Chapter 19: How to use a JavaScript library without a type definition file

While some existing JavaScript libraries have type definition files, there are many that don't.

TypeScript offers a couple patterns to handle missing declarations.

Section 19.1: Make a module that exports a default any

For more complicated projects, or in cases where you intend to gradually type a dependency, it may be cleaner to create a module.

Using JQuery (although it does have typings available) as an example:

```
// place in jquery.d.ts
declare let $: any;
export default $;
```

And then in any file in your project, you can import this definition with:

```
// some other .ts file
import $ from "jquery";
```

After this import, `$` will be typed as `any`.

If the library has multiple top-level variables, export and import by name instead:

```
// place in jquery.d.ts
declare module "jquery" {
   let $: any;
   let jQuery: any;

   export { $ };
   export { jQuery };
}
```

You can then import and use both names:

```
// some other .ts file
import {$, jQuery} from "jquery";

$.doThing();
jQuery.doOtherThing();
```

Section 19.2: Declare an any global

It is sometimes easiest to just declare a global of type `any`, especially in simple projects.

If jQuery didn't have type declarations (it does), you could put

```
declare var $: any;
```

Now any use of `$` will be typed `any`.

Section 19.3: Use an ambient module

If you just want to indicate the *intent* of an import (so you don't want to declare a global) but don't wish to bother with any explicit definitions, you can import an ambient module.

```
// in a declarations file (like declarations.d.ts)
declare module "jquery";    // note that there are no defined exports
```

You can then import from the ambient module.

```
// some other .ts file
import {$, jQuery} from "jquery";
```

Anything imported from the declared module (like $ and `jQuery`) above will be of type `any`