Coding Bootcamp

Angular

TECHNOLOGY

Diving Deep into Angular

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

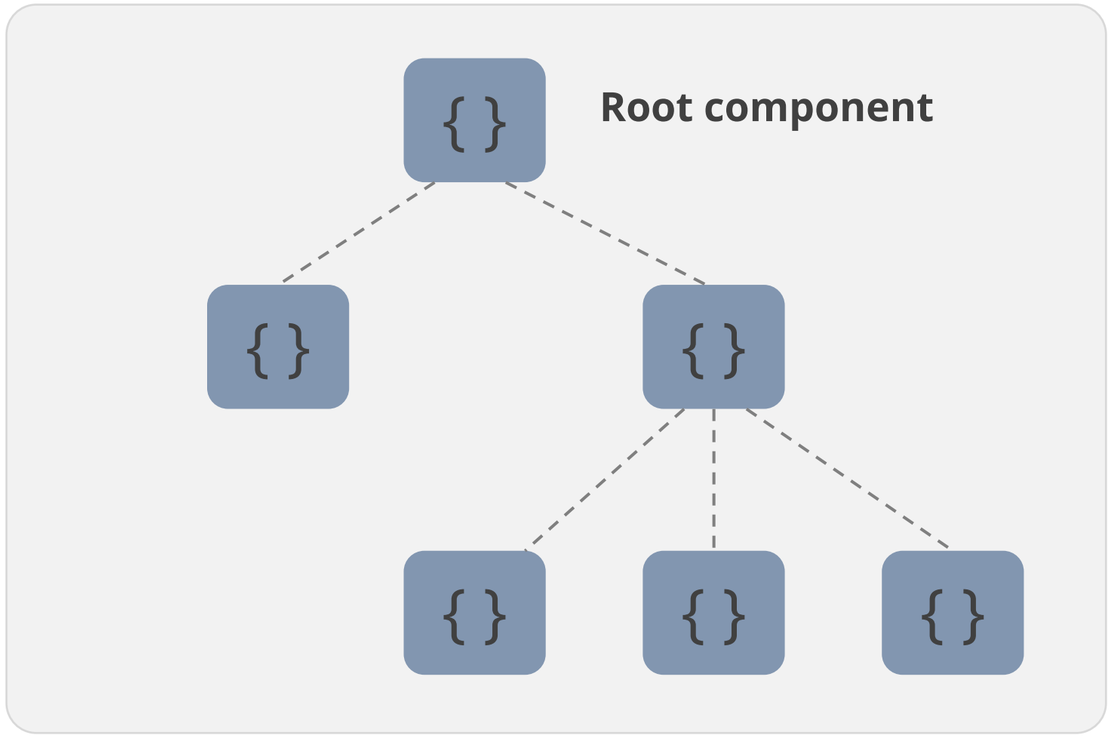
- Create multiple Angular components to provide a structure to the application
- Implement data binding for a seamless component-level interactions
- Develop Directives to manipulate DOM of the web page
- Execute various Pipes to transform data into templates
- Develop template-driven forms

TECHNOLOGY

Angular Components

Components

Components in an Angular application encapsulate the template, data, and behavior of view.

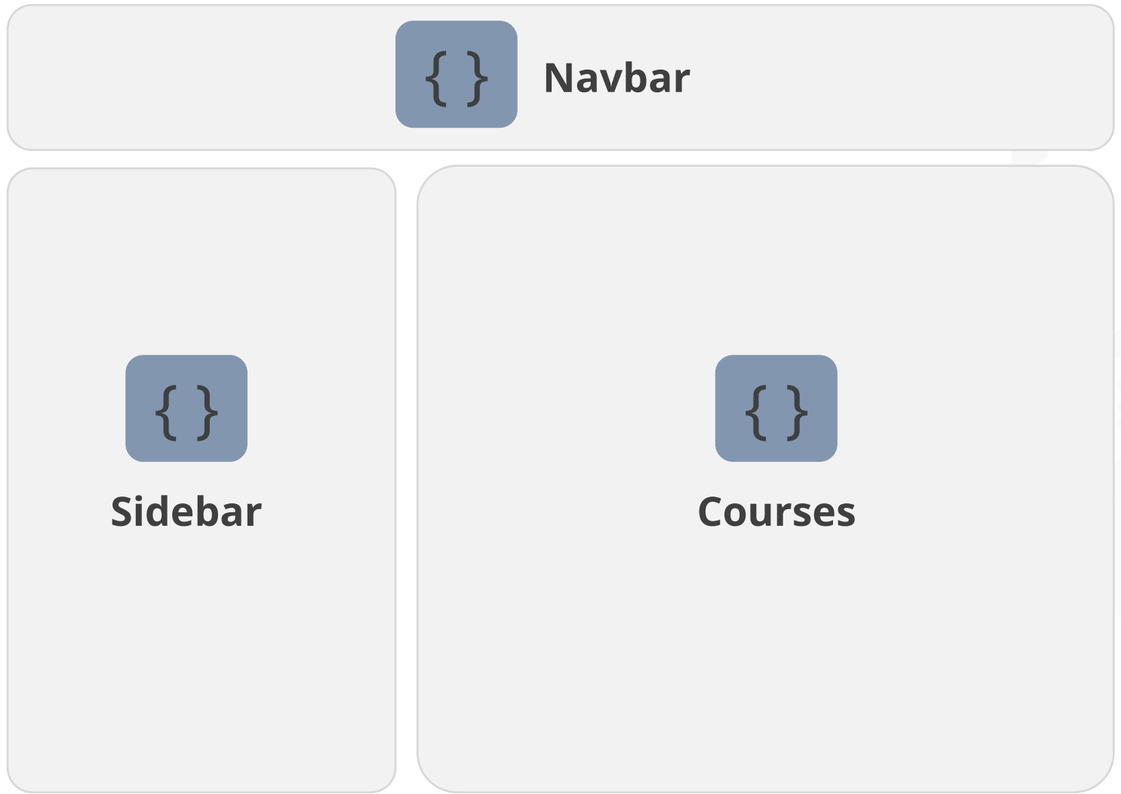


Every app has at least one component called the root component.

Components

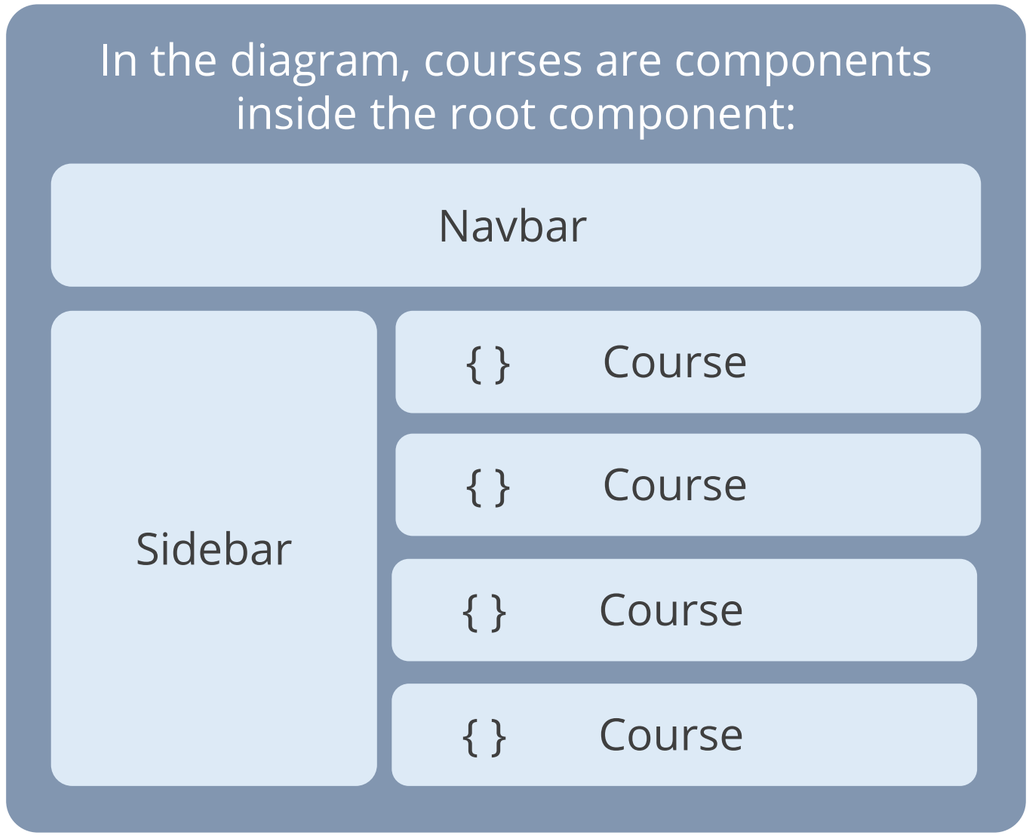
In the real world, an application encapsulates many components.

Here is an example:



Nested Components

Each component has a template for the view and data or logic behind the view. Components can also contain other components, which are called nested components.



Component-level Interactions

Component-level interactions can be done between two components at a similar branch of hierarchy or a parent-child level interaction.

The list of data shared among Angular components are:

- **Parent to child:** via Input()
- **Child to parent:** via Output() and EventEmitter
- **Child to parent:** via ViewChild
- **Unrelated components:** via a Service

@Input and @Output

@Input and **@Output** are mechanisms to receive and send data from one component to another.

Example:

```
@Component({
Selector: 'App-items'
…..
})

export class addTask{
          @Input() item_name
@Output() onNameChanger = new
EventEmitter()
 }
```

- The **@Input** decorator indicates that data will be received from a component. The received data will be stored in **item_name**.
- The **@Output** decorator indicates that data will be sent to another component via the **onNameChanger** property.

Component-level Interaction

Duration: 15 min.

Problem Statement:

You have been assigned a task to implement component-level interactions by passing he data from one component to other.

Outcome:

By following these steps, you will be able to establish communication between nested components and get comprehensive understanding on @input and @output decorators.

Note: Refer to the demo document for the detailed steps: 01_Component_level_interactions

ASSISTED PRACTICE

Assisted Practice: Guidelines

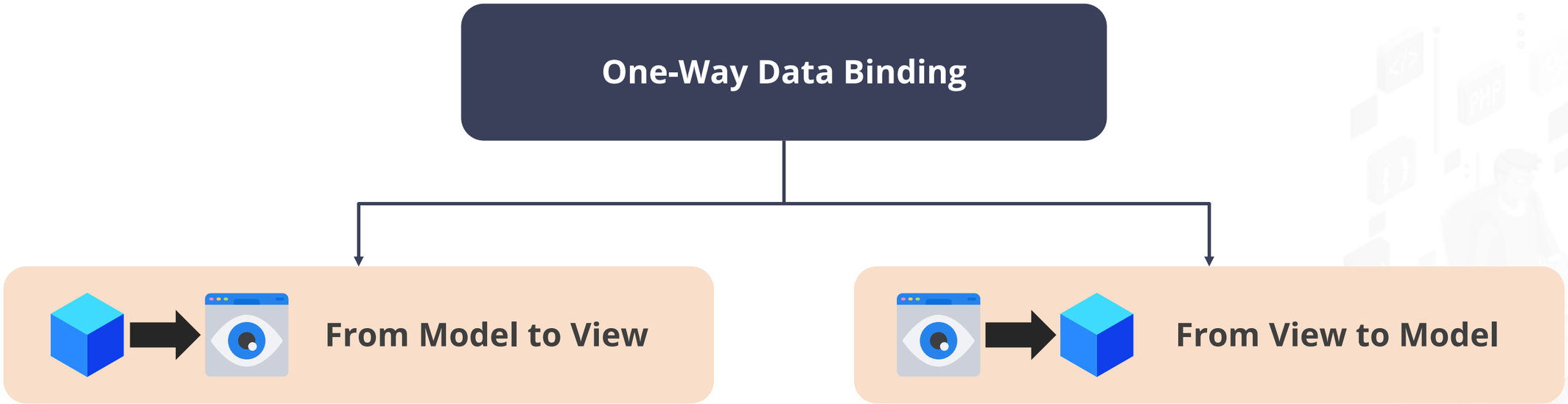
Steps to be followed:

1. Implementing the nested component that can communicate using the @input and @output decorators

One-Way Data Binding

One-Way Data Binding

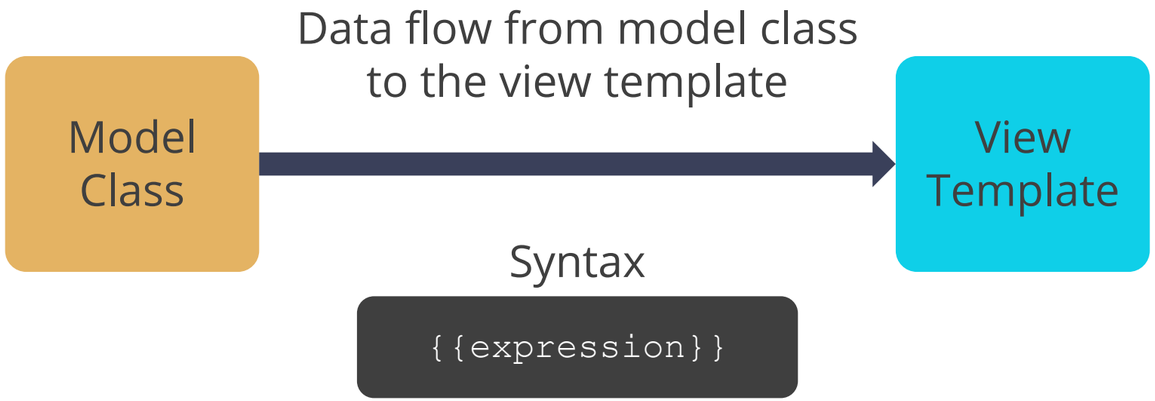
One-way data binding is a way to bind data, either from **model to view** or from **view to model**.



It displays information to the end-user, which automatically synchronizes with each underlying data change.

Interpolation

Angular interpolation is also called as string interpolation.



Example:

```
// app.component.ts
import { Component } from
'@angular/core';

@Component({
  selector: 'app-root',
  template: '<h1>{{ title }}</h1>',
})
export class AppComponent {
  title = 'Hello, Angular!';
}
```

Interpolation

Interpolation is used to bind a component property to a part of the template, usually to display data as text.

Interpolation

- Is used to bind data from model class to the view template
- Allows the user to include expressions as part of any string that can be used in HTML
- Converts the expressions into a string, replaces them in the original string, and updates the view
- Allows users to use interpolation wherever they use a string literal in the view

Interpolation Syntax

Interpolation syntax uses the double curly braces (**{{ }}**) in the template to denote the interpolation.

The content inside the double curly braces is called **template expression.**

Double curly braces indicate the interpolation.

Interpolation: Example

Component

```
Import {Component} from
'@angular/core';
@Component ({
   selector: 'app-root',
   templateURL:'./app.Component.html',
   styleUrls: ['./app.Component.css']
)}
Export class AppComponent{
     First_Name='Yuvraj';
     Last_Name='singh';
```

Template

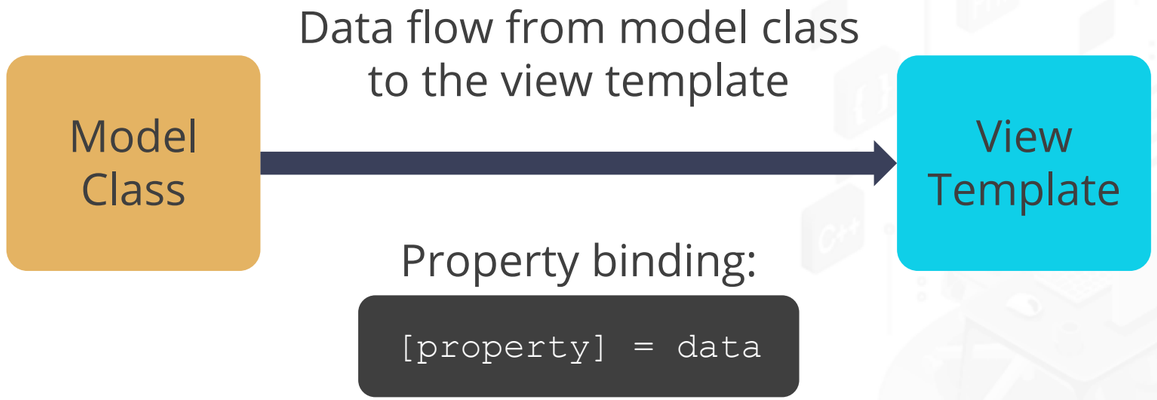
```
<p>Welcome
{{First_name,{{Last_name}}</p>
```

When the component values change, Angular updates the view. However, if the value in the view changes, the component is not updated.

Property Binding

Property binding is a one-way data-binding method.

- It is used to bind a property of a DOM element to a field, which is a defined property in the model TypeScript code
- It allows users to bind HTML element property to a property in the model
- It sets the properties in the model, and it will be used at the view
- Examples of properties are class and href.



Property Binding Syntax

Property binding is enclosed in square brackets (**[ ]**).

The content inside the square brackets is the name of the property.

```
<element [ binding-target ]=”binding-source”>
```

The content inside the double quotes is the property of the model.

The binding source must be a template expression.

Property Binding: Example

The given example has two property bindings:

Component

```
import {Component}from'@angular/core';
@Component({
    Selector:'app-root',
    templateURL:'./app.Component.html',
    styleUrls:['./app/Component/css']
})
export class AppComponent{
    title="Angular Property Binding Example"
    isDisabled=true;
}
```

Template

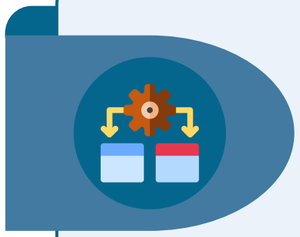
```
<h1 [innerText]="title"></h1>
<h2>Example</h2>
<button
[disabled]="isDisabled">I am
disabled</button>
```

- The component class of the **title** property is linked to the h1 tag of the **innerText** property.
- The button for the **Disabled** property is linked to the **isDisabled** property of the component.

When users change the **title** or **isDisabled** in the component, Angular automatically updates the view.

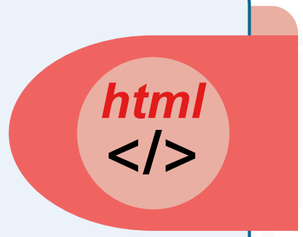
Property Binding

Property binding has a separate syntax for class, style, and attribute binding



Class and Style Binding

- Classes and styles are unique in that they contain a list of classes or styles.
- Bindings need to be more flexible in managing them.



Attribute Binding

- Attribute binding covers specific HTML attributes of the properties.

Class Binding

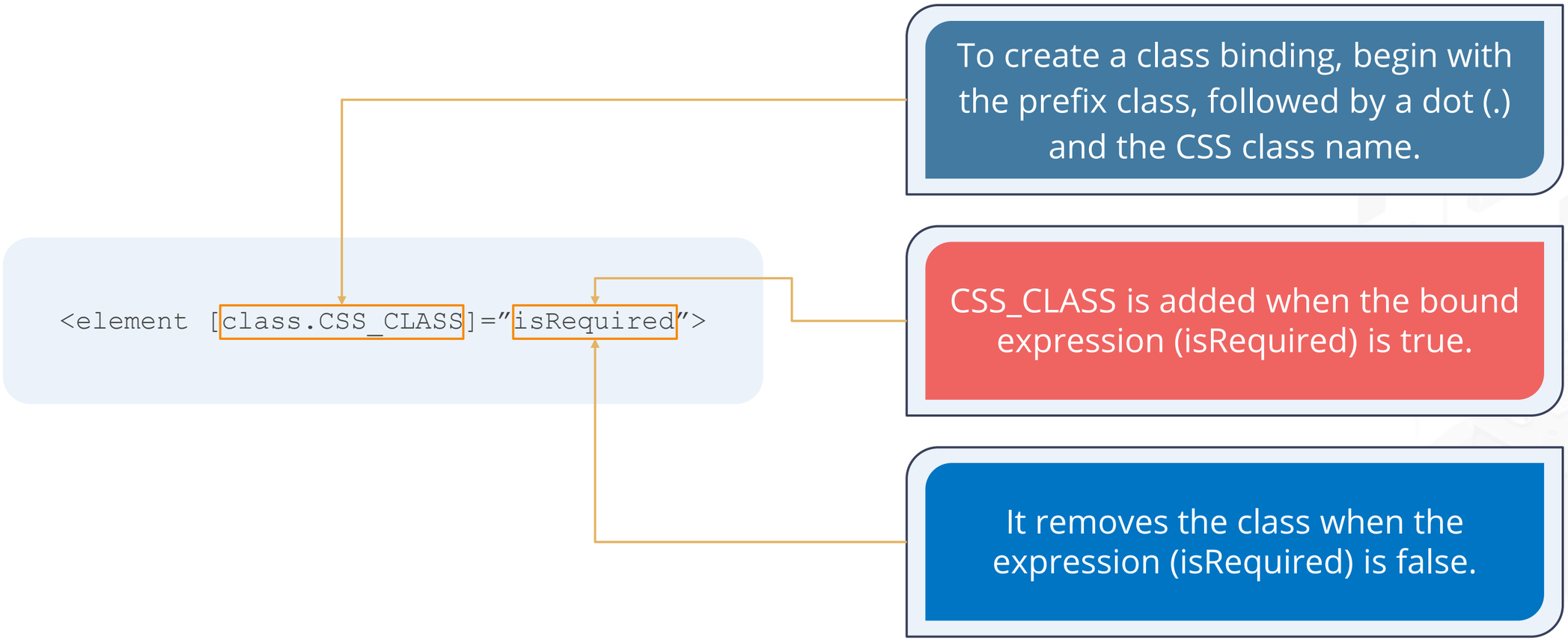
Class binding is a way to set a class property of a view element to store data and manage the state of an application.

Users can dynamically add and remove CSS classes from an element's class attribute.

Users bind a class of a DOM element to the corresponding property in the Typescript Code.

Class Binding Syntax

Class binding syntax is like property binding, containing a square bracket (**[ ]**).



Class Binding: Example

This is a class binding, that is **[class.myClass]="isTrue".**

This is a template

This is a component.

```
13  import { Component } from "@angular/core";
14
15  @Component({
16      selector: 'app-example',
17      template: `
18                  <div>
19                  <h1 [class.myClass]="isTrue">This class binding is for true value</h1>
20                  <h1 [class.myClass]="!isTrue">This class binding is for false value</h1>
21                  </div>
22                  `
23  })
24  export class ExampleComponent {
25      isTrue: boolean = true;
26  }
```

Basically, the expression will get evaluated by Angular.

Style Binding

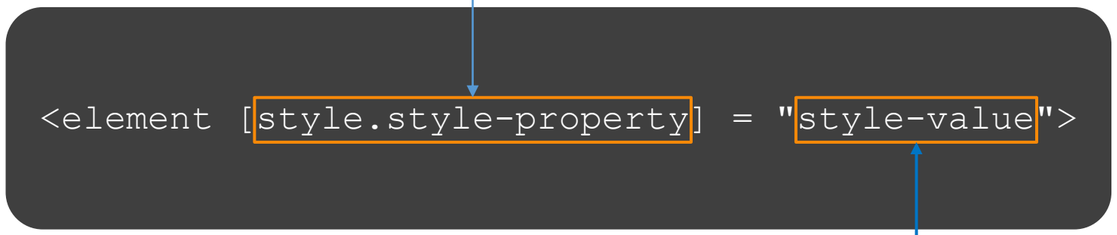
Style binding is used to set the style of a view element.

