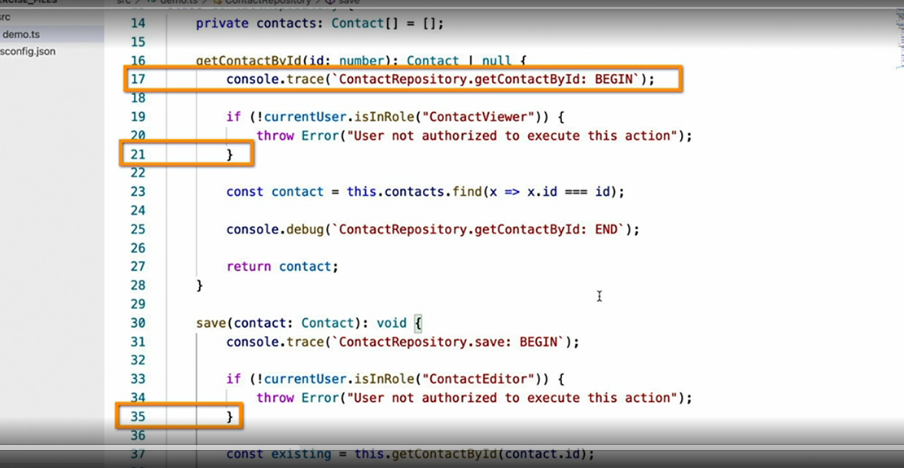
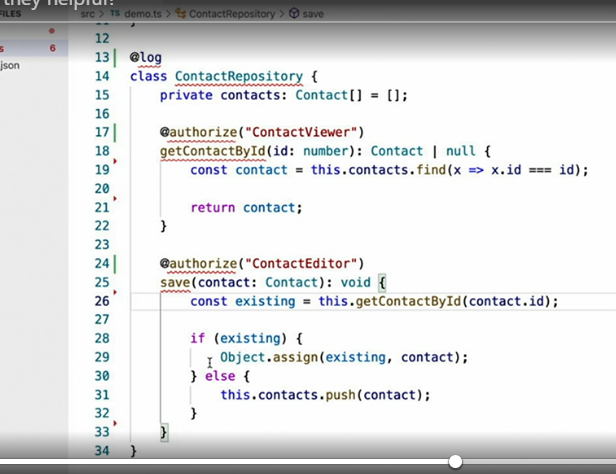
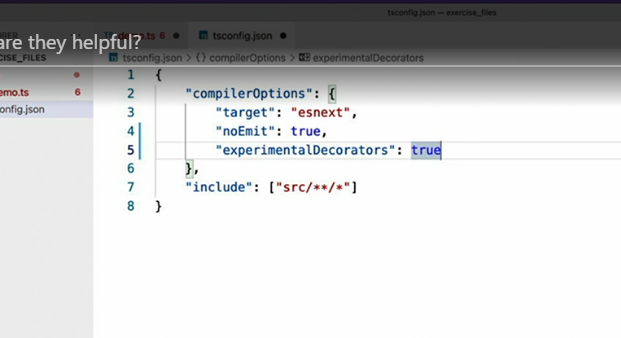
* - [Instructor] In this chapter, I'm going to show how to use an incredibly powerful type-script feature named **decorators**.
* The easiest way to understand decorators is to think of them as **metadata** that you can add to your classes, methods and even, getter and setter properties.
* In fact, you can add far more than just information.
* You can actually extend these elements with additional behavior, allowing you to add that behavior to your code without actually changing your code.
* The typical example of decorators in action is a logging decorator.
* Let me show you.



