Components

main.ts

- This is the starting point of the application
- Angular exports a function named `platformBrowserDynamic` that can be used for targeting the browser. This comes from the platform browser dynamic scope package

```
import { platformBrowserDynamic }
  from '@angular/platform-browser-dynamic';
```

- The returned object has a `bootstrapModule` function that runs the bootstrap. This function is expecting a root module, in this case a module called `AppModule`.

```
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

app.module.ts

@NgModule Decorator

- Recall that a decorator is an expression that evaluates to a function allowing annotation of classes
- They allow us to configure/modify code like classes method and fields at design time

- A decorator called `NgModule` is used to inform Angular that the class code in here is intended to be an Angular module, it uses a decorator.
- Angular imports the `NgModule` definition from its core package.

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


app.module.ts

- The imports property is used to bring in other Angular modules that your module will need.
- The declarations property is used to make components, directives, and pipes available to other modules.
- The bootstrap property is used for a root module and lets Angular know which component or components will be the entry point for your app code.

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

app.module.ts

Browser

- We want to make use of the browser module that the Angular platform has available in the platform-browser scope package.
  - Contains directives, pipes, and other tools for working with the DOM
- The module is imported and added to the list of imports for our application

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

...

imports: [
  BrowserModule
],
```


app.module.ts

The Bootstrap

- Finally, we need a starting point for our application
- It is in a file called app.component.ts and is imported via

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

- app.component must be used in the starting code for the app, it's the root component

app.component.ts

@Component Decorator

- Angular uses the `@Component()` decorator that allows us to specify a class as an Angular **component**
- It provides additional metadata that determines how the **component** should be rendered

app.component.ts

- app.component.ts is found inside the app directory.
- To build an Angular component, we need to use the component decorator on a class which comes from the core scope package

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

- To decorate / define a component, you need to provide at least two properties: selector and template (or template URL).
  - The component decorator takes in an object literal.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

The Component's Selector

- By default, the root component has the selector `app-root`
- You can change it but must use at least one dash
- In index.html:

```
<body>
  <app-root></app-root>
</body>
```


The Component's Template

Template and Styles

- The are two ways to create a template for a component:
  - Using a `template` property OR
  - Using a `templateURL` property and creating a template file (ie. app.component.html')
    - This will resolve relative paths
- Similarly, there are two ways to style
  - Using the `styles` property OR
  - Using `styleURLs` and creating an external CSS file(s)
    - The `stylesURLs` property takes in an array of CSS files
    - It will automatically be injected into the <head> tag at load time
    - The style is automatically **scoped to the component** you're currently writing

Nested Components

Using components inside other components

- To create a subcomponent, we need to:
  1. Use the CLI to generate (or just add the files yourself)

     ```
     $ ng generate component <subcomp_name>
     ```

     this generates a folder that contains Html, Css, Spec (testing) files, Component info (<subcomp_name>.component.ts)
  2. In the root (bootstrap) component, import and declare subcomponent

     ```
     import { PersonComponent }
       from './person/person.component'
     ```

     ```
     @NgModule({
       declarations: [
         AppComponent,
         PersonComponent
       ],
     ```
  3. Add selector to parent components and use properties to pass variables (optional)

Angular Templates

Syntax and Features

- In a template, we can the following things:
  - Interpolation and Binding
  - Expressions and Expression operators
  - Conditionals
  - Template variables

Data Interacting with templates

Interpolation and Expressions

- Interpolation means binding data to a template
- Data binding is the basis of Angular, binding data to templates means to make data available to a particular template
- Data can bind to templates in three ways:
  - {{ }} expressions
  - By using attributes or methods in the exported component class
  - Property binding (i.e. [textContent] = "variable")

Component Inputs

@Input decorator

- To pass data from one level component to the next, we must do the following things:
  1. Import the Input decorator from the @angular/core scoped package

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

  2. use the @Input decorator to indicate this to the subcomponent

```
export class PersonComponent implements OnInit {
  @Input() people_array;
```

Component Inputs

@Input decorator

- To pass data from one level component to the next, we must do the following things:
  3. Set a property with an existing variable in the parent component in the selector tag

```
<app-person [people_array]="people"></app-person>
```

```
...
export class AppComponent {
  people = [
    {
      name: "Bobby",
      birthday: "1980-04-19"
    }
  ];
```

  4. Use the data in the subcomponent's HTML file