The inline styles of an HTML element can be set using the style binding in Angular.

A dynamically styled element can be created by conditioning styles to an element.

Style Binding Syntax

Style binding syntax is like class and property binding, containing a square bracket (**[ ]**).



To create a style binding, begin with the prefix style, followed by a dot (.) and the CSS style name.

The style-property is the binding target that will be set to the value of the bound expression, "style-value" in this case.

Style Binding: Example

This is the style.

This is the style name property.

This is the style value expression.

```
28  import { Component } from "@angular/core";
29  @Component({
30    selector: 'app-example',
31    template: `
32              <div>
33              <span [style.font-size.px]="isTrue? 20 : 12">This style binding is set for true value</span>
34              <span [style.font-size.%]="!isTrue? 120 : 30">This style binding is set for false value</span>
35              </div>
36              `
37  })
38  export class ExampleComponent {
39      isTrue: boolean = true;
40  }
41
```

Attribute Binding

Attribute binding is used to bind an attribute property of a view element.

Attribute binding is mainly useful where users don’t have any property view for an HTML element attribute.

Users can increase accessibility, dynamically style an application, and manage multiple CSS classes or styles simultaneously.

Attribute Binding Syntax

To write the attribute binding, the user needs to write the keyword **attr.thenameoftheproperty**

```
<p [attr.attribute-you-are-targeting]="expression"></p>
```

The attribute value is set by users using an expression that resolves to a string.

Attribute Binding: Example

To bind a value to the colspan property of the element

This is the attribute.

This is the name of the property.

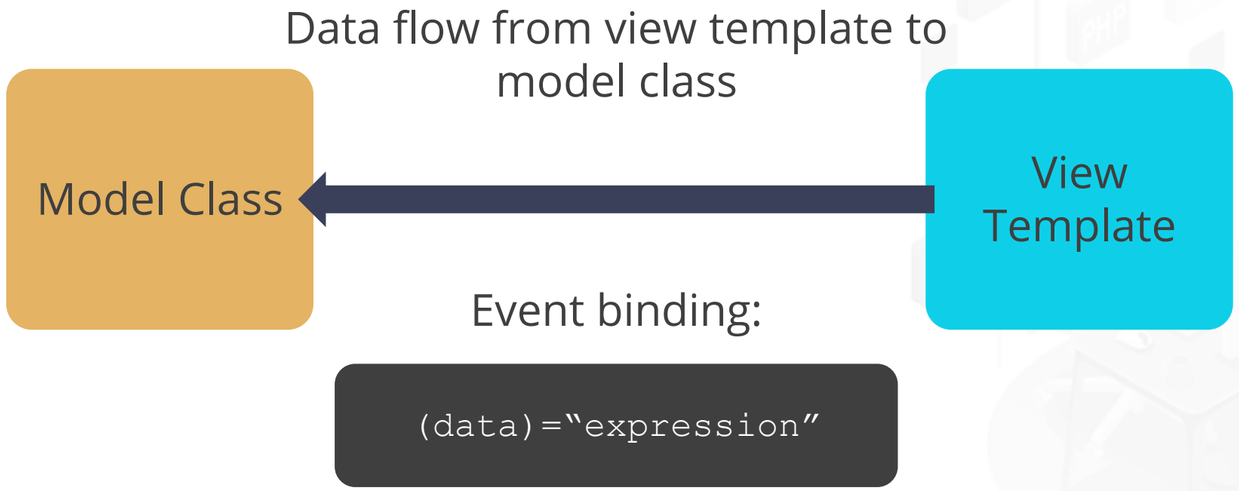
This is the expression.

```
69   <table>
70       <tr><td [attr.colspan]="3">three</td></tr>
71       <tr><td>1</td><td>2</td><td>3</td></tr>
72   </table>
```

Event Binding

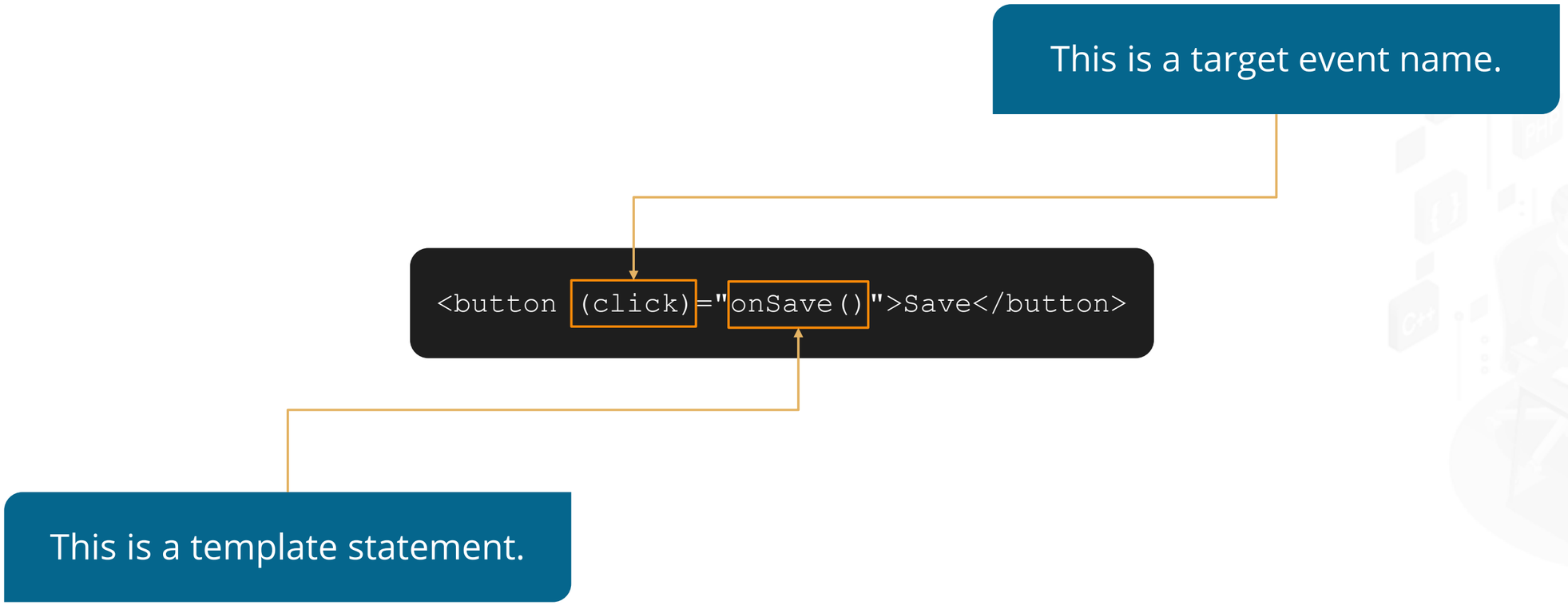
Event binding can pass data from view to component.

- Allows to bind webpage events to a model property or methods
- Binds the events like clicks, touch, keystroke, and hover
- Allows users to keep their components in sync with the view by tracking and responding to events in the view



Event Binding Syntax

Event binding listens for click events and calls the component's onSave() method when a button is clicked.



Data Binding

Duration: 15 min.

Problem Statement:

You have been assigned a task to implement data binding in various components.

Outcome:

By following the steps, you will be able to successfully implement data binding using string interpolation method.

Note: Refer to the demo document for the detailed steps: 02_Data_binding

ASSISTED PRACTICE

Assisted Practice: Guidelines

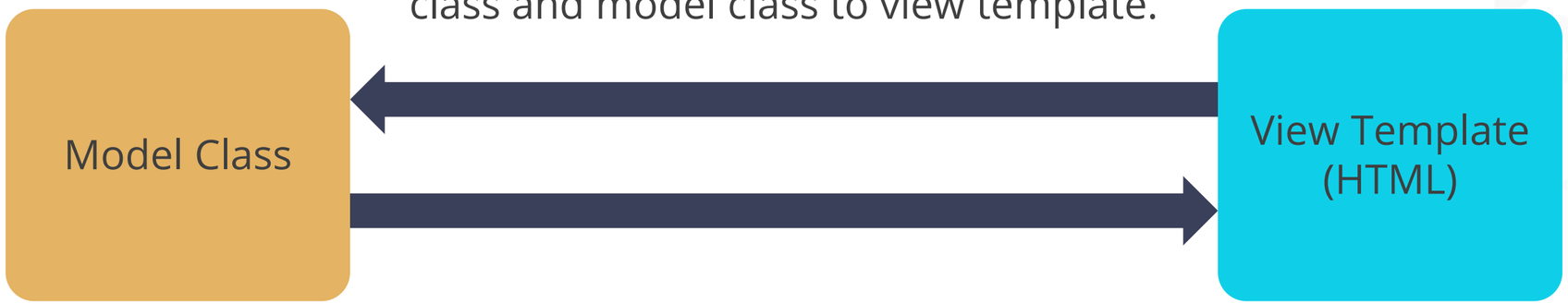
Steps to be followed:

1. Create an Angular application with data binding

Two-Way Data Binding

Two-Way Data Binding

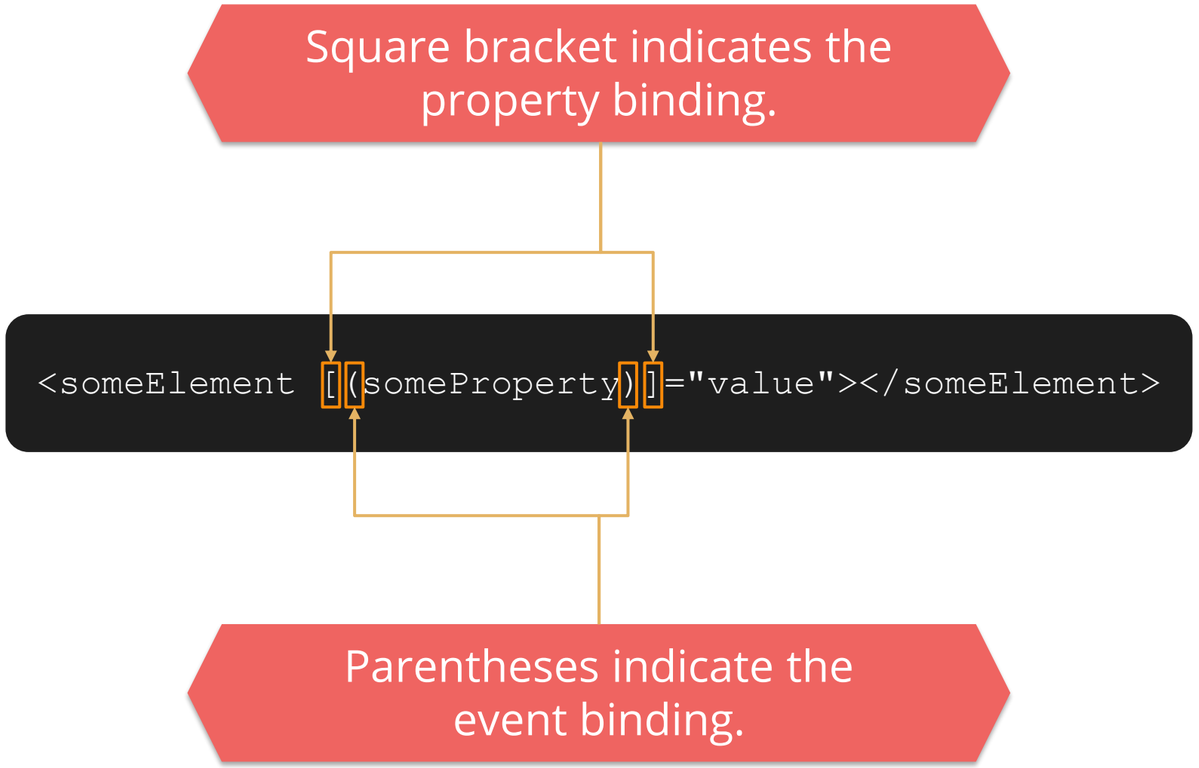
Binds data from model class to view template and view template to model class



This is a combination of event binding and property binding.

