Displaying data in views

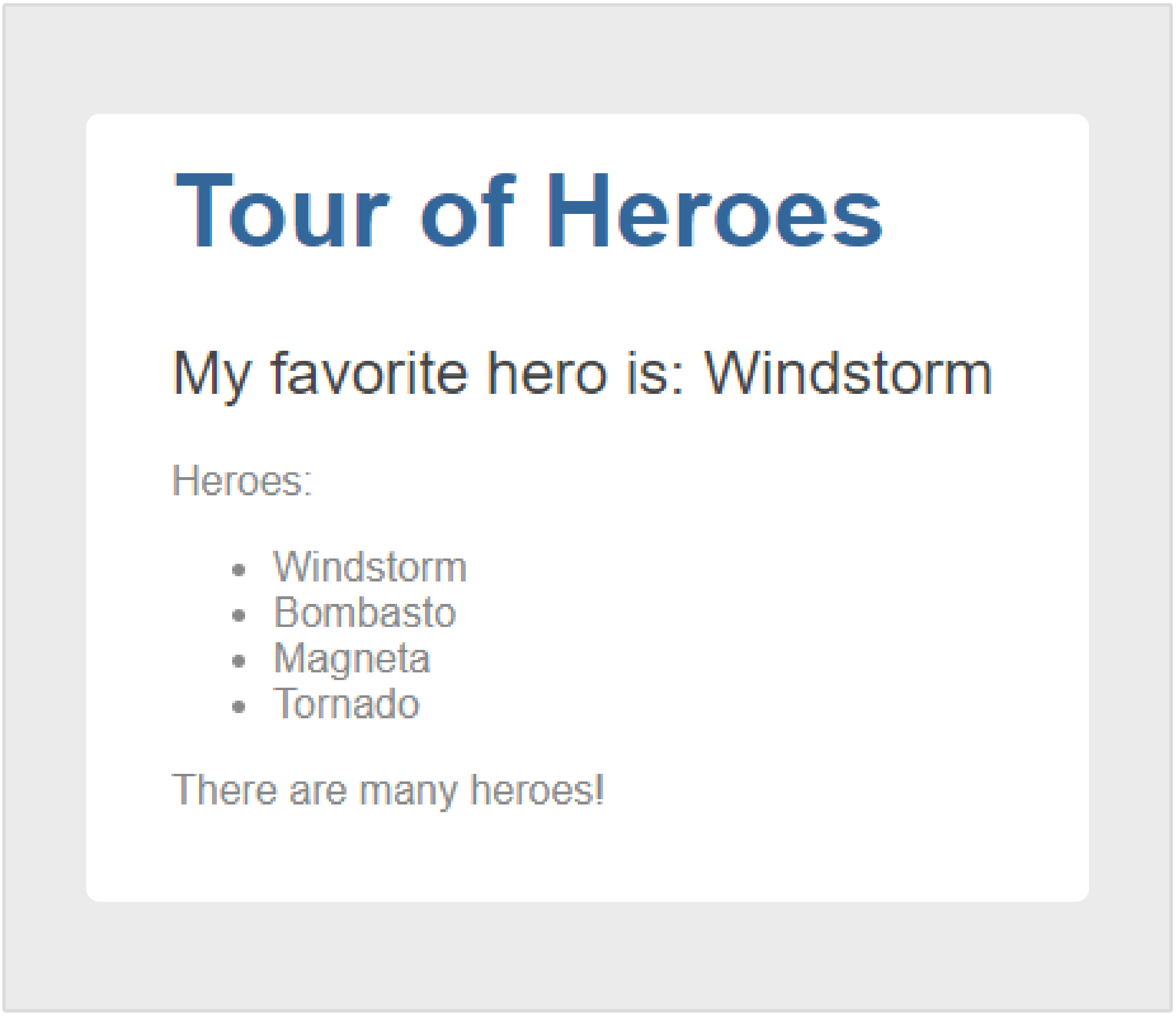
Angular components form the data structure of your application. The HTML template associated with a component provides the means to display that data in the context of a web page. Together, a component's class and template form a view of your application data.

The process of combining data values with their representation on the page is called data binding. You display your data to a user (and collect data from the user) by *binding* controls in the HTML template to the data properties of the component class.

In addition, you can add logic to the template by including directives, which tell Angular how to modify the page as it is rendered.

Angular defines a *template language* that expands HTML notation with syntax that allows you to define various kinds of data binding and logical directives. When the page is rendered, Angular interprets the template syntax to update the HTML according to your logic and current data state. Before you read the complete template syntax guide, the exercises on this page give you a quick demonstration of how template syntax works.

In this demo, you'll create a component with a list of heroes. You'll display the list of hero names and conditionally show a message below the list. The final UI looks like this:


Tour of Heroes

My favorite hero is: Windstorm

Heroes:

- Windstorm
- Bombasto
- Magneta
- Tornado

There are many heroes!

> The live example / download example demonstrates all of the syntax and code snippets described in this page.

Showing component properties with interpolation

The easiest way to display a component property is to bind the property name through interpolation. With interpolation, you put the property name in the view template, enclosed in double curly braces: `{{myHero}}`.

Use the CLI command `ng new displaying-data` to create a workspace and app named `displaying-data`.

Delete the `app.component.html` file. It is not needed for this example.

Then modify the `app.component.ts` file by changing the template and the body of the component.

When you're done, it should look like this:

src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
    `
})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
}
```

You added two properties to the formerly empty component: `title` and `myHero`.

The template displays the two component properties using double curly brace interpolation:

src/app/app.component.ts (template)

```
template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is: {{myHero}}
</h2>
  `
```

> The template is a multi-line string within ECMAScript 2015 backticks (`` ` ``). The backtick (`` ` ``)—which is *not* the same character as a single quote (`'`)—allows you to compose a string over several lines, which makes the HTML more readable.

Angular automatically pulls the value of the `title` and `myHero` properties from the component and inserts those values into the browser. Angular updates the display when these properties change.

> More precisely, the redisplay occurs after some kind of asynchronous event related to the view, such as a keystroke, a timer completion, or a response to an HTTP request.

Notice that you don't call **new** to create an instance of the `AppComponent` class. Angular is creating an instance for you. How?

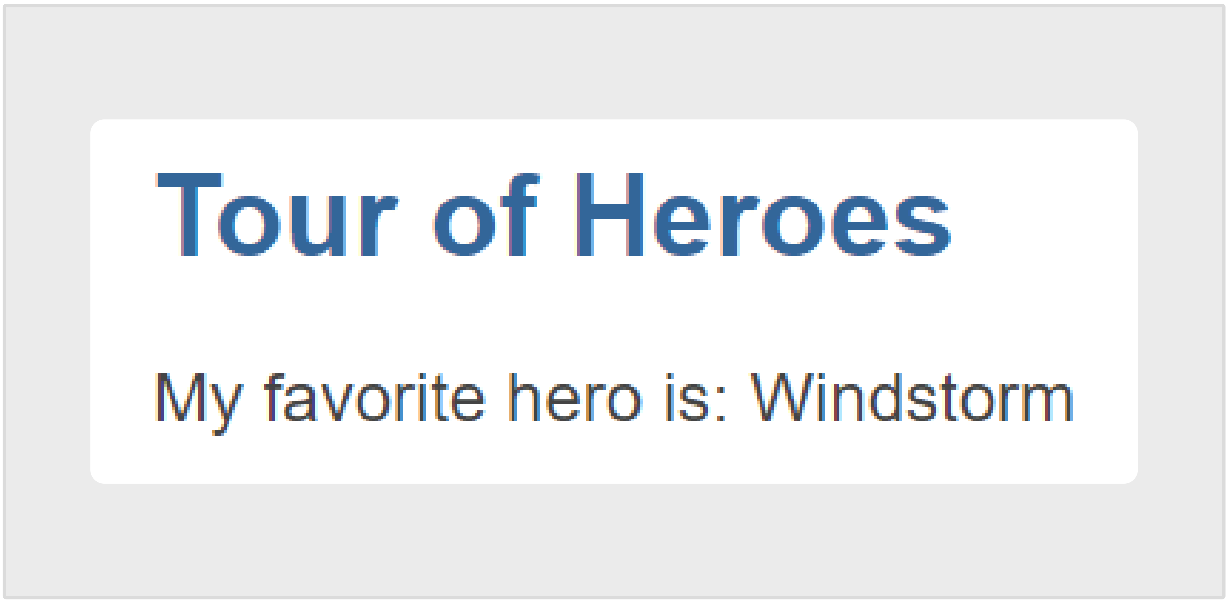
The CSS `selector` in the `@Component` decorator specifies an element named `<app-root>`. That element is a placeholder in the body of your `index.html` file:

src/index.html (body)

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

When you bootstrap with the `AppComponent` class (in `main.ts`), Angular looks for a `<app-root>` in the `index.html`, finds it, instantiates an instance of `AppComponent`, and renders it inside the `<app-root>` tag.

Now run the app. It should display the title and hero name:



Tour of Heroes

My favorite hero is: Windstorm

The next few sections review some of the coding choices in the app.

Choosing the template source

The `@Component` metadata tells Angular where to find the component's template. You can store your component's template in one of two places.

- You can define the template *inline* using the `template` property of the `@Component` decorator. An inline template is useful for a small demo or test.
- Alternatively, you can define the template in a separate HTML file and link to that file in the `templateUrl` property of the `@Component` decorator. This configuration is typical for anything more complex than a small test or demo, and is the default when you generate a new component.

In either style, the template data bindings have the same access to the component's properties. Here the app uses inline HTML because the template is small and the demo is simpler without the additional HTML file.

By default, the Angular CLI command `ng generate component` generates components with a template file. You can override that by adding the "-t" (short for `inlineTemplate=true`) option:

```
ng generate component hero -t
```

Initialization

The following example uses variable assignment to initialize the components.

```
export class AppComponent {
  title: string;
  myHero: string;

  constructor() {
    this.title = 'Tour of Heroes';
    this.myHero = 'Windstorm';
  }
}
```

You could instead declare and initialize the properties using a constructor. This app uses more terse "variable assignment" style simply for brevity.

Add logic to loop through data

The `*ngFor` directive (predefined by Angular) lets you loop through data. The following example uses the directive to show all of the values in an array property.

To display a list of heroes, begin by adding an array of hero names to the component and redefine `myHero` to be the first name in the array.

src/app/app.component.ts (class)

```
export class AppComponent {
  title = 'Tour of Heroes';
  heroes = ['Windstorm', 'Bombasto',
'Magneta', 'Tornado'];
  myHero = this.heroes[0];
}
```

Now use the Angular `ngFor` directive in the template to display each item in the `heroes` list.

src/app/app.component.ts (template)

```
template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is: {{myHero}}
</h2>
  <p>Heroes:</p>
  <ul>
    <li *ngFor="let hero of heroes">
      {{ hero }}
    </li>
  </ul>
`
```

This UI uses the HTML unordered list with `<ul>` and `<li>` tags. The `*ngFor` in the `<li>` element is the Angular "repeater" directive. It marks that `<li>` element (and its children) as the "repeater template":

src/app/app.component.ts (li)

