```
import { NgModule } from '@angular/core';

@NgModule({
})
export class AppModule { }
```

Like the `@Component` decorator, it takes a configuration object.

As we are building an app for the browser, the root module imports `BrowserModule`. This is not the only target possible for Angular, you could choose to render the app on the server for example, and therefore import another module. `BrowserModule` contains all kinds of useful stuff we will use later. A module can choose to *export* components, directives and pipes. When you import a module, you will make all the directives, components and pipes exported by the imported module usable in your module. Our root module won't be imported in another module, so we don't have `exports`, but we will have several `imports` in the end.

The terminology is not beginner friendly here. We were talking about ES6 and TS modules in the first chapters, which define imports and exports. And now we are talking about Angular modules, which also have imports and exports… I'm not a fan of having the same terms for different things, so let me explain a little bit more.

You can see an ES6 or TS import purely as a language feature, like an import statement in Java: it allows using the imported classes/functions in your source code. It also declares a dependency for the bundler or module loader (Webpack or SystemJS, for example), which knows that if `a.ts` is loaded, then `b.ts` must also be loaded since `a` imports `b`. You have to use imports and exports with ES6 and TypeScript, whether or not you're using Angular or any other framework.

On the other hand, importing an Angular module (for example `BrowserModule`) in your own Angular module (`AppModule`), has a functional meaning. It tells Angular: all the components, directives and pipes that are exported by `BrowserModule` should be made available to my Angular components/templates. It has no special meaning for the TypeScript compiler.

Back to `NgModule`: in its configuration object, we must declare the components that belong to our root module, with the `declarations` field. You can see it contains the previous component we talked about: `AppComponent`.