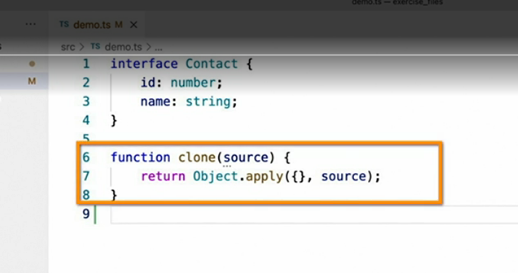
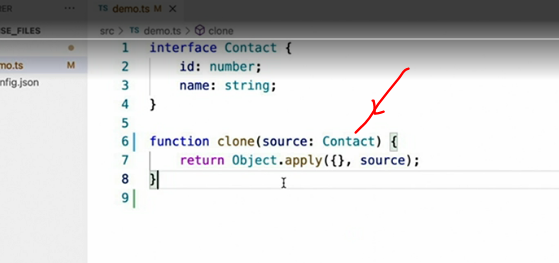
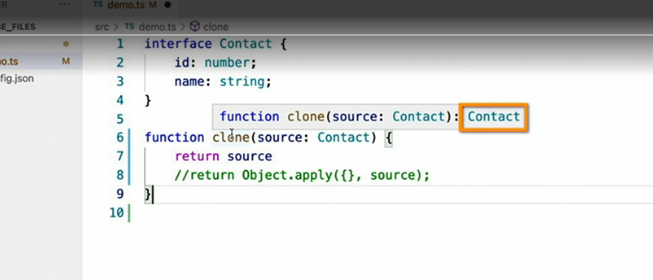
* - [Instructor] In a previous video, I showed you how to apply type information to variables, but seeing as how functions are a very important part of JavaScript development, it's nice to know that this same syntax can be applied to them as well.
* For example, this is a JavaScript function that ***accepts an object, and returns a clone of that object.***



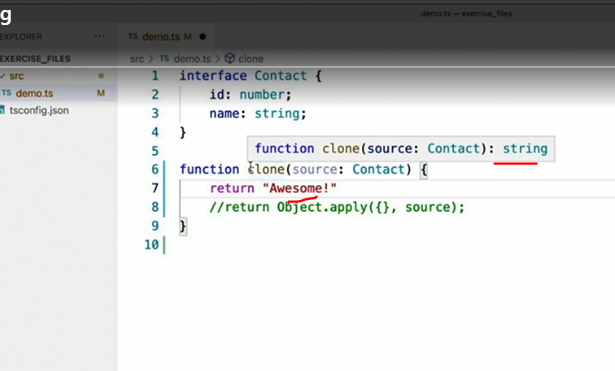
* If we wanted to restrict this method to only allow contact types as input parameters, we could simply **apply a type to the parameter** like this.



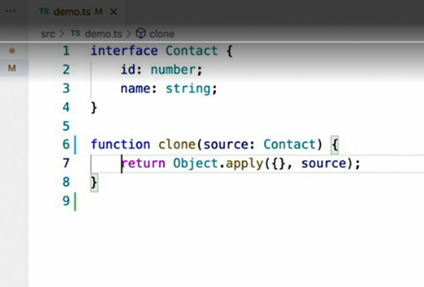
* (keyboard keys tapping) Now, for the return type, since TypeScript is incredibly intelligent about reading your code and inferring types, for many functions, you can usually get away with not supplying a return type at all.



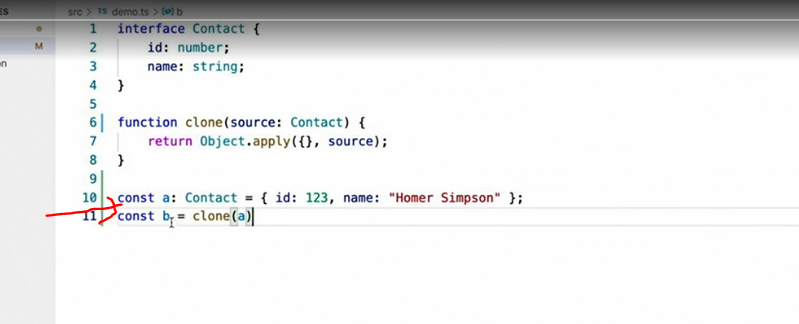
* For example, if this method were simply to return the parameter it was given.
* (keyboard keys tapping) I can see that TypeScript has correctly determined the return type will be contact.
* Likewise, if I were to return a string value.



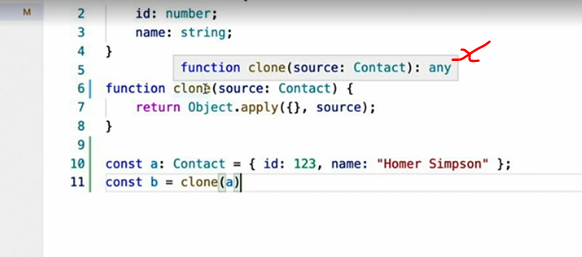
* (keyboard keys tapping) TypeScript says the return type is string, but let's go back to the original code.



