## Angular Directives

we are going to look at the Angular Directives. We will look at three types of directives that Angular supports like Component, Structural and Attribute Directives. We also look at the few of the most commonly used Angular directives.

- What is Angular Directive
- Component Directive
- Structural Directives
- Commonly used structural directives
- ngFor
- ngSwitch
- ngIf
- Attribute Directives
- Commonly used Attribute directives

## What is Angular Directive

The Angular directive helps us to manipulate the DOM. You can change the appearance, behavior or layout of a DOM element using the Directives. They help you to extend HTML

There are three kinds of directives in Angular:

1. Component Directive

2. Structural directives

3.Attribute directives

## Component Directive

Components are special directives in Angular. They are the directive with a template (view) We covered how to create Components in Angular tutorial.

## Structural Directives

Structural directives can change the DOM layout by adding and removing DOM elements. All structural Directives are preceded by Asterix symbol

### Commonly used structural directives

### ngFor

The ngFor is an Angular structural directive, which repeats a portion of the HTML template once per each item from an iterable list (Collection). The ngFor is similar to ngRepeat in AngularJS

Example of ngFor

```
<tr *ngFor="let customer of customers;">
   <td>{{customer.customerNo}}</td>
   <td>{{customer.name}}</td>
   <td>{{customer.address}}</td>
   <td>{{customer.city}}</td>
   <td>{{customer.state}}</td>
</tr>
```

**ngSwitch**

The ngSwitch directive lets you add/remove HTML elements depending on a match expression. ngSwitch directive used along with ngSwitchCase and ngSwitchDefault

The example of ngSwitch

```
<div [ngSwitch]="Switch_Expression">
   <div *ngSwitchCase="MatchExpression1”> First Template</div>
   <div *ngSwitchCase="MatchExpression2">Second template</div>
   <div *ngSwitchCase="MatchExpression3">Third Template</div>
   <div *ngSwitchCase="MatchExpression4">Third Template</div>
   <div *ngSwitchDefault?>Default Template</div>
</div>
```

**ngIf**

The ngIf Directives is used to add or remove HTML elements based on an expression. The expression must return a boolean value. If the expression is false then the element is removed, else the element is inserted

Example of ngIf

```
<div *ngIf="condition">
   This is shown if condition is true
</div>
```

## Attribute Directives

An Attribute or style directive can change the appearance or behavior of an element.

## Commonly used Attribute directives

### ngModel

The ngModel directive is used the achieve the two-way data binding. We have covered ngModel directive in Data Binding in Angular Tutorial

### ngClass

The ngClass is used to add or remove the CSS classes from an HTML element. Using the ngClass one can create dynamic styles in HTML pages

Example of ngClass

```
1
2 <div [ngClass]="'first second'">...</div>
3
```

### ngStyle

ngStyle is used to change the multiple style properties of our HTML elements. We can also bind these properties to values that can be updated by the user or our components.

Example of ngStyle

```
1
2 <div [ngStyle]="{'color': 'blue', 'font-size': '24px', 'font-weight': 'bold'}"
3    some text
4 </div>
5
```

**Building Custom Directives**

You can also build custom directives in Angular. The Process is to create a JavaScript class and apply the **@Directive** attribute to that class. You can write the desired behavior in the class.

Summary

In this tutorial, we introduced you to the **directives in Angular**. In the next few tutorials, we will look at some of the important directives in detail

**Angular ngFor Directive**

The **ngFor is an Angular structural directive**, which repeats a portion of HTML template once per each item from an iterable list (Collection). The **ngFor** is similar to **ngRepeat** in AngularJS

In this tutorial, we will look at the syntax of **ngFor**. We will show you how to use ngFor using an example code. We further look at the local variables like **Index**, **First**, **Last**, **odd** and **even**, which is exported by the **ngFor directive**. The use of local variables is demonstrated by creating a simple code to Format odd & even rows of a table by assigning different classes to them. Finally, we look at the **Track By (trackBy) clause**, which enhances the performances of the **ngFor.**

## Table of Content

### Syntax of ngFor

The simplified syntax for the ngFor is as shown below

```
1
2 <li *ngFor="let item of items;”> .... </li>
3
```

### *ngFor

The syntax starts with *ngFor. The * here represents the Angular template syntax.

### let item of items.

Here the items are a collection that we need to show to the user. The **let keyword** creates a **local variable** named item. You can then use this variable to reference the individual item in the items collection. You can use this variable anywhere inside your template.

### How to Use ngFor Directive

To Use **ngFor**, First, you need to create a block of HTML elements, which can display a single item of the items collection. Then use the **ngFor** directive to tell angular to repeat that block of HTML elements for each item in the list.

### Example of ngFor

First, you need to create an angular Application. If you are new to angular , then you should read How to Create Angular Application Tutorial.

You can also download the code for this tutorial from the gitHub. The starting point for the project is found in Start folder and the completed project is in the folder ngFor.

Open the app.component.ts and add the following code. The following Code contains a list of Top 5 movies in a movies array. We will build a template to display these movies in a tabular form.

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

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

export class AppComponent
{
   title: string ="Top 10 Movies" ;

   movies: Movie[] =[

      {title:'Zootopia',director:'Byron Howard, Rich Moore',cast:'Idris El
      {title:'Batman v Superman: Dawn of Justice',director:'Zack Snyde
      {title:'Captain American: Civil War',director:'Anthony Russo, Joe R
      {title:'X-Men: Apocalypse',director:'Bryan Singer',cast:'Jennifer La
      {title:'Warcraft',director:'Duncan Jones',cast:'Travis Fimmel, Robe
      {title:'Avengers: Age of Ultron',director:'Joss Whedon',cast:'Rober
      {title:'Star Wars: The Force Awakens',director:'Colin Trevorrow',ca
      {title:'The Hunger Games: Mockingjay - Part 2',director:'Francis La
      {title:'Jurassic World',director:'Colin Trevorrow',cast:' Chris Pratt,
```

The next step is to create HTML template. Open the **app.component.html** and add the following code

```
<div class='panel panel-primary'>
  <div class='panel-heading'>
    {{title}}
  </div>

  <div class='panel-body'>
    <div class='table-responsive'>
      <table class='table'>
        <thead>
          <tr>
            <th>Title</th>
            <th>Director</th>
            <th>Cast</th>
            <th>Release Date</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let movie of movies;">
            <td>{{movie.title}}</td>
            <td>{{movie.director}}</td>
            <td>{{movie.cast}}</td>
```

We are using **Bootstrap CSS classes** to make the table look good.

The most important part of the code is

```
1
2 <tr *ngFor="let movie of movies;">
3    <td>{{movie.title}}</td>
4    <td>{{movie.director}}</td>
5    <td>{{movie.cast}}</td>
6    <td>{{movie.releaseDate}}</td>
7 </tr>
8
```

Our purpose is to display the list of movies. Hence we build a **tr element** of a table, which can display a single movie. We want the **tr element** to be repeated for each item in the movies collection. Hence we apply **ngFor directive to the tr element of the table**.

The **“let movie of movies”** creates the **local variable movie**. You can use the variable to create the template

Run the project using **npm start** command and you should be able to see the list of movies displayed in your browser

### Local Variables in ngFor

**ngFor** also provides several values to help us manipulate the collection. We can assign the values of these exported values to the local variable and use it in our template

The list of exported values provided by **ngFor directive**

### index

This is a zero-based index and set to the current loop iteration for each template context.

### first

This is a boolean value, set to true if the item is the first item in the iteration

### last

This is a boolean value, set to true if the item is the last item in the iteration

### even

This is a boolean value, set to true if the item is the even-numbered item in the iteration

### odd

This is a boolean value, set to true if the item is the odd-numbered item in the iteration

### How to Use Local Variables

Using additional let operator, we can create an additional local variable and assign the value of the index to it. The syntax is shown below

```
1
2 <li *ngFor="let item of items; let i=index;”> .... </li>
3
```

The following code shows the list of movies along with the index.

```
<tr *ngFor="let movie of movies; let i=index;">
   <td> {{i}} </td>
   <td>{{movie.title}}</td>
   <td>{{movie.director}}</td>
   <td>{{movie.cast}}</td>
   <td>{{movie.releaseDate}}</td>
</tr>
```

**Formatting odd & even rows**

We can use the odd & even values to format the odd & even rows alternatively. To do that create two local variable o & e. Assign the values of odd & even values to these variables using the let statement. Then use this to change the class name to either odd or even. The example code is shown below

```
<tr *ngFor="let movie of movies; let i=index; let o= odd; let e=even;"
[ngClass]="{ odd: o, even: e }">
   <td> {{i}} </td>
   <td>{{movie.title}}</td>
   <td>{{movie.director}}</td>
   <td>{{movie.cast}}</td>
   <td>{{movie.releaseDate}}</td>
</tr>
```

Open the app.component.css and add the appropriate background colour to the odd and even classes as shown below

```
.even { background-color: red; }
.odd { background-color: green; }
```

**Track By**

The angular includes **Track By clause**, just like AngularJS did. Track By clause allows you to specify your own key to identify objects.

Angular 2 generates unique IDs.for each item in your collection to track the items collection.This has a huge performance implication. For instance, Consider you have items in your DOM and want to refresh them with the new items from the database. Although the retrieved items are similar to those available in the DOM, the Angular have no means to identify them. The Angular to simply remove these elements from DOM and recreates the new elements from the new data.

**trackBy clause** eliminates this problem, by telling angular how to identify the similar elements. The Angular will use the value returned trackBy to match the elements returned by the database and updates the DOM Elements without recreating them.

We should always specify the primary key or unique key as the trackBy clause.

The syntax for using Track By is as follows

