

Angular 2

Lesson 7—Directives



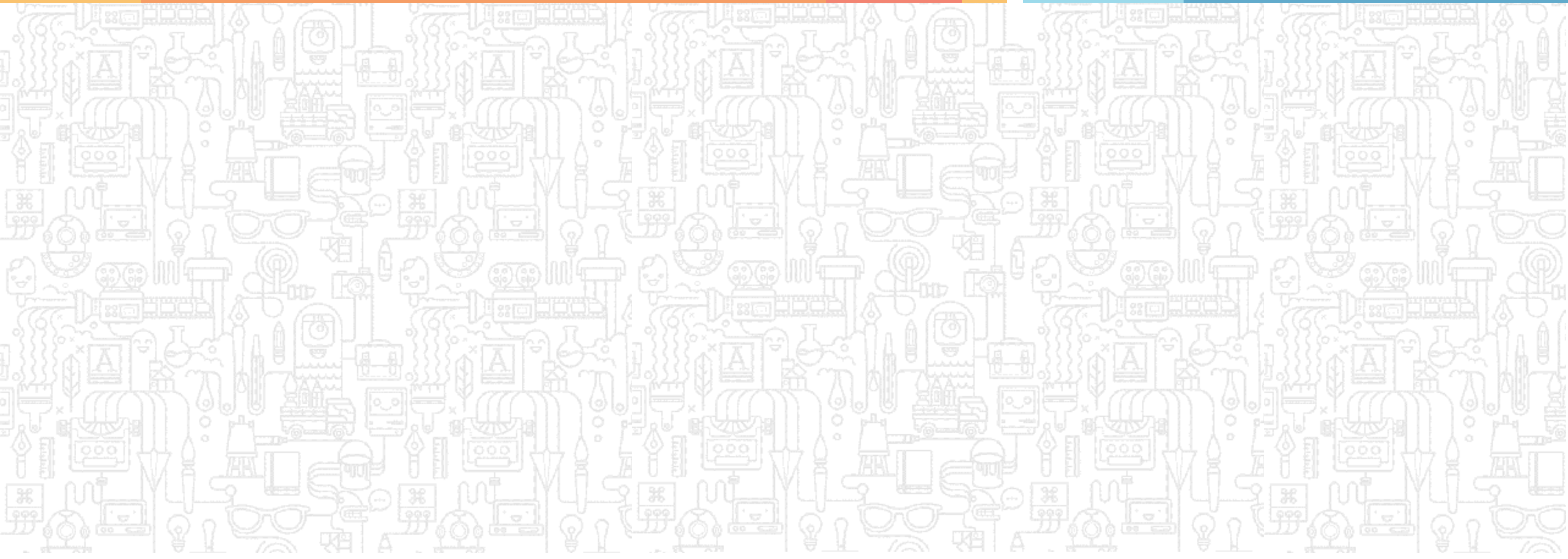
Learning Objectives



- ✔ Angular Directives
- ✔ Types of Angular Directives
- ✔ Built-in Angular Directives
- ✔ Working with Custom Directives

Directives

Topic 1 — Angular Directives



What Are Angular 2 Directives?

Directives are used to separate the re-usable functions and features of Angular 2.

This is an angular decorator. We use this decorator to define a directive (the same meaning as @Component).



Note

Unlike components, directives don't have a HTML template.
They add behavior to an existing DOM element.

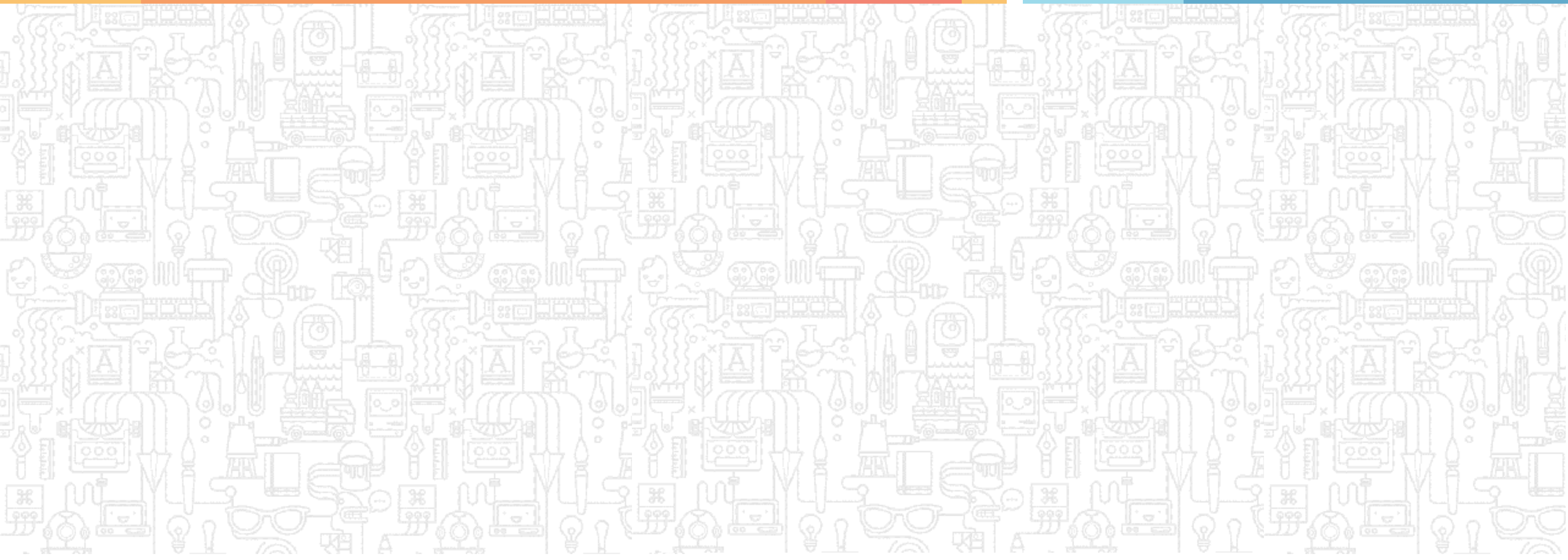
Angular 2 Directives—Example

The example below is used to display the added behavior to an existing DOM element, where the textbox size is enlarged on mouseover event.

```
<input type="text" autoGrow />
```

Directives

Topic 2—Types of Directives



Types of Directives

There are three types of directives, which include:

Components:

The component directive includes directives with the Template.

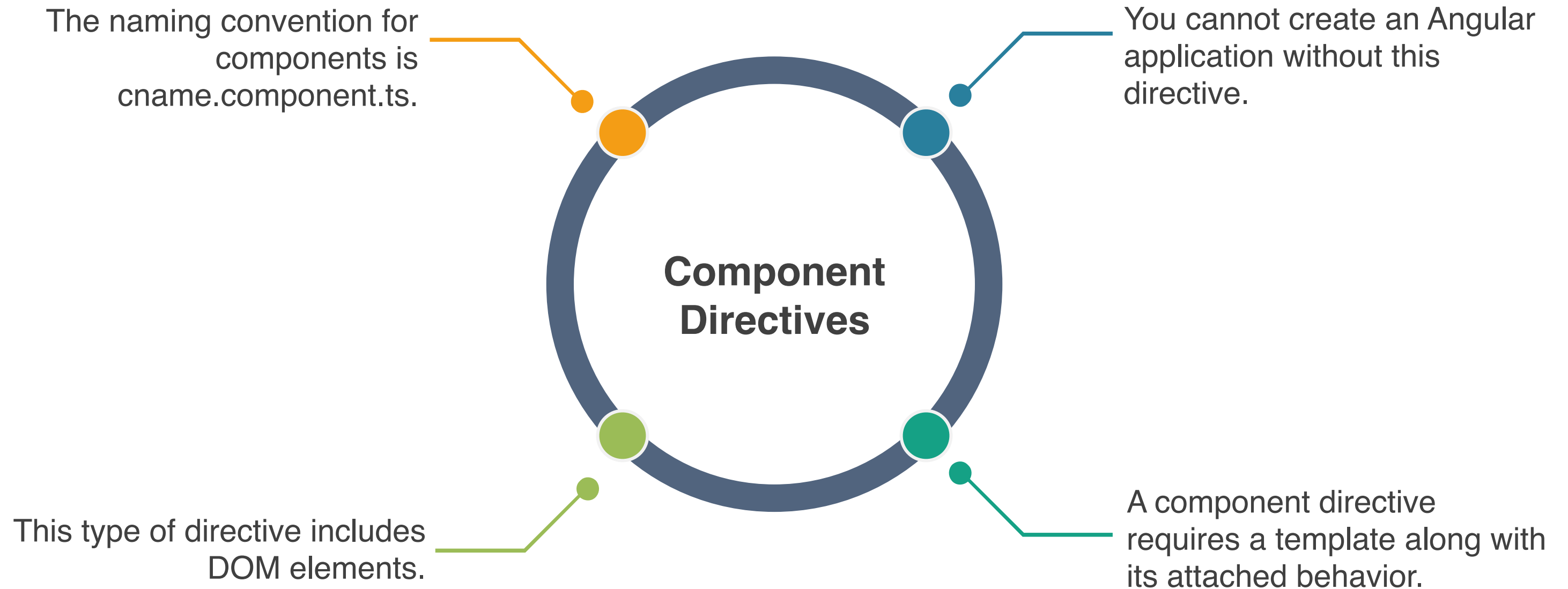
Structural Directives:

This directive changes the behavior of a component or element by affecting how the template is rendered.

Attributes Directives:

This type of directive changes the behavior of a calling component or element but doesn't affect the template.

Types of Directives—Component Directive



Types of Directives—Component Directive

This is how components look:

```
training@localhost:~  
app.component.ts  
    import {Component} from '@angular/core';  
    @Component({  
        selector: 'my-app',  
        template: `  
            <h1>Angular demo Boilerplate</h1>  
            <p>Hello World batch good afternoon friends!!  
</p>  
            <rb-mycomp></rb-mycomp>  
        `,  
    })  
    export class AppComponent{  
    }
```

Types of Directives—Attribute Directive

Attribute directives are used to change the appearance or behavior of a component or a native DOM element.

An attribute directive requires building a controller class annotated with `@Directive`, which specifies the selector that identifies the attribute.

The controller class implements the desired directive behavior.

In this sample below, we can implement the `colorChange` directive by using "colorChange."

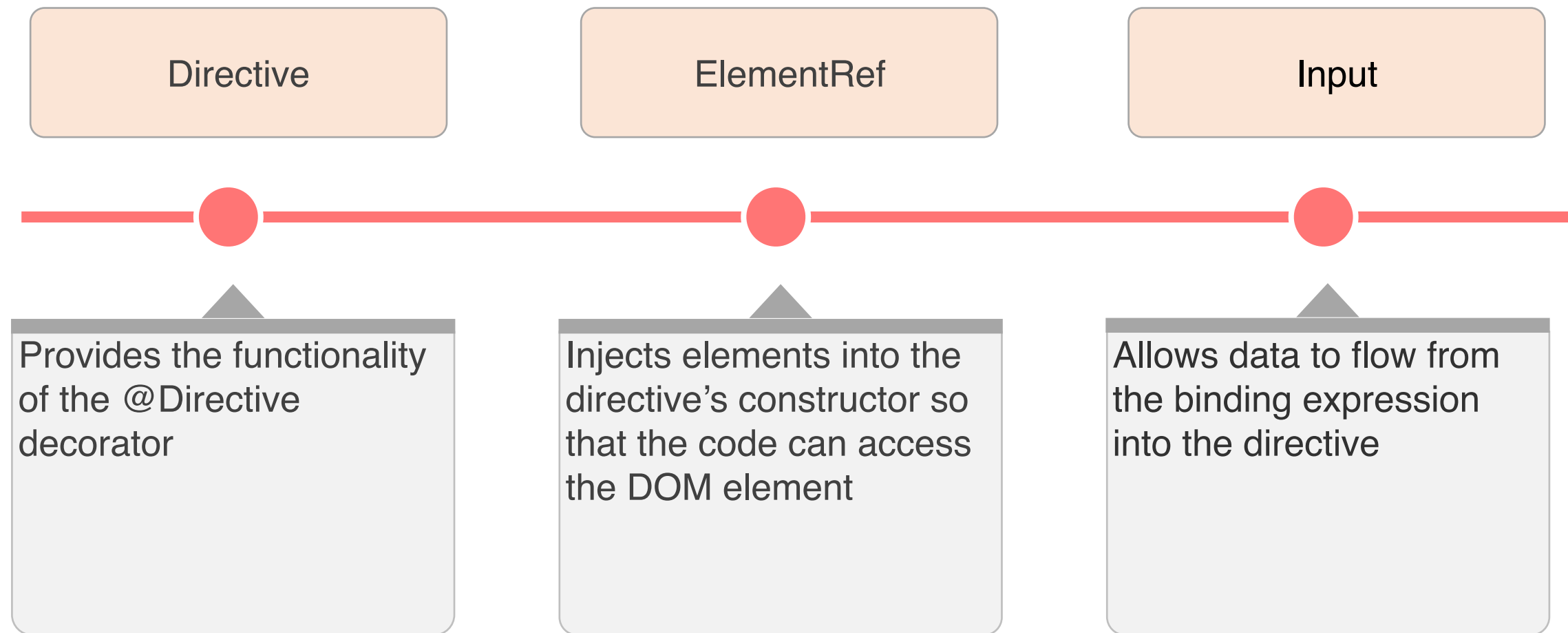
```
<div colorChange ><div>
```

```
<p colorChange ></p>
```

Types of Directives—Attribute Directive

There are predefined attribute directives.

The import statement specifies symbols from the Angular core.



Types of Directives—Attribute Directive

This is how the attribute looks:

 training@localhost:~

```
import { Directive, ElementRef, Input } from '@angular/core';

@Directive({ selector: '[ColorChange]' })

export class ColorChangeDirective {
  constructor(er: ElementRef) {
    er.nativeElement.style.backgroundColor = 'green';
  }
}
```

Types of Directives—Attribute Directive

There are two predefined, built-in attributes in Angular 2:



ngStyle

ngClass

Types of Directives—Attribute Directive

ngStyle

- Creating dynamic styles in web applications can be difficult.
- In Angular2, there are multiple ways to handle Dynamic CSS and CSS classes with the new template syntax.
- These directives offer syntactic sugar for more complex ways of altering element styles.
- It is possible to bind properties to values that can be updated by the user.

```
<div [ngStyle]="{'color': 'green', 'font-size':  
'24px', '}'">
```

working with style using ngStyle

```
</div>
```

Types of Directives—Attribute Directive

ngClass

- The ngClass directive changes the class attribute that is bound to the component or attached element.
- The ngClass Directive is an Angular 2 Attribute Directive that allows you to add or remove an HTML element of CSS class.
- Using ngClass, you can create dynamic styles in HTML pages.
- The ngClass attribute is applied to a DOM element; it is then bound to an expression.
- The ngClass attribute evaluates the expression and changes the class attribute of the element to which it is applied.

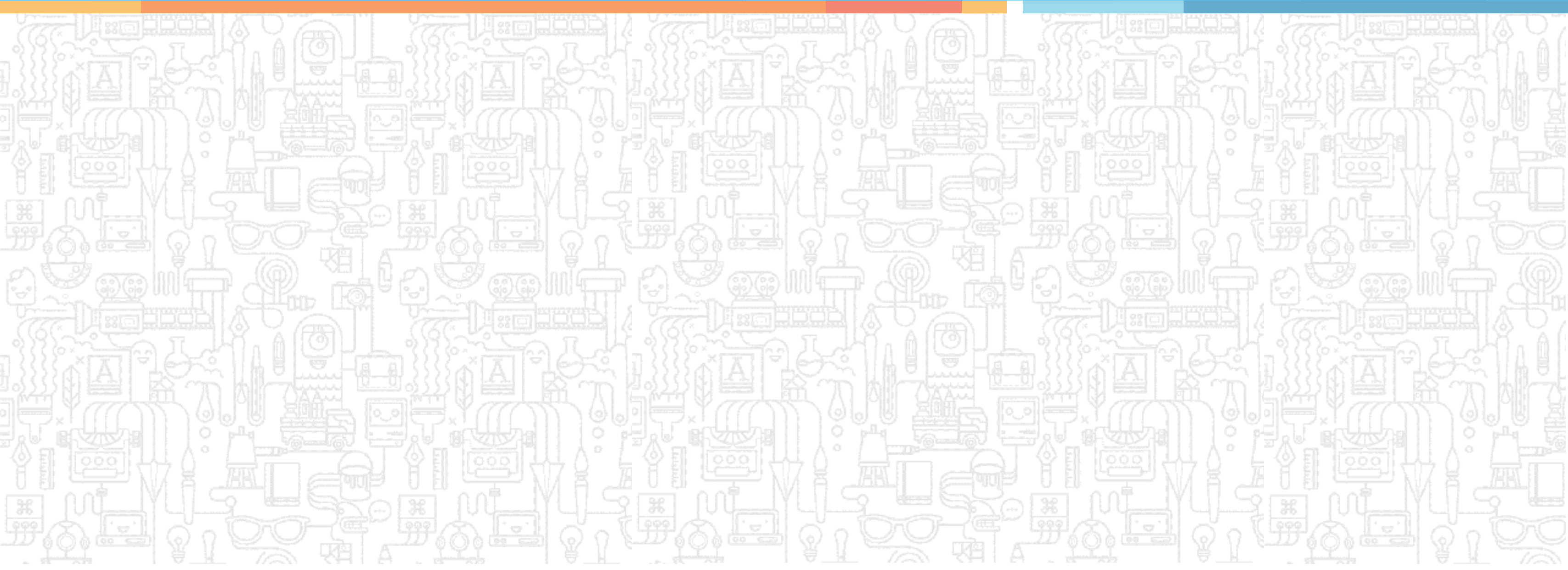
Types of Directives—Attribute Directive

ngClass

- The ngClass allows multiple ways to add and toggle CSS.
- It is possible to bind these classes directly to component properties to update them dynamically as needed.
- The expression of ngClass can be evaluated using 3 different options:
 - i. String
 - ii. Array
 - iii. Object

Directives

Topic 3—Built-in Angular Directives



Types of Directives—Binding to a String

- You can use the String as an expression and can bind it directly to the ngClass attribute.
- If multiple classes must be assigned, then separate each class with a space.

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

Example:

```
<div [ngClass]=" 'c11 c12' ">
```

```
User data Text with the default classes
```

```
</div>
```

Note: c11 and c12 are css classes

Types of Directives—Binding to an Array

The same result can be achieved by using an array instead of a string.

Example:

```
<div [ngClass]="['cl1' , 'cl2']">  
User data Text withthe default classes  
</div>
```

Note: cl1 and cl2 are css classes

Types of Directives—Binding to an 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.

Example:

```
div [ngClass]="['bold-text', 'green']">array of classes</div>
```

```
<div [ngClass]="['italic-text blue']">string of classes</div>
```

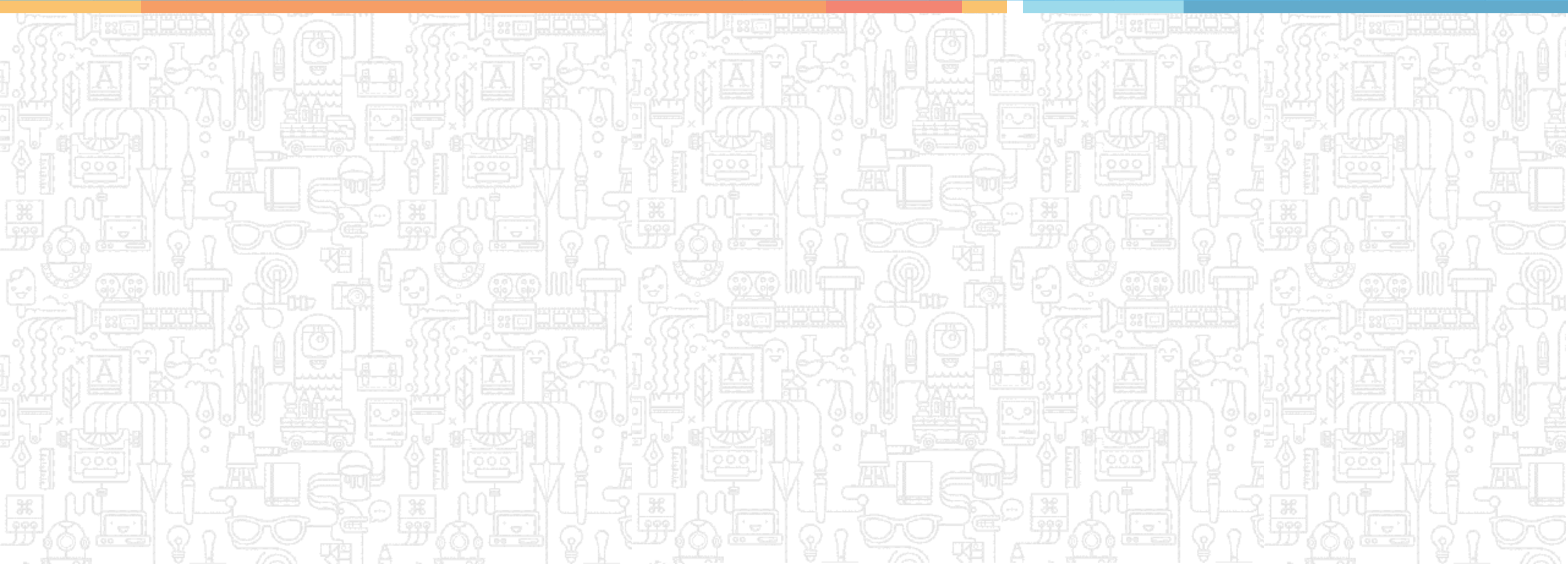
```
<div [ngClass]="{'small-text': true, 'red': true}">object of classes</div>
```

Demo—Attribute Directive



Directives

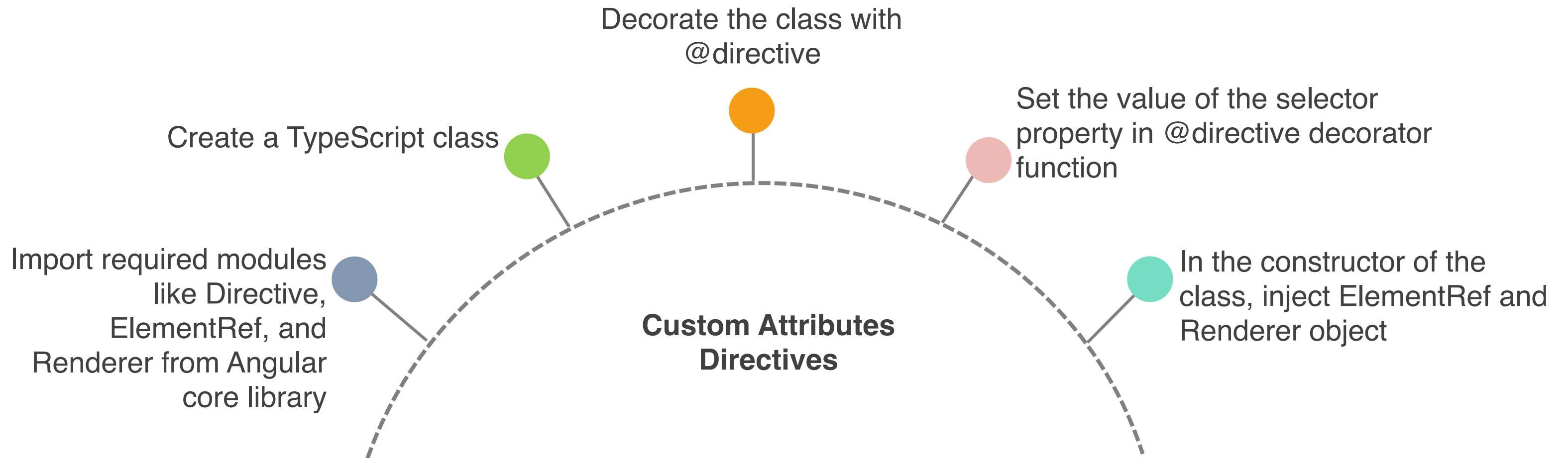
Topic 4—Custom Attribute Directives



Custom Attributes Directives

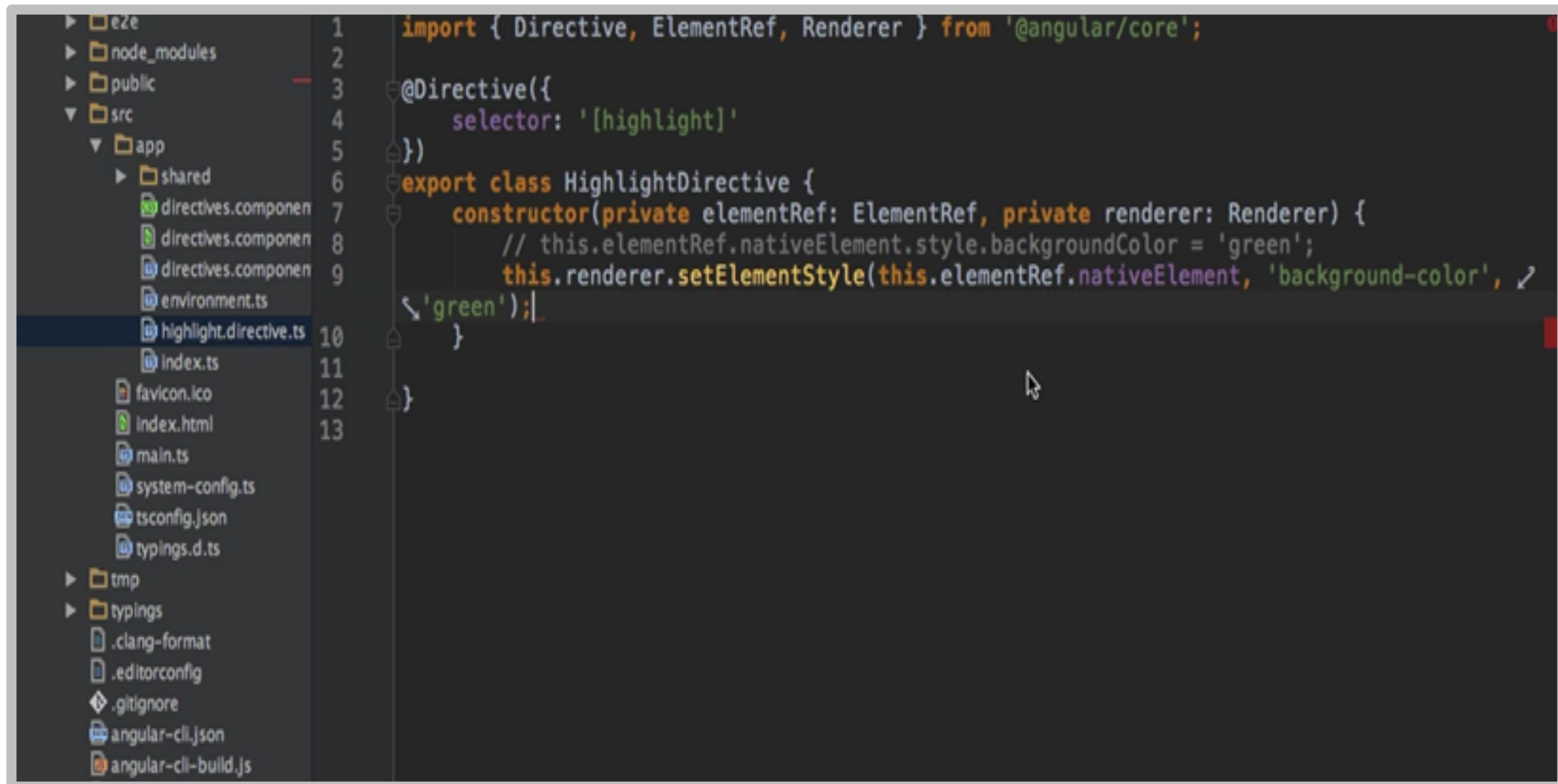
- Let's start with creating the Attribute Directive. To do this, you need to create a class and decorate it with **@directive** decorators.

There are few important points to remember:



Creating Custom Directive

This class is created by overriding ElementRef and Renderer.

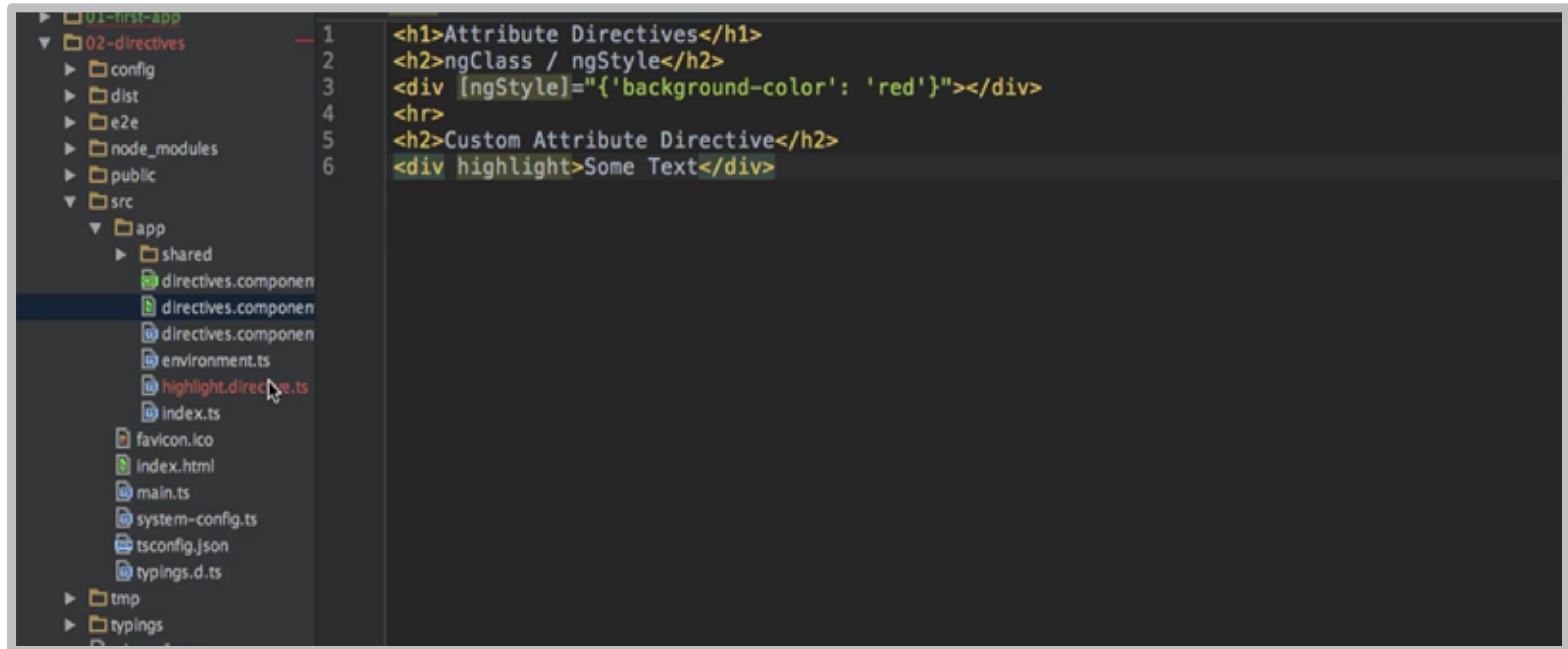


```
1  import { Directive, ElementRef, Renderer } from '@angular/core';
2
3  @Directive({
4      selector: '[highlight]'
5  })
6  export class HighlightDirective {
7      constructor(private elementRef: ElementRef, private renderer: Renderer) {
8          // this.elementRef.nativeElement.style.backgroundColor = 'green';
9          this.renderer.setStyle(this.elementRef.nativeElement, 'background-color',
10             'green');
11      }
12  }
```

The screenshot shows a code editor with a file explorer on the left. The file explorer displays a project structure with folders like 'e2e', 'node_modules', 'public', 'src', and 'tmp'. Under 'src/app', there's a 'shared' folder containing 'directives.component.ts', 'environment.ts', and 'highlight.directive.ts'. The 'highlight.directive.ts' file is selected, and its content is displayed in the editor. The code defines a custom directive 'HighlightDirective' that uses 'ElementRef' and 'Renderer' from '@angular/core'. It has a selector '[highlight]' and a constructor that takes 'elementRef' and 'renderer' as arguments. The constructor calls 'renderer.setStyle' to set the 'background-color' of the element to 'green'.

Creating Custom Directive

This is the method to “highlight” a directive in Template.



```
1 <h1>Attribute Directives</h1>
2 <h2>ngClass / ngStyle</h2>
3 <div [ngStyle]='{"background-color": "red"}'></div>
4 <hr>
5 <h2>Custom Attribute Directive</h2>
6 <div highlight>Some Text</div>
```

Structural Directive

- Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements.
- Angular has a powerful template engine that lets you easily manipulate the DOM structure of elements.
- Structural directives are easy to recognize. An asterisk (*) precedes the directive attribute name.

While an **Attribute** directive modifies the appearance or behavior of an element, a **Structural** directive modifies the DOM layout.

Structural Directive

Structural Directives handle how a component or element renders through the use of the template tag.

This helps you to run some code that decides what the final rendered output will be.

Angular has a few built-in structural directives:

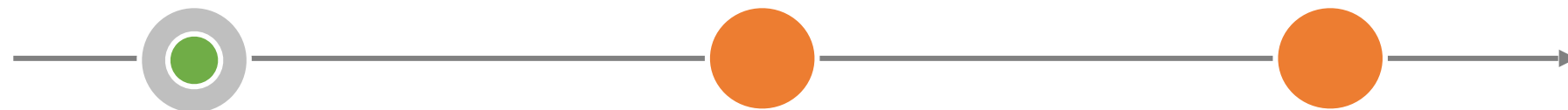


ngif Directive

- ngIf is the simplest structural directive and the easiest to understand.
- It takes a Boolean expression and makes an entire chunk of the DOM appear or disappear.
- The ngIf directive doesn't hide elements through CSS. It adds and removes them physically from the DOM.
- Confirm this using browser developer tools to inspect the DOM.

Example:

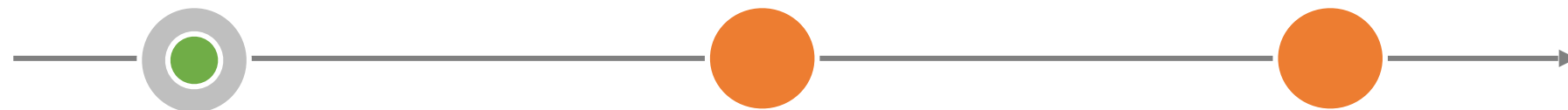
```
<app-if-example *ngIf="exists">
```



ngif Directive

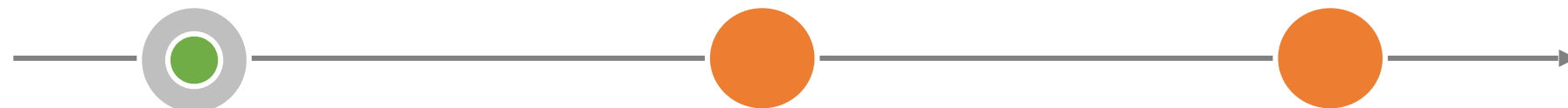
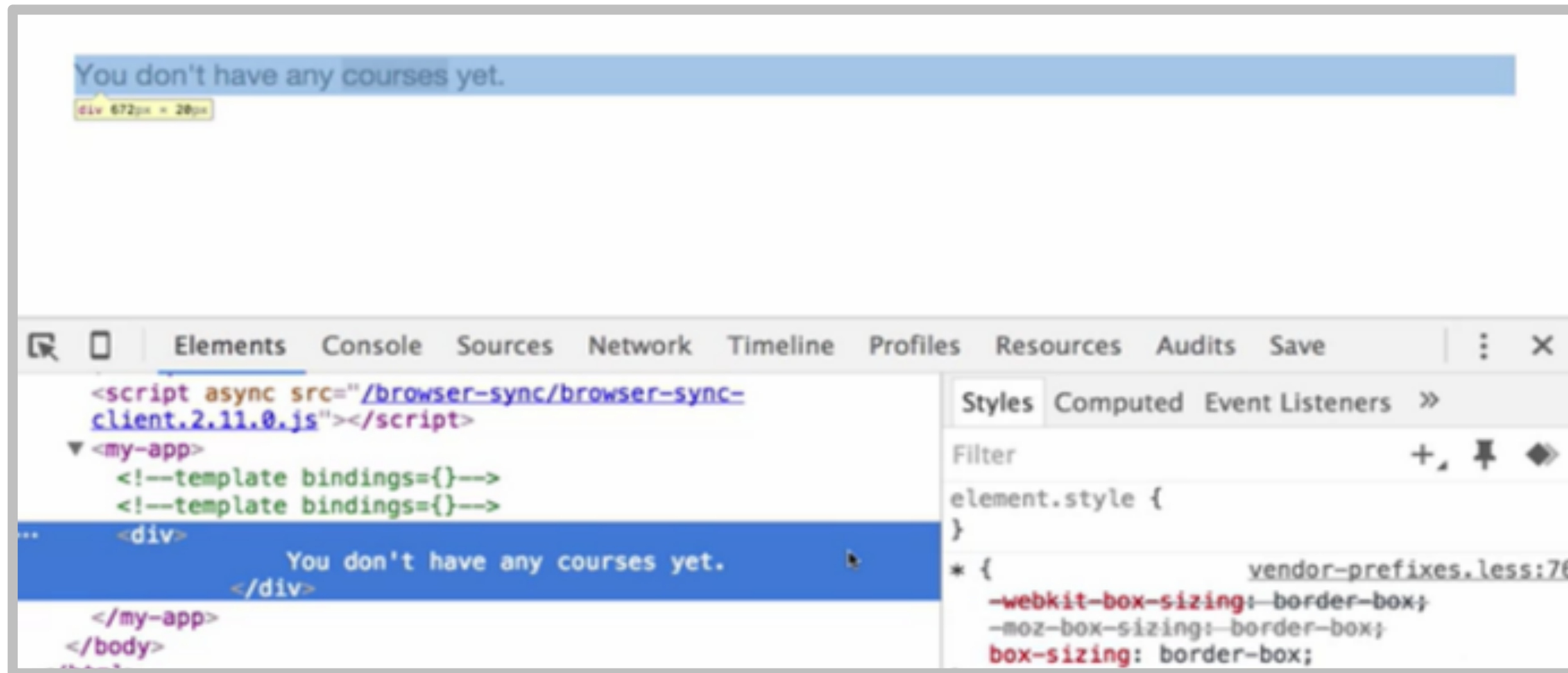
This example displays only the courses that have some content within.

```
2
3 @Component({
4   selector: 'my-app',
5   template: `
6     <div *ngIf="courses.length > 0">
7       List of courses
8     </div>
9     <div *ngIf="courses.length == 0">
10      You don't have any courses yet.
11    </div>
12  `
13 })
14 export class AppComponent {
15   courses = [];
16 }
```



ngif Directive

If you inspect the DOM, you will find that content is physically removed from DOM as well.

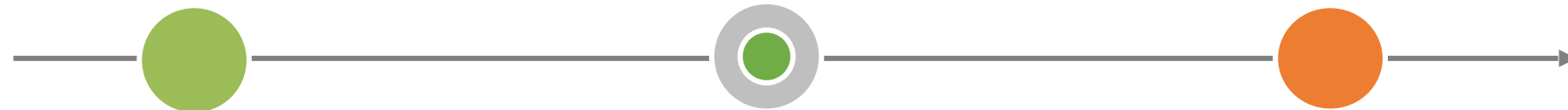


ngswitch Directive

- The Angular ngSwitch is actually a set of cooperating directives: ngSwitch, ngSwitchCase, and ngSwitchDefault.
- ngSwitch is actually comprised of two directives: an attribute directive and a structural directive.
- It's very similar to a switch statement in JavaScript and other programming languages, but in the template.

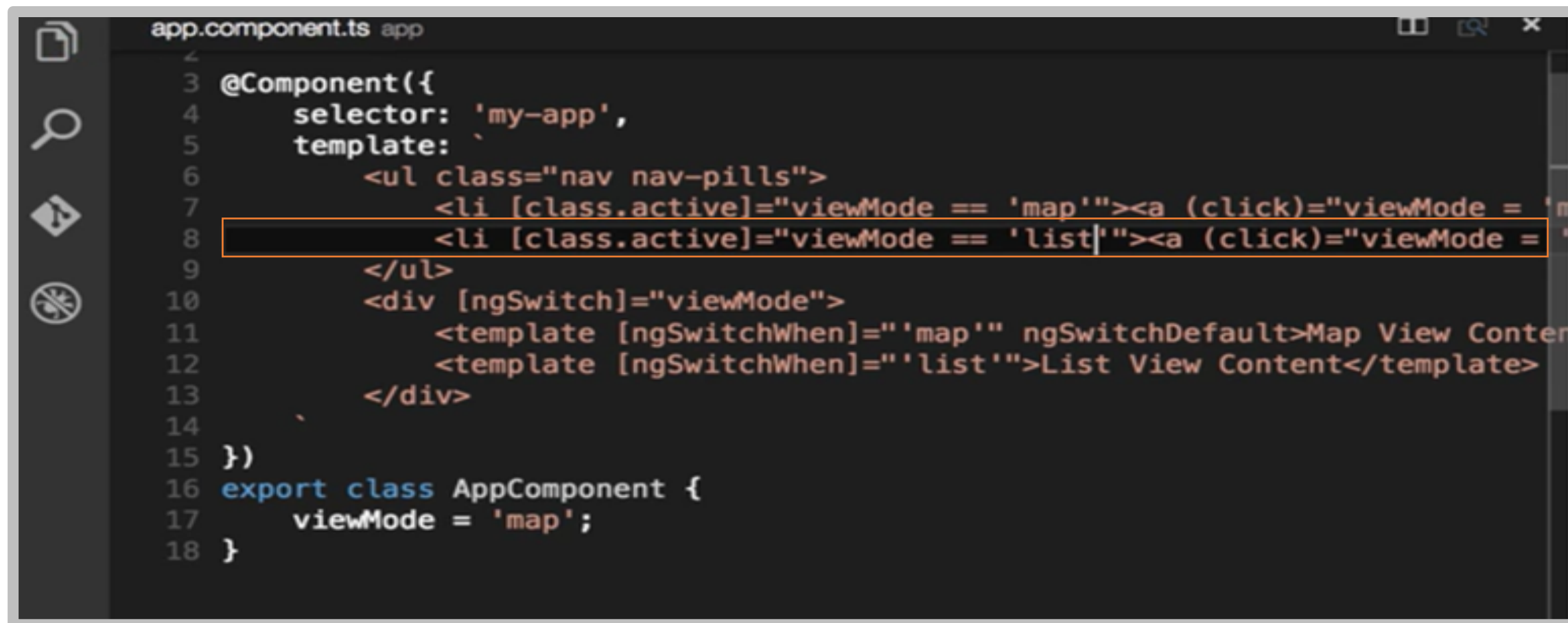
Example:

```
<div [ngSwitch]="tab">  
  
<app-tab-content *ngSwitchCase="1">content 1</app-tab-content>  
  
<app-tab-content *ngSwitchCase="2"> content 2</app-tab-content>  
  
</div>
```

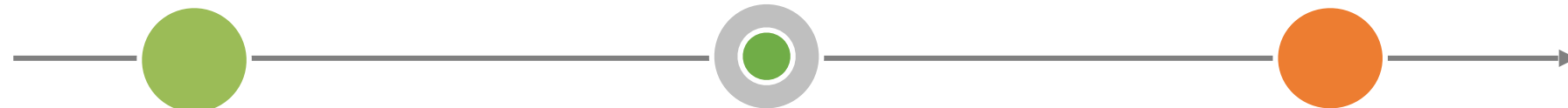


ngswitch Directive

- Here, you can see the ngSwitch attribute directive being attached to an element.
- The expression bound to the directive defines what it will be compared against in the switch structural directives.

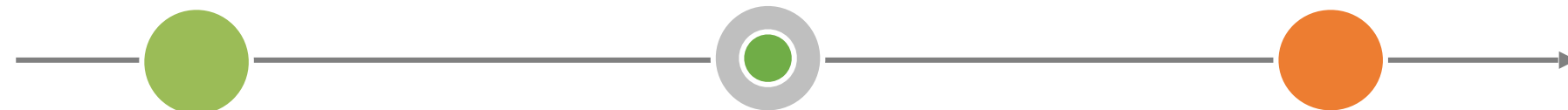


```
app.component.ts app
1
2
3 @Component({
4   selector: 'my-app',
5   template: `
6     <ul class="nav nav-pills">
7       <li [class.active]="viewMode == 'map'"><a (click)="viewMode = 'm
8       <li [class.active]="viewMode == 'list'"><a (click)="viewMode = '
9     </ul>
10    <div [ngSwitch]="viewMode">
11      <template [ngSwitchWhen]="'map'" ngSwitchDefault>Map View Conten
12      <template [ngSwitchWhen]="'list'">List View Content</template>
13    </div>
14  `
15 })
16 export class AppComponent {
17   viewMode = 'map';
18 }
```



ngswitch Directive

This example shows how you can switch between two different views using this directive.

