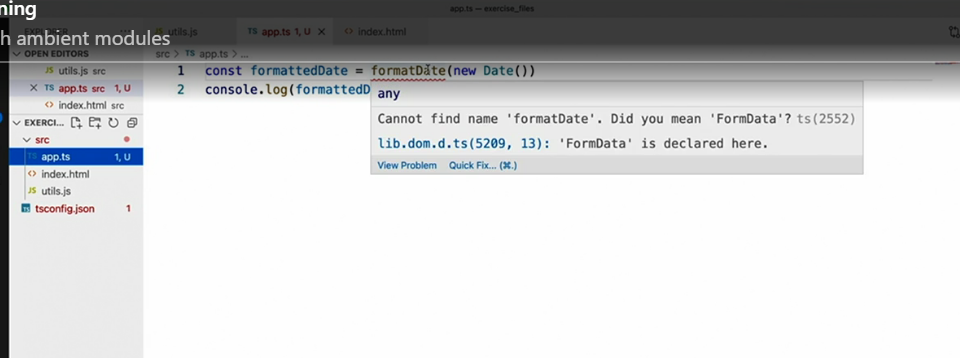
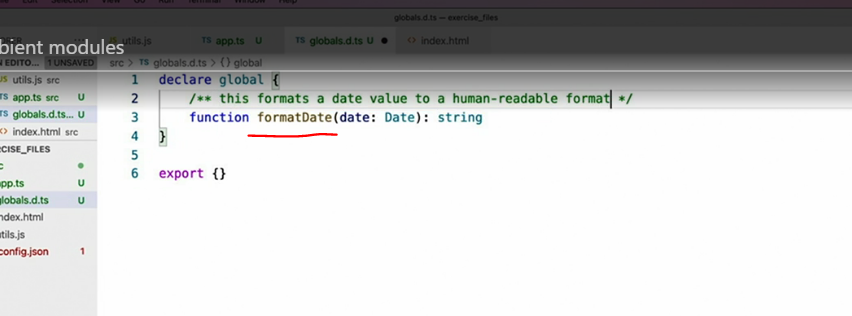
* - [Instructor] In the imports and exports video, I showed you how to convert an existing application that relied on global functions, to modules using imports and exports.
* However, if you have a large existing codebase that relies on global imports for existing JavaScript files that can't be converted, don't lose hope.
* In this video, I'll show you how to document that JavaScript code, so you can continue to use those dependencies without a problem.
* To begin, I'll go back to the same application that I just converted, back to before I converted it.
* Remember this was the version of the application that was explicitly loading all of the JavaScript files.
* I'll start out by converting the app.
* js file to TypeScript, just like last time, and I'll see that error again.



* TypeScript doesn't know about a function named formatDate, just like last time.
* However, in order to fix this problem this time, I'm going to create my own version of the TypeScript declaration files, I showed you way back in the first chapter.
* As you might recall, these files have a d.
* ts extension.
* Since TypeScript is going to read the contents of this file, no matter what name I give it prior to the d.
* ts extension, I'll just call it globals.
* d.
* ts.
* Then I can create whatever type definitions I want in this file.
* In order to tell TypeScript that what I'm about to describe will be in the global namespace, I'll write the following.
* **This tells TypeScript that everything I write inside these braces, should be available in the global namespace.**
* Notice the error here.
* This is TypeScript telling me that I can't use this syntax just anywhere, it has to be inside of a module.
* That means I need to export something in order for TypeScript to treat this as a module.
* So I'll just define an empty set of exports.
* With that out of the way, I can now define whatever global definitions I like, for example, my function.



* Notice how I'm just writing the definition of the function and not the implementation of itself.
* And of course, this is TypeScript after all, so I can give it a parameter type and a return type.
* And while I'm at it, I can add some documentation too.
* Once I'm all done with the definition, I can switch back to the TypeScript code that uses it, and see that not only has TypeScript found my definition just fine, but I even get the full documentation in IntelliSense as well.
* That's all I need to do in order to address this particular example, but rest assured that you can use any of the TypeScript concepts that I have shown in this course inside of these declaration files to describe any situation you may come across in your custom code.
* It's also important to note that writing type declarations like this, doesn't actually ensure that the code is available at runtime.
* For example, if I were to forget to include the script tag that loads the util.
* js file into my browser, my TypeScript code would continue to compile without any errors, only to crash at runtime, when it goes to execute a function that doesn't exist.
* Regardless of that caveat, having this ability to define types for JavaScript code, that lives outside of your TypeScript files, is a great way to get full TypeScript support when integrating with existing JavaScript codebases.