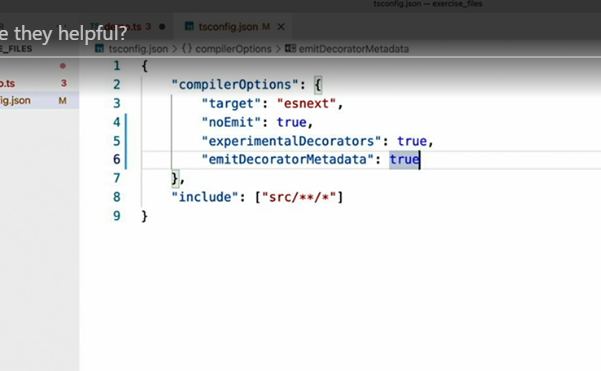
* This example class has logging statements at the beginning and end of each function.
* This helps diagnose problems more quickly because you can determine which functions have run and whether or not they completed.
* It also features a mechanism that protects the method from being called by users who don't have proper access.
* Though this is the behavior that we want, there are a few things that are less than ideal about this code the way that it's written now.
* First off, this has a decent amount of duplicated code.
* Next, it's tedious to write, not to mention error prone, considering the fact that developers are required to type the method name free hand and finally, this logging and auditing code is an auxiliary function.
* It doesn't actually have anything to do with the actual purpose and function of these methods, which is to save and retrieve contacts in this case.
* In other words, it's just noise.
* This scenario, however, is a perfect opportunity to apply decorators to both centralize and simplify this code.
* For example, what if we could just write this and then eliminate the logging statements? and this and eliminate access control statements? Doesn't this code look so much cleaner and easier to read? Plus, it has the added benefit of being able to easily change how and where those log and audit entries are stored.



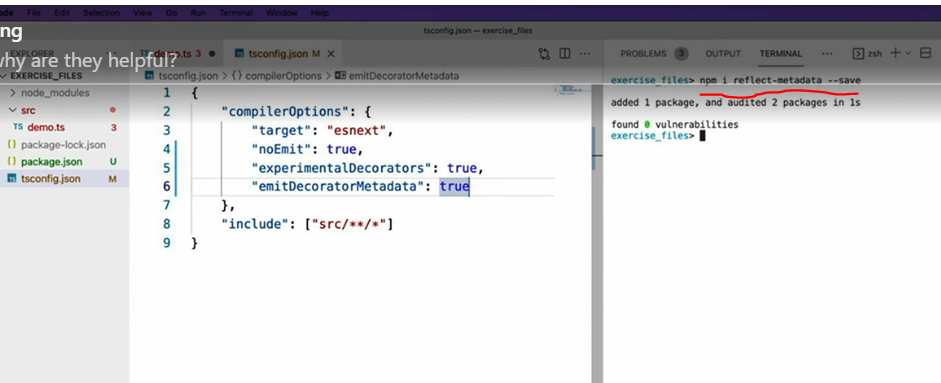
* Plus, it has the added benefit of being able to easily change how and where those log entries are stored or how we apply our authorization logic.
* These are exactly the kinds of scenarios where decorators shine, and in this chapter, I'll show you how to create these decorators.
* Before I continue, however, I must warn you that **decorators are actually a proposed addition to the JavaScript language** itself, and as this proposal evolves, it's final implementation may actually begin to differ from typescripts implementation.
* If you're worried about writing code that may conflict with a future JavaScript standard, then you may want to simply view this chapter to learn what decorators can do but wait to implement the techniques in your own code until the standard has been finalized.
* If, however, you trust that type script will support you in writing code that will at least compile to JavaScript compatible code, then by all means, please continue on.
* I certainly use decorators in my code.
* In order to really make this point clear, type script, (chuckles) actually encourages developers to opt in to decorator support via a **configuration setting.**
* So be sure to do that right now.
* Simply, open up your tsconfig file and set the experimentalDecorator settings to true.



* There's also one more thing, which is not technically required to use decorators, but it's a really valuable tool, and that is **runtime metadata.**
* I'll explain exactly what this is in later videos, but while we're here, let's go ahead and add support for it, so it will be available to us later.



* To enable it, you'll first need to set another setting, emitDecoratorMetadata, to true, and then you'll need to install the **reflect-metadata** library like so.



* This library implements polyfills for another set of proposed ECMAScript features, which are considered experimental now, but should hopefully be approved and implemented around the same time that decorators become widely available and now, with all this code in place, continue on to the next video where I'll show you how to create your first decorator.