Two-Way Data Binding Syntax

Two-way binding uses the unique syntax known as a banana in a box **[()].**



Handling Events in Angular

Duration: 30 min.

Problem Statement:

You have been assigned a task to implement event handling in Angular

Outcome:

By following the steps, you have successfully implemented event binding and handling the click event in the view.

Note: Refer to the demo document for the detailed steps:
03_Handling_Events_in_Angular

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Implementing event handling by binding the event in Angular

Angular Forms

Forms

Forms are used to handle user input data.

Applications use forms to enable users to:

- Log in
- Update a profile
- Enter information
- Perform data-entry tasks

Forms

Angular supports two types of forms:

Template driven forms

- Created using Directives in the template
- To create a simple form application

Reactive forms

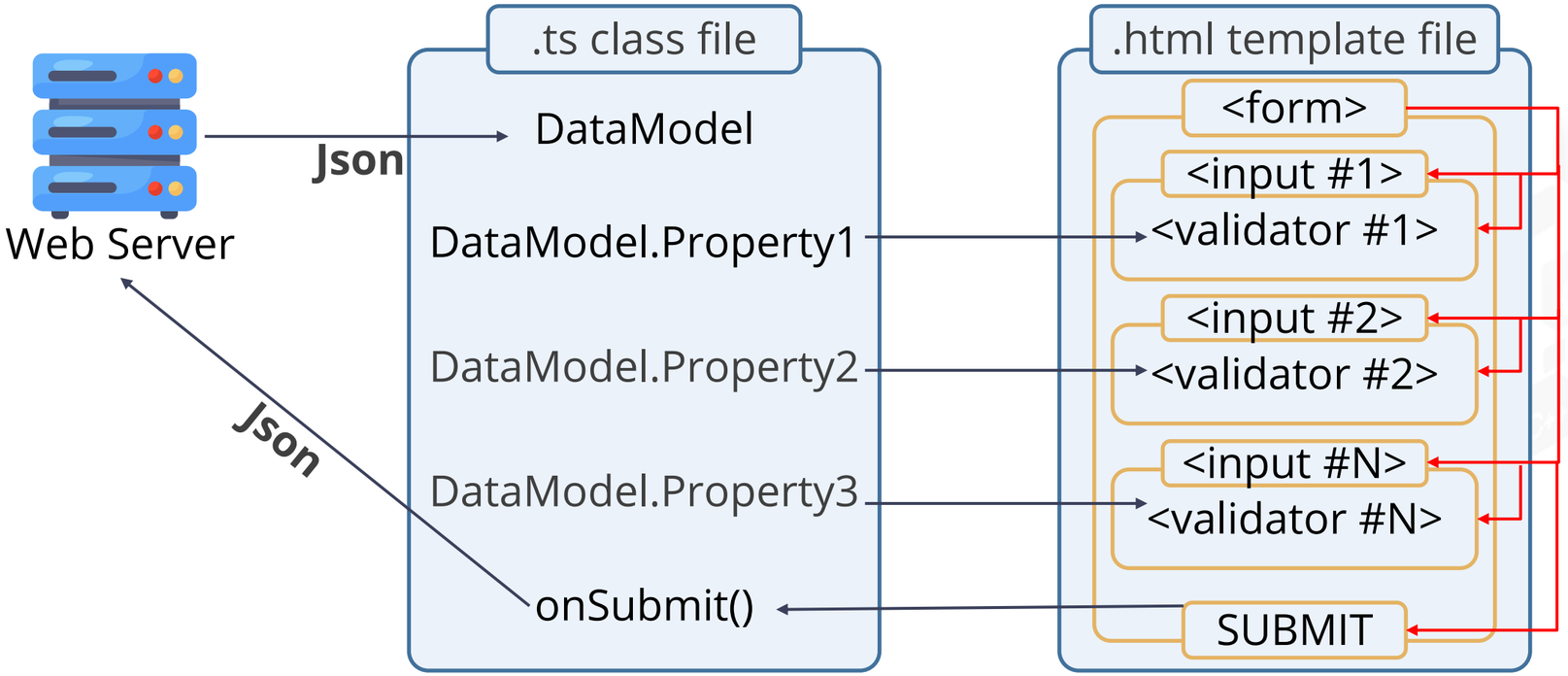
- Created inside component class
- Based on the structured data model

Template Driven Forms vs. Reactive Forms

Template Driven Forms	Reactive Forms
Easy to use	More flexible
Suitable for simple scenarios	Handles any complex scenarios
Similar to Angular	Reactive transformations can be made possible, such as •Handling an event based on a debounce time •Handling events when the components are distinct until changed •Adding elements dynamically
Two-way data binding (using [(NgModel)] syntax)	No data binding is required (immutable data model preferred by most developers)
Unit testing is tough	Unit testing is easy

Template Driven Forms

Diagram of Template Driven Forms



Template Driven Forms: Example

HTML

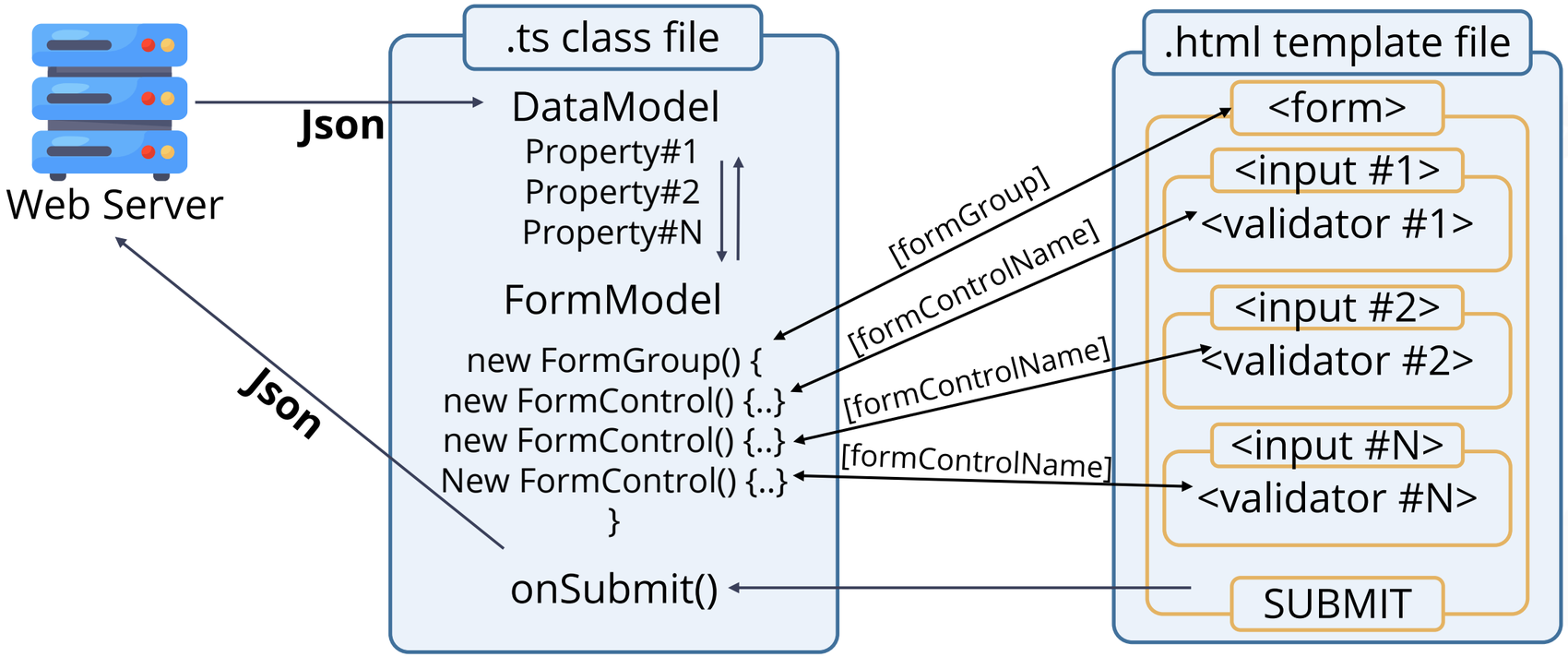
```
<form #userName="ngForm"
(ngSubmit)="onClickSubmit(userName.value)">
   <input type="text" name="username" placeholder="username"
ngModel>
   <br/>
   <br/>
   <input type="submit" value="submit">
</form>
```

Component

```
import { Component, OnInit } from '@angular/core';
@Component({
   selector: 'app-test',
   templateUrl: './test.component.html',
   styleUrls: ['./test.component.css']
})
export class TestComponent implements OnInit {
  ngOnInit() {
   }
   onClickSubmit(result) {
      console.log("You have entered : " + result.username);
   }}
```

Reactive Forms

Diagram of Reactive Forms



Reactive Forms: Example

AppComponent

```
import { Component, OnInit } fr '@angular/core';
import { FormGroup, FormControl } om from '@angular/forms';

@Component({
   selector: 'app-test',
   templateUrl: './test.component.html',
   styleUrls: ['./test.component.css']
})
export class TestComponent implements OnInit {
   userName;
   formdata;
   ngOnInit() {
      this.formdata = new FormGroup({
         userName: new FormControl("Tutorialspoint")
      });
   }
   onClickSubmit(data) {this.userName = data.userName;}
}
```

Reactive Forms: Example

HTML

```
<div>
   <form [formGroup]="formdata"
(ngSubmit)="onClickSubmit(formdata.value)" >
      <input type= "text"  name="userName"
placeholder="userName"
         formControlName = "userName">
      <br/>
      <br/>
      <input type="submit"  value="Click here">
   </form>
</div>
<p> Textbox result is: {{userName}} </p>
```

Implementing Reactive Forms in Angular

Duration: 30 min.

Problem Statement:

You have been assigned a task to implement Reactive forms in Angular.

Outcome:

By following the steps, you will successfully create reactive forms in Angular and handle data from the forms.

Note: Refer to the demo document for the detailed steps:
04_Implementing_Reactive_Form_in_Angular

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new Angular Application
2. Create Reactive form using Angular

Implementing Template-driven Forms

Duration: 30 min.

Problem Statement:

You have been assigned a task to implement Template-driven forms in Angular.

Outcome:

By following the steps, you will be able to create template-driven forms and work with data associated with it.

Note: Refer to the demo document for the detailed steps:
05_Implementing_Template_Driven_Form

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a Template-driven form in Angular
2. Run the code on the browser

Directives

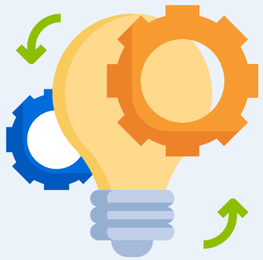
Directives

Directives are classes that can change or add new behavior to template components.

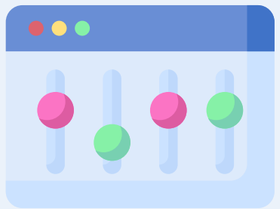
Used to manipulate the DOM, whether by adding, removing, or changing the DOM elements

Extend HTML's behavior by creating custom HTML elements, attributes, and classes with application-specific functionality

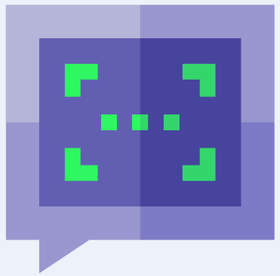
Directives



Directives are meant to be a function that executes the DOM by the Angular compiler to extend the power of the HTML with new syntax.



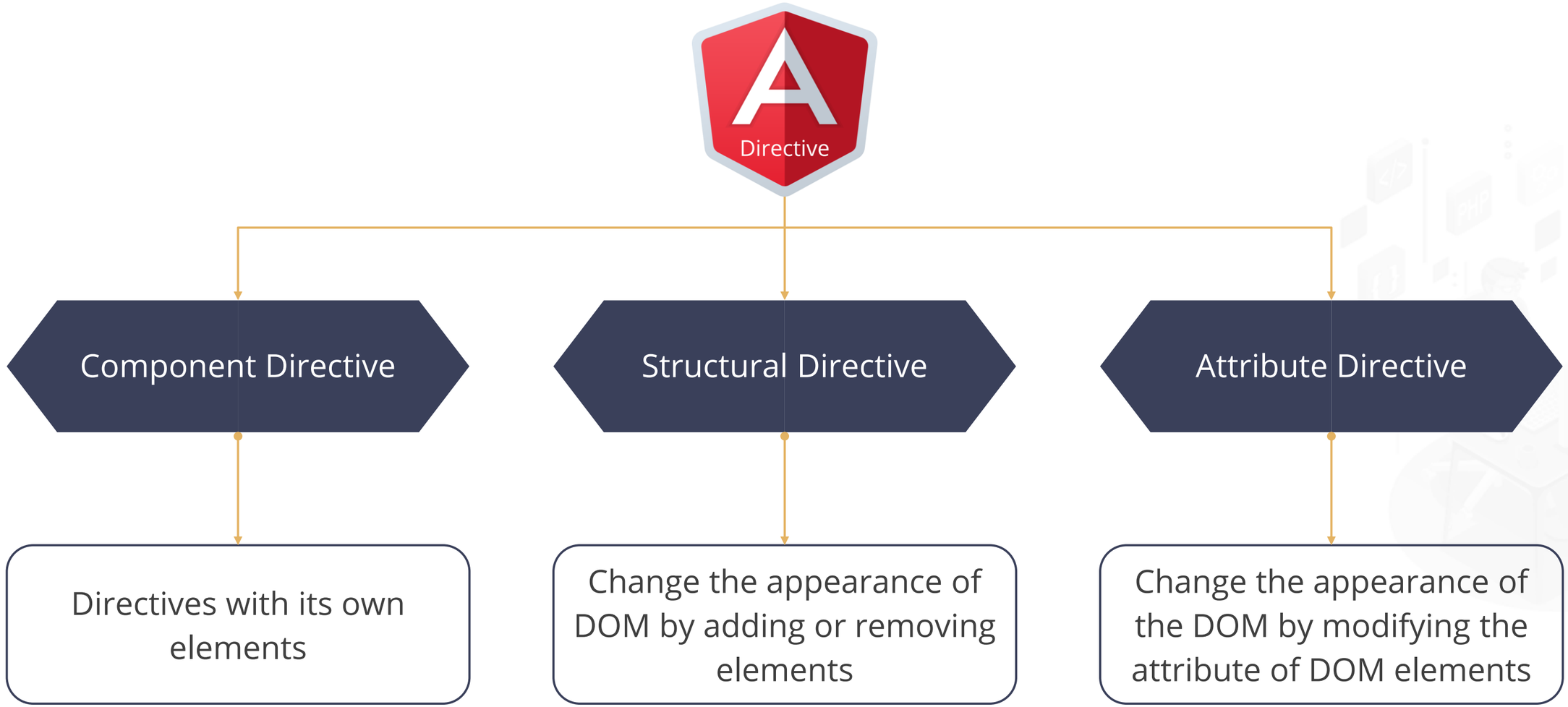
Directives have names and can be predefined or custom-defined, allowing them to be called anything.



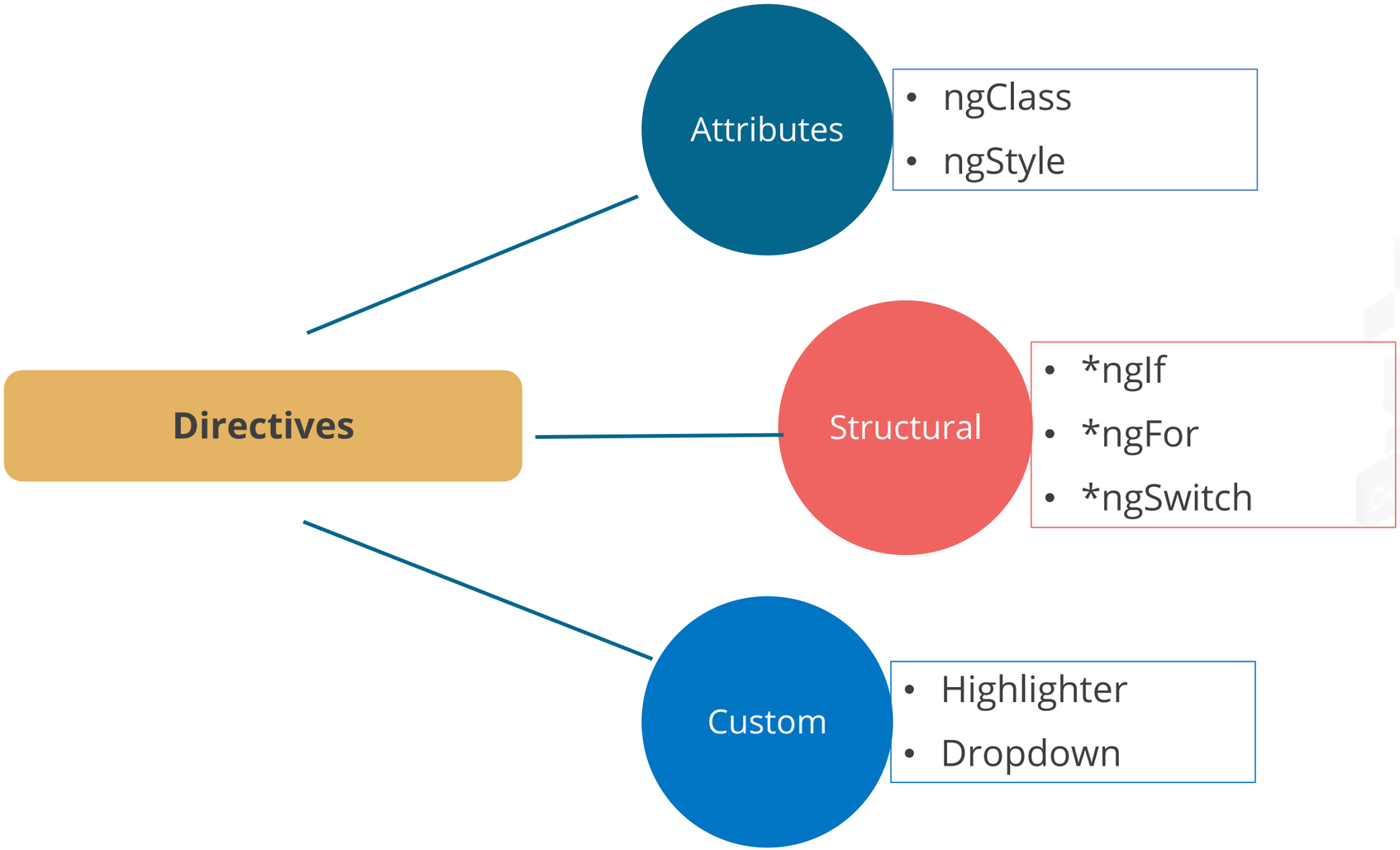
Directives are purposeful depending on the predefined directive, like attribute, comment, element, or class.

Types of Directives in Angular

There are three kinds of Directives in Angular:

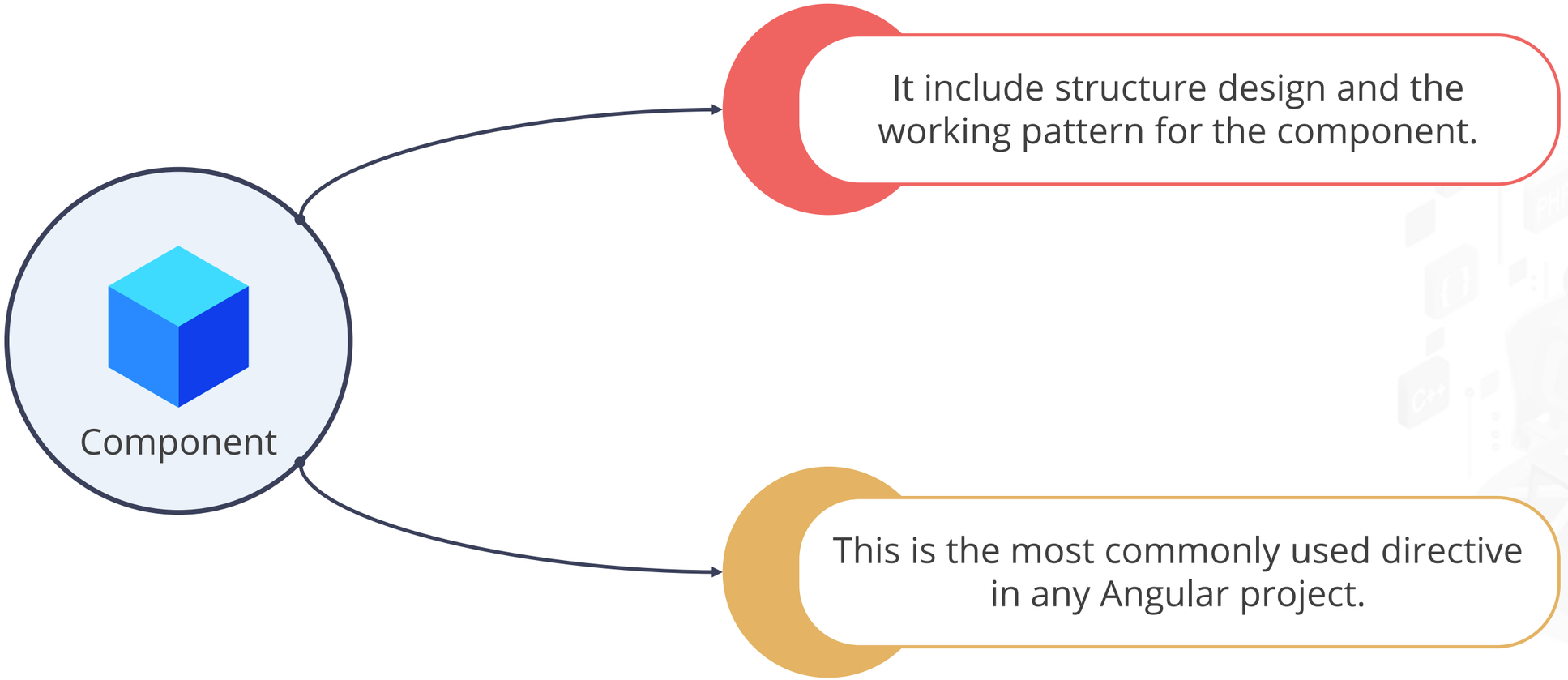


Structure of Directives



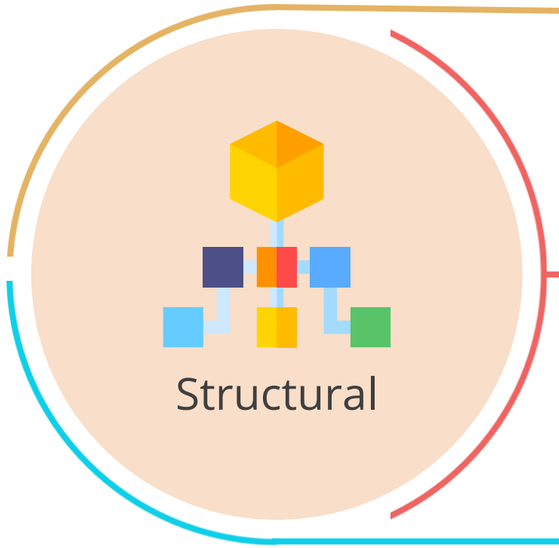
Component Directives

The HTML templates are specified using component Directives.



Structural Directives

The structural directive is used to change the DOM layout by adding and removing DOM elements.



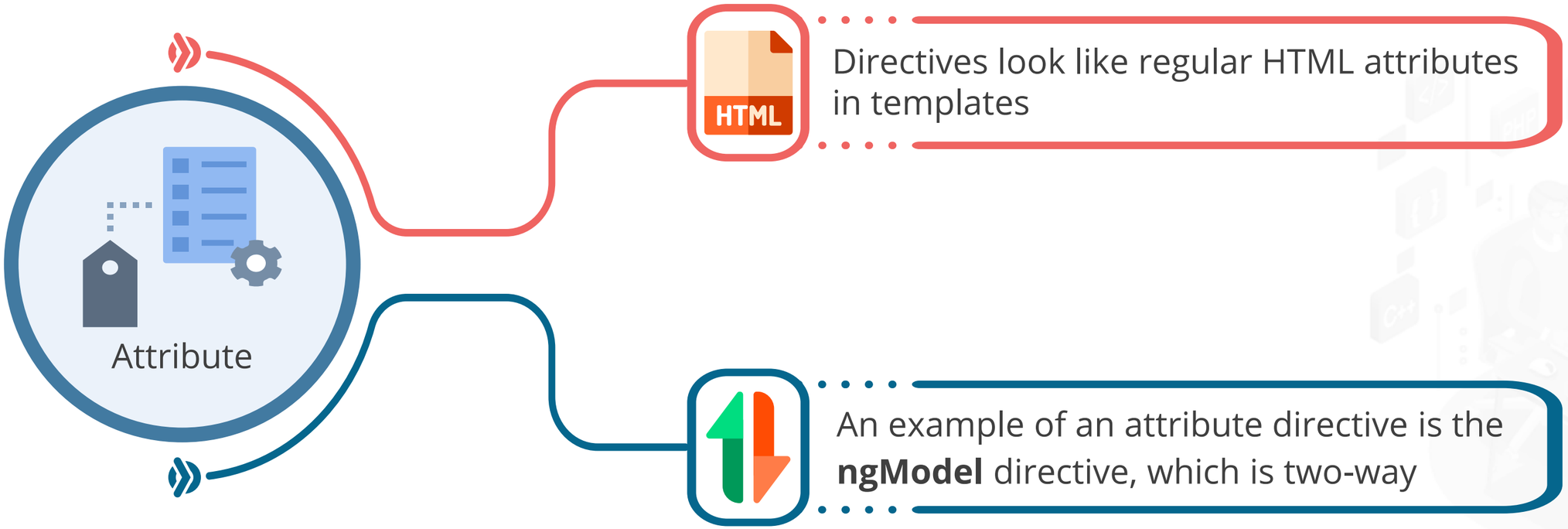
The angular framework provides three built-in structural Directives, like **ngFor**, **ngIf**, and **ngSwitch.**

Directives prefixed with ***ngFor**, ***ngIf**, or ***ngSwitch** Angular internally transform this * into **ng-template.**

Component Template is used by the respective Directive to add or remove a template inside a DOM tree.

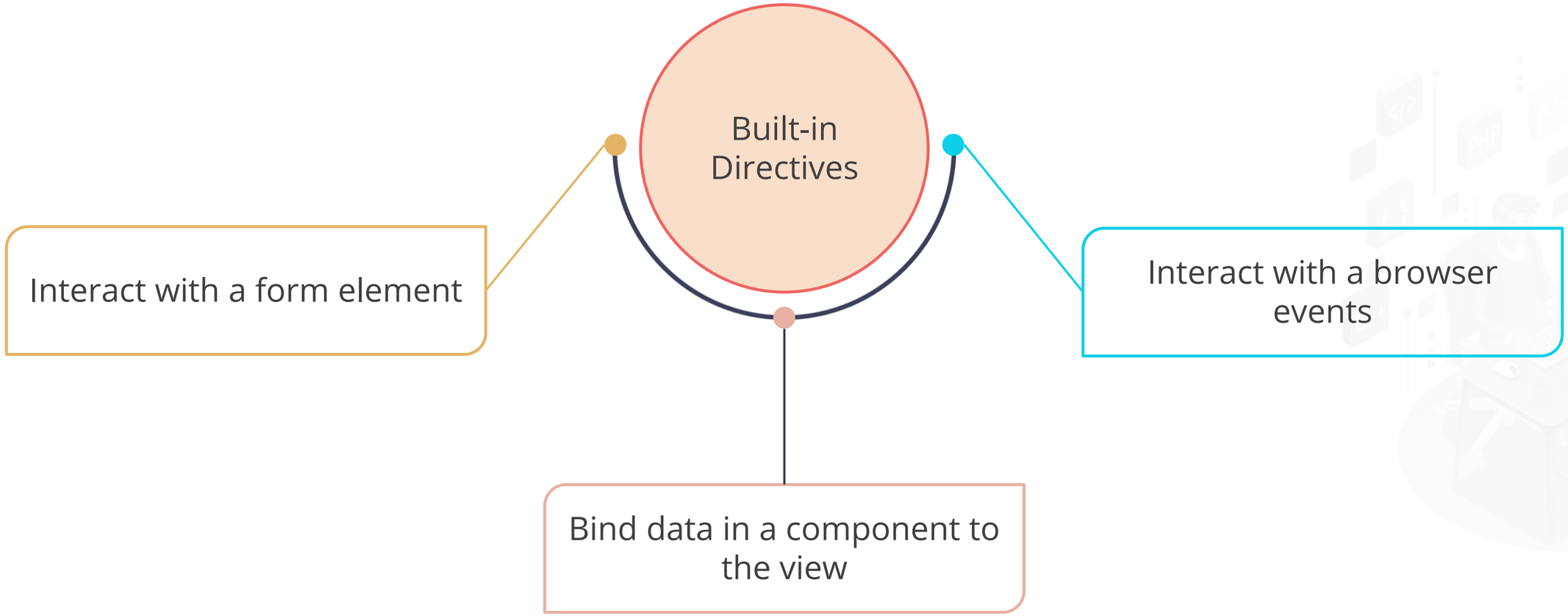
Attribute Directives

An attribute directive is used to change the appearance or behavior of a component, element, or another directive.



Built-in Directives

Angular includes various built-in Directives. A user can also create a directive in Angular.



Working with Built-in Directives

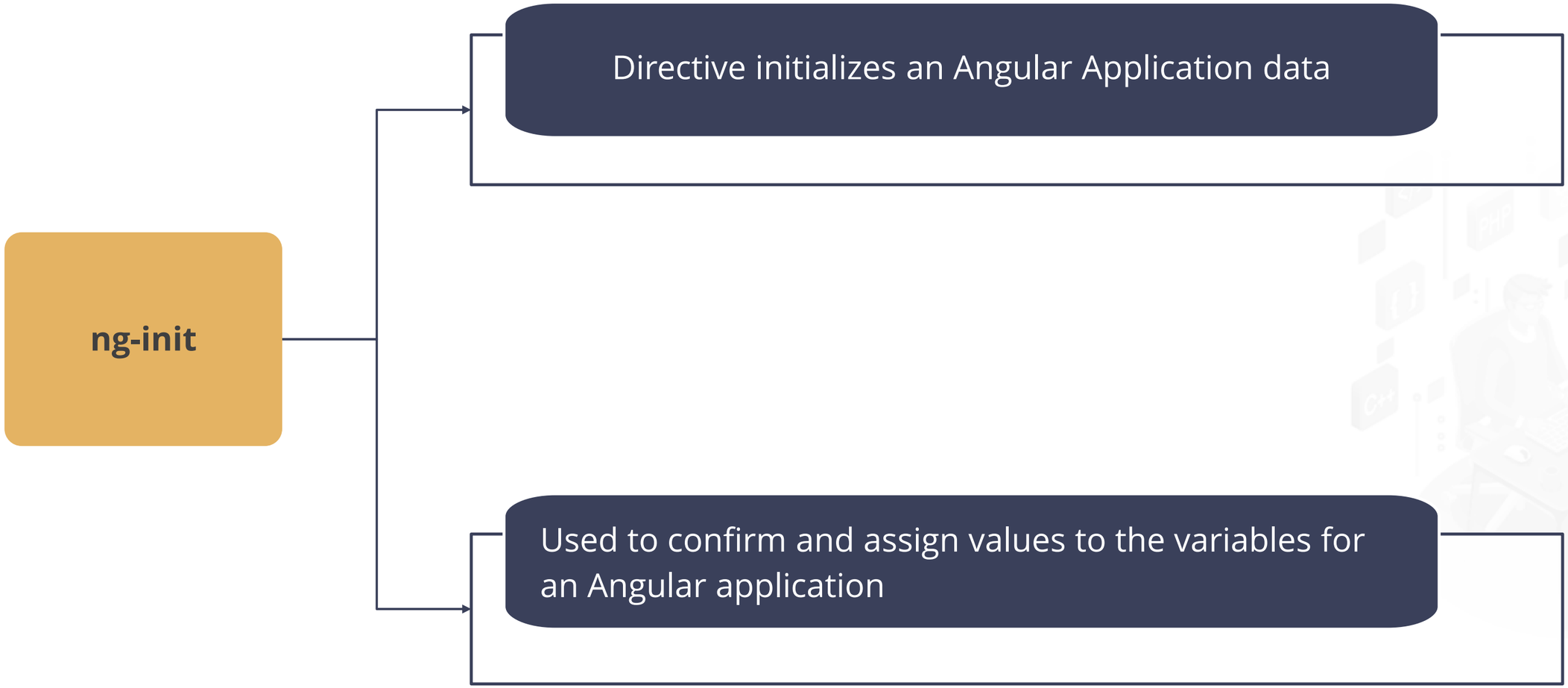
Built-in Directives are unique attributes starting with the **ng-** prefix, where ng stands for Angular.

ng-app	It is combined to set the Angular section.
ng-init	It sets the absence variable value.

Working with Built-in Directives

ng-class	It is used for the CSS class dynamically.
ng-model	It is used to bind the value of HTML controls (input, select, and textarea) to application data.
ng-repeat	It is used to loop through any item in the collection to make a new template.

Built-in Directives: ng-init



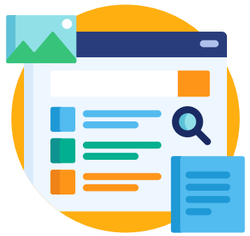
Example of ng-init

User initializes the variable's first and last names and allows values to it.

```
<div ng-app="" ng-init="firstName:'Ravi';lastName:'Jadhav'}">
…….
</div>
```

Built-in Directives: ng-model

The ng-model directive is used for two-way data binding in Angular.



The ng-model is used to take the value of input controls like textbox and label, then use them on web pages.

Example of ng-model

A user defines a model named **myname**.

```
<div ng-app=“”>
…..
  <input type=“text” ng-model=“myname”>
  <p>My name is(enter it):{{myname}}</p>
</div>
```

Working with Directives

Duration: 10 min.

Problem Statement:

You have been assigned a task to implement directives in Angular.

Outcome:

By following the steps, you will be able to create and work with ngif directive and transfer data from .ts to views of the components.

Note: Refer to the demo document for the detailed steps:
06_Working_with_Directives

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a directive in Angular
2. Run the code on the browser

Working with ngapp Directive

Duration: 20 min.

Problem Statement:

You have been assigned a task to implement ngapp directive in Angular.

Outcome:

By following the steps, you will be able to create and work efficiently with ngapp directive.

Note: Refer to the demo document for the detailed steps:
07_Working_with_ngapp_Directive

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a ngapp directive in Angular
2. Run the code on the browser

Working with ngrepeat Directive

Duration: 30 min.

