We have setup webpack just enough to be performing the most basic function of webpack - creating a dist folder with a main.js file from our entry point. And all of that is great - but none of it is useful yet.

What's wrong? Let's take a look:

1. The distribution folder has no connection whatsoever to our app. If you start the express server, our app is still functioning exactly the same way it did in part 0.
2. The main.js file of our distribution folder contains none of the javascript or other assets we wrote for our webpage.
   In short - there are some things wrong with our distribution folder. So it's time to take a look at customizing the webpack output. The "output" of webpack is - no surprise here - the distribution folder. It is where webpack drops or "outputs" the neat bundles of assets it creates from the individual files we point it to.

So we are going to solve the issues above by setting up our webpack output, along with a few other tasks required to make it all work.

Switch the Git Branch

If you want to practise the steps mentioned in the video demonstration below, then stay on the `2-add-webpack-entry` branch.
Else, you can switch to the branch `3-webpack-output-and-loaders` corresponding to the current exercise where all the steps have already been carried out.

```
git checkout 3-webpack-output-and-loaders
git branch
```

About the `dist` Directory, Installing the Babel Tool and Loading the JS Dependencies

Reflect

If you were told to install a node module called awesomest-linter for your development dependencies, what command would you write?
Hint - Use `npm` or `yarn`.

Your reflection

npm

Things to think about

It’s easy to get confused with the many commands and differences between `yarn` and `npm`, but in frontend development, it’s helpful to be fluent with the basics in both package management systems. Don’t forget you may want a `-D` or `--dev` with npm or yarn here as well.

There are a handful of valid solutions here.

- `npm install --dev awesomest-linter`
- `yarn add --dev awesomest-linter`
- `npm i -D awesomest-linter`
- `yarn add -D awesomest-linter`

QUESTION 2 OF 6

When using webpack, if you need to change the button background color, which of the following theoretical files would you choose to edit?

- bundle.min.css
- ui.css
- main.css

SUBMIT

Steps Demonstrated in the Video Above

It might seem strange that we have to add ANOTHER library before we’re even finished setting up the first library - but believe me, that’s how it goes. Babel itself is also not an easy tool to use necessarily, it requires a bit of setup but it is widely used throughout the javascript world to translate new ES syntax into vanilla js that can run on browsers etc. They describe their tool like this:

Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.

Once you have the hang of this setup, I have personally found it helpful to have babel even on projects that don't have Webpack. Sometimes I'll install it just for the convenience … like the convenience of no semicolons or being able to use import/export syntax.

First off, we need to install Babel via npm. Babel occasionally changes its install requirements, but at the moment these are the configurations that work and seem to be pretty stable:

`npm i -D @babel/core @babel/preset-env babel-loader`

Remember that the -D will install these as **development** dependencies.
And now, just like webpack, babel also requires a config file. In this case, it is a fairly simple one.

Create a new file `.babelrc` in the root of the project. Fill it with this code:

```
{ 'presets': ['@babel/preset-env'] }
```

Now, we're about to go through a whole rigamarole of settings, these aren't the kind of settings I would commit to memory necessarily, but I will try to explain the steps as we go along.

First, we have both webpack and babel installed, now we have to get webpack to use babel. Doing that forces us to use a part of webpack we haven't explained yet - we will explain them I promise - but that would jump us just a little ahead of where we are, so for now, you can copy and paste this part. We will observe what it does and then circle back around to explain it.

So, back we go to the webpack config, because we have some things to add.

First, we could specify the "output" of our webpack config, it would look something like this:

`output: { ...output options }`

But at the moment, we don't need to add any custom settings there. The default settings are good enough for us - creating a dist folder in the root of our project.

So, output is set, but there's still the matter of getting webpack to use babel. For that we'll use a webpack loader. Loaders are what we will circle back around to in a second, but for now, add this to your webpack config.

```
    module: {
                rules: [
                        {
                    test: '/\.js$/',
                    exclude: /node_modules/,
                    loader: "babel-loader"
                        }
                ]
    }
```

Now, with that loader in place, we should be able to get going.

Go to your index.js file and import our two javascript files (also make sure you export them in the original files!). Then console log one of the functions. Here is an example:

```
import { checkForName } from './js/nameChecker'
import { handleSubmit } from './js/formHandler'

console.log(checkForName);

alert("I EXIST")
```

Really we don't need the alert any more, but either way, delete the current distribution folder and rerun the build command.

You have been successful if you see your console logged function in the main.js that is output.

QUESTION 3 OF 6

Babel is a javascript library that allows us to:

- Use any version of javascript we want, in any file
- Use javascript in any type of file like .html, .css, etc...
- Use the latest version of javascript, even if our browser or runtime doesn't allow it
- Use extra javascript features that don't yet exist in the language spec

SUBMIT

Quiz
What file would you look for in a project to see if it is using babel?

.babelrc
RESET

QUESTION 5 OF 6

Why are webpack and babel seen together so frequently?

- Babel allows developers to use convenient es6 syntax that works well with Webpack in a front end application
- Webpack requires Babel and Babel is made to be run with Webpack.
- Babel requires no extra setup when used with webpack
- They are not frequently used together, this example is only being used because we are not using a framework.

SUBMIT

Interview Question
What is the difference between source code and built code?

Your reflection
source code is used during development only but built code is the one which distributed to run our code
Things to think about
This is a good question to know the answer to, not because you will be asked it point blank like this necessarily, but because knowing the answer means you understand enough about the front end ecosystem and are probably able to jump into a larger project and contribute meaningful front end work.

Resource - Webpack Introductory Documentation

Have a look at the **Webpack Introductory Documentation**, where you can understand its core concepts - Entry, Output, Loaders, and Plugins.