```
1
2 <li *ngFor="let item of items; let i = index; trackBy: trackBy
3
```

**Example**

In our movie list example, let us make the title of the movie as the identifier. This is done by assigning the movie.title to trackBy clause as shown below

```
<tr *ngFor="let movie of movies; trackBy:movie?.title;">
   <td>{{movie.title}}</td>
   <td>{{movie.director}}</td>
   <td>{{movie.cast}}</td>
   <td>{{movie.releaseDate}}</td>
</tr>
```

**Note that we have used movie?.title instead of movie.title. ? is known as Safe navigation operator ( Elvis operator ). The ? returns null if the movie is null otherwise returns the value of the title. Without this, the null-reference exception is thrown by the system**

### Track By In case of Multiple Identity

In case of your model containing composite key, that can be handled by first creating a method, which returns the unique identifier and assigning this method to trackBy

For example, if want both title and director to be used as the identifier, first create a method say CompositeKey in the component as shown below. The function concatenates the title and director and returns it as string

```
CompositeKey (index,item){
   return item.title + item.director ;
}

```

Then assign this function to trackBy clause as shown below

```
<tr *ngFor="let movie of movies; trackBy:CompositeKey;" >
   <td>{{movie.title}}</td>
   <td>{{movie.director}}</td>
   <td>{{movie.cast}}</td>
   <td>{{movie.releaseDate}}</td>
</tr>

```

**Conclusion**

In this tutorial, You can download the source code from here

**ngSwitch, ngSwitchcase, ngSwitchDefault Angular Example**

The ngSwitch is an Angular structural directive, which allows us to add or remove DOM elements. It works in conjunction with ngSwitchcase, & ngSwitchDefault directives. It is similar to the switch statement of JavaScript. In this tutorial, we will look at the syntax of ngSwitch, ngSwitchcase & ngSwitchDefault. We will show you how to use these directives using an example.

## Table of Content

## Angular ngSwitch Directive

The ngSwitch is an Angular directive, which allows us to display one or more DOM elements based on some pre-defined condition.

The following is the syntax of ngSwitch. It contains three separate directives. ngSwitch, ngSwitchCase & ngSwitchDefault.

Syntax

```
<container_element [ngSwitch]="switch_expression">
  <inner_element *ngSwitchCase="match_expresson_1">...
  <inner_element *ngSwitchCase="match_expresson_2">...
  <inner_element *ngSwitchCase="match_expresson_3">...
  <inner_element *ngSwitchDefault>...</element>
</container_element>
```

### ngSwitch

ngSwitch is bound to container_element like div etc. We assign a switch-expression to the ngSwitch via property binding syntax. Angular evaluates the switch_expression at runtime and based on its value displays or removes the elements from the DOM.

### ngSwitchCase

ngSwitchCase is bound to an inner_element, which we must place inside the container_element. We use * (Asterix symbol), because it is a structural directive. We also assign a match_expression, which

Angular evaluates at runtime. The Angular displays the inner_element only when the value of the match_expression matches the value of the switch_expression else it is removed from the DOM.

If there is more than one match, then it displays all of them.

Note that the ngSwitchCase does not hide the element, but removes them from the DOM.

**ngSwitchDefault**

ngSwitchDefault is also bound to an inner_element, which we must place inside the container_element. But it does not have any match_expression. If none of the ngSwitchCase match_expression matches the switch_expression, then the angular displays the element attached to the ngSwitchDefault

You can place ngSwitchDefault anywhere inside the container element and not necessarily at the bottom.

You are free to add more than one ngSwitchDefault directive. Angular displays all of them.

**Important Points**

- You must place ngSwitchCase & ngSwitchDefault inside the ngSwitch directive
- Angular displays every element, that matches the switch_expression
- If there are no matches, angular displays all the elements, which has ngSwitchDefault directive

- You can place one or more than one ngSwitchDefault anywhere inside the container element and not necessarily at the bottom.

- Any element within the ngSwitch statement but outside of any NgSwitchCase or ngSwitchDefault directive is displayed as it is.

- The elements are not hidden but removed from the DOM.

**ngSwitch Example**

**Component Class**

Create a variable **num** in your Angular Component class

```

num: number= 0;

```

Template

```
Input string : <input type='text' [(ngModel)] ="num"/>
<div [ngSwitch]="num">
   <div *ngSwitchCase="'1'">One</div>
   <div *ngSwitchCase="'2'">Two</div>
   <div *ngSwitchCase="'3'">Three</div>
   <div *ngSwitchCase="'4'">Four</div>
   <div *ngSwitchCase="'5'">Five</div>
   <div *ngSwitchDefault>This is Default</div>
</div>
```

Now let us examine the code in detail

```
Input string : <input type='text' [(ngModel)] ="num"/>
```

We bind the num variable to the input box.

```
<div [ngSwitch]="num">
```

We attach the ngSwitch directive to the div element, then bind it to the expression num.

```

<div *ngSwitchCase="'1'">One</div>

```

Next, we have a few ngSwitchCase directives attached to the div element with matching expressions “1”,”2” etc. Whenever the num matches these expressions, the ngSwitchCase displays the element attached to it else it removes it from DOM.

```

<div *ngSwitchDefault>**This is Default**</div>

```

The ngSwithcDefault does not take any expression, but it is displays only when all other ngSwitchCase match expressions fail.

## More Examples

The following uses the array of objects instead of a variable.

## Component class

```
1
2
3  class item {
4      name: string;
5      val: number;
6  }
7
8  export class AppComponent
9  {
1      items: item[] = [{name: 'One', val: 1}, {name: 'Two', val
0      selectedValue: string= 'One';
1  }
1
1
2
```

**Template**

Note that we have two matches for `*ngSwitchCase="'Two'"`. `ngSwitchcase` renders both.

```
div class='panel panel-primary'>
 <div class='panel-heading'>
  <h1>ngSwitch Example 2</h1>
 </div>

 <div class="panel-body">
  <div class='row'>
   <div class='col-md-6'>
    <select [(ngModel)]="selectedValue">
     <option *ngFor="let item of items;" [value]="item.name">{{ite
    </select>
   </div>
  </div>

  <div class='col-md-6'>
   <div class='row' [ngSwitch]="selectedValue">
    <div *ngSwitchCase="'One'">One is Selected</div>
    <div *ngSwitchCase="'Two'">Two is Selected</div>
    <div *ngSwitchCase="'Two'">Two Again used in another elemen
    <div *ngSwitchDefault>This is Default</div>
   </div>
  </div>

```

You can also make use of ng-template directly instead of *ngSwitchCase. In fact *ngSwitchCase="'One'" is a shortcut to ng-template [ngSwitchCase]="'One'".

```
1
2   <h2>ngSwitch using ng-template</h2>
3   <div class='col-md-6'>
4    <div class='row' [ngSwitch]="selectedValue">
5     <ng-template [ngSwitchCase]="'One'">One is Selected</ng-tem
6     <ng-template [ngSwitchCase]="'Two'">Two is Selected</ng-tem
7     <ng-template ngSwitchDefault>This is Default</ng-template>
8    </div>
9   </div>
10
```

More than one ngSwitchDefault. Works perfectly ok.

```
<h2>ngSwitchDefault</h2>
<div class='col-md-6'>
  <div class='row' [ngSwitch]="selectedValue">
    <div *ngSwitchCase="'One'">One is Selected</div>
    <div *ngSwitchDefault>This is Default 1</div>
    <div *ngSwitchCase="'Two'">Two is Selected</div>
    <div *ngSwitchDefault>This is Default 2</div>
  </div>
</div>
```

Source Code

You can download the source code from GitHub

## Angular ngIf then else directive by example

The ngIf is an Angular Structural Directive, which allows us to add/remove DOM Element based on some condition. In this Tutorial let us look at the syntax of ngIf. We will show you how to use ngIf using example code. We will also learn how to use the optional else clause & optional then clause using the ng-template

## Table of Content



### ngIf Syntax

### ngIf

```
<p *ngIf="condition">
   content to render, when the condition is true
</p>
```

The ngIf is attached to a DOM element ( p element in the above example). ngIf  is a structural directive, which means that you can add it to any element like div, p, h1, component selector, etc. Like all structural directive, it is prefixed with * asterisk

It is then bound to an expression (a condition in the above example). The expression is then evaluated by the ngIf directive. The expression must return either true or false.

If the expression evaluates to false then the entire element is removed from the DOM. If true then the element is inserted into the DOM.

**Hidden attribute Vs ngIf**

```
<p [hidden]="condition">
   content to render, when the condition is true
</p>
```

The above achieves the same thing, with one vital difference.

ngIf  does not hide the DOM element. It removes the entire element

along with its subtree from the DOM. It also removes the corresponding state freeing up the resources attached to the element.

hidden attribute does not remove the element from the DOM. But just hides it.

The difference between [hidden]='false' and *ngIf='false' is that the first method simply hides the element. The second method with ngIf removes the element completely from the DOM.

By using the **Logical NOT** (!), you can mimic the else condition as shown here.

