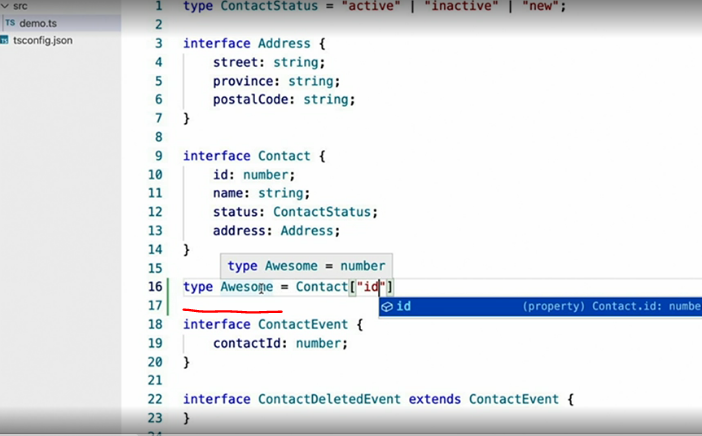
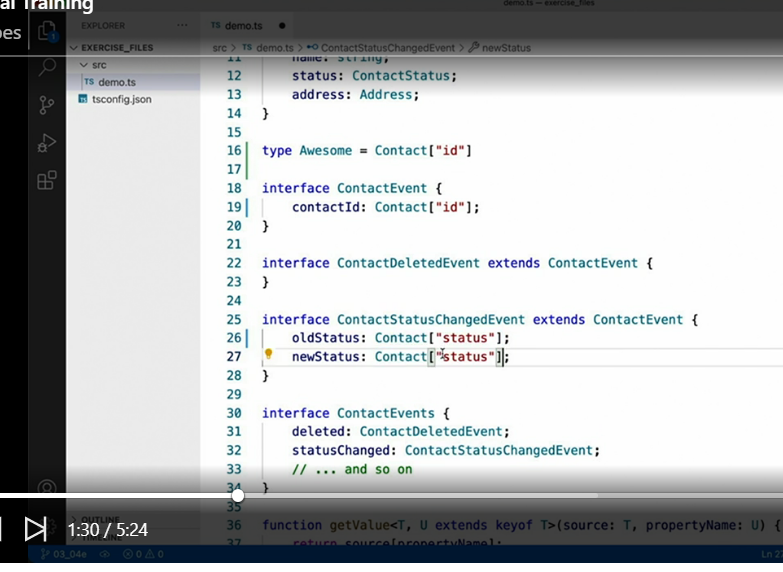
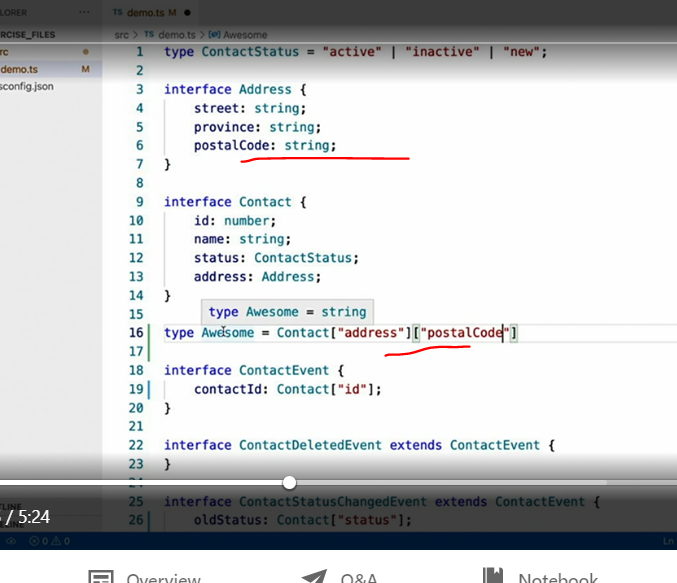
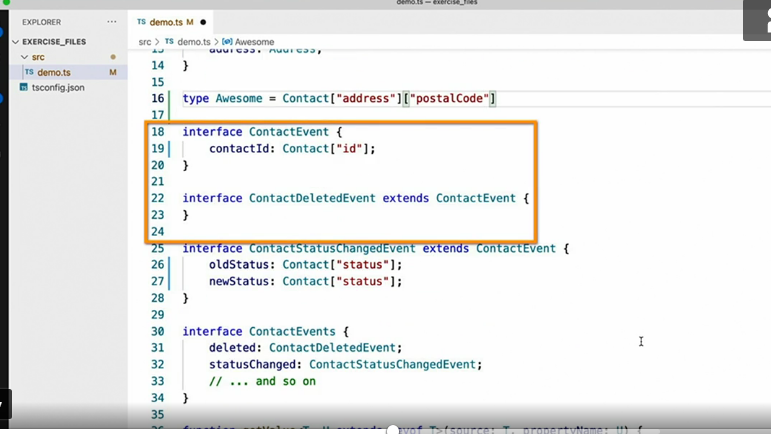
* - [Instructor] In this video, I'm going to show you how ***to determine the type of a certain property or multiple properties of a target object*** using something called an **indexed access type.**
* As its name implies*, the syntax for an indexed access type is the same as the JavaScript syntax for accessing a property of an object using the index syntax like this.*



