* - [Instructor] In the previous chapter, I showed you how to add TypeScript to your application.
* I also showed you how to configure TypeScript to inspect your JavaScript files and provide some pretty impressive type checking simply by inferring or implying type information from your existing code using nothing but basic JavaScript syntax.
* In this chapter, I'll show you how to not only explicitly provide TypeScript with type information but also move beyond those simple built-in types and create your own custom types.
* Before I show you how to build custom types however, let's start by showing the basic TypeScript syntax.
* I'll start with regular old JavaScript.

Graphical user interface, text, application

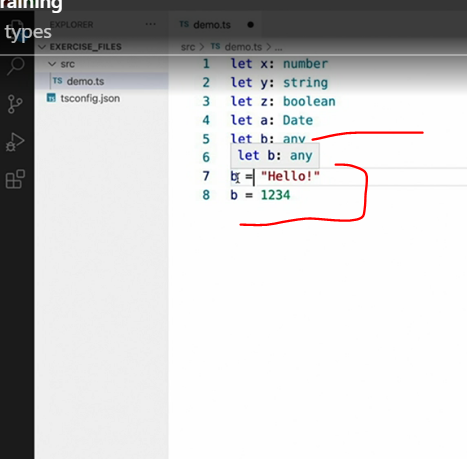
Description automatically generated

* Here, I'm creating a variable named X and assigning it the numeric value five.
* And just like I showed in the last chapter, If I hover over X, I can see that TypeScript has inferred that because I initialized the variable with a numeric value, the type of this variable is number.
* This is called type inference and it's one of TypeScript's most powerful and impressive features.

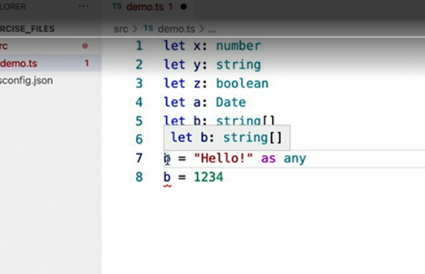
Graphical user interface, application

Description automatically generated

* However, if I remove the assignment, TypeScript no longer has enough information to infer the type.
* So in this case or in any case that I need to explicitly define a variables type, I can do it with this syntax; a colon followed by the name of the type, in this case number.
* I can use this syntax to define any types that I want like this.
* (instructor typing) I can even identify array types.
* (instructor typing) And if I try assign one of these variables to a value of the wrong type, TypeScript shows me errors.
* So that's the basics syntax that lets me give TypeScript the information it needs to help keep me for making mistakes.
* But let's back up a minute.
* What happens when it's not a mistake? What happens when I actually want to assign a different type to an existing variable? That kind of dynamic behavior is after all part of the power of JavaScript, right? Enter the any type.



* By defining this variable as the **any** type, I've told TypeScript that this variable can literally hold any value of any type.
* In addition to defining a variable as the any type, I can also use **TypeScript type casting** to tell it to treat any value as the type any.
* For example, rather than defining the variable B as type any I could cast the string value I wanted to assign to the any type instead.
* For example, rather than defining the variable B as type any, I could cast the string value I wanted to assign to the any type instead.
* And now we can see that the error goes away.
* Now using any can seem like a useful way to reduce the many type checking errors you may see when you first start introducing TypeScript to your projects.



* And there may be many times that you may feel you have no other way to get rid of those pesky errors other than using the any type like this.
* However, the any type has a **significant downside.**
* By using the any type, you're effectively telling TypeScript *don't bother including this type in type checking*.
* Trust me, I've got it handled.
* However, this goes against the entire point of using TypeScript in the first place.
* Although this is a perfectly legitimate syntax and the any type is a perfectly legitimate type, you can think of the rest of this course as providing you with the tools you'll need in order to express the exact types you want and allow you to avoid ever having to use the any type in your projects.