```
/// <reference path="angular.d.ts" />
angular.module(10, []); // the module name should be a string
// so when I compile, I get:
// Argument of type 'number' is not assignable to parameter of type 'string'.
```

`/// <reference path="angular.d.ts" />` is a special comment recognized by TS, telling the compiler to look for the interface `angular.d.ts`. Now, if you misuse an AngularJS method, the compiler will complain, and you can fix it on the spot, without having to manually run your app!

Even cooler, since TypeScript 1.6, the compiler will auto-discover the interfaces if they are packaged in your `node_modules` directory in the dependency. More and more projects are adopting this approach, and so is Angular. So you don't even have to worry about including the interfaces in your Angular project: the TS compiler will figure it out by itself if you are using NPM to manage your dependencies!

5.9. Decorators

This is a fairly new feature, added only in TypeScript 1.5, to help supporting Angular. Indeed, as we will shortly see, Angular components can be described using decorators. You may not have heard about decorators, as not every language has them. A decorator is a way to do some meta-programming. They are fairly similar to annotations which are mainly used in Java, C# and Python, and maybe other languages I don't know. Depending on the language, you add an annotation to a method, an attribute, or a class. Generally, annotations are not really used by the language itself, but mainly by frameworks and libraries.

Decorators are really powerful: they can modify their target (method, classes, etc.), and for example alter the parameters of the call, tamper with the result, call other methods when the target is called or add metadata for a framework (which is what Angular decorators do). Until now, it was not something possible in JavaScript. But the language is evolving and there is now an official proposal for `decorators`, which may be standardized one day in the future (possibly in ES7/ES2016). Note that the TypeScript implementation goes slightly further than the proposed standard.

In Angular, we will use the decorators provided by the framework. Their role is fairly basic: they add some metadata to our classes, attributes or parameters to say things like "this class is a component", "this is an optional dependency", "this is a custom property", etc. It's not required to use them, as you can add the metadata manually (if you want to stick to ES5 for example), but the code will definitely be more elegant using decorators, as provided by TypeScript.

In TypeScript, decorators start with an `@`, and can be applied to a class, a class property, a function or a function parameter. They can't be applied to a constructor, but can be applied to its parameters.

To have a better grasp on this, let's try to build a simple decorator, `@Log()`, that will log something every time a method is called.

It will be used like this: