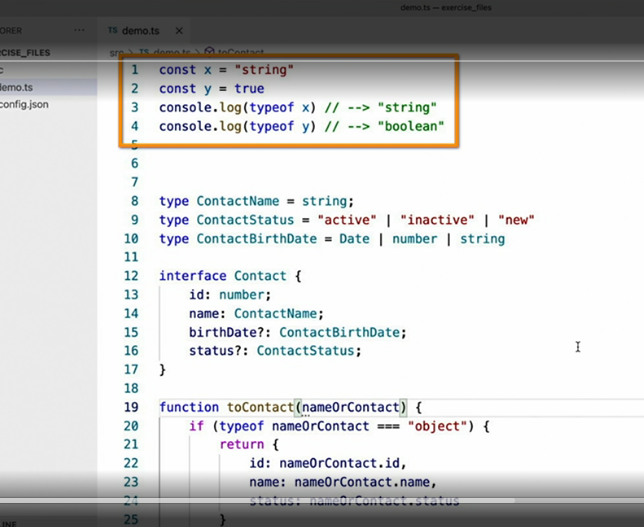
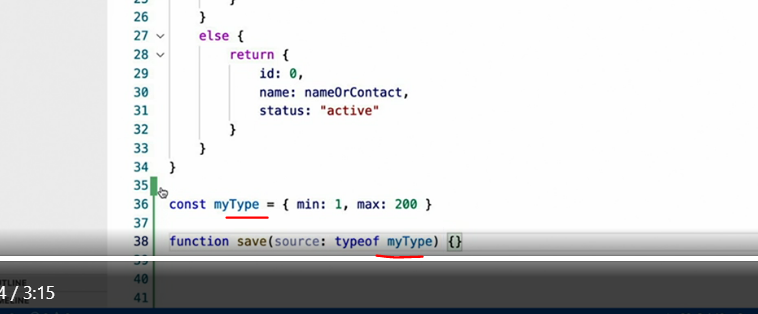
* - [Instructor] In this video, I'm going to show you how to take advantage of a JavaScript feature that's been around for a long time, the **typeof** operator.
* If you've never used JavaScript's typeof operator, it looks like this.



* If I were to execute the code on lines one through four, I would see in the console, the word string and Boolean printed out because those are the types of the X and Y variables.
* Now, this is typical JavaScript behavior.
* It has nothing to do with type script.
* In fact, I've been writing JavaScript code like this for years.
* This is, once again, another example of using JavaScript's dynamic typing system to write very flexible logic.
* In this case, this function accepts a single parameter, named name or contact, the value of which, from what we can tell from the code, it expects to be either a string or a contact object, but here's the problem.
* We can understand, from looking at this code, that the parameter should be either a string or a contact value, but type script can't.
* So let's apply some static typing to this method signature Here, I've said that this function returns a contact type, and I've used a union type to indicate that it accepts either a string or a contact value as a parameter.



* When I do this, something interesting happens.
* Because type script now knows that the parameter can only be one of those two types, it can now make much smarter inferences, such as the fact that if the JavaScript typeof operator returns object, then the parameter must be a contact, or if it's something other than an object then the value must be a string value, but that's not all.
* ***Type script also allows you to use the typeof operator to define or determine static types dynamically.***
* Consider the following example.
* First, I define a variable and assign it to a JavaScript object.
* I don't explicitly set this variable type, but type script is able to infer what it is nonetheless.
* That's great, but how do I refer to this same type later? Well, like this.
* With this code, I can make sure that any time someone tries to call this save function, the parameter that they passed must match the same structure as what I've defined on the myType variable on line 36.



* In all honesty, in most cases, (laughs) I would probably just elect to simply create an explicit interface or a type alias to address this situations but sometimes, using the typeof operator in this way is a helpful shortcut.
* However, this approach can be useful in that more dynamic code that you write when you're accepting values of types you don't actually know about until run time.