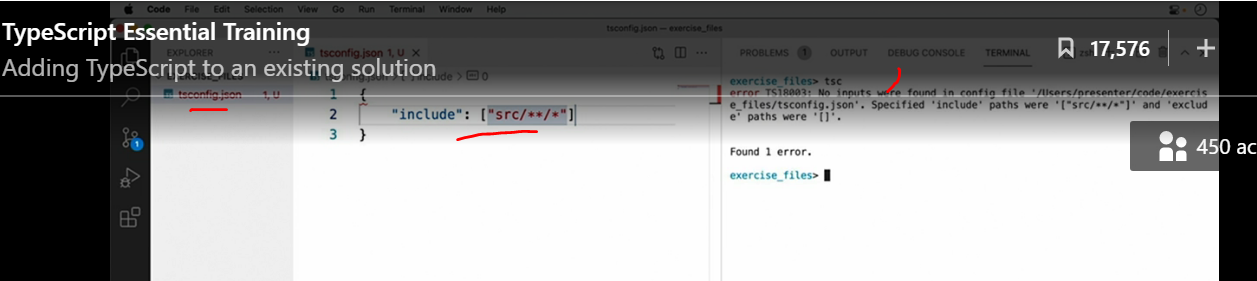
* - [Instructor] In the introduction to this course, I showed you TypeScript in action, adding valuable static typing to your projects.
* In order to gain access to this information, however, you'll need to configure TypeScript in your application.
* If you're using an editor that has built-in support for TypeScript, such as Visual Studio Code, like I'm using here, the only thing you'll need to add is the TypeScript configuration file to the root of your application's code base.
* This file is called ***tsconfig.json.***
* There are plenty of configuration options available but I'm going to start with just one: **include**.
* And I'll supply it a **parameter of the paths to the files** that I'd like to include.
* Here I'm telling it to look for all of the files under *the src or source folder* under this current folder.
* This defines the **scope** of this TypeScript project.



* Now I can save this file and run the TypeScript compiler to see what happens.
* And I see the output no inputs were found in config file, which ***means that the compiler didn't find any files to compile.***
* So let's create one now.
* First, I'll create that src or source folder that I mentioned earlier.
* And then I'll create a new TypeScript file, which is just a file with the .
* ts extension.
* Then I'll add some JavaScript to it.

Graphical user interface, text, application

Description automatically generated

* Now if I run that tsc command again, I'll see there's no output, which actually means that everything ran just fine.
* How do I know? Well, if I look in the src folder, I now see two files, the demo.
* ts file that I created and the new demo.
* js file that's the **output of TypeScript's compilation process.**
* If I open up this new file, I can see that TypeScript has taken my code and transpiled it down into JavaScript that is more compatible with a wider variety of browsers and runtimes.
* In fact, you're able to ***control which JavaScript version TypeScript targets in the tsconfig file.***
* You can find almost all of the **TypeScript settings** in the **compilerOptions** section.
* The option I'm most interested in right now is the target option.
* As you can see here, this option defaults to ES3, which stands for ECMAScript version 3.
* As of the time of this recording, the majority of browsers in use today support ECMAScript 6 or ES6, so that's usually a great choice.

Graphical user interface, text, application

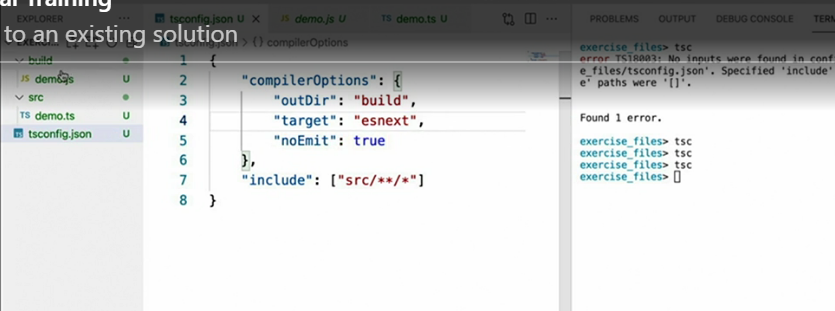
Description automatically generated

* Now when I run the tsc command again, and I inspect the output, I can see the JavaScript that TypeScript has emitted has changed.
* Since ES6 supports the class keyword natively, TypeScript knows that there is no conversion to be done and simply writes the class out as is.

Graphical user interface, text, application

Description automatically generated

* It can also tell that the three properties I defined on the class were never used.
* So it knows that it's safe to strip them out from the final output.
* TypeScript's default behavior is to render JavaScript files right next to the TypeScript files.
* However, if TypeScript is the only tool you'll be using, *you'll probably want to direct the output to another folder.*
* To do this, simply **set the outDir setting to the folder where you want the generated files**.
* Then delete the generated demo.
* js and run the tsc command again.
* Once it's finished, I can see the file has now been created in the new build folder.



* If, however, you plan on using TypeScript only for its type checking ability, and will rely on another tool, such as the **Babel compiler** to transpile the final output, then an even better value for the target setting is **esnext**.
* This setting tells TypeScript to emit code that is compatible **with whatever the latest version of JavaScript happens to be**, which effectively just means that it *should strip out all the type annotations and pass the rest of the code onto the other tools.*
* And finally, if you do plan on using TypeScript only for type checking and nothing more, it's also a good idea to set the noEmit setting to true, which tells TypeScript not to write out anything to disk at all.

Graphical user interface, application

Description automatically generated

* If I do this and then delete the generated folder, then run the tsc command again, I see that my type checking is working perfectly fine **but no files are being generated anymore.**
* Now that I have a basic TypeScript configuration in my project, it's time to move on and see what else it can do.