Problem Statement:

You have been assigned a task to implement ngrepeat directive in Angular.

Outcome:

By following the steps, you will successfully create and implement ngrepeat directive in Angular and work on the data associated with it.

Note: Refer to the demo document for the detailed steps:
08_Work_with_ngrepeat_Directive

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a ngrepeat directive in Angular
2. Run the code on the browser

Working with ng-class Directive

Duration: 30 min.

Problem Statement:

You have been assigned a task to implement ng-class directive in Angular.

Outcome:

By following the steps, you will successfully create, implement and work with ng-class directive in Angular and data associated with it.

Note: Refer to the demo document for the detailed steps:
09_Work_with_ng-class_Directive

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a ng-class directive in Angular
2. Run the code on the browser

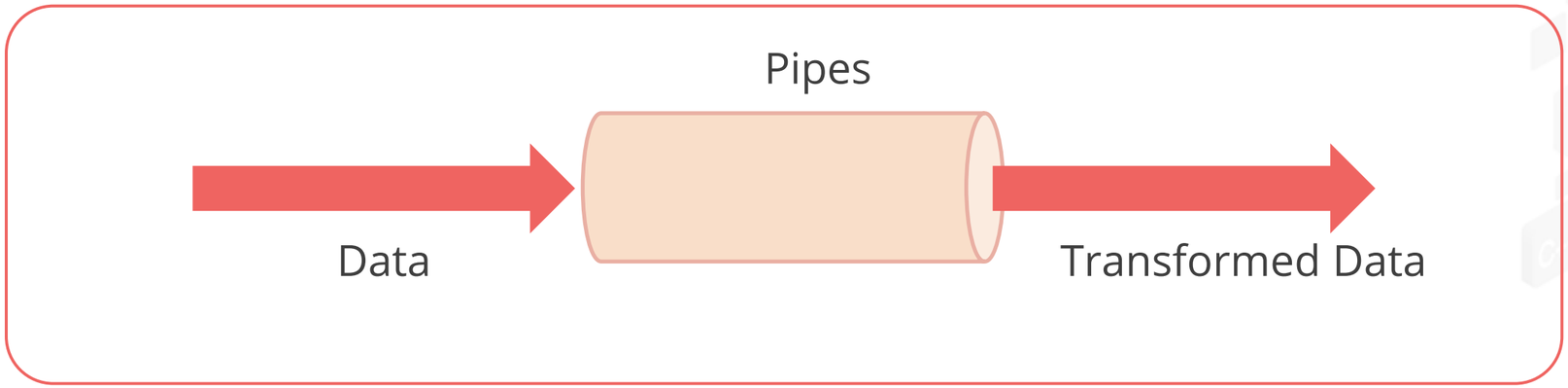
TECHNOLOGY

Introduction to Angular Pipes

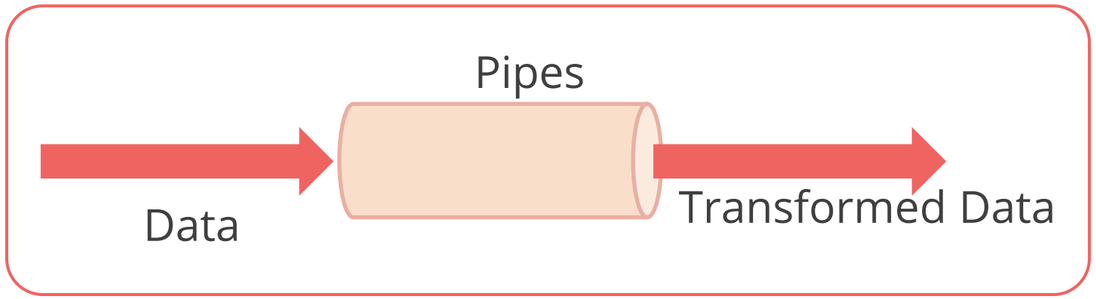
simplilearn

Angular Pipes

Pipes are simple functions that accept a string and transform data into desired information.

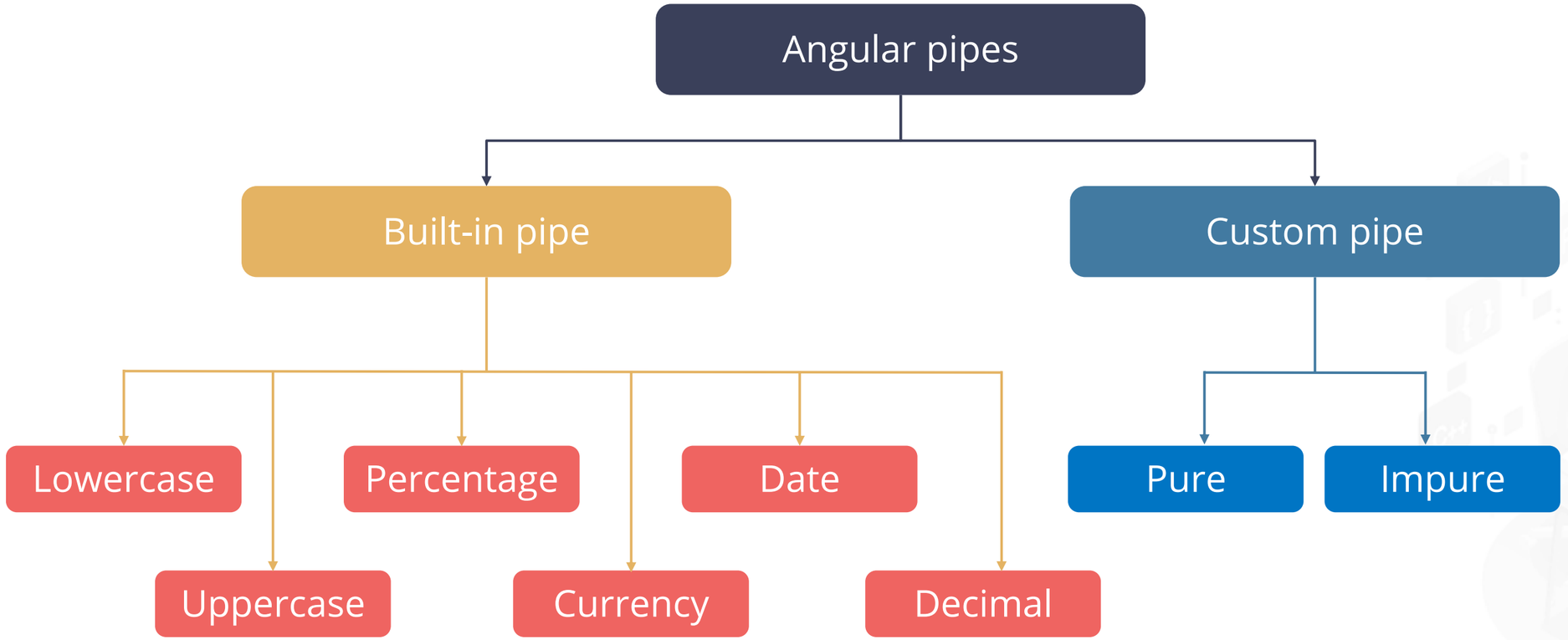


Angular Pipes: Features



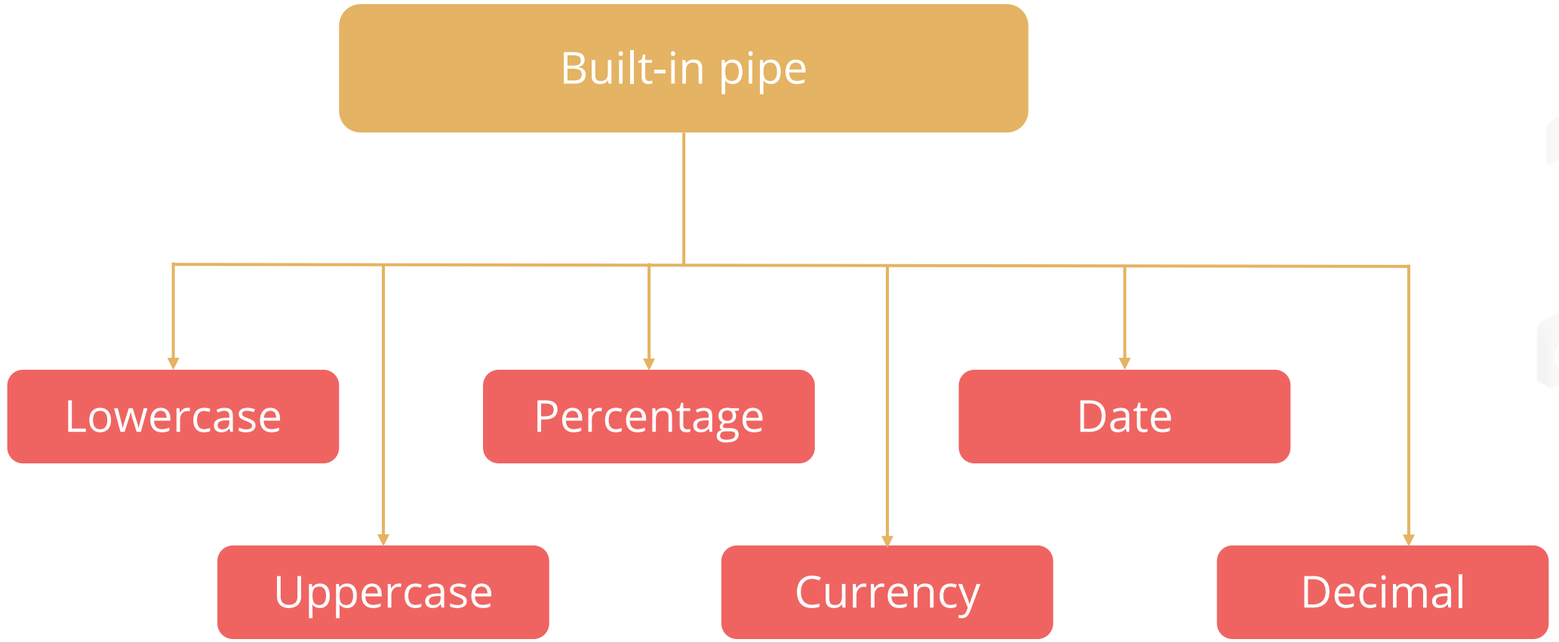
- Pipes are specified using the pipe "|" symbol.
- Pipes can be chained to other pipes.
- Pipes can be provided with arguments by using the colon (:) sign.

Types of Angular Pipes



Built-in Pipe

Angular provides built-in pipes for typical data transformations, including transformations for internationalization, which uses locale information to format data.

