**Angular** reads the HTML within that root and compiles it into an internal representation. This reading and compiling is the **bootstrapping** process. Manual **bootstrapping** is when you write code to execute the **bootstrapping** process instead of using the ng-app directive.

A basic understanding of the following:

* JavaScript Modules vs. NgModules.

An NgModule describes how the application parts fit together. Every application has at least one Angular module, the *root* module that you bootstrap to launch the application. By convention, it is usually called AppModule.

If you use the Angular CLI to generate an app, the default AppModule is as follows:

/\* JavaScript imports \*/

import { BrowserModule } from '@angular/platform-browser';

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

import { FormsModule } from '@angular/forms';

import { HttpClientModule } from '@angular/common/[http](https://angular.io/api/common/http)';

import { AppComponent } from './app.component';

/\* the AppModule class with the @[NgModule](https://angular.io/api/core/NgModule) decorator \*/

@NgModule({

declarations: [

AppComponent

],

imports: [

[BrowserModule](https://angular.io/api/platform-browser/BrowserModule),

[FormsModule](https://angular.io/api/forms/FormsModule),

[HttpClientModule](https://angular.io/api/common/http/HttpClientModule)

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

After the import statements is a class with the **@**[**NgModule**](https://angular.io/api/core/NgModule) [decorator](https://angular.io/guide/glossary#decorator).

The @[NgModule](https://angular.io/api/core/NgModule) decorator identifies AppModule as an [NgModule](https://angular.io/api/core/NgModule) class. @[NgModule](https://angular.io/api/core/NgModule) takes a metadata object that tells Angular **how to compile and launch the application**.

* ***declarations***—this application's lone component.
* ***imports***—import [BrowserModule](https://angular.io/api/platform-browser/BrowserModule) to have browser specific services such as DOM rendering, sanitization, and location.
* ***providers***—the service providers.
* ***bootstrap***—the *root* component that Angular creates and inserts into the index.html host web page.

The default application created by the Angular CLI only has one component, AppComponent, so it is in both the declarations and the bootstrap arrays.

**The declarations array**

The module's declarations array tells Angular which components belong to that module. As you create more components, add them to declarations.

You must declare every component in exactly one [NgModule](https://angular.io/api/core/NgModule) class. If you use a component without declaring it, Angular returns an error message.

The declarations array only takes declarables. Declarables are components, [directives](https://angular.io/guide/attribute-directives) and [pipes](https://angular.io/guide/pipes). All of a module's declarables must be in the declarations array. Declarables must belong to exactly one module. The compiler emits an error if you try to declare the same class in more than one module.

These declared classes are visible within the module but invisible to components in a different module unless they are exported from this module and the other module imports this one.

An example of what goes into a declarations array follows:

declarations: [

YourComponent,

YourPipe,

YourDirective

],

A declarable can only belong to one module, so only declare it in one @[NgModule](https://angular.io/api/core/NgModule). When you need it elsewhere, import the module that has the declarable you need in it.

**Only @**[**NgModule**](https://angular.io/api/core/NgModule) **references** go in the imports array.

**Using directives with @**[**NgModule**](https://angular.io/api/core/NgModule)

Use the declarations array for directives. To use a directive, component, or pipe in a module, you must do a few things:

1. Export it from the file where you wrote it.
2. Import it into the appropriate module.
3. Declare it in the @[NgModule](https://angular.io/api/core/NgModule) declarations array.

Those three steps look like the following. In the file where you create your directive, export it. The following example, named ItemDirective is the default directive structure that the CLI generates in its own file, item.directive.ts:

src/app/item.directive.ts

import { [Directive](https://angular.io/api/core/Directive) } from '@angular/core';

@[Directive](https://angular.io/api/core/Directive)({

selector: '[appItem]'

})

export class ItemDirective {

// code goes here

constructor() { }

}

The key point here is that you have to export it so you can import it elsewhere. Next, import it into the NgModule, in this example app.module.ts, with a JavaScript import statement:

src/app/app.module.ts

import { ItemDirective } from './item.directive';

And in the same file, add it to the @[NgModule](https://angular.io/api/core/NgModule) declarations array:

src/app/app.module.ts

declarations: [

AppComponent,

ItemDirective

],

Now you could use your ItemDirective in a component. This example uses AppModule, but you'd do it the same way for a feature module. For more about directives, see [Attribute Directives](https://angular.io/guide/attribute-directives) and [Structural Directives](https://angular.io/guide/structural-directives). You'd also use the same technique for [pipes](https://angular.io/guide/pipes) and components.