```
<li *ngFor="let hero of heroes">
  {{ hero }}
</li>
```

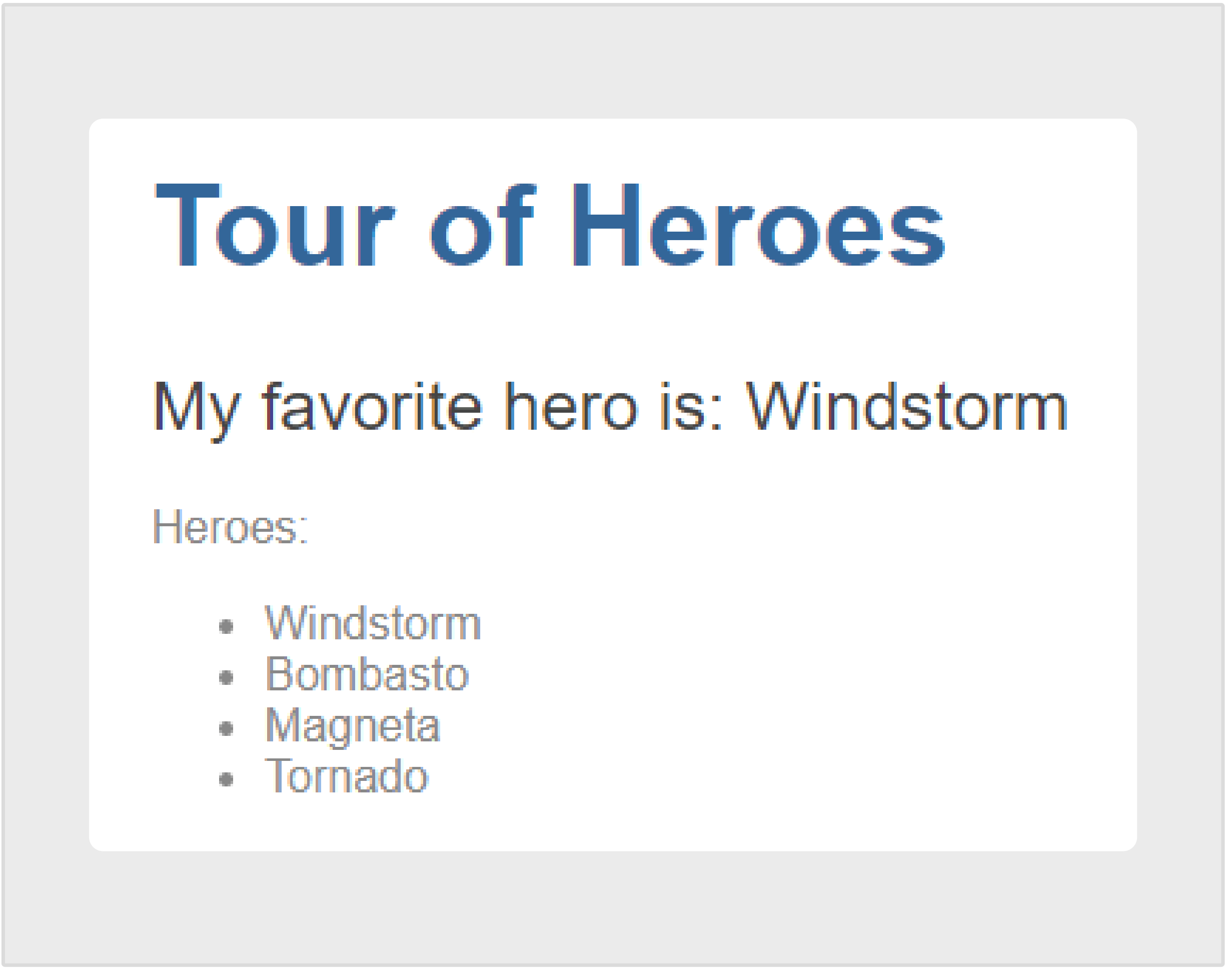
Don't forget the leading asterisk (*) in `*ngFor`. It is an essential part of the syntax. For more information, see the Template Syntax page.

Notice the `hero` in the `ngFor` double-quoted instruction; it is an example of a template input variable. Read more about template input variables in the microsyntax section of the Template Syntax page.

Angular duplicates the `<li>` for each item in the list, setting the `hero` variable to the item (the hero) in the current iteration. Angular uses that variable as the context for the interpolation in the double curly braces.

> In this case, `ngFor` is displaying an array, but `ngFor` can repeat items for any iterable object.

Now the heroes appear in an unordered list.


Tour of Heroes

My favorite hero is: Windstorm

Heroes:

- Windstorm
- Bombasto
- Magneta
- Tornado

Creating a class for the data

The app's code defines the data directly inside the component, which isn't best practice. In a simple demo, however, it's fine.

At the moment, the binding is to an array of strings. In real applications, most bindings are to more specialized objects.

To convert this binding to use specialized objects, turn the array of hero names into an array of `Hero` objects. For that you'll need a `Hero` class:

```
ng generate class hero
```

This command creates the following code.

src/app/hero.ts

```
export class Hero {
  constructor(
    public id: number,
    public name: string) { }
}
```

You've defined a class with a constructor and two properties: `id` and `name`.

It might not look like the class has properties, but it does. The declaration of the constructor parameters takes advantage of a TypeScript shortcut.

Consider the first parameter:

src/app/hero.ts (id)

```
public id: number,
```

That brief syntax does a lot:

- Declares a constructor parameter and its type.
- Declares a public property of the same name.
- Initializes that property with the corresponding argument when creating an instance of the class.

Using the Hero class

After importing the `Hero` class, the `AppComponent.heroes` property can return a *typed* array of `Hero` objects:

src/app/app.component.ts (heroes)

```
heroes = [
  new Hero(1, 'Windstorm'),
  new Hero(13, 'Bombasto'),
  new Hero(15, 'Magneta'),
  new Hero(20, 'Tornado')
];
myHero = this.heroes[0];
```

Next, update the template. At the moment it displays the hero's `id` and `name`. Fix that to display only the hero's `name` property.

src/app/app.component.ts (template)

```
template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is:
{{myHero.name}}</h2>
  <p>Heroes:</p>
  <ul>
    <li *ngFor="let hero of heroes">
      {{ hero.name }}
    </li>
  </ul>
`
```

The display looks the same, but the code is clearer.

Conditional display with NgIf

Sometimes an app needs to display a view or a portion of a view only under specific circumstances.

Let's change the example to display a message if there are more than three heroes.

The Angular `ngIf` directive inserts or removes an element based on a *truthy/falsy* condition. To see it in action, add the following paragraph at the bottom of the template:

src/app/app.component.ts (message)

```
<p *ngIf="heroes.length > 3">There are
many heroes!</p>
```

Don't forget the leading asterisk (*) in `*ngIf`. It is an essential part of the syntax. Read more about `ngIf` and `*` in the ngIf section of the Template Syntax page.

The template expression inside the double quotes, `*ngIf="heroes.length > 3"`, looks and behaves much like TypeScript. When the component's list of heroes has more than three items, Angular adds the paragraph to the DOM and the message appears. If there are three or fewer items, Angular omits the paragraph, so no message appears.

For more information, see template expressions.

> Angular isn't showing and hiding the message. It is adding and removing the paragraph element from the DOM. That improves performance, especially in larger projects when conditionally including or excluding big chunks of HTML with many data bindings.

Try it out. Because the array has four items, the message should appear. Go back into `app.component.ts` and delete or comment out one of the elements from the heroes array. The browser should refresh automatically and the message should disappear.

Summary

Now you know how to use:

- **Interpolation** with double curly braces to display a component property.
- **ngFor** to display an array of items.
- A TypeScript class to shape the **model data** for your component and display properties of that model.
- **ngIf** to conditionally display a chunk of HTML based on a boolean expression.

Here's the final code:

src/app/app.component.ts *src/app/hero.ts*

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

import { Hero } from './hero';

@Component({
  selector: 'app-root',
  template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is:
{{myHero.name}}</h2>
```

```
  <p>Heroes:</p>

  <ul>
    <li *ngFor="let hero of heroes">
      {{ hero.name }}
      </li>
  </ul>
  <p *ngIf="heroes.length > 3">There
are many heroes!</p>
`
})
export class AppComponent {
  title = 'Tour of Heroes';
  heroes = [
    new Hero(1, 'Windstorm'),
    new Hero(13, 'Bombasto'),
    new Hero(15, 'Magneta'),
    new Hero(20, 'Tornado')
  ];
  myHero = this.heroes[0];
}
```