# Angular 4 Templating Basics

In Angular, your views are defined within HTML templates. Templates are defined within the **@Component** decorator. You're able to define inline HTML templates as well as external templates within HTML files.

You're also able to display data defined within the component through interpolation, as well as use various conditionals within the template.

Let's see how it all works.

## Defining External Templates

Using the same project that we've been using based on our **Free Angular 4 Course**, this is what our app.component.ts file looks like:

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

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

export class AppComponent {
  title = 'app works!';
}
```

When you use the Angular CLI to generate an Angular project, it uses the **templateUrl** metadata property to define an external template by default.

As a general rule of thumb, you use **templateUrl** to define a template whenever there's a considerable amount of HTML associated with the given component. Otherwise, if you use inline HTML within the component itself, it will become quite unorganized and hard to navigate.

But what if your component only has a few lines of HTML? In this case, it may make sense to define inline HTML.

## Defining Inline HTML Templates

Using the code above, let's change the **templateUrl** property to **template,** and declare a single line of HTML.

```
    template: '<h1>Hey guys!</h1>',
```

If you want the ability to define multiple lines of inline HTML, single quotes will not work. Instead, you will need to change them to backticks .`

```
    template: `
    <h1>Hey guys!</h1>
    <p>How are you dong?</p>
    `,
```

So, in terms of defining templates; that is all there is to it! Now, let's move onto the next logic step: how do we display dynamic data in the template?

## Interpolation

Interpolation is just a fancy word for displaying data in the template. This data is usually defined within the component class.

When you use the Angular CLI to generate an Angular project, it displays a **title** property through interpolation within the **app.component.html** template file. It looks like this:

```
<h1>
  {{title}}
</h1>
```

So, interpolation works by wrapping double curly brackets around a template expression. A template expression can represent a component property or even mathematical equations.

If your component property consists of an object, you can reference a specific object property like this:

```
@Component({
// Other component properties removed
  template: `
  <h1>Hey guys!</h1>
  <p>{{ myObject.gender }}</p>
  `,

})

export class AppComponent {

  myObject = {
    gender: 'male',
    age: 33,
    location: 'USA'
  };

}
```

## *ngFor and Iterables

What if you have an array or an array of objects? How would you display those in a template?

That's where the *ngFor directive comes in handy. Here's how you display a basic array:

```
@Component({
// Other component properties removed.
  template: `
  <h1>Hey guys!</h1>
  <ul>
    <li *ngFor="let arr of myArr">{{ arr }}</li>
  </ul>
  `,

})
export class AppComponent {

  myArr = ['him','hers','yours','theirs'];

}
```

The process is exactly the same for an array of objects as well.

## *ngIf and Else

Sometimes you only want to display a given template if an expression matches some condition. In these cases, you would use the *ngIf directive. Using the example above:

```
<li *ngIf="myArr">Yeah, I exist.</li>
```

Because the object myArr is defined, it evaluates to true and displays the HTML element that it has been applied to.

You can also specify the ! not operator and it won't display the given HTML element:

```
<li *ngIf="!myArr">Yeah, I exist.</li>
```

Additionally, you can use other operators such as == and !=

```
<li *ngIf="myArr == 'something'">Yeah, I exist.</li>
```

New in Angular 4, you can also declare **else** as conditions:

```
  template: `
  <h1>Hey guys!</h1>
  <ul>
    <li *ngIf="myArr; else otherTmpl">Yeah, I exist.</li>
  </ul>

  <ng-template #otherTmpl>No, I do.</ng-template>
  `,
```

So, we add **; else otherTmpl**. The **otherTmpl** is defining a local variable. You reference that local variable on an **ng-template** element. This allows you to display a template when the expression evaluates to false.

We can also use **then** inside of our expression to define 2 different templates based on truth or false.

```
  template: `
  <h1>Hey guys!</h1>
  <div *ngIf="myArr; then tmpl1 else tmpl2"></div>

  <ng-template #tmpl1>I'm here</ng-template>
  <ng-template #tmpl2>I'm not here</ng-template>
  `,
```

And this concludes the basics of Templating in Angular 4. With the knowledge above, you will be able to handle most tasks associated with your templates.

In the next lesson as a part of our Free Angular 4 Course, we're going to learn about property binding.