Remember, components, directives, and pipes belong to one module only. You only need to declare them once in your app because you share them by importing the necessary modules. This saves you time and helps keep your app lean.

**The imports array**

The module's imports array appears exclusively in the @[NgModule](https://angular.io/api/core/NgModule) metadata object. It tells Angular about other NgModules that this particular module needs to function properly.

This list of modules are those that export components, directives, or pipes that the component templates in this module reference. In this case, the component is AppComponent, which references components, directives, or pipes in [BrowserModule](https://angular.io/api/platform-browser/BrowserModule), [FormsModule](https://angular.io/api/forms/FormsModule), or [HttpClientModule](https://angular.io/api/common/http/HttpClientModule). A component template can reference another component, directive, or pipe when the referenced class is declared in this module or the class was imported from another module.

**The providers array**

The providers array is where you list the services the app needs. When you list services here, they are available app-wide. You can scope them when using feature modules and lazy loading. For more information, see [Providers](https://angular.io/guide/providers).

**The bootstrap array**

The application launches by bootstrapping the root AppModule, which is also referred to as an entryComponent. Among other things, the bootstrapping process creates the component(s) listed in the bootstrap array and inserts each one into the browser DOM.

Each bootstrapped component is the base of its own tree of components. Inserting a bootstrapped component usually triggers a cascade of component creations that fill out that tree.

While you can put more than one component tree on a host web page, most applications have only one component tree and bootstrap a single root component.

This one root component is usually called AppComponent and is in the root module's bootstrap array.

# JavaScript Modules vs. NgModules

JavaScript and Angular use modules to organize code, and though they organize it differently, Angular apps rely on both.

## JavaScript modules

In JavaScript, modules are individual files with JavaScript code in them. To make what’s in them available, you write an export statement, usually after the relevant code, like this:

export class AppComponent { ... }

Then, when you need that file’s code in another file, you import it like this:

import { AppComponent } from './app.component';

JavaScript modules help you namespace, preventing accidental global variables.

For more information on JavaScript modules, see [JavaScript/ECMAScript modules](https://hacks.mozilla.org/2015/08/es6-in-depth-modules/).

## NgModules

NgModules are classes decorated with @[NgModule](https://angular.io/api/core/NgModule). The @[NgModule](https://angular.io/api/core/NgModule) decorator’s imports array tells Angular what other NgModules the current module needs. The modules in the imports array are different than JavaScript modules because they are NgModules rather than regular JavaScript modules. Classes with an @[NgModule](https://angular.io/api/core/NgModule) decorator are by convention kept in their own files, but what makes them an [NgModule](https://angular.io/api/core/NgModule) isn’t being in their own file, like JavaScript modules; it’s the presence of @[NgModule](https://angular.io/api/core/NgModule) and its metadata.

The AppModule generated from the [Angular CLI](https://angular.io/cli) demonstrates both kinds of modules in action:

/\* These are JavaScript import statements. Angular doesn’t know anything about these. \*/

import { [BrowserModule](https://angular.io/api/platform-browser/BrowserModule) } from '@angular/platform-browser';

import { [NgModule](https://angular.io/api/core/NgModule) } from '@angular/core';

import { AppComponent } from './app.component';

/\* The @[NgModule](https://angular.io/api/core/NgModule) decorator lets Angular know that this is an NgModule. \*/

@[NgModule](https://angular.io/api/core/NgModule)({

declarations: [

AppComponent

],

imports: [ /\* These are [NgModule](https://angular.io/api/core/NgModule) imports. \*/

[BrowserModule](https://angular.io/api/platform-browser/BrowserModule)

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

The NgModule classes differ from JavaScript module in the following key ways:

* An NgModule bounds [declarable classes](https://angular.io/guide/ngmodule-faq#q-declarable) only. Declarables are the only classes that matter to the [Angular compiler](https://angular.io/guide/ngmodule-faq#q-angular-compiler).
* Instead of defining all member classes in one giant file as in a JavaScript module, you list the module's classes in the @[NgModule.declarations](https://angular.io/api/core/NgModule#declarations) list.
* An NgModule can only export the [declarable classes](https://angular.io/guide/ngmodule-faq#q-declarable) it owns or imports from other modules. It doesn't declare or export any other kind of class.
* Unlike JavaScript modules, an NgModule can extend the entire application with services by adding providers to the @[NgModule.providers](https://angular.io/api/core/NgModule#providers) list.