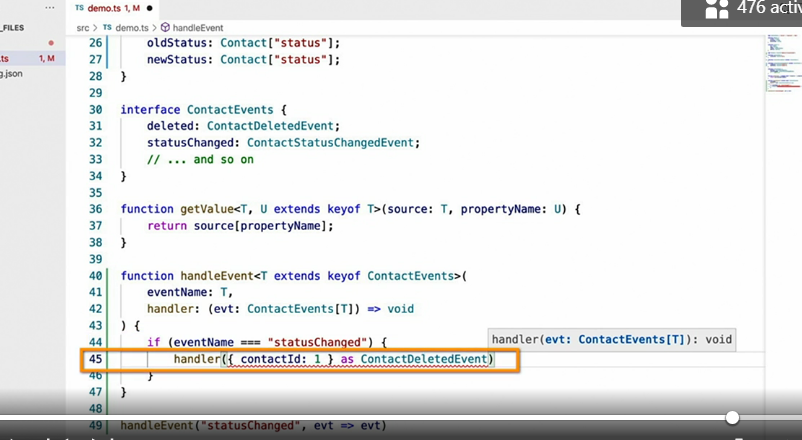
* If I hover over the type, I can see that the type of Awesome matches the type of the id property on the Contact type, number.
* How is this useful? Well, take a look at line 19 where I have a property named contactId, which references the ID property of the contact type.
* Well, at least its name references the ID property but its type just happens to be the same.
* There's no direct connection between the two.
* So let's make one.
* While I haven't really changed the type of the contactId property here, I have done a much better job of showing my intention that this type should match the type of the ID property on the contact interface.
*  I can do the same thing with these references to ContactStatus.
* Now, let's go back to our original example and explore a little deeper.
* Deeper into the object structure that is.
* Not only can I use this syntax to reference the type of properties, I can actually dig into the properties of those properties.



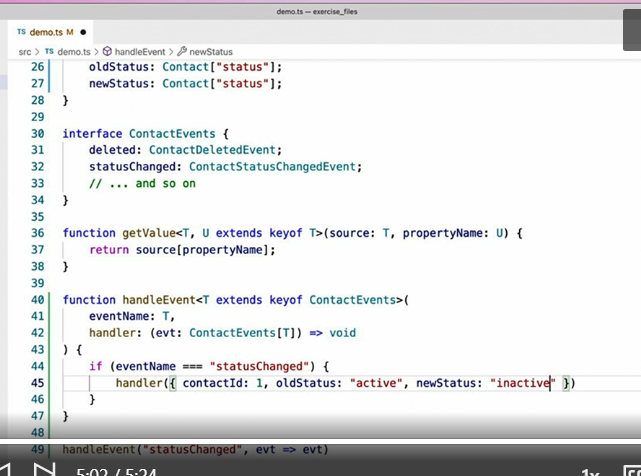
* For example, if I change the index to point to the address property of the contact type, which itself is a complex type, I can keep using the same syntax again to drill down into the address type, like when I choose the postalCode property, I can see that the Awesome type now matches the postalCode type, string.
* Now let's turn our attention to the getValue method here on line 36.
* But first, let me change what this function does just a little bit.
* Take a look at the handful of contact event interfaces I've defined here.



* I've also collected all these types into yet another type called ContactEvents, which you can think of as one place to find all of the events having to do with a contact.
* With that in mind, I'm going to change the getValue method not to read a value from an object but execute a method.
* Just bear with me.
* Okay, I lied.
* I'm not going to use that getValue method at all.
* But I lied for a good reason.
* You see, I know this looks pretty complex but hopefully, it'll be a little easier to digest since you watched me convert the method I already explained.
* Whether you're mad at me or not, let's go through this.
* First, I define a generic type T using the key of constraint like I showed in the earlier video.
* Then I use this generic type as the type of the first parameter to the method, eventName.
* In other words, this parameter will accept a string containing the name of any of the properties you can find on the ContactEvents type, deleted, StatusChanged, et cetera.
* Then comes the interesting part.
* The second parameter here is a function, which accepts a single parameter and that parameter matches the type of the property indicated by the first parameter, eventName.
* To really appreciate it and actually make sense of what I just said, you really have to see it in action.
* Here I've called the method and asked to handle the statusChanged eventName.
* This is nothing new.
* You've seen this already.
* The magic comes in the second parameter.
* Notice how TypeScript has determined the type of this event parameter is ContactStatusChangedEvent, even though I didn't specify it in this call.
* That's because I used index access of the ContactEvents type to figure it out on its own.
* And of course, TypeScript is also smart enough to figure out the types inside the method as well.



* Here I've handled the statusChanged event whose handler expects a ContactStatusChangedEvent, however, the code I've written tries to pass the handler a ContactDeletedEvent, which is not correct, so TypeScript yells at me.



* However, the moment I provide an object that matches the correct ContactStatusChangedEvent interface, TypeScript is happy and those errors go away.
* Hopefully you can see now that even though it may have seemed like a somewhat trivial syntax, the indexed access typing technique can actually help produce really advanced and powerful type checking.