

## **Directives in Angular:**

We will see,

1. What is angular directive?
2. Why use angular directives?
3. Types of Angular directive?
4. Hands on: Directives in Angular

### **What is directive in Angular?**

In Angular, Directives are defined as classes that can add new behaviour to the elements in the template or modify existing behaviour.

### **Why we need directives?**

The purpose of Directives in Angular is to makeover the DOM, by adding new elements to DOM or removing elements and even changing the appearance of the DOM elements.

**Example:** Let's assume that we are getting some data from any remote resource or from any service. If data is not a single valued data. Consider we are getting list of customers or employees and we must display all of them on our html page, so in order to achieve this scenario one type directives can be useful. Sometimes on some events we will have to change style so for such kind of purpose we can also use directives.

Similarly we can also create our own directives too.

## **Types of Directives in Angular**

Directive in angular can be categorized into the following types: Component Directive, Structural Directive, and Attribute directive.

Let's consider each of these Angular Directive types separately.

### **Component Directive**

Special directives in Angular are called Components since this type of directive has a template or template URLs. In effect, it is a component directive that shows something in DOM.

### **Structural Directive**

This type of directive is used to make changes in the layout of the DOM. Elements can be added or removed, hence changing the structure of the DOM. An example would be `*ngIf`(adding or removing element from DOM) or `*ngFor`(lists elements of every iteration).

## **Attribute directive**

This type of angular directive is used to make behaviour or element changes in the appearance and behaviour of an element. For example, `ngStyle( applying styles)` or `ngClass( applying CSS classes)`.

## **Custom Directives**

We can create our own directive too. We will see practically how to create custom directive.  
Example: We will create a directive to change text. Which will be our custom directive.

Angular allows us to extends html syntax using custom directive for providing some customer behaviour but for this purpose we will have register custom directives with angular compiler. So, when compiler comes across custom directive, angular compiler will get to know it's nothing new it's extended html syntax.

## **Component Directive:**

Let's start with component directive. We have already discussed about component directive before but in today's session we will see some additional information about directives. So, all of us know when we create angular component, we will get four files among which one file is ts that is

class file. So, in class file we have @Component decorator. Which has following properties:

```
@Component({
  selector: 'app-root',
  template: '<h1>Welcome To Citiustech</h1>',
  styleUrls: ['./app.component.css']
})
```

Here we can say component as directive because here the selector which we have is going to display our template page into other html page. Because if in normal html page we will add <app-root> then it will be printed as it is because html doesn't know what exactly it is but as it's inbuilt directive of angular so, while compiling angular application, compiler will know what is meaning of <app-root>. And app.component.html page will be integrated to index.html. So, component is also called as directive which is going to change behaviour of html. Similarly, if we don't have to embed the template for <app-root> selector and have to just display only one or two elements then we can also specify those elements in template: instead of templateUrl. Which is as given below:

```
TS app.component.ts X
src > app > TS app.component.ts > AppComponent
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    template: '<h1>Welcome To Citiustech</h1>',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'directiveandpipes';
10 }
11
```

## Structural Directive:

Structural directives are directives which change the DOM layout by adding and removing DOM elements.

Angular provides a set of built-in structural directives (such as `NgIf`, `NgForOf`, `NgSwitch` and others) which are commonly used in all Angular projects.

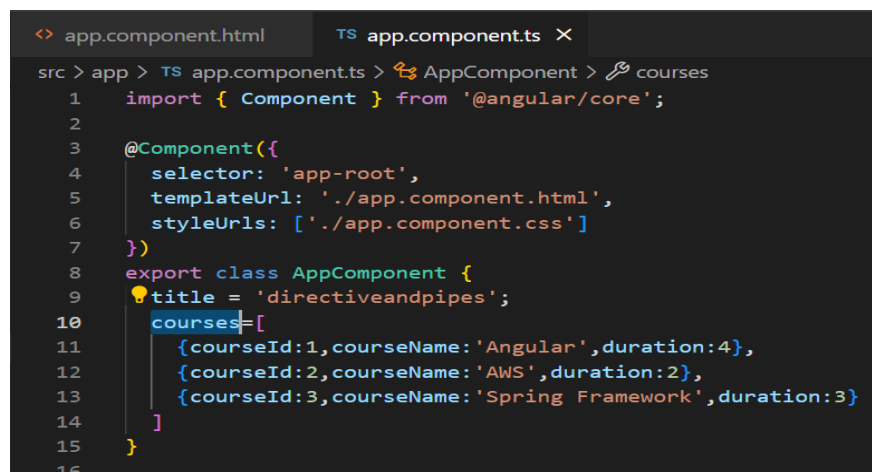
Make a note for all structural directives, are starting with asterisk `*`.

First, we will talk about `ngIf`. So, when you will be working on real project, you may have some scenarios where you will have to show or hide some part of page based on some conditions then we can use `*ngIf` structural directive.

Let's start with example:

In `app.component.ts` file we will declare and define an array

**Step 1:** add one object in `app.component.ts`



```
<> app.component.html TS app.component.ts X
src > app > TS app.component.ts > AppComponent > courses
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'directiveandpipes';
10   courses=[
11     {courseId:1,courseName:'Angular',duration:4},
12     {courseId:2,courseName:'AWS',duration:2},
13     {courseId:3,courseName:'Spring Framework',duration:3}
14   ]
15 }
16
```

Add given below code in app.component.html file

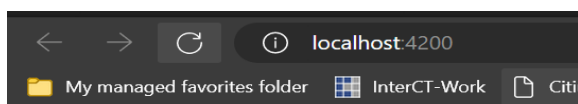
```
app.component.html X TS app.component.ts
src > app > app.component.html > div
1 <h1>Directives Understanding</h1>
2 <div *ngIf="courses.length >0">
3 <h2>List Of Courses</h2>
4 </div>
5 <div *ngIf="courses.length ==0">
6 <h2>No Courses</h2>
7 </div>
```

This will print List of courses because our courses object has values. But this approach is older one. It was using till angular 4.0.

If we go back to app.component.ts and empty our array and run code again then we will get output as No courses.

```
app.component.html X TS app.component.ts X
src > app > TS app.component.ts > AppComponent > courses
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'directiveandpipes';
10  courses=[]
11
12
13 }
14
```

```
app.component.html X TS app.component.ts
src > app > app.component.html > div
1 <h1>Directives Understanding</h1>
2 <div *ngIf="courses.length >0">
3 <h2>List Of Courses</h2>
4 </div>
5 <div *ngIf="courses.length ==0">
6 <h2>No Courses</h2>
7 </div>
```



## Directives Understanding

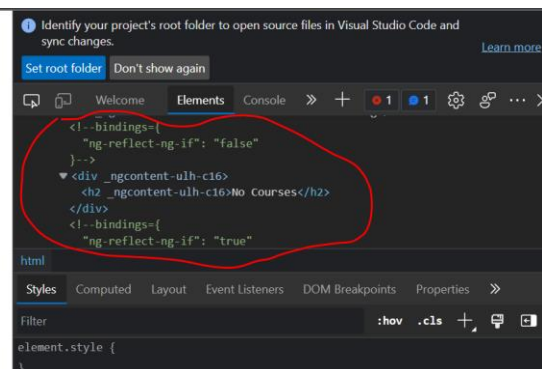
No Courses

Now, right click on browser and click on inspect. So, in elements we

can see only No course div will be only there.  
Can't see other div.

## Directives Understanding

No Courses



So, from this we can conclude, structural directives add or remove elements from DOM. So, if condition is false it has removed above element. Same will happen if courses will have courses. It will add only above div which is holding List Of Courses.

→if else

From Angular 4, instead of repeating \*ngIf statement multiple times, we have approach which is like if else which is present in other programming languages.

Let's see how to achieve if else:

```
<!-- Way 2 -->
<div *ngIf="courses.length > 0 then ifcourses else ifNocourses"></div>
<ng-template #ifcourses>
  <h2>List Of Courses</h2>
</ng-template>
<ng-template #ifNocourses>
  <h2>No Courses Available</h2>
</ng-template>
```

## Extra demo for multiple conditions: **(Skip)**

```
<!-- Form multiple conditions -->

<!-- <div *ngIf="courses.length ==3 || courses.length ==1; then thcourses;
else nocourse"></div>
<ng-template #thcourses>
  Three Courses
</ng-template>
<ng-template #onecourse>
  One Courses
</ng-template>
<ng-template #nocourse>
  No Courses
</ng-template> -->
```

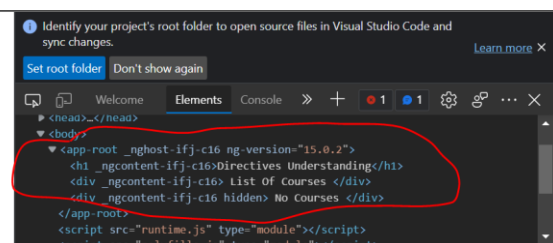
Let's see one more approach for achieving same requirement using hidden property.

```
<!-- Way 3 using hidden property -->
<div [hidden]="courses.length>0">
  List Of Courses
</div>
<div [hidden]="courses.length ==0">
  No Courses
</div>
```

Above course will also work properly as per our requirement but if we will inspect it then we will get to know that hidden property is not removing element from DOM, it's just hiding element.

### Directives Understanding

List Of Courses



This is difference between \*ngIf and hidden. In \*ngIf element won't be added to DOM but using



hidden, element will be just hidden but element will be present in DOM.

## **\*ngSwitchCase:**

Here we got one more structural directive that is \*ngSwitch. This is same which we have in other programming language.

Here upto some extent we can use ngIf for same but, ngIf works with only true and false but when we will have to work with multiple choices/conditions then we should use ngSwitch.

Let try to Understand it with simple example:

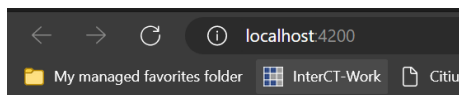
### **Step 1. Open app.component.html**

```
<!-- ngSwitch -->
<ul>
  <li [class.active]="viewMode=='laptop'"><button (click)="viewMode='laptop'">Laptop</button></li>
  <li [class.active]="viewMode=='mobile'"><button (click)="viewMode='mobile'">Mobiles</button></li>
</ul>
<div [ngSwitch]="viewMode">
  <div *ngSwitchCase="'laptop'">
    <ol>
      <li>Lenovo Thinkpad</li>
      <li>Mackbook Pro</li>
      <li>Dell Inspiron</li>
      <li>HP Pavilion</li>
    </ol>
  </div>
  <div *ngSwitchCase="'mobile'">
    <ol>
      <li>Iphone 12 Mini</li>
      <li>Iphone 13 Mini</li>
      <li>Iphone 12 Max Pro</li>
      <li>Iphone 13 Mini</li>
    </ol>
  </div>
  <div *ngSwitchDefault> Select From Category </div>
</div>
```

## Step 2. Open app.component.ts

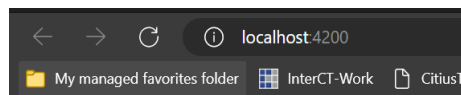
```
<> app.component.html X TS app.component.ts X
src > app > TS app.component.ts > AppComponent > viewMode
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'directiveandpipes';
10   viewMode = 'option';
11   courses = [
12
13   ]
14 }
```

Output of above demo will be like:



### Directives Understanding

- Laptop
  - Mobiles
1. Iphone 12 Mini
  2. Iphone 13 Mini
  3. Iphone 12 Max Pro
  4. Iphone 13 Mini



### Directives Understanding

- Laptop
  - Mobiles
1. Lenovo Thinkpad
  2. Mackbook Pro
  3. Dell Inspiron
  4. HP Pavilion

## \*ngFor:

We use this structural directive to render list of objects. We have already seen ngFor but let's see again with some additional information.

Let's understand it with example:

**Step 1.** In app.component.ts file , we will create new property courses and set this to an array of objects.

```
src > app > TS app.component.ts > AppComponent > courses > courseDuration
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'directiveandpipes';
10   viewMode = 'option'
11   courses = [
12     {courseId:101,courseName:'AWS', courseDuration:'3 months'},
13     {courseId:102,courseName:'Angular', courseDuration:'1 months'},
14     {courseId:103,courseName:'Spring', courseDuration:'1 months'}
15   ]
16 }
17
```

**Step 2.** Open app.component.html file and add given below code.

```
<!-- *ngFor -->
<div >
  <ol *ngFor="let course of courses">
    <li>{{course.courseName}}</li>
  </ol>
</div>
```

Which will give us output:

### Directives Understanding

- Laptop
- Mobiles

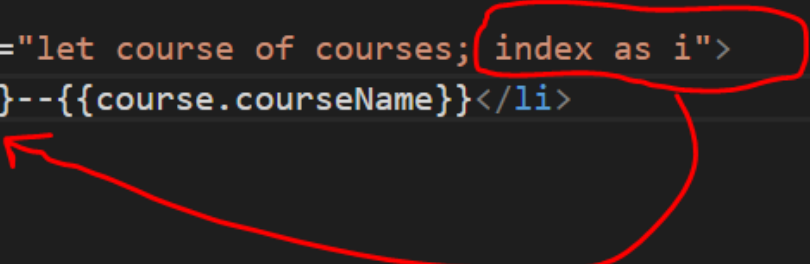
Select From Category

- AWS
- Angular
- Spring

Along with ngForOf we have some additional functionalities too. Example, we have exported data from a table and we have to provide index to data then we can use index method.

See code given below:

```
<!-- *ngFor -->
<div>
  <ol *ngFor="let course of courses; index as i">
    <li>{{i}}--{{course.courseName}}</li>
  </ol>
</div>
```



Similarly, we have some more too. Like even, is first, last and many more.

For reference visit:

<https://angular.io/api/common/NgForOf>

- **index: number** : The index of the current item in the iterable.
- **count: number** : The length of the iterable.
- **first: boolean** : True when the item is the first item in the iterable.
- **last: boolean** : True when the item is the last item in the iterable.
- **even: boolean** : True when the item has an even index in the iterable.
- **odd: boolean** : True when the item has an odd index in the iterable.

Index and count functions return us number but rest all return Boolean value.

One more example for even:

# App.component.html

```
<!-- *ngFor -->
<div>
  <ol *ngFor="let course of courses; index as i; even as isEven">
    <li>{{i}}--{{course.courseName}} <span *ngIf="isEven">(Even)</span></li>
  </ol>
</div>
```

Select From Category

1. 0--AWS (Even)

1. 1--Angular

1. 2--Spring (Even)

## ngFor directive and change detection:

ngFor directive also respond to change. How does it takes place, for this we will see an example.

To Check it, we will add a button on top of our previous demo to add new course.

**Step 1.** Open app.component.html file.

```
<!-- *ngFor -->
<button (click)="addCourse()">Add Course</button>
<div>
  <ol *ngFor="let course of courses; index as i; even as isEven">
    <li>{{i}}--{{course.courseName}} <span *ngIf="isEven">(Even)</span></li>
  </ol>
</div>
```

**Step 2.** Open app.component.ts file

```
<> app.component.html TS app.component.ts X
src > app > TS app.component.ts > AppComponent > addCourse > courseDuration
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'directiveandpipes';
10   viewMode = 'option'
11   courses = [
12     {courseId:101,courseName:'AWS', courseDuration:'3 months'},
13     {courseId:102,courseName:'Angular', courseDuration:'1 months'},
14     {courseId:103,courseName:'Spring', courseDuration:'1 months'}
15   ]
16   addCourse(){
17     this.courses.push({courseId:105,courseName:'Core Java',courseDuration:'20 days'});
18   }
19 }
20
```

So, now on click event new course will be added and in background angular will keep detecting changes and update too so we can also see newly added page on our html page.

### **Attribute Directives:**

Angular provides us two in built attribute directives and those are ngClass and ngStyle. They are called as attribute directives because they are use like attribute in normal html attribute. Attribute directives are use to change appearance of template.

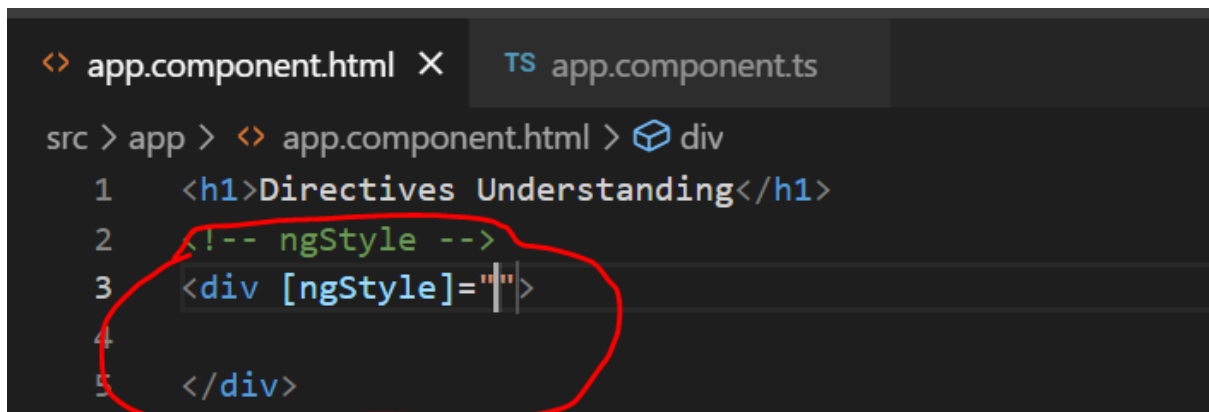
If you are familiar with java script we have one method that is addClass() which helps us to add new style to element but angular provides us ngClass directive which helps us to add multiple classes too, similarly ngStyle will also helps us to apply multiple style to elements.

Let's try to understand how we can implement it.

### **ngStyle:**

Let's implement ngStyle first. ngStyle is used to implement multiple styling to any element. To use ngStyle we will have to put it in square bracket [ngStyle]=" and have to provide expression. This expression is of type object.

