]
},
```

29. If we want our JavaScript files to be transpiled through TypeScript, we have to give them a ".ts" extension. Rename them to index.ts and utilities.ts.
30. webpack is still looking for "index.js" as its entry. So edit webpack.config.js and change
```
  entry: './index.js',
```
… to this …
```
  entry: './index.ts',
```
31. Lastly, tell TypeScript how you want it to process files by creating tsconfig.json:
```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
  },
  "exclude": [
    "node_modules"
  ]
}
```

32. We're finally ready. Go ahead and attempt a build.

The build should fail with an error message saying that you're trying to use an argument that is not assignable to a parameter of type 'number'. If it fails for another reason, you might want to do some debugging.

33. If you go back to index.js and change the "3" back to a real number, then build and start your server, it should work again.