Writing our first app

We want to create a simple app that displays a joke to the user.

Learning Outcomes

- That *Components* are the building blocks of an Angular application.
- What annotations are and how to use them in TypeScript.
- How to import code from other files so we can use it in our file.
- How to package our application into an Angular Module.
- How to bootstrap an Angular application so is starts on a web page

Components

To begin writing our application we open up `script.ts` and create a class called `JokeComponent`.

```
class JokeComponent {
}
```

A `class` is a new feature of ES6 which we will explain in much more detail in the next section, but to summarise it's a blueprint for creating objects with specific functions and properties already attached to it.

The word `Component` isn't random. Components are a feature of Angular that let us create a new HTML *language* and they are how we structure Angular applications.

HTML comes with a bunch of pre-built tags like `<input>` and `<form>` which look and behave a certain way. In Angular we create new *custom* tags with their own look and behaviour.

An Angular application is therefore just a set of custom tags that interact with each other, we call these tags *Components*.

If you are coming from Angular 1 then Components are the same as Components in Angular 1.5 and the same as element directives in Angular <1.5

The code that controls a component we put into a class like the `JokeComponent` above, but how do we *link* this class with a HTML tag, say a tag called `<joke>`?

We use a new feature of TypeScript called *annotations*, and specifically an annotation called `@Component`, like so:

```
@Component({
  selector: 'joke'
})
class JokeComponent {
}
```

The `@Component` is an annotation, an annotation automatically adds some boilerplate code to the class, function or property its attached to.

You can write Angular without using annotations you would just have to write the boilerplate code yourself.

We are going to use other annotations later on, however the main one for working with components is `@Component`.

You can configure the @Component annotation by passing it an object with various parameters. In our example above @Component has one parameter called `selector`, this tells Angular which tag to link this class too.

By setting the selector to `joke` we've told angular that whenever it finds a tag in the HTML like `<joke></joke>` to use an instance of the JokeComponent class to control it.

Imports

Before we can use `@Component` though we need to import it, like so:

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

The line above is saying we want to import the `Component` code from the module `@angular/core`.

We leave to SystemJS to figure out *how* to load that component from @angular/core or even *where* @angular/core is.

The above might not look like javascript but it is, the `{ Component }` part is something called destructuring and that's a feature of ES6, more on that later.

If you are coming from a language like Python or Java you'll be used to the concept of imports. Basically we are pulling in dependencies from another file and making it available in this file.

Template

To use our brand new custom component we add the tag `<joke></joke>` to our HTML file, like so:

```
<body>
  <joke></joke>
</body>
```

So far this isn't doing much yet though, we want Angular to replace <joke></joke> with some template HTML. To do that we use another attribute of the Component decorator called template, like so:

```
@Component({
  selector: 'joke',
  template: '<h1>What did the cheese say when it looked in the mirror?</h1><p>Halloumi
(hello me)</p>'
})
```

That's hard to read though, the HTML is all written on one line but i'd like to read it on multiple lines.

There is a new feature of ES6 JavaScript called `template strings` which lets us define multi-line strings, lets use it:

```
@Component({
  selector: 'joke',
  template: `
  <h1>What did the cheese say when it looked in the mirror?</h1>
  <p>Halloumi (hello me)</p>
  `
})
```

The string uses a special character ` it's called a back-tick, we'll be digging into this in much more detail in the next section. For now just accept that it lets us define strings on multiple lines like the above.

Angular Modules

If we ran this code now we would see it's still not working!

We've defined a component with a custom tag, added the tag to our HTML but we haven't told Angular that we want to use Angular on this page.

To do that we need to do something called *bootstrapping*.

In Angular 1 when we added `np-app="module-name"` to the top of the HTML page it bootstrapped the application for us. When Angular 1 loaded it first checked for this tag, looked for the module that was associated with that tag and loaded the code from it. However with Angular we need to do all of this manually, for good reasons which we'll explain later.

In Angular your code is structured into `packages` called Angular Modules, or `NgModules` for short. Every app requires at least one module, the root module, that we call `AppModule` by convention.

We are using the term *module* for two different concepts. In JavaScript the term module generally refers to code which exists in a single file. An NgModule is a different concept, it combines code from different files together into one package. An NgModule therefore contains functionality from multiple files a module refers to functionality in a single file.

Lets create our root Angular Module, like so:

```
@NgModule({
  imports: [BrowserModule],
  declarations: [JokeComponent],
  bootstrap: [JokeComponent]
})
export class AppModule {
}
```

To define an Angular Module we first create a class and then annotate it with a decorator called `@NgModule`.

You'll notice this follows a similar pattern to when we created a component. We created a class and then annotated it with a decorator, this pattern is a common one in Angular.