**Step 1.** Open app.component.html and add simple line.

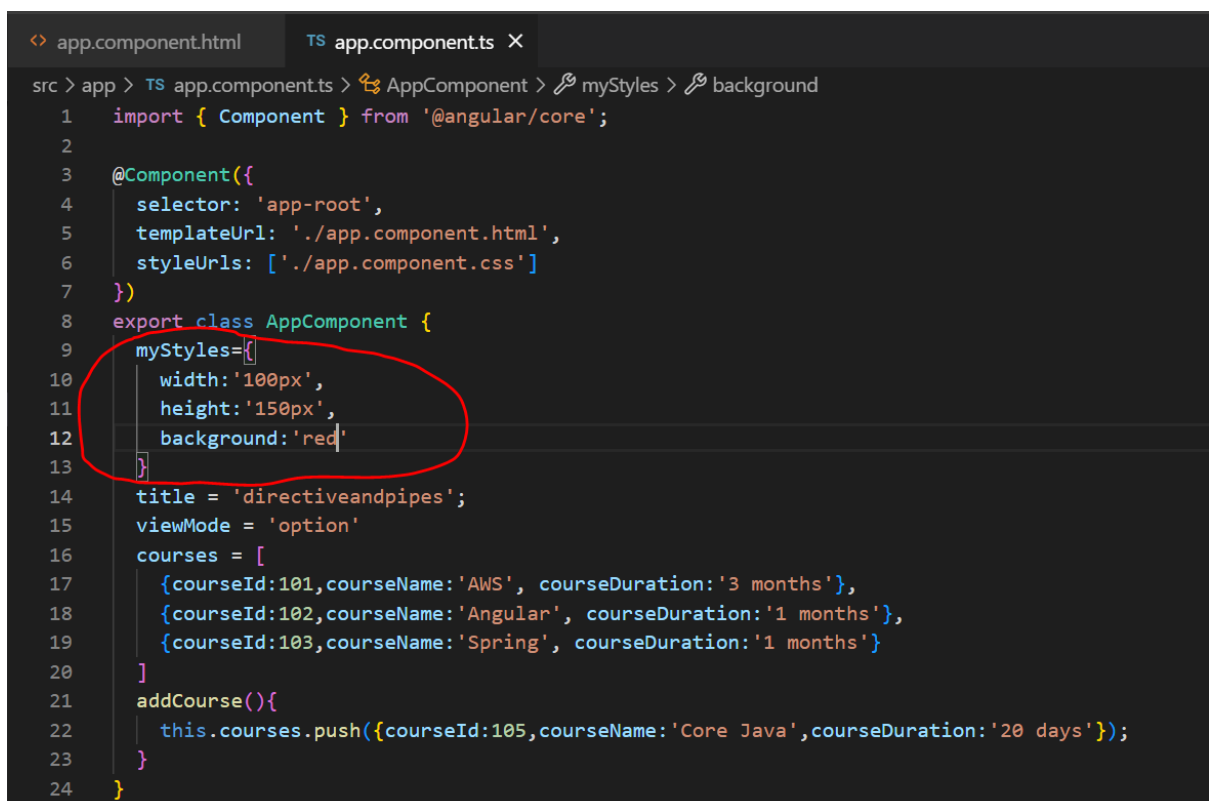


```
<> app.component.html X TS app.component.ts

src > app > <> app.component.html > div
1 <h1>Directives Understanding</h1>
2 <!-- ngStyle -->
3 <div [ngStyle]="">
4
5 </div>
```

In order to provide expression for ngStyle, we will create an object in app.component.ts

**Step 2.** Open app.component.ts file and add given below code.



```
<> app.component.html TS app.component.ts X

src > app > TS app.component.ts > AppComponent > myStyles > background
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   myStyles={
10     width:'100px',
11     height:'150px',
12     background:'red'
13   }
14   title = 'directiveandpipes';
15   viewMode = 'option'
16   courses = [
17     {courseId:101,courseName:'AWS', courseDuration:'3 months'},
18     {courseId:102,courseName:'Angular', courseDuration:'1 months'},
19     {courseId:103,courseName:'Spring', courseDuration:'1 months'}
20   ]
21   addCourse(){
22     this.courses.push({courseId:105,courseName:'Core Java',courseDuration:'20 days'});
23   }
24 }
```

**Step 3.** Now, open app.component.html file and add this object as an expression to [ngStyle]

```
<> app.component.html X TS app.component.ts
src > app > <> app.component.html > div
1 <h1>Directives Understanding</h1>
2 <!-- ngStyle -->
3 <div [ngStyle]="myStyles">
4
5 </div>
```

Directives Understanding



We can also add such styles dynamically too means on button's click event we can implement ngStyle.

**Step 4.** Open app.component.html and add given below code.

```
<> app.component.html X TS app.component.ts
src > app > <> app.component.html > button
1 <h1>Directives Understanding</h1>
2 <!-- ngStyle -->
3 <div [ngStyle]="myStyles">
4 | | Welcome to Citiustech
5 </div>
6 <button (click)="changeStyle()">Change Style</button>
```

**Step 5.** Open app.component.ts file and add given below code.

```
<> app.component.html TS app.component.ts X
src > app > TS app.component.ts > AppComponent > changeStyle
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   myStyles={
10     width: '100px',
11     height: '150px',
12     background: 'red',
13     color: 'black'
14   }
15   changeStyle(){
16     this.myStyles['background']='green';
17     this.myStyles['color']='white';
18   }
```



## ngClass:

ngClass in angular directive is used to add multiple classes based on condition.

As I told you in java script, we can only one class but Angular's ngClass directive helps us to add multiple classes. Let's see how to do it.

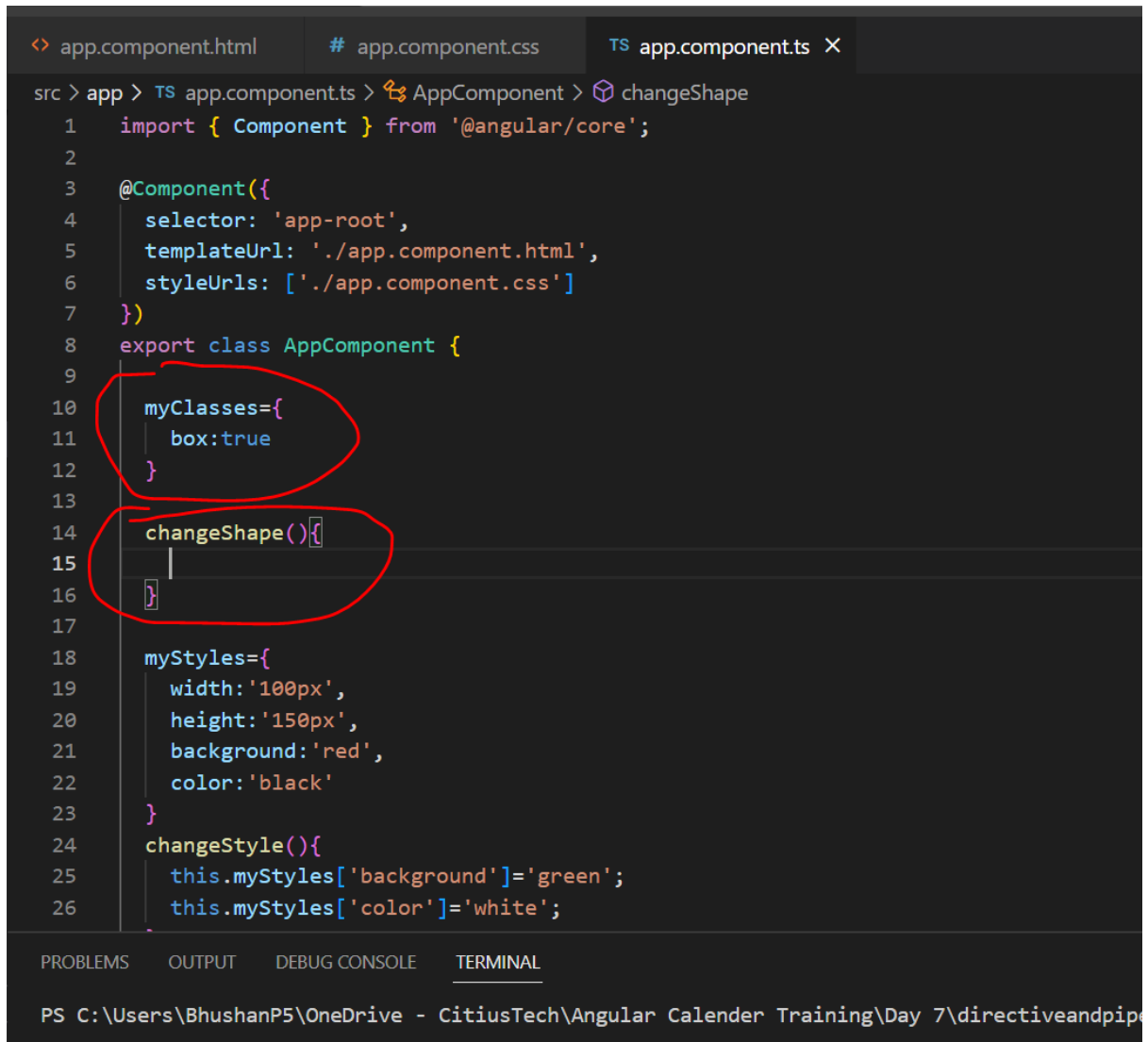
**Step 1.** Open app.component.html file and add given below code.

```
<> app.component.html X TS app.component.ts
src > app > <> app.component.html > button
1  <h1>Directives Understanding</h1>
2  <!-- ngClass -->
3  <div>
4
5  </div>
6  <button (click)="changeShape()">Change Shape</button>
7
```

**Step 2.** Open app.component.css file and add given below style.

```
<> app.component.html X # app.component.css X TS ap
src > app > # app.component.css > .circle
1  .box{
2    width: 100px;
3    height: 150px;
4    background-color: tomato;
5  }
6  .border{
7    border: 2px solid black;
8  }
9
10 .circle{
11   border-radius: 50%;
12 }
```

**Step 3.** Now, we will have to get classes defined in .css file in .ts file and set condition whether true or false.

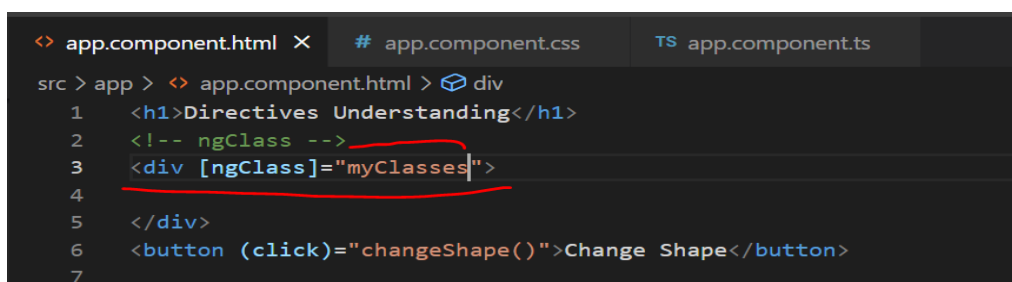


```
<> app.component.html # app.component.css TS app.component.ts X
src > app > TS app.component.ts > AppComponent > changeShape
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9
10   myClasses={
11     box:true
12   }
13
14   changeShape(){
15   }
16
17
18   myStyles={
19     width:'100px',
20     height:'150px',
21     background:'red',
22     color:'black'
23   }
24   changeStyle(){
25     this.myStyles['background']='green';
26     this.myStyles['color']='white';
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\BhushanP5\OneDrive - CitiusTech\Angular Calendar Training\Day 7\directiveandpipe

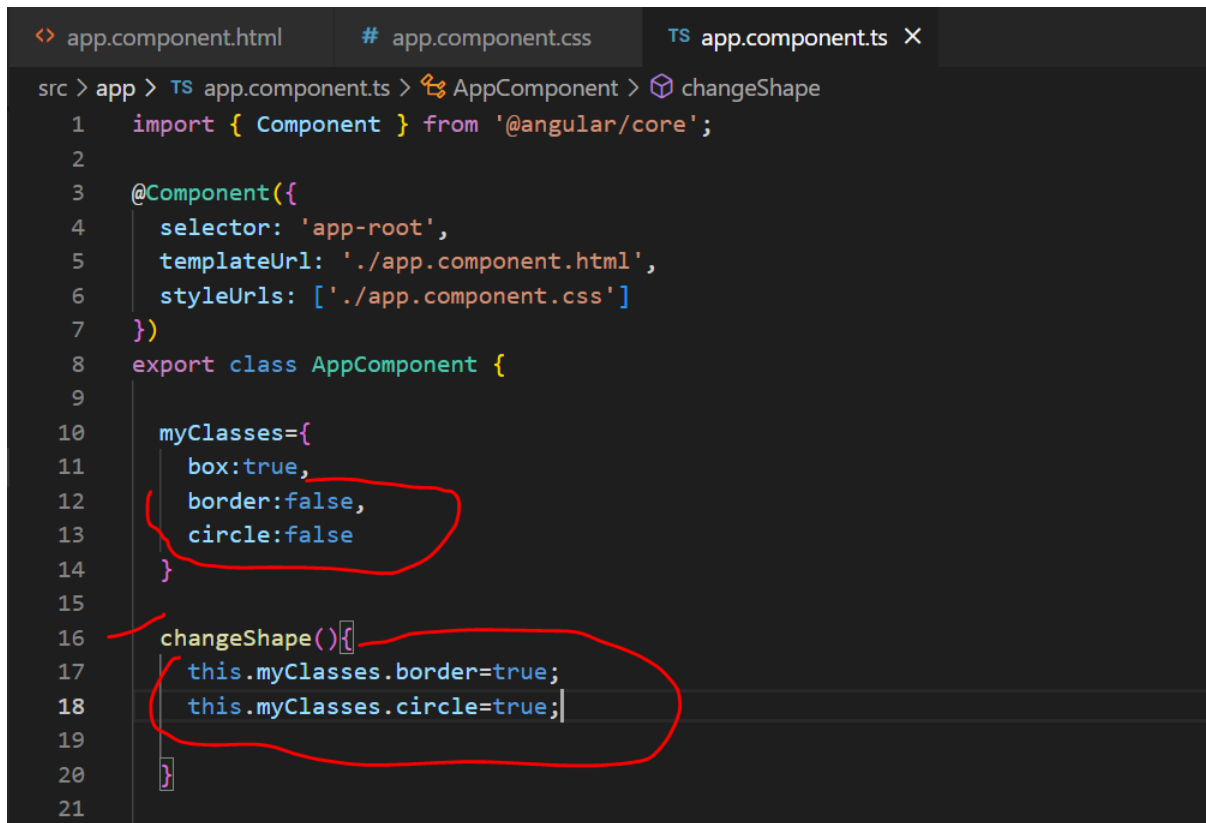
**Step 4.** Open app.component.html file and set myClasses object



