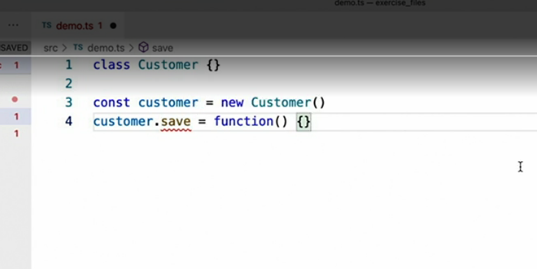
* - [Instructor] In the previous video, I showed you how to document code that TypeScript isn't aware of.
* In this video, I'm going to show you how to extend the definition of a type that TypeScript is aware of.
* What do I mean? Well, let's say that you have this class, and you use some good, old-fashioned, dynamic JavaScript to add some behavior to it after an instance of it has been defined.

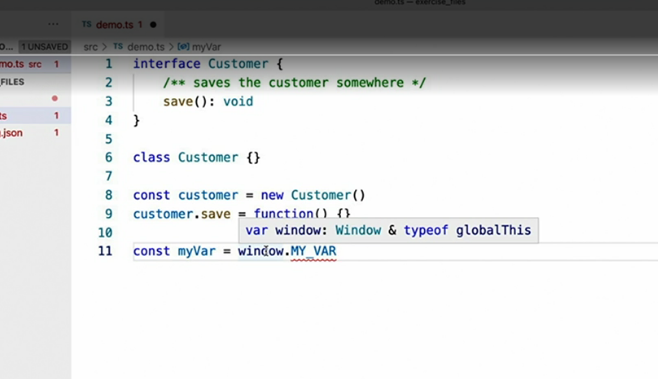


* Well, TypeScript doesn't like that you didn't tell it about that save property when you created the original class definition.
* Now you could just go back to the definition and define the function with an empty implementation, but what if you don't want to do that? Or what if that class definition is a third-party library and not your code?
* **In order to extend a class definition, whether you own the class definition or not, you can simply *define an interface with the same name as the class*, like this.**
* Then fill this interface with whatever you would like that class to expose.

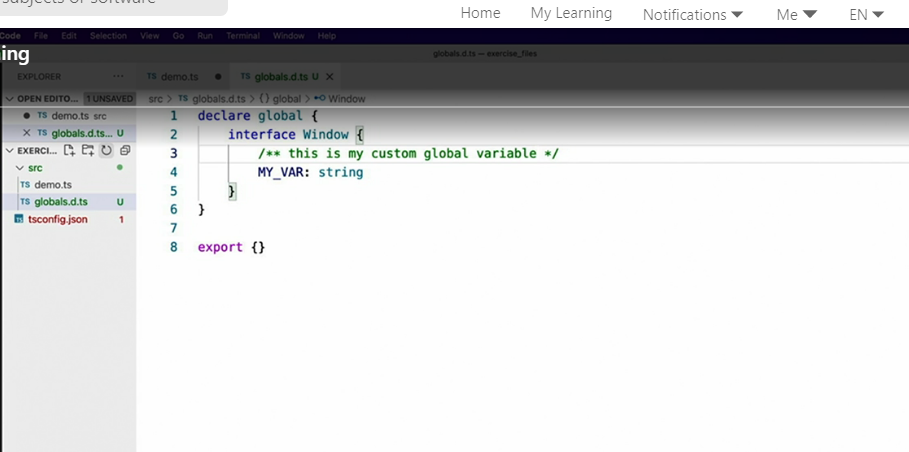
Graphical user interface, text, application

Description automatically generated

* And notice that the error goes away, and it's replaced with full type information.
* This is a TypeScript feature called **declaration merging**, which means that TypeScript is perfectly happy to allow you to extend an existing interface with a new one.
* Unlike JavaScript code, ***creating another interface with the same name as an existing interface merges the new interface into the existing one rather than overriding or replacing it.***
* And yes, I did say extend an existing interface, even though the example I'm showing here extends a class definition.
* That's because ***TypeScript considers every class definition to also be an interface.***
* It's just an interface that happens to have an implementation.
* To be honest, I don't use this type of declaration merging too much, usually only when I'm adding behavior to a third-party library.
* However, the one scenario where I do use it quite a bit is to **strongly type my global variables in a web application.**
* You know, the ones that look like this.



* Notice that if I hover over the window variable, I'll see that the built-in TypeScript types include an interface definition for the browser's window object, one that I can extend.
* I'll just create another globals.d.ts file like I did in the previous video and add my global declaration here, just like I did in the last video.
* Then I'll define an interface with the same name as the existing interface window.
* and once I have that, I can put anything I want in there, just like I did with my global function.



* With that in place, I can switch back to my application and see that the error has been replaced with full type information.
* Note that you can use this trick to extend any class or interface that you reference in your application, regardless of whether it's your code or a third-party library, and you should feel free to do so.