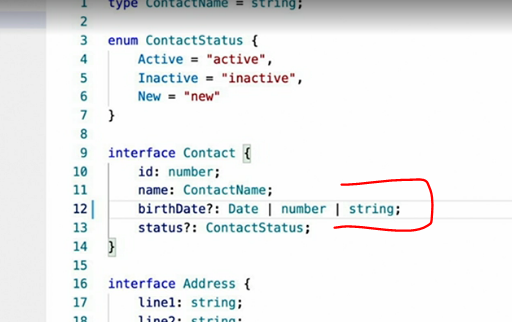
* - [Instructor] In a previous video, I showed you how to use the type keyword to define simple type aliases or pointers to another type like the one online one here.

Graphical user interface, text, application

Description automatically generated

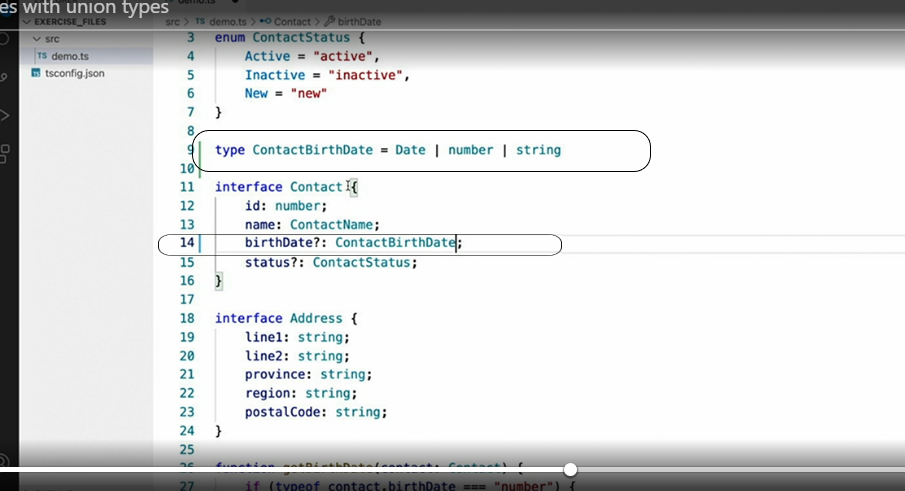
* *Type aliases referring to a single value* like this are helpful.
* ***However, defining type aliases that combine multiple types together to create new, more dynamic and powerful types is far more useful.***
* For example, let's revisit the contact type that I've been building throughout this course.
* Here on line 11, I've used a simple type alias called ContactName to give a better name to a string type.
* However, this interface also includes an even better example in the birthDate field.
* Right now, this field is defined as the JavaScript date type.
* However, this field is a great example of how many applications take advantage of JavaScript's dynamic nature to allow fields like this to hold both a date value and a number value which is the output of the date.
* now function.
* Or perhaps some applications may even want to accept a string value in this field as well.
* As it's defined now, this contact interface would not accept anything but a date value.
* However, I can update this field to accept any of those three types **using the pipe syntax** like this.
* This syntax **allows you to provide a list of as many types as you want to support in a field.**



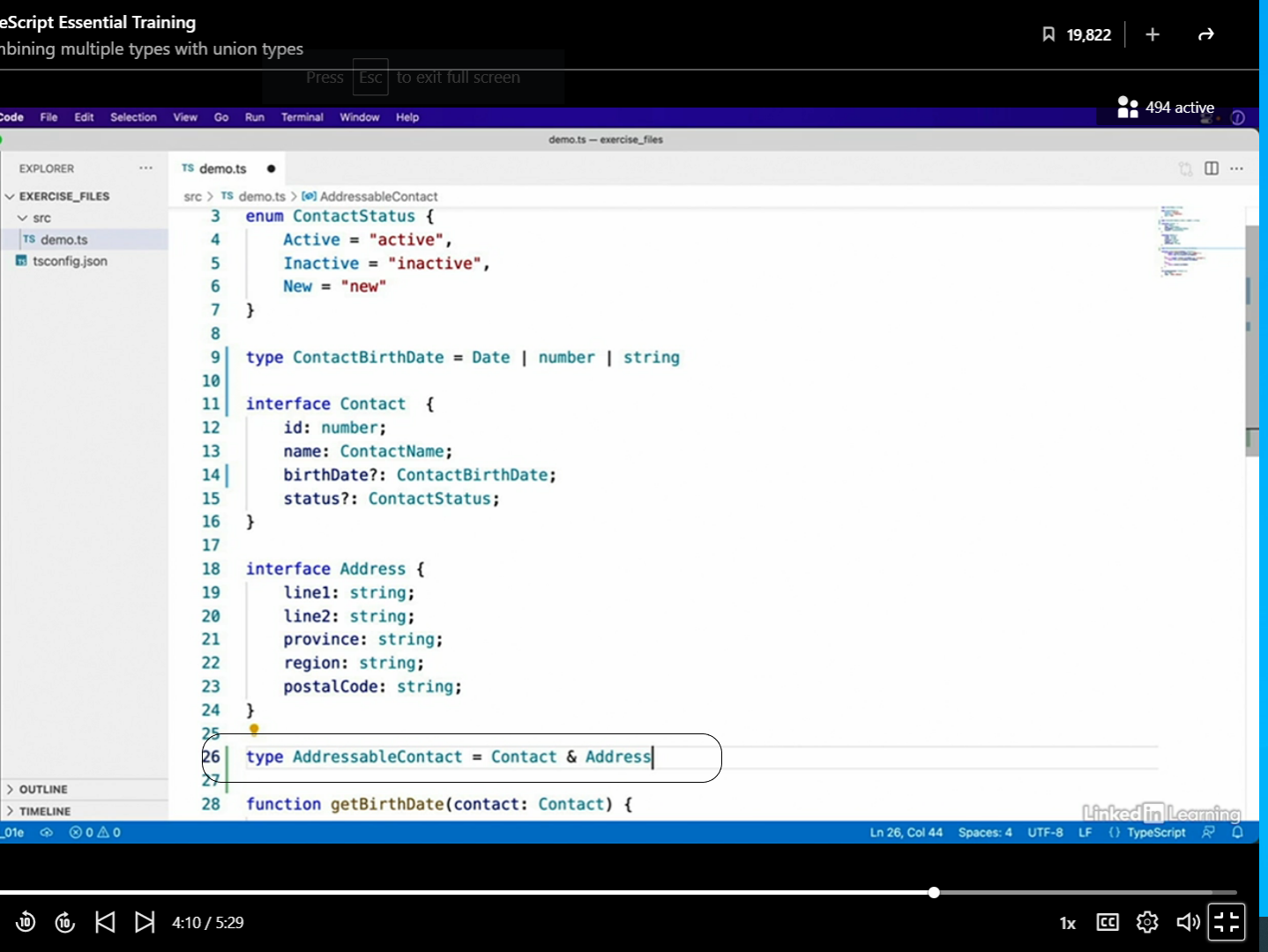
* What's more, TypeScript is intelligent enough to apply this information to the logic in your code.
* For example, take a look at this function that might operate on this field.
* This function uses standard JavaScript operators to inspect the type of the values at runtime to determine how to treat them.
* The first two conditions contain logic to handle number and string values.



