Always import

app/heroes/heroes.component.ts (initial version)

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

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  hero = 'Windstorm';
  constructor() { }

  ngOnInit() {
  }

}
```

→ css selector

life cycle hook
Good place to put initialization logic

Always export the Component class so that you can import it elsewhere

Create a Hero class in its own file

src/app/hero.ts

```
export class Hero {
  id: number;
  name: string;
}
```

src/app/heroes/heroes.component.ts

```ts
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  hero: Hero = {
    id: 1,
    name: 'Windstorm'
  };

  constructor() { }

  ngOnInit() {
  }

}
```

refactor the hero property to be type IHero. Initialize with an id of 1 and name 'Windstorm

→ Page no longer displays properly you changed the hero from a string to an Object.

heroes.component.html (HeroesComponent's template)

```html
<h2>{{ hero.name }} Details</h2>
<div><span>id: </span>{{hero.id}}</div>
<div><span>name: </span>{{hero.name}}</div>
```

Format with uppercase pipe

```
<h2>{{ hero.name | uppercase }} Details</h2>
```

hero name turns capital

Two Way Binding

src/app/heroes/heroes.component.html (HeroesComponent's template)

```
<div>
    <label>name:
      <input [(ngModel)]="hero.name" placeholder="name">
    </label>
</div>
```

Angular's two way data binding syntax

ngModel is not available by default.

It belongs to FormsModule

→ AppModule

app.module.ts (FormsModule symbol import)

```
import { FormsModule } from '@angular/forms'; // <-- NgModel lives
here
```

AppModule

Angular needs to know how the pieces of your application fit together and what other files and libraries the app requires. This information is called *metadata*

Some of the metadata is in the `@Component` decorators that you added to your component classes. Other critical metadata is in `@NgModule` decorators.

The most important `@NgModule` decorator annotates the top-level **AppModule** class.

The Angular CLI generated an `AppModule` class in `src/app/app.module.ts` when it created the project. This is where you *opt-in* to the `FormsModule`.

Then add `FormsModule` to the `@NgModule` metadata's `imports` array, which contains a list of external modules that the app needs.

app.module.ts (@NgModule imports)

```
imports: [
  BrowserModule,
  FormsModule
],
```

When the browser refreshes, the app should work again. You can edit the hero's name and see the changes reflected immediately in the `<h2>` above the textbox.

Declare *HeroesComponent*

Every component must be declared in *exactly one* NgModule.

You didn't declare the `HeroesComponent`. So why did the application work?

It worked because the Angular CLI declared `HeroesComponent` in the `AppModule` when it generated that component.

Open `src/app/app.module.ts` and find `HeroesComponent` imported near the top.

```
import { HeroesComponent } from './heroes/heroes.component';
```

The `HeroesComponent` is declared in the `@NgModule.declarations` array.

```
declarations: [
  AppComponent,
  HeroesComponent
],
```

Note that `AppModule` declares both application components, `AppComponent` and `HeroesComponent`.