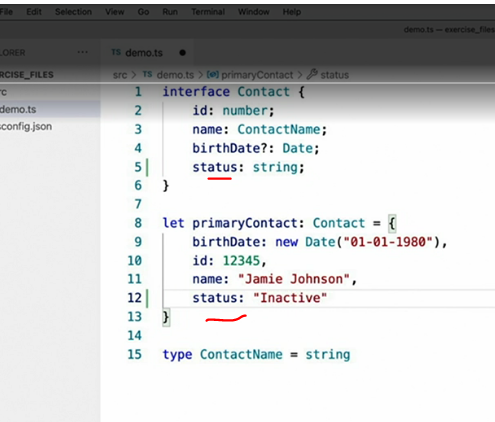
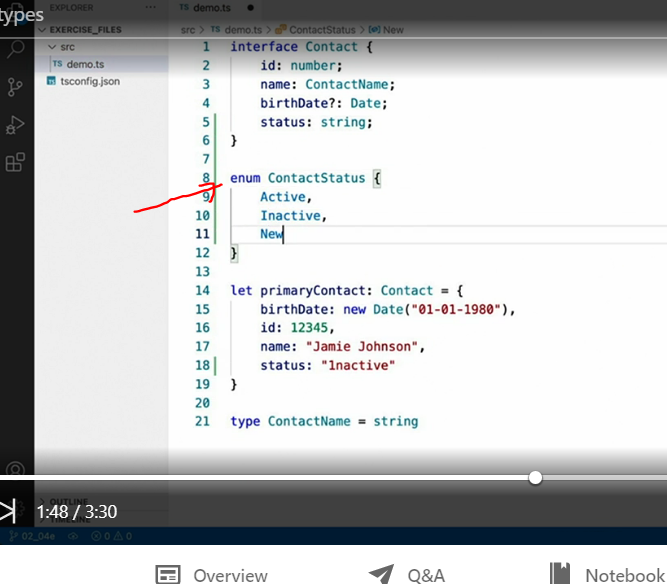
* - [Educator] When you write code, you'll often need to reference various kinds of hard-coded values.
* For instance, let's say that in our application a contact can also have a status value, which indicates the various states that a contact can be in, such as Active, Inactive, New, or otherwise.
* In order to add this property, I could define it as a string property, like this, which would allow me to assign the values I need, like this, like Active or Inactive.



* The problem with this approach is that the string type allows me to assign any string such as, or consider a more nefarious example.
* This one is a misspelling with a 1 instead of an I that would introduce a variety of bugs that would be very difficult to spot.
* Now, our fictitious company and our application may be concerned about happy customers and maybe we hope that as developers we never make typos, but I don't ever want to allow either of those two values to be assigned to this field and TypeScript can help me with that in a few ways.
* The first technique that TypeScript offers to avoid mistakes like these is called an **enum.**
* ***Enums are a special type that has a hard-coded list of values and is defined like this.***
* Once an enum is defined, I can refer to it just as I would any other type.



* In our contact case I can replace my string type with the ContactStatus enum.

Graphical user interface, text, application

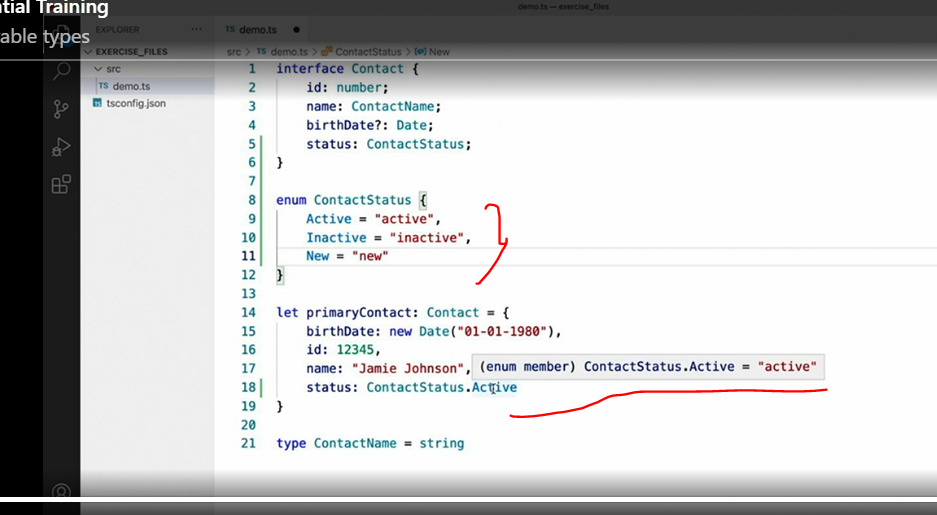
Description automatically generated

* Unlike many other TypeScripts syntaxes that get stripped away at compile time, enums do get compiled along with the rest of your code.
* This allows you to refer to these types at runtime like this.
* This is not a drop in replacement for string values however.

Text, timeline

Description automatically generated

* If I hover over the value, I see that TypeScript tells me that this enum represents the value zero since it's the first value that I defined in the enum.
* TypeScript does ***however allow you to use string values in your enums.***
* It's as simple as this.
* Here I've used the same words in lowercase.
* However, you can use whatever string or number values you like, as long as every value in the enum is the same type.



* With this change in place, I can now hover over the value again and see that the value at runtime will be the *string value Active.*
* This value and all other enum values you define will be the same everywhere you use them.
* More importantly, because TypeScript knows all of the possible values, it can give you much better autocomplete suggestions and even output compilation errors if you manage to use an enum property that doesn't exist or is spelled wrong.