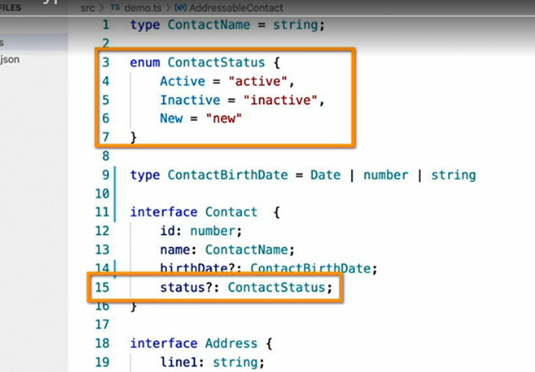
* And since we've already told TypeScript that this value can only be one of three types and we've already handled two of those types, then TypeScript is smart enough to derive that this last case must be the date type.
* Now that we've seen how powerful this approach can be, *let's take this type that's been defined inline and use a type alias to give it a name.*



* I'll simply copy it, use the type keyword to create an alias and then paste the inline type.
* I'll replace the existing definition with our new type alias.
* ***Another use of the type alias syntax is to combine multiple types together to create an entirely new type.***
* For example, I'll revisit the interfaces video where I use the **extends** keyword to extend the contact interface from the address interface to bring in all of the fields from the address interface into a contact.
* But what if **we want to have two different types of contacts: one with address properties and one without?** Well I could use the **extends** keyword to create a new interface for that.
* (instructor typing) Or since this is a situation where I'm not actually defining any additional fields, I could use another type alias syntax to accomplish the same thing.



* **Unlike the pipe syntax which provides alternative types, using the ampersand syntax like this combine the two types together to create an entirely new type.**
* One last usage for type aliases also happens to be my favorite and one that I use all the time as an alternative to enum values.
* To demonstrate, let's revisit the enum example I gave previously.

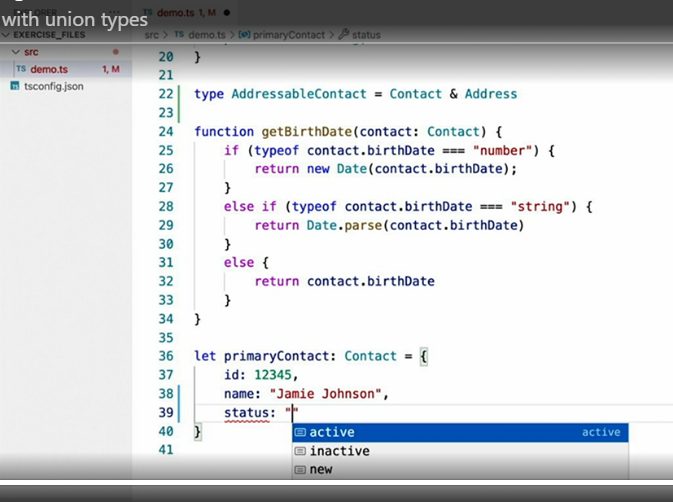


* In this example, I introduced an enum type to define the three valid string values that could be used for this field: active, inactive and new.
* The downside of this approach as I mentioned in that video, is **that defining enums like this actually adds more code to your final output.**
* Often that may be what you want.
* However, in examples like this where you want to just restrict certain fields to a limited set of known values, I find **type aliases to be far more effective.**
* To rewrite this enum using a type alias, simply change it to this.

Text

Description automatically generated

* And now when you use it, (instructor typing) you no longer have to reference the enum type because using a type alias for this field will allow TypeScript to restrict those values for you.



* All you have to do is type the string values directly.
* And there it is, my favorite alternative to enum values.