`@NgModule` has a few params:

imports

The other Angular Modules that export material we need in this Angular Module. Almost every application's root module should import the BrowserModule.

declarations

The list of components or directives belonging to *this* module.

bootstrap

Identifies the root component that Angular should bootstrap when it starts the application.

We know `NgModule` but `BrowserModule` is the Angular Module that contains all the needed Angular bits and pieces to run our application in the browser.

Angular itself is split into separate Angular Modules so we only need to import the ones we really use. Some other common modules you'll see in the future are the `FormsModule`, `RouterModule` and `HttpModule`.

We also need to remember to import NgModule and BrowserModule, like so:

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

Bootstrapping

Now we have defined our root Angular Module called AppModule we need to bootstrap the application using it, like so:

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
.
.
.
platformBrowserDynamic().bootstrapModule(AppModule);
```

You might be asking yourself why Angular doesn't do this for us like it did in Angular 1?

In Angular bootstrapping is *platform specific*.

Angular 1 assumed that Angular would only ever be run in a browser, Angular makes no such assumption. We could be writing Angular code for a mobile device using a solution like Ionic. We could be loading up Angular on a node server so we can render HTML for web crawlers that don't run JavaScript.

Angular isn't limited to only working in the browser which is why we need to tell Angular exactly *how* we want it to bootstrap itself, in our case we are running in the browser so we use the platformBrowserDynamic function to bootstrap our application.

Component Tree

An Angular application is architected as a tree of Components stemming from one root Component.

Your root component is the component you configured on your root `NgModule` in the bootstrap property, so in our case it's the `JokeComponent`.

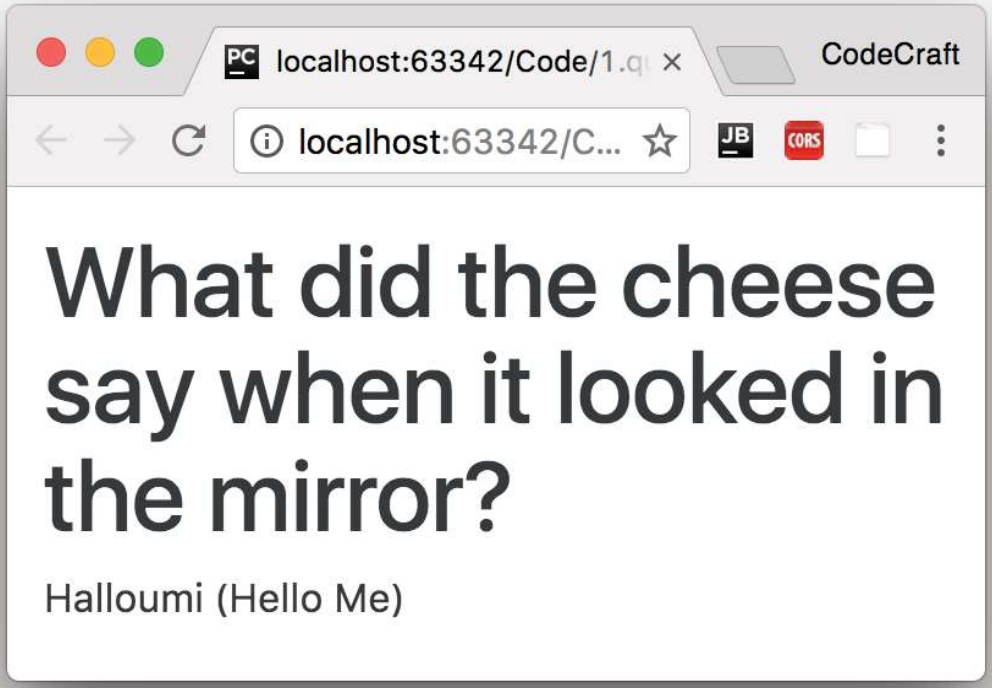
By bootstrapping with `JokeComponent` we are saying that it's the root component for our application.

In the template for our `JokeComponent` we would add tags for other Components, in the template for those Components we would add tags for others... and so on and so on.

However in our `index.html` file we will never see anything other than the tag for our root component, like so:

```
<body>
  <joke></joke>
</body>
```

Running the application in our browser we would see:



Troubleshooting

If when looking at the browser console you see see an error like the below:

```
> The selector "joke" did not match any elements
```

This means you forgot to add the tag for your root component to your `index.html` file.

Summary

A *Component* is the building block of an Angular application.

It lets us create a new HTML language of custom tags and link them with javascript classes which describe the behaviour of that tag.

An application is composed of a tree of such Components glued together all depending from one root component.

We package together related Components and supporting code into something called an Angular Module which we use to bootstrap Angular onto a webpage.