```
<p *ngIf="!condition">
   content to render, when the condition is false
</p>
```

The better solution is to use the optional else block as shown in the next paragraph.

### Condition

The condition can be anything. It can be a property of the component class. It can be a method in the component class. But it must evaluate to true/false. The ngIf directive tries to coerce the value to Boolean.

### ngIf else

The ngIf allows us to define optional else block using the ng-template

```
<div *ngIf="condition; else elseBlock">
   content to render, when thee condition is true
</div>

<ng-template #elseBlock>
   content to render, when the condition is false
</ng-template>
```

The expression starts with a condition followed by a semicolon.

Next, we have `else` clause bound to a template named `elseBlock`. The template can be defined anywhere using the `ng-template`. Typically it is placed right after ngIf for readability.

When the condition evaluates to false, then the `ng-template` with the name `#elseBlock` is rendered by the `ngIf` Directive.

**ngIf then else**

You can also define `then` `else` block using the `ng-template`

```
<div *ngIf="condition; then thenBlock else elseBlock">
   This content is not shown
</div>

<ng-template #thenBlock>
   content to render when the condition is true.
</ng-template>

<ng-template #elseBlock>
   content to render when condition is false.
</ng-template>
```

Here, we have then clause followed by a template named thenBlock.

When the condition is true, the template thenBlock is rendered. If false, then the template elseBlock is rendered

### ngIf Example

Create a new Angular project by running the command ng new ngIf

### Component Class

Create a boolean variable showMe in your app.component.ts class as shown below

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

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 title: string = 'ngIf Example' ;
 showMe: boolean;
}
```

**Template**

Copy the following code to the `app.component.html`.

```
<h1>Simple example of ngIf </h1>



<div class="row">
 Show <input type="checkbox" [(ngModel)]="showMe" />
</div>

<h1>ngIf </h1>

<p *ngIf="showMe">
 ShowMe is checked
</p>
<p *ngIf="!showMe">
 ShowMe is unchecked
</p>

<h1>ngIf Else</h1>

<p *ngIf="showMe; else elseBlock1">
 ShowMe is checked
</p>

<ng-template #elseBlock1>
```

Now let us examine the code in detail

```
Show <input type="checkbox" [(ngModel)] ="showMe"/>
```

This is a simple checkbox bound to `showMe` variable in the component

```
<div *ngIf="showMe">
   ShowMe is checked
</div>
```

The `ngIf` directive is attached to the `div` element. It is then bound to the expression “showMe”. The expression is evaluated and if it is `true`, then the `div` element is added to the DOM else it is removed from the DOM.

```
<p *ngIf="showMe; else elseBlock1">
 ShowMe is checked
</p>

<ng-template #elseBlock1>
 <p>ShowMe is checked Using elseBlock</p>
</ng-template>
```

If else example.

```
<p *ngIf="showMe; then thenBlock2 else elseBlock2">
 This is not rendered
</p>

<ng-template #thenBlock2>
  <p>ShowMe is checked Using thenblock</p>
</ng-template>

<ng-template #elseBlock2>
  <p>ShowMe is unchecked Using elseBlock</p>
</ng-template>
```

If then else example. Note that the content of `p` element, to which `ngIf` is attached is never rendered

```
<h1>using hidden </h1>


<p [hidden]="showMe">
   content to render, when the condition is true  using hidden pr
</p>


<p [hidden]="!showMe">
   content to render, when the condition is false. using hidden p
</p>
```

The property binding on the hidden attribute. You can open the developer console and see that both the elements are rendered and only one of them is visible and the other one is hidden

### Module

Import FormsModule in app.module.ts as we are using ngModal directive

```
1
2 import { BrowserModule } from '@angular/platform-browser';
3 import { NgModule } from '@angular/core';
4 import { FormsModule } from '@angular/forms';
5
6 import { AppRoutingModule } from './app-routing.module';
7 import { AppComponent } from './app.component';
8
9 @NgModule({
10  declarations: [
11    AppComponent
12  ],
13  imports: [
14    BrowserModule,
15    AppRoutingModule,
16    FormsModule
17  ],
18  providers: [],
19  bootstrap: [AppComponent]
20 })
21 export class AppModule { }
22
23
```

# Angular NgClass Directive

The **Angular ngClass** Directive is an Angular Attribute Directive, which allows us to add or remove CSS classes to an HTML element. Using ngClass you can create dynamic styles in angular components by using conditional expressions.

**Table of Content**

### NgClass

The ngClass directive adds and removes CSS classes on an HTML element. The syntax of the ngClass is as shown below.

```
1
2 <element [ngClass]="expression">...</element>
3
```

Where

**element** is the DOM element to which class is being applied

**expression** is evaluated and the resulting classes are added/removed from the element. The expression can be in various formats like string, array or an object. Let us explore all of them with example

### NgClass with a String

You can use the String as expression and bind it to directly to the ngClass attribute. If you want to assign multiple classes, then separate each class with space as shown below.

```
<element [ngClass]="'cssClass1 cssClass2'">...</element>
```

**Example**

Add the following classes to the app.component.css

```
.red { color: red; }
.size20 { font-size: 20px; }
```

Add the following to the app.template.html

```
<div [ngClass]="'red size20'"> Red Text with Size 20px </div>
```

The above example code adds the two CSS Classes red & size20 to the div element.