```
<> app.component.html X # app.component.css TS app.component.ts
src > app > <> app.component.html > div
1  <h1>Directives Understanding</h1>
2  <!-- ngClass -->
3  <div [ngClass]="myClasses">
4
5  </div>
6  <button (click)="changeShape()">Change Shape</button>
7
```

Above code will display red rectangle but now we have to change shape on button click event so, we will have to write given below code in app.component.ts file.

**Step 5.** Open app.component.ts file and given below code.



```
<> app.component.html # app.component.css TS app.component.ts X
src > app > TS app.component.ts > AppComponent > changeShape
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9
10   myClasses={
11     box:true,
12     border:false,
13     circle:false
14   }
15
16   changeShape(){
17     this.myClasses.border=true;
18     this.myClasses.circle=true;
19   }
20
21 }
```

Now, on button click event shape of rectangle will be changed.

In this way we can use attribute directives to change appearance of elements.

## Custom Directive:

So far, we have seen built in directives. Now we will see how to create custom directive. Let's consider a scenario where you are responsible for creating a custom directive which will be responsible to change colour of text.

In order to achieve this we will have to follow steps given below:

**Step 1.** We will have to create/add directive. To do so, we will have to use command.

**ng g directive changecolor**

**Step 2.** Once our directive will be created then open it. Here you will see that this directive class file is having new decorator that is @Directive. Which include selector. Here selector is playing same role means it's identifier for this directive. Using this selector, we can use this directive anywhere we want.

For implementing our requirement, we will have to use one class here and it's name is ElementRef.

So,

1. what is **ElementRef**? --ElementRef is a class that can hold a reference to a DOM element.
2. How to use it? –First we will have to add it in import at top of our directive.ts file and inject it using constructor.

```
TS changecolor.directive.ts X
src > app > TS changecolor.directive.ts > ChangecolorDirective > constructor
1  import { Directive , ElementRef } from '@angular/core';
2
3  @Directive({
4    selector: '[appChangecolor]'
5  })
6  export class ChangecolorDirective {
7
8    constructor(private element:ElementRef) { }
9
10 }
11
```

To complete our requirement I will have to add given below code here.

```
TS changecolor.directive.ts X # app.component.css <> app.component.html
src > app > TS changecolor.directive.ts > ChangecolorDirective
1  import { Directive ,ElementRef } from '@angular/core';
2
3  @Directive({
4    selector: '[appChangecolor]'
5  })
6  export class ChangecolorDirective {
7
8    constructor(private element:ElementRef) {
9      this.element.nativeElement.style.color='blue';
10     this.element.nativeElement.style.background='Yellow'
11   }
12
13 }
14
```

So, now I can say my custom directive is ready now. Wherever I will have to use this styling I can simply use this custom directive just by using it's selector.

## Pipes:

Use pipes to transform strings, currency amounts, dates, and other data for display. Pipes are simple functions to use in template expressions to accept an input value and return a transformed value.

Pipes are useful because you can use them throughout your application, while only declaring each pipe once.

## App.component.ts:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'day7demos';
  userName='BhushanP';
  object: Object = {name: 'John', email: 'john@gmail.com',
    additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789,
987654321]}};
  a: number = 0.259;
  currentDate = new Date();
}
```

## App.component.html

```
<h1>Welcome to Citiustech</h1>
<!-- Uppercase Pipe -->
<h2>Welcome {{userName | uppercase}}</h2>
<!-- Lowercase pipe -->
<h2>Welcome {{userName | lowercase}}</h2>
<!-- Json Pipe -->
<div>
```

```

    <p>Without JSON pipe:</p>
    <pre>{{object}}</pre>
    <p>With JSON pipe:</p>
    <pre>{{object | json}}</pre>
  </div>

  <!-- Percent Pipe -->
  <div>
    <!--output '26%'-->
    <p>A: {{a | percent}}</p>
  </div>

  <!-- Date Pipe -->
  <h2>{{currentDate}}</h2>
  <h2>{{currentDate | date: 'short'}}</h2>
  <h2>{{currentDate | date: 'medium'}}</h2>

```

For detail information about pipe visit.

<https://angular.io/api?query=pipe>

## Custom Pipes

What we have seen for built in pipes, same way we can create our own pipe.

### What is reason for having custom pipes?

Let's consider a scenario that there is data which we have to display but instead of displaying data as it we have to do some changes. Example: if string is very long and we have to display only 8 characters followed by ... then in order to achieve this we will create our pipe.

### How to create custom pipe?

Either we can add using angular cli or we can create it manually.

Command to create new pipe is

ng g p pipe\_name

```
PS C:\Users\BhushanP5\OneDrive - CitiusTech\Angular Calender Training\Day 8\Day 8> cd .\day7demos\  
PS C:\Users\BhushanP5\OneDrive - CitiusTech\Angular Calender Training\Day 8\Day 8\day7demos> ng g p cit  
ipipe  
CREATE src/app/citipipe.pipe.spec.ts (195 bytes)  
CREATE src/app/citipipe.pipe.ts (221 bytes)  
UPDATE src/app/app.module.ts (459 bytes)
```

Let's open our pipe and try to understand

```
1  import { Pipe, PipeTransform } from '@angular/core';  
2  
3  @Pipe({  
4    name: 'citipipe'  
5  })  
6  export class CitipipePipe implements PipeTransform {  
7  
8    transform(value: any) {  
9      // transform(value: any, limit:number) {  
10       // console.log(value);  
11       return value.substr(0, 5)+'...';  
12     }  
13  }
```

After creating pipe, we will get one pipe.

Pipe\_name.pipe.ts. This file has **@Pipe** decorator in which we will have to provide name for our pipe. This class implements **PipeTransform** interface.


This interface has one method which we will have to implement that method is

**transform(value:any){ }**. In this method here is one parameter and also can have one more optional parameter. Whatever transformation we will have to implement that transformation has to develop in transform () method.



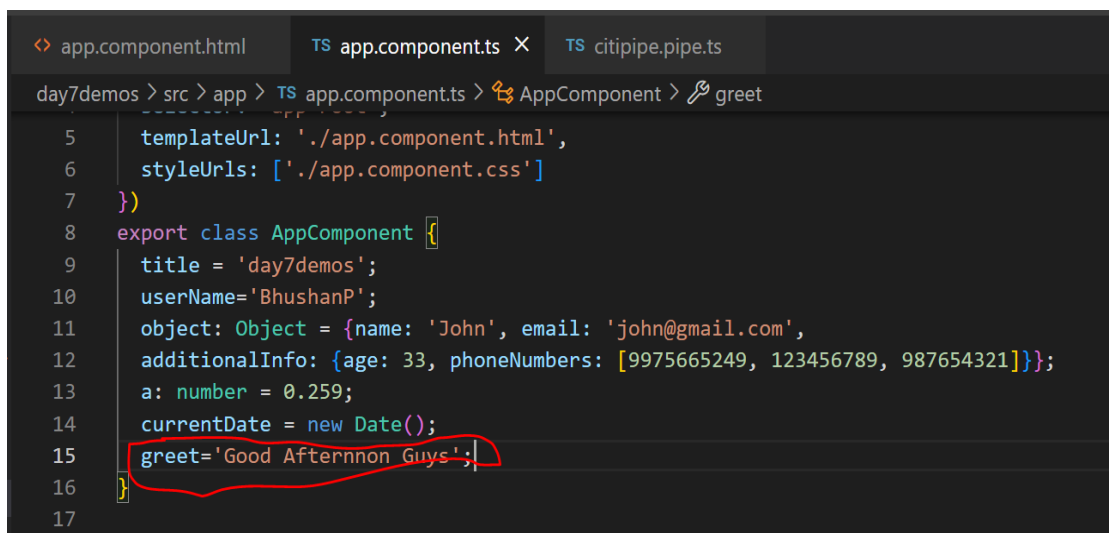
Let's try it by implementing it.

**Step 1.** We have created new pipe, open this pipe and add given below code.



```
<> app.component.html TS app.component.ts TS citipipe.pipe.ts X
day7demos > src > app > TS citipipe.pipe.ts > CitipipePipe > transform
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'citipipe'
5  })
6  export class CitipipePipe implements PipeTransform {
7
8    transform(value: any) {
9      // transform(value: any, limit:number) {
10     // console.log(value);
11     return value.substr(0, 5)+'...';
12   }
13
14 }
```

Now, add one property in app.component.ts.



```
<> app.component.html TS app.component.ts X TS citipipe.pipe.ts
day7demos > src > app > TS app.component.ts > AppComponent > greet
5  templateUrl: './app.component.html',
6  styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'day7demos';
10   userName='BhushanP';
11   object: Object = {name: 'John', email: 'john@gmail.com',
12   additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789, 987654321]}};
13   a: number = 0.259;
14   currentDate = new Date();
15   greet='Good Afternoon Guys';
16 }
17
```

Open app.component.html

```
app.component.html X TS app.component.ts TS citipipe.pipe.ts
day7demos > src > app > app.component.html > h2
16 <!--output '26%'-->
17 <h2>A: {{a | percent}}</h2>
18 </div>
19
20 <!-- Date Pipe -->
21 <h2>{{currentDate}}</h2>
22 <h2>{{currentDate | date: 'short'}}</h2>
23 <h2>{{currentDate | date: 'medium'}}</h2>
24
25 <h2>{{greet | citipipe}}</h2>
```

**Step 2.** Now, we will implement pipe to accept parameter. For this we will create an array of employees name and we want to display only 5 characters from name and if name will be having more 5 characters then our pipes should show only 5 characters followed by three dots. ...

#### a. Add an array in **app.component.ts**

```
app.component.html TS app.component.ts X TS citipipe.pipe.ts
day7demos > src > app > TS app.component.ts > AppComponent > employees
7 })
8 export class AppComponent {
9   title = 'day7demos';
10  userName = 'BhushanP';
11  object: Object = {name: 'John', email: 'john@gmail.com',
12  additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789, 987654321]}};
13  a: number = 0.259;
14  currentDate = new Date();
15  greet = 'Good Afternoon Guys';
16  employees = [
17    'Paul Adams',
18    'Brian',
19    'Tom',
20    'Keanu Reeves',
21    'John Connor'
22  ]
```

#### b. Display this array in app.component.html file.

```
<h2>{{greet | citipipe}}</h2>
<ul *ngFor="let emp of employees">
  <li>{{emp}}</li>
</ul>
```

Now, add our pipe on it

```
<h2>{{greet | citipipe}}</h2>
<ul *ngFor="let emp of employees">
  <li>{{emp | citipipe}}</li>
</ul>
```

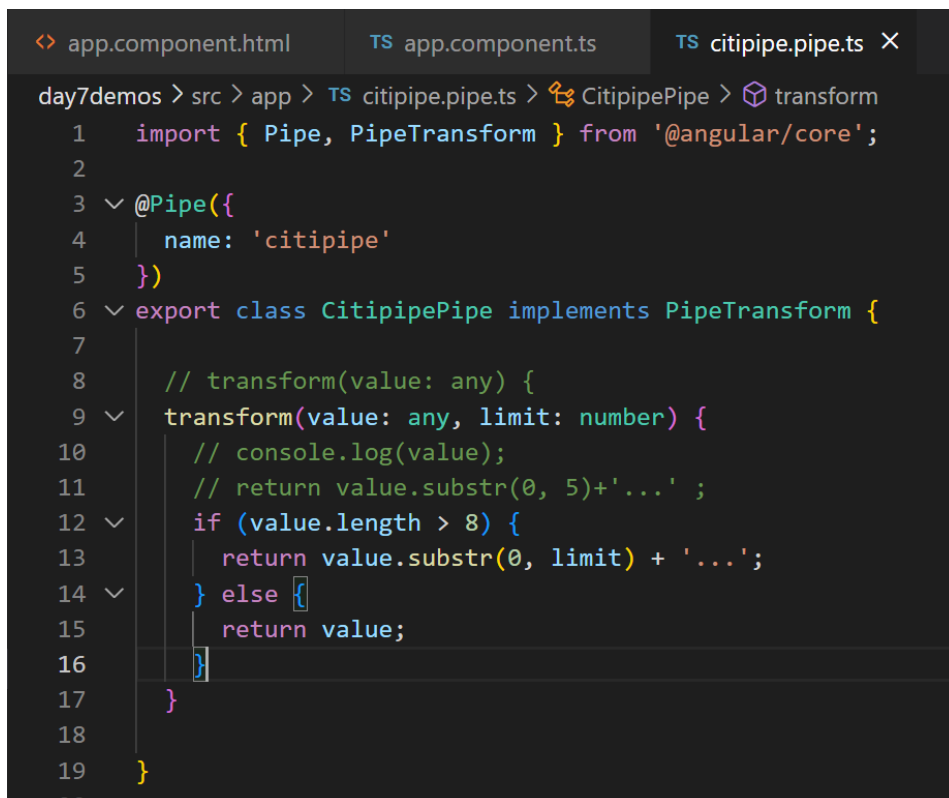
It will transform output.

c. Now, we will have to provide parameters.