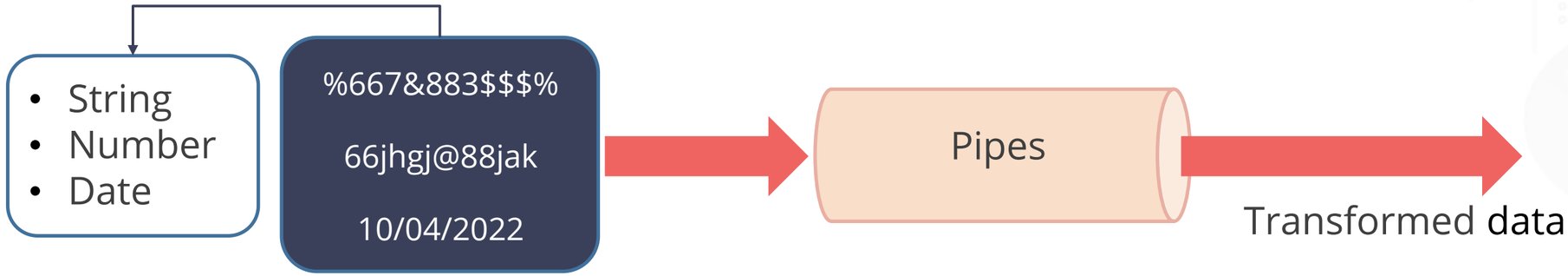


Built-in Pipe: Date Pipe

Date pipe

Formats a date value

```
{{ value_expression | date [ : format [ : timezone [ : locale ] ] ] }}
```



Built-in Pipe: Date Pipe

Example:

'long'	'MMMM d, y, h:mm:ss a z'	June 15, 2015 at 9:03:01 AM GMT+1
'full'	'EEEE, MMMM d, y, h:mm:ss a zzzz'	Monday, June 15, 2015 at 9:03:01 AM GMT+01:00

'short Date'	'M/d/yy'	6/15/15
'mediumDate'	'MMM d, y'	Jun 15, 2015
'longDate'	'MMMM d, y'	June 15, 2015
'fullDate'	'EEEE, MMMM d, y'	Monday, June 15, 2015

'shortTime'	'h:mm a'	9:03 AM
'mediumTime'	'h:mm:ss a'	9:03:01 AM
'longTime'	'h:mm:ss a z'	9:03:01 AM GMT+1
'fullTime'	'h:mm:ss a zzzz'	9:03:01 AM GMT+01:00

Built-in Pipe: Upper-case Pipe

Upper-case pipe

Transforms text to uppercase

```
{{ value_expression | uppercase }}
```

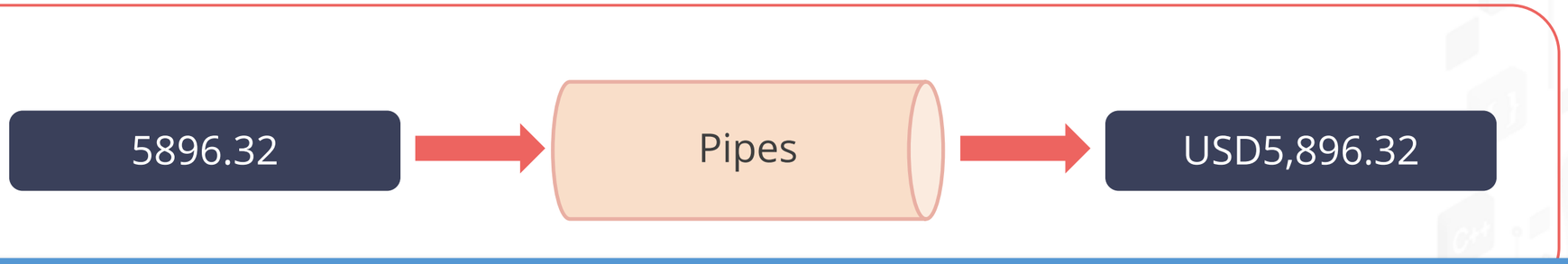
Built-in Pipe: Lower-case Pipe

Lower-case pipe

Transforms text to lowercase

```
{{ value_expression | lowercase }}
```

Built-in Pipe: Currency Pipe




```
{{ value_expression|currency[ : currencyCode [ : display[ : digitsInfo[ : locale ]]]] }}
```

ISO 4217 Currency code

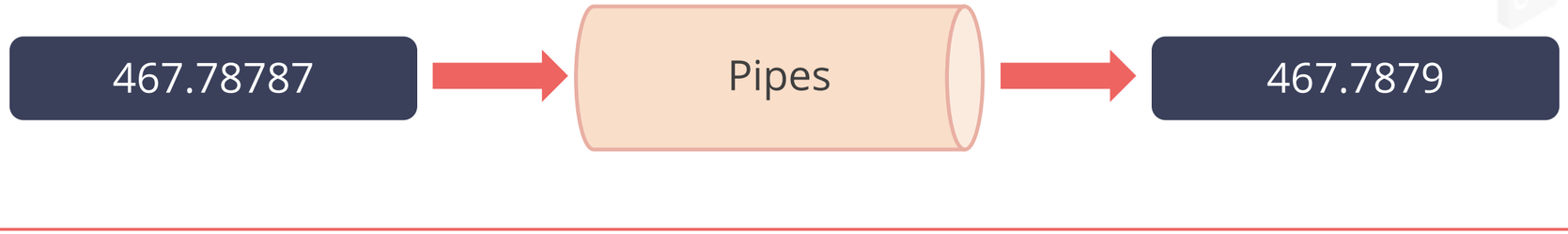
CAD	Canadian Dollar
USD	US Dollar
INR	Indian Rupees

Built-in Pipe: Decimal Pipe

Decimal pipe

Transforms a number into a decimal point string

```
{{ value_expression | number [ : digitsInfo [ : locale ] ] }}
```



Built-in Pipe: Percent Pipe

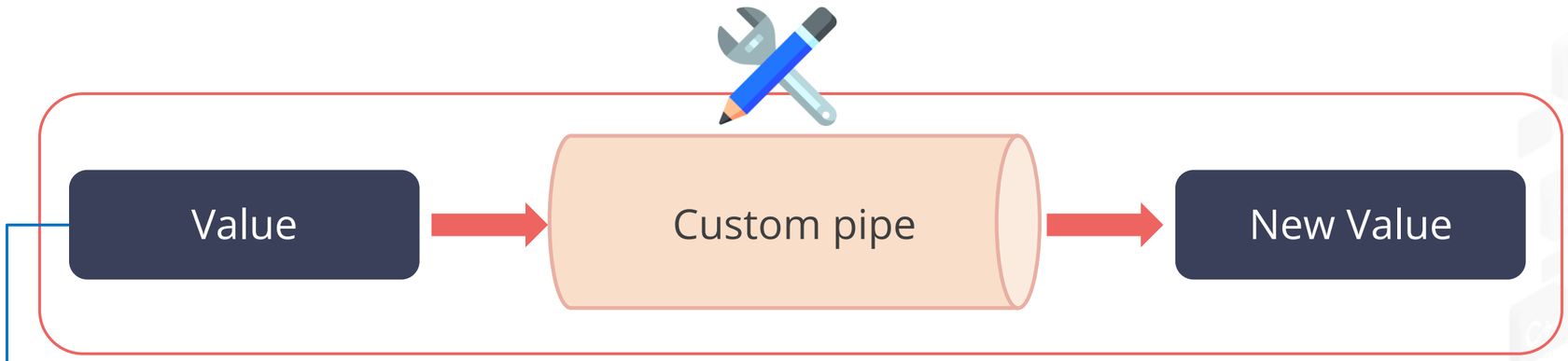
Percent pipe

Transforms a number into a percentage string

```
{{ value_expression | percent [ : digitsInfo [ : locale ] ] }}
```

Custom Pipe

Custom pipes transform the data into the desired format for the users.



- String
- Number
- Date

```
{{ value_expression | custompipe}}
```

Custom Pipe with Parameter

Name of the class

Name of the pipe

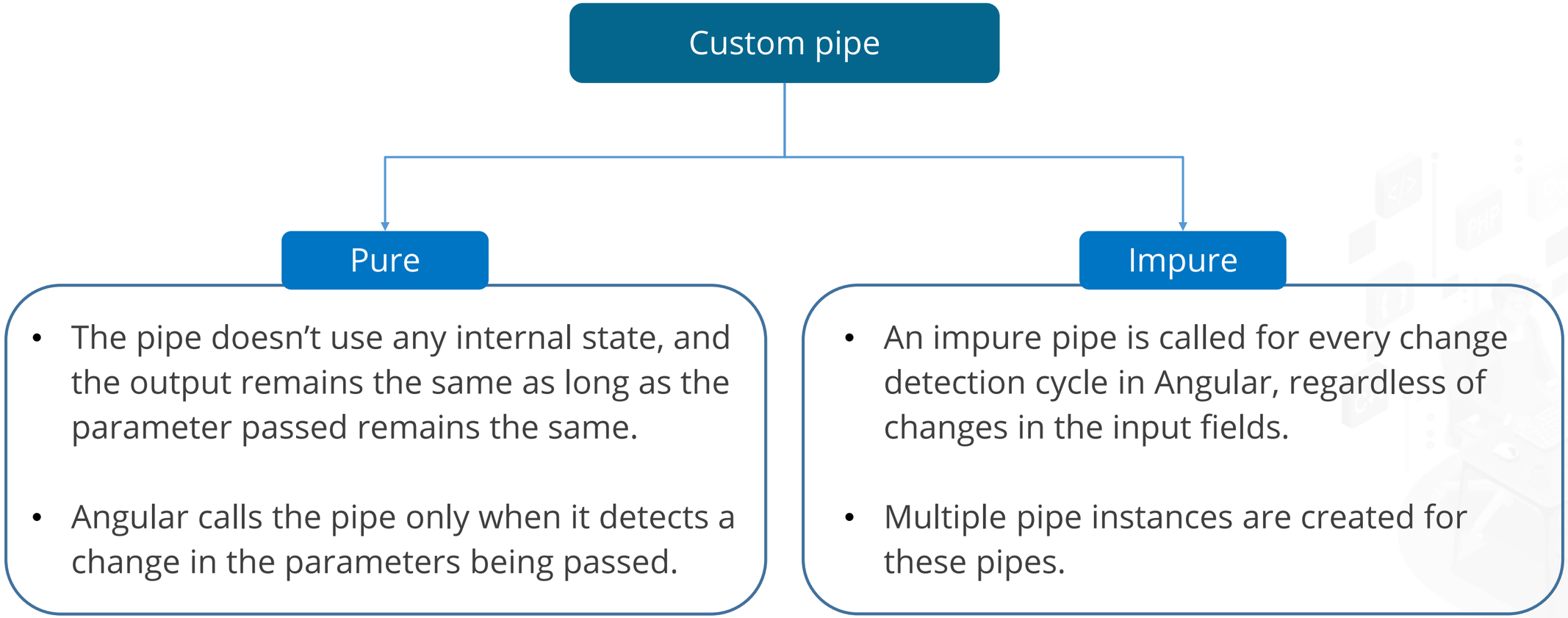
```
import {Pipe, PipeTransform} from '@angular/core';
@Pipe ({
   name : 'Pipename'
})
export class Pipeclass implements PipeTransform {
   transform (Parameters): returntype { }
}
```

Parameters that are passed to the pipe

Function to work with the pipe

returntype of the pipe

Types of Custom Pipe



Pure Pipe: Example

```
import { Pipe, PipeTransform } from '@angular/core’;
@Pipe({
name: 'currency‘
pure:true
})
export class CurrencyPipe implements PipeTransform {
transform(value: any, args?: any): any {
if (!value) {
return '1.00';
}
return value;
}
}
```

Impure Pipe: Example

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
name: 'currency',
pure:false
})
export class CurrencyPipe implements PipeTransform {
transform(value: any, args?: any): any {
if (!value) {
return '1.00’;
}
return value;
}
}
```

Chaining Pipes

Multiple operations can be carried out within a single expression using the chaining pipe.

For example:

To display the birthday in upper case, use the inbuilt date pipe and upper-case pipe.

```
{{birthday | date | uppercase}}
<!- The output is - Friday, May 11, 2000 -->
```

Pipe Transform Interface

It is an interface that is implemented by pipes to perform a transformation.

```
interface PipeTransform {
transform(value: any, ...args: any[]): any
}
```

Implementing Angular Pipes

Duration: 30 min.

Problem Statement:

You have been assigned a task to implement pipes in Angular.

Outcome:

By following the steps, you will successfully create and work with pipes in Angular.

Note: Refer to the demo document for the detailed steps:
10_Working_with_Angular_Pipes

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a pipe code in Angular
2. Run the code on the browser

Key Takeaways

- Components in an Angular application encapsulate the template, data, and behavior of view.
- Angular CLI is used to initialize, develop, and maintain Angular applications directly from a command shell.
- Two-way binding binds data from model class to the view template and from view template to the model class.
- Pipes are simple functions that accept a string and transform data into desired information.
- There are two types of forms in Angular, namely reactive forms and template-driven forms.

TECHNOLOGY

Thank You

simplilearn