You can also use the ngClass without a square bracket. In that case, the expression is not evaluated but assigned directly to the class attribute. We also need to remove the double quote around the expression as shown below.

```
<div class="row">
   <div ngClass='red size20'>Red Text with Size 20px </div>
</div>
```

### NgClass with Array

You can achieve the same result by using an array instead of a string as shown below. The syntax for ngClass array syntax is as shown below

```
<element [ngClass]="['cssClass1', 'cssClass2']">...</element>
```

### Example

All you need to change the template as shown below

### NgClass with Object

You can also bind the ngClass to an object. Each property name of the object acts as a class name and is applied to the element if it is true. The syntax is as shown below

```

<element [ngClass]="{'cssClass1': true, 'cssClass2': true}">...</eleme

```

Example of objects as CSS Classes

```

<div class="row">
  <div [ngClass]="{'red':true,'size20':true}">Red Text with Size 20px<
</div>

```

In the above example, an object is bound to the ngClass. The object has two properties red and size20. The property name is assigned to the div element as a class name.

### Dynamically updating Class names

We can dynamically change the CSS Classes from the component.

### Using strings

To do that first create a string variable `cssStringVar` in your component code and assign the class names to it as shown below.

```
1
2 cssStringVar: string= 'red size20';
3
```

You can refer to the `cssStringVar` in your template as shown below

```
1
2 <div class="row">
3   <div [ngClass]="cssStringVar">Red Text with Size 20px : from comp
4 </div>
5
```

### Using arrays

Instead of string variable, you can create a **array of string** as shown

below.

```
cssArray:string[]=['red','size20'];
```

And, then use it in `ngClass` directive

```
<div class="row">
  <div [ngClass]="cssArray">
    Red Text with Size 20px  : from CSS Array
  </div>
</div>
```

**Using JavaScript object**

Create a class as shown below in your component

```

class CssClass {
 red: boolean= true;
 size20: boolean= true;
}

```

Next, create the instance of the CssClass in the component as shown below. You can change the value of the property true as false dynamically

```

cssClass: CssClass = new CssClass();

```

And then refer to the cssClass in your template.

```

<div class="row">
 <div [ngClass]="cssClass"> Red Text with Size 20px : from componer
</div>

```

You can download the source code from GitHub

# Angular ngStyle Directive

The Angular ngStyle directive allows us to set the many inline style of a HTML element using an expression. The expression can be evaluated at run time allowing us to dynamically change the style of our HTML element. In this tutorial, we learn how to use the ngStyle with an example.

**Table of Content** [hide]



## ngStyle Syntax

```
1
2 <element [ngStyle]="{'styleNames': styleExp}">...</element>
3
```

**Where**

element is the DOM element to which style is being applied

styleNames are style names ( ex: ‘font-size’, ‘color’ etc). with an optional suffix (ex: ‘top.px’, ‘font-style.em’),

styleExp is the expression, which is evaluated and assigned to the styleNames

We can add more than one key value pairs 'styleNames': styleExp each separated by comma.

In the following example, some-element gets the style of font-size of 20px.

```
1
2 <some-element [ngStyle]="{'font-size': '20px'}">Set Font size to 20px
3
```

The units (for example px, em) are prefixed to the styleName.

```
1
2 Syntax:
3 <element [ngStyle]="{'styleName.unit': widthExp}">...</element>
4
5 Example:
6 <some-element[ngStyle]="{'font-size.em': '3'}">...</some-element>
7
```

### ngStyle Example

### Change Style Dynamically

Initialize a variable `color` and add it to your component

```
color: string= 'red';
```

And in your template, add the following

```
<input [(ngModel)]="color" />
<div [ngStyle]="{'color': color}">Change my color</div>
```

In the above example, we apply `ngStyle` directive to the `div` element. We assign JavaScript object `{'color': color}` to the `ngStyle`directive. The variable `color` is dynamically changed by the user input and it is applied instantly to the `div` element

The following code uses the ternary operator to set the background color to **red** if the `status` variables indicator is set to “error” else **blue**.

```
1
2 div [ngStyle]="{'background-color':status === 'error' ? 'red' : 'blue' }">
3
```

**ngStyle multiple attributes**

We can change multiple style as shown in the following example

```
1
2 <p [ngStyle]="{'color': 'purple',
3             'font-size': '20px',
4             'font-weight': 'bold'}">
5    Multiple styles
6 </p>
7
8
```

The JavaScript object is assigned to the ngStyledirective containing multiple properties. Each property name of the object acts as a class name. The value of the property is the value of the style.

