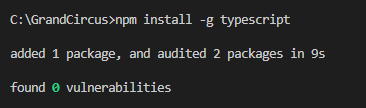
**Installing TypeScript + Setup Instructions**

12/07/22

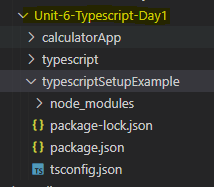
To install TypeScript, we went into our Grand Circus folder (this can be anywhere, since we are installing it globally) and typed “npm install -g typescript”



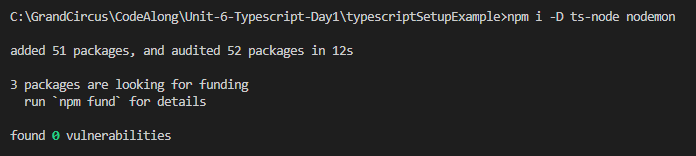
The following steps have to be done for every TypeScript file created, which can be frustrating.

We created a new folder and then in the Terminal where the file will be located, we typed “npm init”. This creates a new JSON file (you can also use “npm init –yes” to avoid having to answer questions.





After that, we typed “npm i -D ts-node nodemon” and it created the “node\_modules” folder.

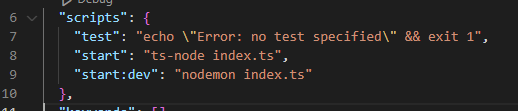


We then typed in “npx tsc –init” and it created the “tsconfig.json” file above

Text

Description automatically generated

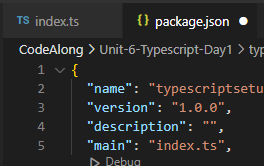
We then went into our JSON file and changed out the “scripts” with the code shown below.



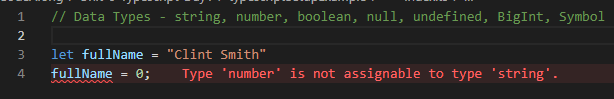
Finally, we created a TypeScript file “index.ts” and changed the “main” in our JSON file to match the file name of our Typescript file.

Text

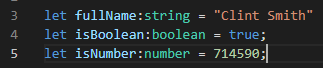
Description automatically generated



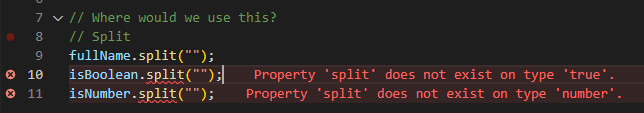
As a simple first test, we created a variable “fullName” and gave it the value of “your name here” (a string). If we then try to assign it the value 0 (a number), we get an error message. TypeScript will not allow us to change a String to a Number. Once assigned to a string, it must remain a string.



To identify the data type associated with the variable, we can add the type after the variable name. You don’t technically have to enter that, but this can be a good way to easily identify what data type each variable is.

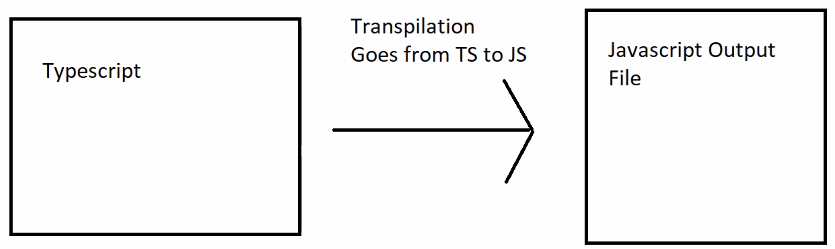


Where would we use something like this? When we want to be explicit about the data type associated with a variable. For example, you can use the .split() method on strings, but not other data types. If we try to use .split() on the variables above, then TypeScript will throw up an error message.



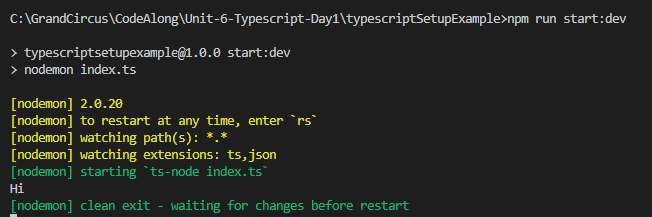
What TypeScript is doing is called “transpilation” (think “compilation”, but different), taking your TypeScript file and changing (compiling) it into a JavaScript output file. This is essentially how TypeScript sits outside your JavaScript file and puts in place new guardrails.

You type a file within TypeScript that has to follow certain rules, and it spits out a JavaScript copy of that file that, because it was written in TypeScript, should work exactly as intended.



Let’s test this out with our new TypeScript file. To do this we added a console.log(“Hi”) to our index.ts and then in the Terminal typed “npm run start:dev”





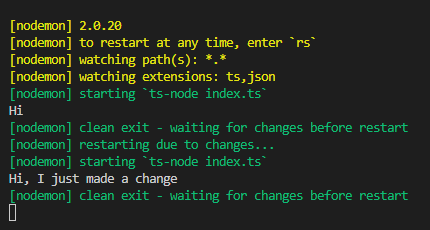
This is calling back to our JSON file “scripts” array

Text

Description automatically generated

With this running, every time you make a change and then Save, it automatically re-runs and updates the results. To exit, you use ‘CTRL+C’.





I ran into a problem running ‘npm test’ in the Mock Assessment #5, but found a solution (npm install --save-dev jest) via Google.

Text

Description automatically generated

Text

Description automatically generated

I ran into some problems in doing the Race Cars lab and Jonathan looked at it with me. I ended up needing to run the following in the Terminal as well.

npx ts-jest config:init

npm i ts-jest

When VS Code says it can’t find Express us this:

npm install express @types/express