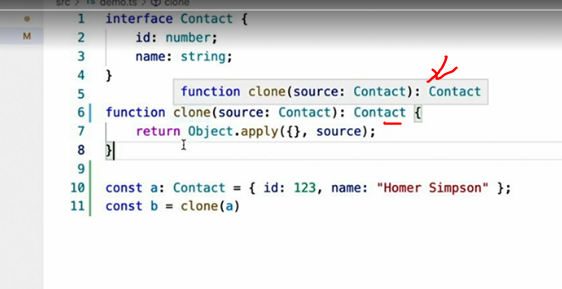
* Now, it's obvious to us as JavaScript developers that the return type of this function should match the type of the parameter passed into it.
* In other words, given this code.



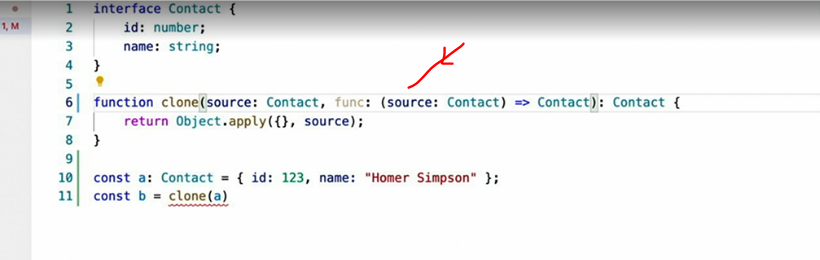
* (keyboard keys tapping) It seems clear that if the type of variable A is contact, and the variable B is a copy of variable A, then the type of variable B should also be contact.
* However, that's not the case.



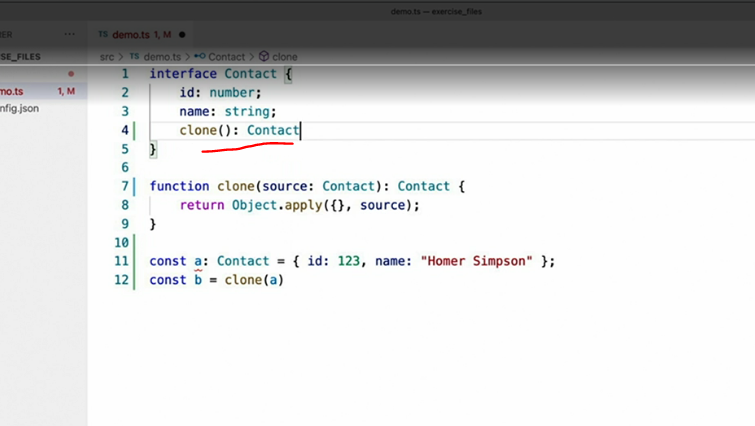
* Since the object.
* apply function used in this clone function returns the any type, that's what TypeScript infers the return type of this whole function to be.
* And since we're assigning that value to variable B, that's the type that TypeScript infers variable B to be as well.
* To fix this, we can apply the same type syntax to state the return value of the function that we apply to variables.



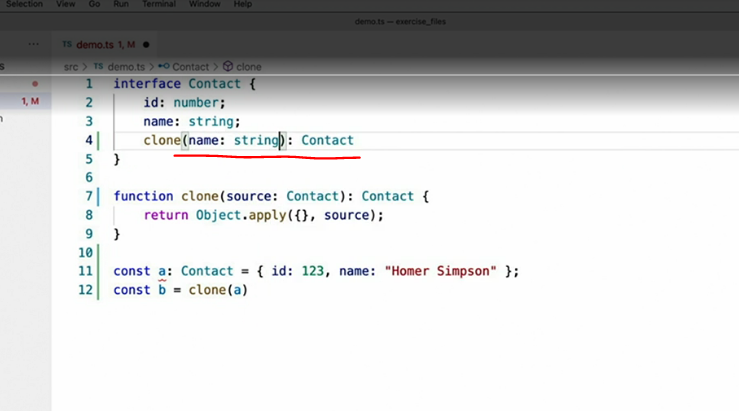
* (keyboard keys tapping) Now, everything is correct.
* In JavaScript, **functions can also be passed as variables.**
* For example, we can rewrite this clone function to **accept a function as a parameter,** rather than an object.
* (keyboard keys tapping) We can, of course, provide type information for function variables as well.



* It looks like this.
* (keyboard keys tapping) This resembles a function signature.
* It starts with a list of parameters contained within a pair of parentheses.
* In this case, a single parameter named source with the type of contact, followed by equals, greater than, and then the return value, contact.
* Finally, there is one last function syntax, and that is the way that you would define a method on an interface.
* It looks like this.



* (keyboard keys tapping) This syntax is similar to what we just saw, but with a colon instead of the arrow syntax we just used.
* Likewise, if this method were to accept any parameters, they would be defined like before.



* (keyboard keys tapping) Now, our functions are fully typed.