**Specifying CSS Units in ngStyle**

CSS has several units for expressing a length, size etc. The units can be em, ex, %, px, cm, mm, in, pt, PC etc. We prefix the units to the StyleName as shown below.

```
<input [(ngModel)]="size" />
<div [ngStyle]="{'font-size.px': size}">Change my size</div>
```

**Using object from Controller**

Create a class as shown below

```
class StyleClass {
  'color': string= 'blue';
  'font-size.px': number= 20;
  'font-weight': string= 'bold';
}
```

And in controller initialize the class

```
styleClass: StyleClass = new StyleClass();
```

Then you can refer it in your template as shown below

```
1
2 <div [ngStyle]="styleClass">size & Color</div>
3
```

Angular Global CSS styles

There are several ways to add Global (Application wide styles) styles to the Angular application. The styles can be added inline, imported in index.html or added via angular-cli.json. The angular allow us to add the component specific styles in individual components, which will override the global styles. In this article we will learn how to add global CSS styles to angular apps. We will also learn how to add custom CSS files & external style sheet to angular application..

**Table of Content**

Example Application

First, create an example application using the following command

```
ng new GlobalStyle
```

Let us add new component

```
ng g c test
```

The above command will create the TestComponent under the folder test and adds it to the AppModule. You can learn more such Angular CLI Commands from the Angular CLI Tutorial.

Open the test.component.html and add the following HTML

```
<p>
 this para is from test component
</p>

```

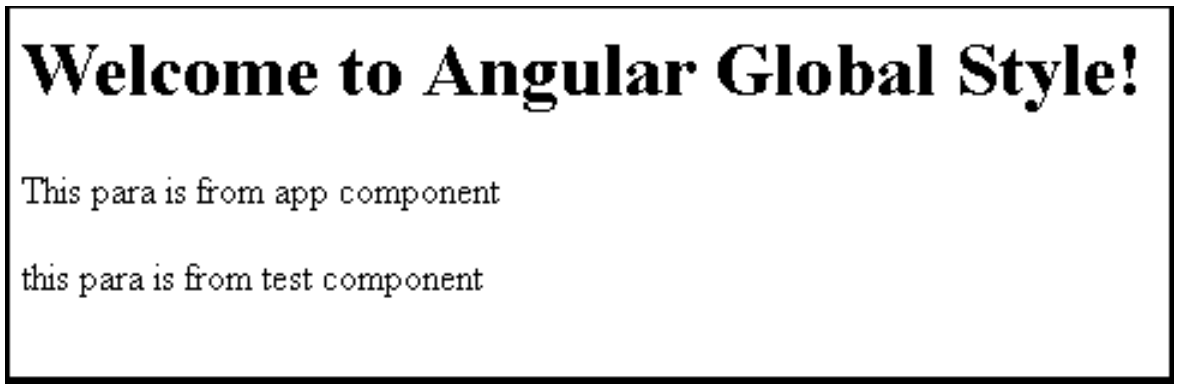
Now open the app.component.html add copy the following HTML

```
<h1>
 Welcome to {{ title }}!
</h1>

<p>This para is from app component</p>

<app-test></app-test>

```

**app.component.ts**

```
1
2
3 import { Component } from '@angular/core';
4
5 @Component({
6  selector: 'app-root',
7  templateUrl: './app.component.html',
8  styleUrls: ['./app.component.css']
9 })
1 export class AppComponent {
0
   title = 'Angular Global Style';
1
1 }
1
2
1
3
```

Run the app and you will see the following



Now let us add the **global CSS Styles** to the above example application

### Adding global CSS styles

### Using Angular-CLI

If you have created the Angular App using Angular CLI, then you can add the custom CSS files in `angular.json` under the `styles` array

`angular.json` was known as `angular-cli.json` in angular 5 and below

You will find it under the node projects-> GlobalStyle -> architect -> build -> options -> styles

By default, the angular adds the `styles.css` under the `src` folder.