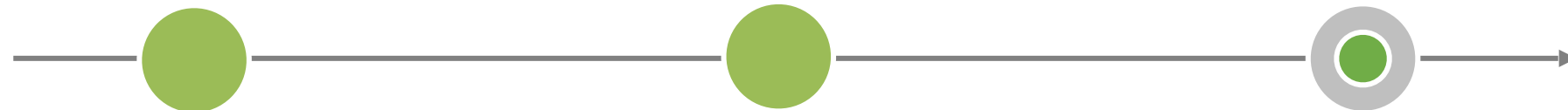


ngfor Directive

- The core directive ngFor allows you to build data presentation lists and tables in HTML templates.
- With ngFor, you can print this data to the screen as a data table by generating HTML.

Example:

```
<div *ngFor="let item of usernames">  
  <div *appMydirective="item">  
    {{item.name}}  
  </div>  
</div>
```

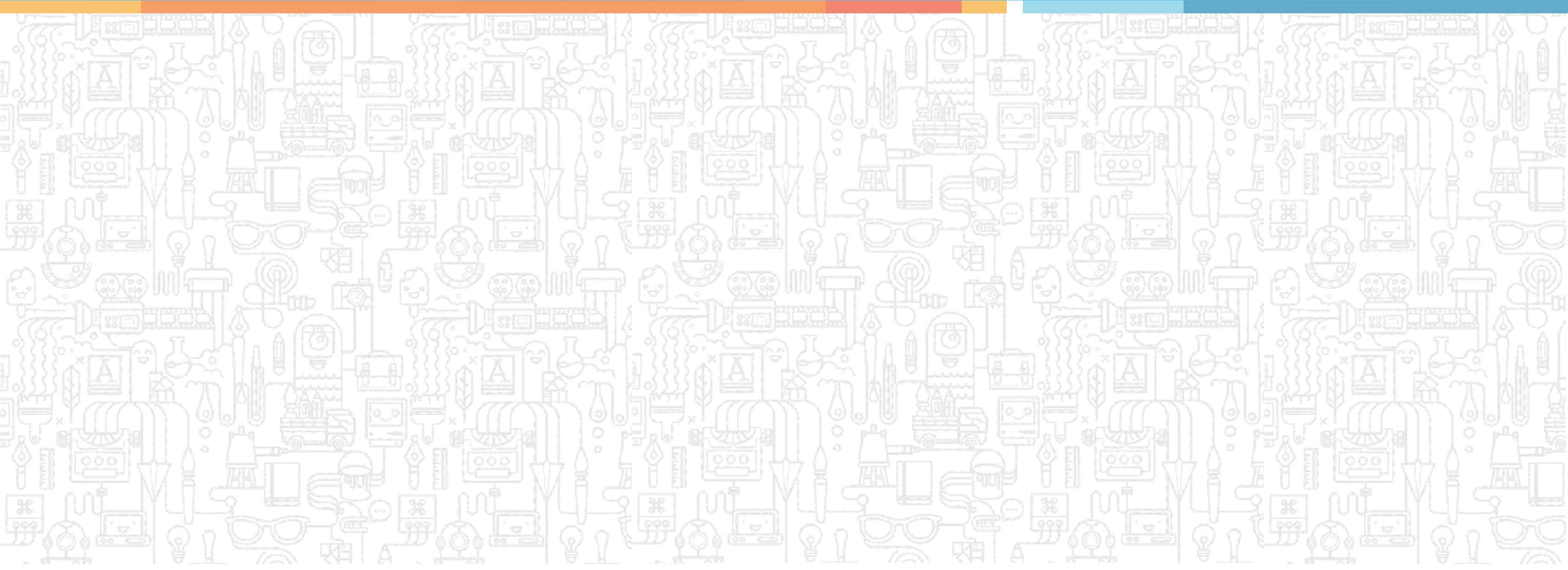


Demo—Attribute Directive



Directives

Topic 5—Custom Structural Directives



Custom Structural Directives

The most important difference in the way you create attribute directives is how they are provided to the DOM.

Attribute directives use ElementRef and Renderer to render and re-render, while structural directives use TemplateRef and ViewContainerRef to update the DOM content.

Custom Structural Directives

This demonstrates how to create a directive by overriding TemplateRef and ViewContainerRef.

```
• app.component.ts
1  import { Directive, Input, TemplateRef, ViewContainerRef } from '@angular/core';
2
3  @Directive({ selector: '[strIf]' })
4  export class strIfDirective {
5    constructor(
6      private templateRef: TemplateRef<any>,
7      private viewContainer: ViewContainerRef
8    ) { }
9
10   @Input() set strIf(shouldAdd: boolean) {
11     if (shouldAdd) {
12       // If condition is true add template to DOM
13       this.viewContainer.createEmbeddedView(this.templateRef);
14     } else {
15       // Else remove template from DOM
16       this.viewContainer.clear();
17     }
18   }
19
20 }
```

Key Takeaways



- ✔ Directive is a concept through which we separate the re-usable functions and features of Angular 2.
- ✔ Unlike components, directives don't have a HTML template.
- ✔ There are 3 types of directives: Component, Attribute, and Structural.
- ✔ A component directive requires a template along with its attached behavior.
- ✔ Attribute directives are used to change the appearance or behavior of a component or a native DOM element.
- ✔ Structural directives are responsible for HTML layout.



QUIZ

1

Why do we use directives?

- a. To Modify DOM
- b. To Build Views
- c. To break down large applications into smaller ones
- d. To make small applications large



QUIZ

1

Why do we use directives?

- a. To Modify DOM
- b. To Build Views
- c. To break down large applications into smaller ones
- d. To make small applications large



The correct answer is **a.**

Directives are used to modify DOM.

QUIZ

2

ngFor is a structural directive.

- a. True
- b. False



QUIZ

2

ngFor is a structural directive.

- a. True
- b. False



The correct answer is **a.**

ngFor is a Structural Directive.

QUIZ

3

Which directive modifies the appearance of an element?

- a. ngSwitch
- b. ngIf
- c. ngFor
- d. ngStyle



QUIZ

3

Which directive modifies the appearance of an element?

- a. ngSwitch
- b. ngIf
- c. ngFor
- d. ngStyle



The correct answer is **d.**

ngStyle directive modifies the appearance of an element.

QUIZ

4

Which Directive allows us to add or remove an HTML element of CSS class?

- a. ngStyle
- b. ngClass
- c. ngApply
- d. ngCSS



QUIZ

4

Which Directive allows us to add or remove an HTML element of CSS class?

- a. ngStyle
- b. ngClass
- c. ngApply
- d. ngCSS



The correct answer is **b.**

ngClass directive allows us to add or remove CSS classes to an HTML element.



Thank You