Means number of characters we will have to provide dynamically.

**Step 3.** Now, our requirement says that if name of employee is less than limit then it shouldn't have three dots ...

Our final code will look like:



```
day7demos > src > app > TS citipipe.pipe.ts > CitipipePipe > transform
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'citipipe'
5  })
6  export class CitipipePipe implements PipeTransform {
7
8    // transform(value: any) {
9    transform(value: any, limit: number) {
10     // console.log(value);
11     // return value.substr(0, 5)+'...' ;
12     if (value.length > 8) {
13       return value.substr(0, limit) + '...';
14     } else {
15       return value;
16     }
17   }
18
19 }
```

Now, use our pipe with value as parameter as shown in given below picture.

A screenshot of a code editor with three tabs: 'app.component.html', 'TS app.component.ts', and 'TS citipipe.pipe.ts'. The active tab is 'app.component.html'. The code shows a series of lines with line numbers 22 through 28. Line 22: <h2>{{currentDate | date: 'short'}}</h2>. Line 23: <h2>{{currentDate | date: 'medium'}}</h2>. Line 24: (empty). Line 25: <h2>{{greet | citipipe:4}}</h2>. Line 26: <ul \*ngFor="let emp of employees">. Line 27: | <li>{{emp | citipipe:2}}</li>. Line 28: </ul>. Red annotations highlight the '4' in line 25 and the '2' in line 27.

In this way, we can create our custom pipe.

## Pure and Impure Pipe.

Whenever any pipe transforms our data that time angular execute that pipe.

Pure pipe are those pipes when pure change will be detected. So, pure pipes check when changes are detecting in values.

But impure pipes are getting executed on any change in our applications. So, impure pipes may affect performance of our application.

It's best practice to avoid impure pipes because it may affect performance of application.

Let's Understand it with an example.

**Step 1.** We will use array employees in app.component.ts file.

```
<> app.component.html • TS app.component.ts X TS citipipe.pipe.ts
day7demos > src > app > TS app.component.ts > AppComponent > employees
7   })
8   export class AppComponent {
9     title = 'day7demos';
10    userName='BhushanP';
11    object: Object = {name: 'John', email: 'john@gmail.com',
12    additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789, 987654321]}};
13    a: number = 0.259;
14    currentDate = new Date();
15    greet='Good Afternnon Guys';
16    employees=[
17      'Paul Adams',
18      'Brian',
19      'Tom',
20      'Keanu Reeves',
21      'John Conor'
22    ]
23  }
24
```

**Step 2.** Open app.component.html file and display this array using interpolation first then add text box and button to add employee runtime.

```
<> app.component.html • TS app.component.ts TS citipipe.pipe.ts
day7demos > src > app > <> app.component.html > button
21 <h2>{{currentDate}}</h2>
22 <h2>{{currentDate | date: 'short'}}</h2>
23 <h2>{{currentDate | date: 'medium'}}</h2>
24
25 <h2>{{greet | citipipe:4}}</h2>
26 <ul *ngFor="let emp of employees">
27   <li>{{emp | citipipe:2}}</li>
28 </ul>
29
30 <!-- pure and impure pipes -->
31 <input type="text" value="" #empName>
32 <button (click)="addEmp(empName.value)">Add</button>
33 <p>{{employees}}</p>
```

**Step 3.** implement this addEmp() method in app.component.ts file.

```
day7demos > src > app > TS app.component.ts > AppComponent > addEmp
10  username= 'brusnaar';
11  object: Object = {name: 'John', email: 'john@gmail.com',
12  additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789, 987654
13  a: number = 0.259;
14  currentDate = new Date();
15  greet='Good Afternnon Guys';
16  employees=[
17    'Paul Adams',
18    'Brian',
19    'Tom',
20    'Keanu Reeves',
21    'John Conor'
22  ]
23  addEmp(value :any){
24    this.employees.push(value);
25  }
26  }
27
```

**Step 4:** Add new custompipe by using command.

**ng g p custompipe.**

After firing command, we will get a custom pipe.

Do one change in it. **Change return null to return value.join().**

```
day7demos > src > app > TS custompipe.pipe.ts > CustompipePipe > transform
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'custompipe'
5  })
6  export class CustompipePipe implements PipeTransform {
7
8    transform(value: unknown, ...args: unknown[]): unknown {
9      return value;
10    }
11  }
12
13
```

```
day7demos > src > app > TS custompipe.pipe.ts > CustompipePipe
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'custompipe',
5    pure:false
6  })
7  export class CustompipePipe implements PipeTransform {
8
9    transform(value: any, ...args: unknown[]): unknown {
10     return value.join();
11   }
12
13 }
```

Now, open app.component.html and add this pipe on our values.

```
<!-- pure and impure pipes -->
<input type="text" value="" #empName>
<button (click)="addEmp(empName.value)">Add</button>
<p>{{employees | custompipe}}</p>
```

**Step 5: Now to go app.component.ts file and log array to console in addEmp() method.**

```

employees=[
  'Paul Adams',
  'Brian',
  'Tom',
  'Keanu Reeves',
  'John Conor'
]
addEmp(value :any){
  this.employees.push(value);
  console.log(this.employees);
}
}

```

So now if we will add value in text box, it will be displayed in console but won't display on html.

Paul Adams,Brian,Tom,Keanu Reeves,John Conor

Console:

```

app.component.ts:25
(6) ['Paul Adams', 'Brian', 'Tom', 'Keanu Reeves', 'John Conor', 'Bhusha n']
app.component.ts:25
(7) ['Paul Adams', 'Brian', 'Tom', 'Keanu Reeves', 'John Conor', 'Bhusha n', 'Rakesh']

```

Now, go to custompipe and set property pure to false.



```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'custompipe',
  pure:false
})
export class CustompipePipe implements PipeTransform {

  transform(value: any, ...args: unknown[]): unknown {
    return value.join();
  }
}
```

In this way pure and impure pipes works.