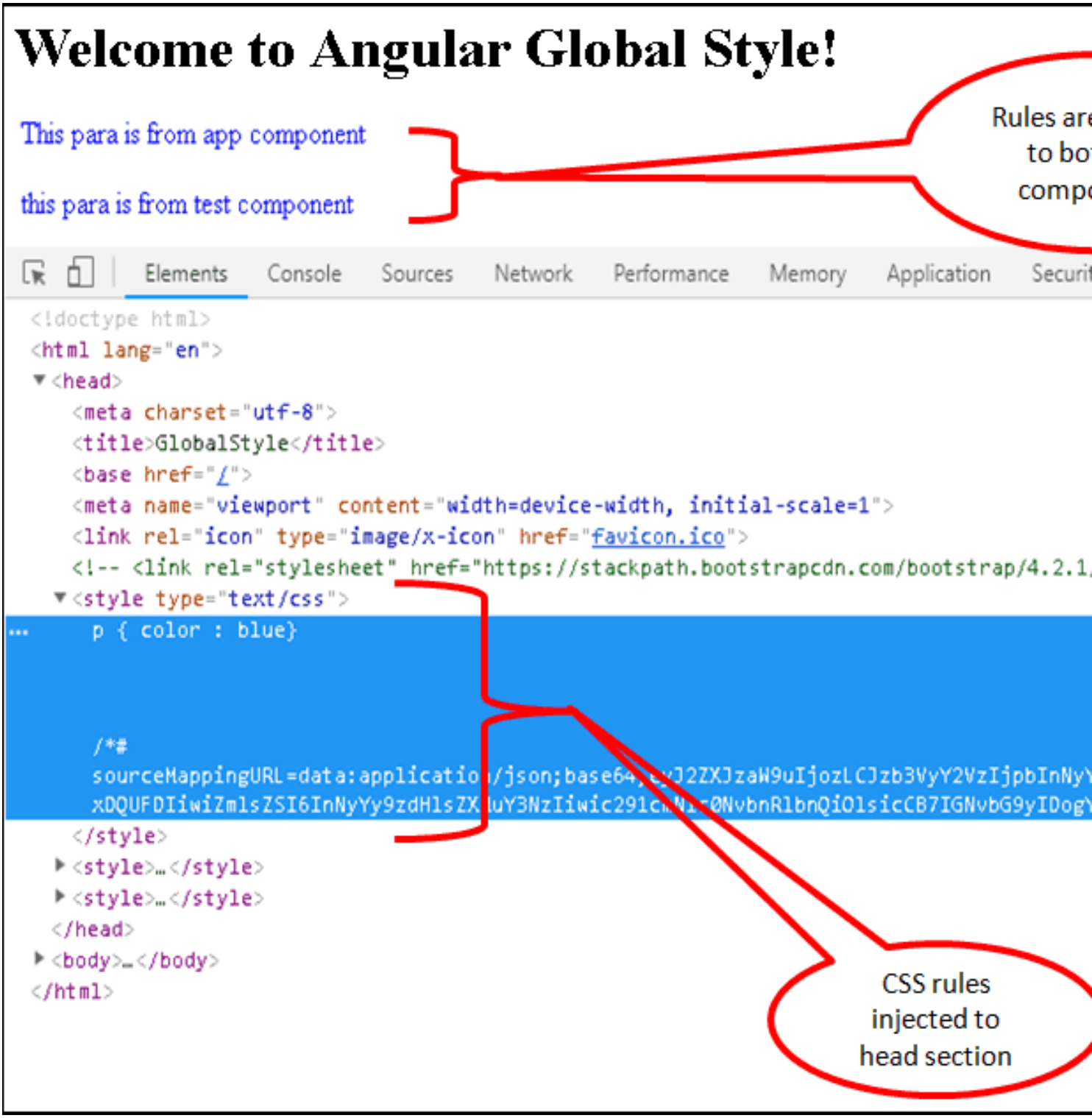
```
        ],
        "styles": [
          "src/styles.css"
        ],

```

The reference to the CSS file is relative to where `angular.json` file is stored. which is project root folder

Open the `styles.css` and add the following CSS rule

```
1
2 p { color : blue}
3
```

When you add CSS files using the `angular.json` configuration file, the CSS rules are bundled into the `styles.bundle.js` and injected into the `head` section




### Adding multiple style sheet

Create a morestyles.css under the folder src/assets/css and add the following CSS style

```
p { color : red}
```

Next, add the CSS file to the angular.json as shown below.

```
"styles": [
  "src/styles.css",
  "src/assets/css/morestyles.css"
],
```

Order of the styles sheets are important as the last one overrides the previously added CSS rules.

### Adding external style sheet

There are three ways you add the external style sheets.

### Copy them locally

For example to include bootstrap 4 you can copy the latest version from the link https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css and copy it under the folder assets/css/bootstrap.min.css

```

"styles": [
  "src/styles.css",
  "src/assets/css/morestyles.css",
  "src/assets/css/bootstrap.min.css"
],

```

The other option is to install the *npm package* provided by the third party libraries. The CSS files are copied under the *node_modules* folder. For Example to install bootstrap run the following *npm* command

```

npm install bootstrap

```

And then add it to the `angular.json` as shown below

```
"styles": [
  "src/styles.css",
  "src/assets/css/morestyles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
```

**Import it in one of the style sheets**

You can import them directly in one of the style sheets. For Example open the styles.css and add the following import statement **at the top.**

```
@import "https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/boots
```

**Adding Styles directly**

If you are not using angular-cli, then you an go old school and link it directly in the index.html file as shown below. You can use this even if you are using the angular-cli.

```
1
2 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootst
3
```

The following includes the local CSS files.

```
1
2 <link rel="stylesheet" href="assets/css/morestyles.css">
3
```

The path must be with reference to the `index.html`

These styles sheets are not included in the bundle, but loaded separately unlike when you are using angular-cli.