Displaying Data

You can display data by binding controls in an HTML template to properties of an Angular component.

In this page, 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:

The demonstrates all of the syntax and code snippets described in this page.

{@a interpolation}

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}}`.

Follow the quickstart instructions for creating a new project 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:

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

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

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:

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:

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

Template inline or template file?

You can store your component's template in one of two places. You can define it *inline* using the `template` property, or you can define the template in a separate HTML file and link to it in the component metadata using the `@Component` decorator's `templateUrl` property.

The choice between inline and separate HTML is a matter of taste, circumstances, and organization policy. Here the app uses inline HTML because the template is small and the demo is simpler without the additional HTML file.

In either style, the template data bindings have the same access to the component's properties.

By default, the Angular CLI generates components with a template file. You can override that with: ng generate component hero -it

Constructor or variable initialization?

Although this example uses variable assignment to initialize the components, you could instead declare and initialize the properties using a constructor:

This app uses more terse "variable assignment" style simply for brevity.

{@a ngFor}

Showing an array property with *ngFor

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.

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

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":

Don't forget the leading asterisk (*) in `*ngFor`. It is an essential part of the syntax. For more information, see the [Template Syntax](guide/template-syntax#ngFor) 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](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols) object.

Now the heroes appear in an unordered list.

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

With the following code:

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:

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:

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.

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

{@a ngIf}

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:

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](guide/template-syntax#ngIf) of the [Template Syntax](guide/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 the template expressions section of the Template Syntax page.

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 hero 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: