Full Stack

Caltech | Center for Technology & Management Education

Post Graduate Program in Full Stack Web Development

simplilearn

Develop a Web Application using frontend stack

Working with an Angular Application

You Already Know

Course(s):

1. An Introduction to TypeScript

1. Angular Training Course

1. MongoDB Developer and Administrator Certification Training

- Setup your environment for TypeScript
  - Install Visual Studio Code
  - Configuring TypeScript via tsconfig.json

- Demonstrate Primitive and Non-Primitive data types
  - Primitive data types: Boolean, Number, and String
  - Non-Primitive data types: Arrays, Tuples, Enum, and Functions

- Explain object-oriented TypeScript functionality
  - Interfaces
  - Extends keyword
  - Type vs interface
  - Classes
  - Access Specifiers

Caltech | Center for Technology & Management Education

simplilearn

- Build Angular components
  - Angular CLI
  - Nested components
  - Lifecycle of Angular components

- Understand Bootstrap
  - Creating a responsive web application

- Explain binding and events
  - Template model
  - Built-in directives
  - Basics of webpack and SystemJS

Caltech | Center for Technology & Management Education

simplilearn

- Explain dependency injection and services
  - Dependency Injection API
  - Creating a service

- Test an Angular app
  - Working with Angular directives
  - Working with custom pipes
  - Working with Angular forms

- Test an Angular app
  - Testing Angular Class
  - Testing service and DOM

- Work with NoSQL databases
  - Benefits and types of NoSQL
  - CAP Theorem

- Understand the basics of MongoDB
  - JSON and BSON
  - Transaction management
  - Scaling in MongoDB
  - Data types

- Perform CRUD operations
  - Data modification in MongoDB
  - Performing various operations like insertion and retrieval
  - Regular expressions

Caltech | Center for Technology & Management Education

simplilearn

- Explain indexing and aggregation
  - Types of index and index creation

- Explain replication and sharding
  - Replica set in MongoDB
  - Write concern levels and read preference models
  - Shard key

- Administer MongoDB cluster operations
  - Capped collection creation
  - TTL collection features
  - Storage engines

A Day in the Life of a Full Stack Developer

In the last sprint, Joe had done a remarkable job. He has developed an application that allows the addition and deletion of items in a cart.

In this sprint, Joe must develop the frontend of the application using an Angular framework. Since he was recently trained in working on this framework, he has been chosen to lead this project and complete the initial part of the application. He agrees to complete the task as early as possible, as the release of the application is next week.

In this lesson, we will learn how to solve this real-world scenario to help Joe complete his task effectively and quickly.

Learning Objectives

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

⊙ List the importance of an Angular framework

⊙ Build an Angular application with customized requirements

⊙ Create responsive forms for user input and validation

⊙ Implement the routing mechanism in your Angular application

Angular Recap

Introduction to Angular JS

• AngularJS was created by Misko Heavery.

• This first version of the framework known as AngularJS was launched in the year 2009.

• Below were the features of Angular JS:

1. Angular JS is a javascript MVW framework.
2. It supports HTML extension by tags, attributes, and expressions.
3. Event Handling can be easily done in Angular JS.
4. It also supports Data Binding.
5. It had a rich library of packages that support and implement built-in templates and Routing.
6. It has various Form-Validations and Animations
7. Reactive and responsive web applications

Angular JS vs. Angular

Angular JS

- It uses Javascript as a base language to build a Single Page Application.
- Development on Angular JS is no more done.
- MVC based architecture.
- It doesn't support mobile compatibility.
- ES5, ES6, and Dart based coding.
- It was developed on the controllers, which is no longer applicable.
- Services consist of factory, provider, value and constant
- Client-side development.
- ng-app and angular bootstrap function are used to initialize

Angular

- Complete rewrite of the AngularJS version.
- It's updated version regularly released because of Semantic Versioning.
- The architecture of Angular 2 is based on service/controller.
- Angular 2 is a mobile-oriented framework.
- We can use ES5, ES6, Typescript to write an Angular 2 code.
- Nowadays, the controllers are replaced by components, and Angular two is completely component-based.
- The class is the only method to define services in Angular2
- Runs on client-side & server-side
- bootstrapmodule() function is used to initialize

Migrating from Angular JS

Upgrading or migrating from Angular JS to Angular is a difficult task

- It is important to upgrade from your current Angular to the latest version after keeping the business needs in mind.

- Moreover, upgrading Angular version should be done at the end of the project, if the project is already in progress.

Ways to Upgrade

Migrating using Gulp

MEAN Stack Developer

Migrating using NgUpgrade
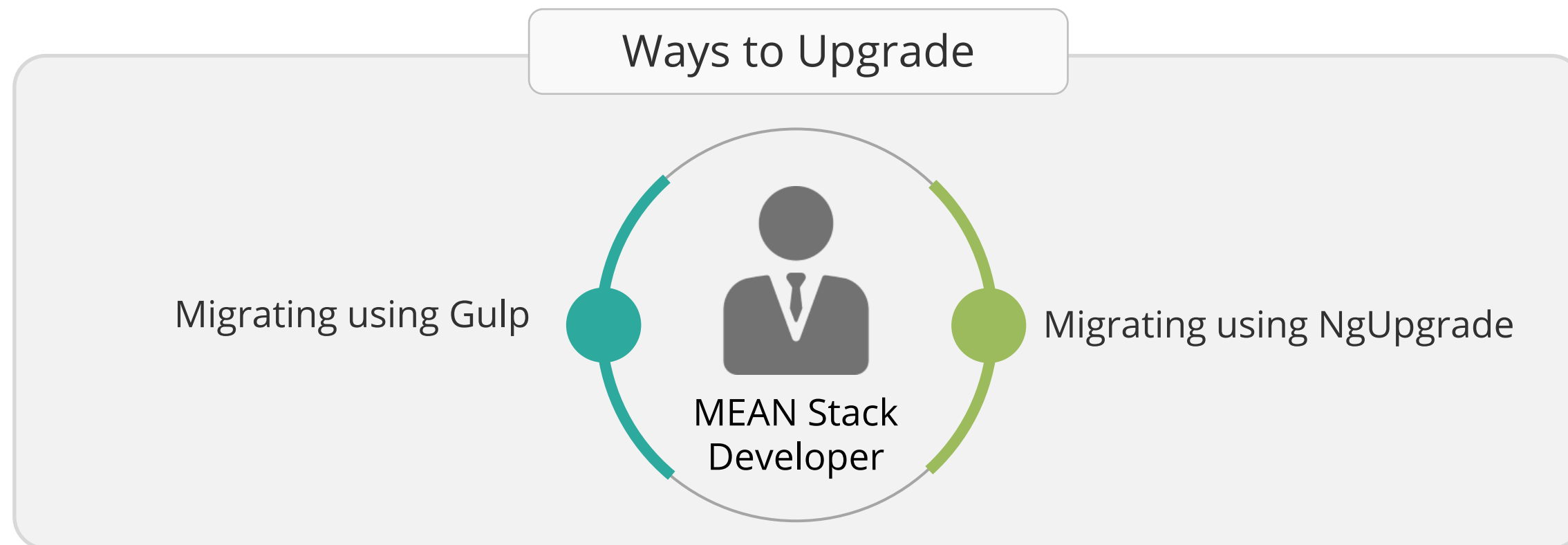
Migrate using NgUpgrading

Ingredients to keep ready before we start migration:

Code	Make sure your current application is features oriented and have one feature per file
Typescript	It is important to install and setup typescript
Modular	Most widely used is Webpack
Angular 2+	All the controllers will be replaced with components in Angular 2+ versions
Setup NgUpgrade	Setting up NgUpgrade will help you run both apps alongside each other and you can make the necessary changes

Introduction to Angular 10

Angular 10

Angular 10 is a smaller version of Angular that adds new features such as a new date range picker in the Angular Material UI component library and CommonJS import warnings.

Why Angular 10?

These are the reasons why we should migrate to the latest version of the Angular framework:

- It is easier to load.

- It also features several workflow and performance improvements, as well as many bug fixes and updates.

- It improves performance, resulting in a speedier Angular build, which is especially beneficial for large projects.

- It is ideal for creating micro front-end applications with a small number of dependencies.

Caltech | Center for Technology & Management Education

simplilearn

Angular 10 Features

In addition to the updates in the Angular versions, these are some features that make Angular 10 easy to use:

- Language service
- New default browser configuration
- Optional stricter settings
- Performance improvements
- Localization

Language Service

The language-service-specific compiler allows multiple type check files to be created using the project interface which builds ScriptInfos as needed.
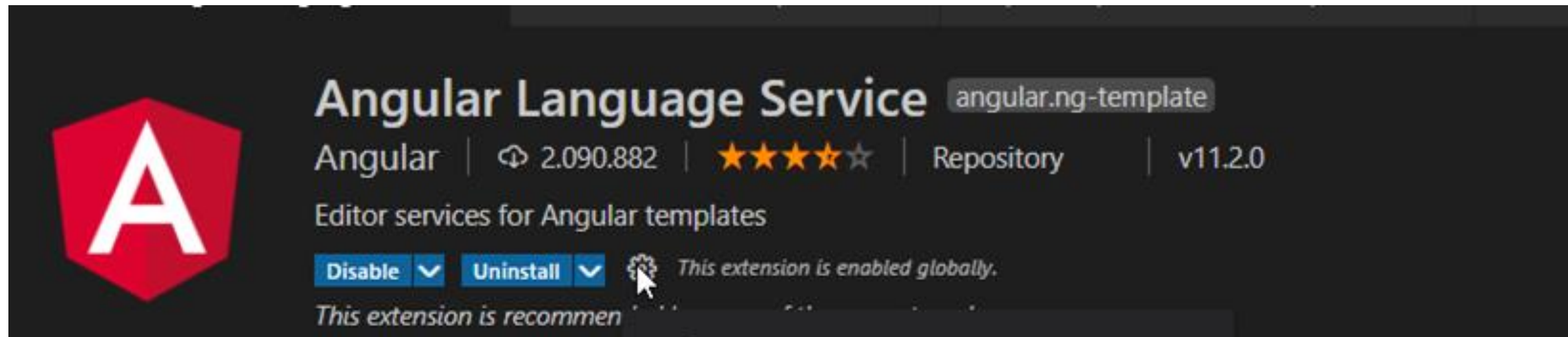
Autocompletion is also removed from the HTML entities such as &, <, and others.

New Default Browser Configuration

Angular 10 modifies the browser configuration for new projects. This revised version has removed the older and less popular browsers.

Users can add the relevant browsers to the .browserslist RC file to enable ES5 builds and differential loading.

Caltech | Center for Technology & Management Education

simplilearn

Optional Stricter Settings

Angular 10 enforces a stricter project setup when a user uses the **ng new** command to establish a new workspace.

We can use the ***ng new --strict*** command to enable this strict feature.

Optional Stricter Settings

The following is a list of various activities that the strict flag performs:

- It allows strict mode in TypeScript.

- It turns the template type checking to Strict.

- It sets up linting rules to keep declarations of type any from being made.

- It reduces the default bundle budgets up to 75%.

Performance Improvements

Following is a list of some resolved issues and bug fixes that we faced in the previous versions:

Router

The UrlMatcher in the upgraded version of Angular 10 can return null.

Core

The derived classes are not decorated if you have undecorated-classes-with-decorated-fields migration.

Modules

Angular 10 can correctly identify modules in TestBed that are affected by overrides.

Performance Improvements

Compiler

The compiler avoids undefined expressions in a holey array when the user imports a nonexistent symbol, and the core avoids a migration error.

Service Workers

Angular 10 prevents app stability from being impacted by SW registration strategies.

Localization

Localization in Angular.js

- The latest angular version allows the merging of many translation documents, whereas previous versions could only load one file.

- Customers can specify a few documents based on their location, and the translations from each of these documents can be blended using a message ID.

TypeScript

TypeScript

TypeScript is the primary language for Angular application development and a JavaScript superset with type safety and tooling assistance at design time.
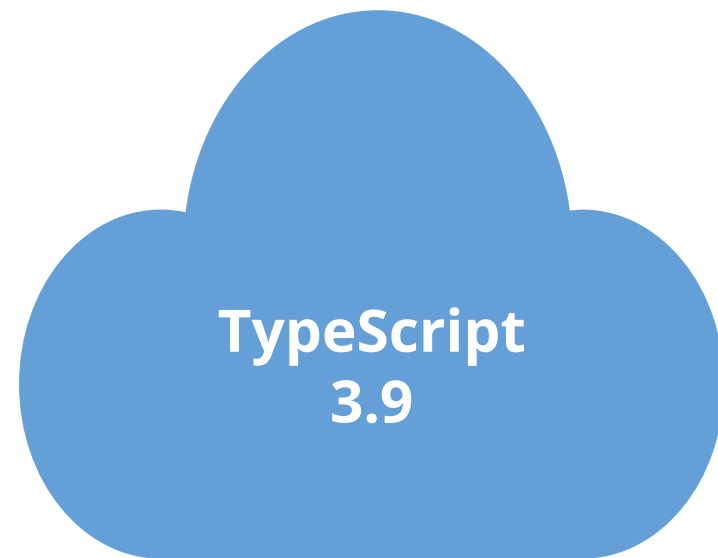

Angular 10 supports the TypeScript version 3.9.

TypeScript 3.9

Typescript version 3.9 enables completions, speedy corrections, and a speedier compiler and editing experience.
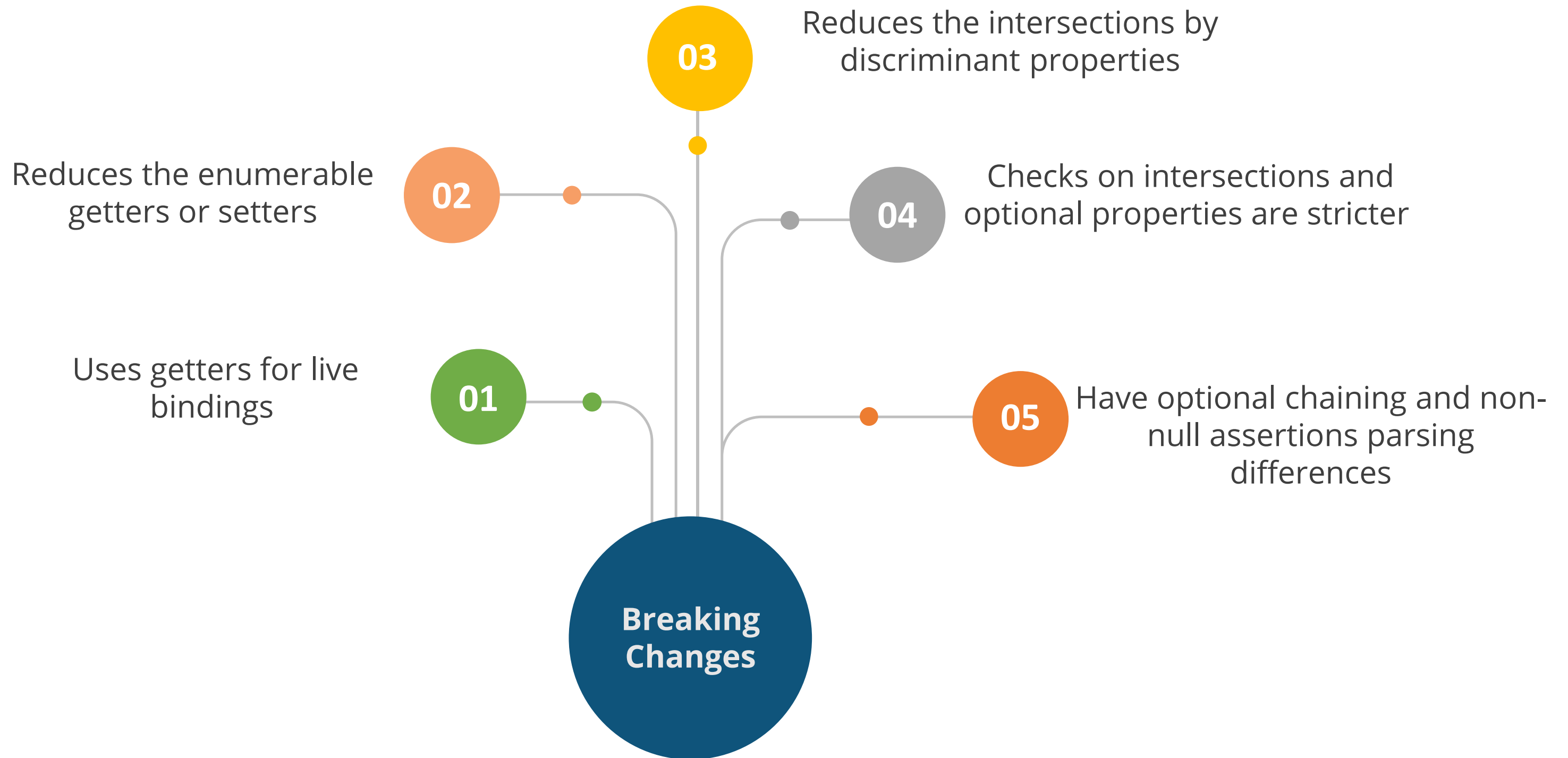
New Features of TypeScript 3.9

TypeScript 3.9

- Speed improvements
- Quick fixes for missing return expressions
- CommonJS auto-imports in JavaScript
- Breaking changes

Caltech | Center for Technology & Management Education

simplilearn

TypeScript 3.9

03 Reduces the intersections by discriminant properties

02 Reduces the enumerable getters or setters

04 Checks on intersections and optional properties are stricter

01 Uses getters for live bindings

05 Have optional chaining and non-null assertions parsing differences

Breaking Changes

Caltech | Center for Technology & Management Education

simplilearn

Speed Improvements

> There were a lot of concerns regarding the compilation and editing speed with older versions of TypeScript, especially with packages like Material-UI.

> This issue is addressed in the latest release by optimizing various pathological instances, which has lowered the Material-UI compilation time by 40%.

> In addition, functionality for renaming files in editors like Visual Studio Code is being developed.

CommonJS Auto-Imports in JavaScript

TypeScript previously required ECMAScript-style imports regardless of the file type. When building JavaScript files, however, most developers use the **CommonJS-style need();** import rather than the ECMAScript-style modules.

ECMAScript style: **import * as** fs **from** "fs";

CommonJS style: **const** fs = require("fs");

TypeScript now recognizes the types of imports users are using to keep their file's style tidy with the current release.

UI Component Libraries for Angular JS Developers

Angular Material

Google released Angular Material, a UI library component, in 2014. It was created with AngularJS developers in mind.

Angular Material

- It attracts consumers and allows them to access the elements or components of our app.

- It also aids in the creation of distinctive styles and forms for our apps.

- These elements improve the application's or website's design consistency and responsiveness. It mixes traditional design concepts with innovation and technology to create a successful product.

Features of Angular Material

- It is an in-built responsive design.

- It has standard CSS.

- It is a cross-platform browser that lets users make web components.

- It has specialized features such as cards, toolbar, speed dial, side nav, swipe, side nav, and more.

Features of Angular Material

These are some more features that make Angular material easy to use:

Cost effective

06

Responsive design

01

Frictionless

05

Common Signals

Extensible

04

Versatile

02

High quality

03
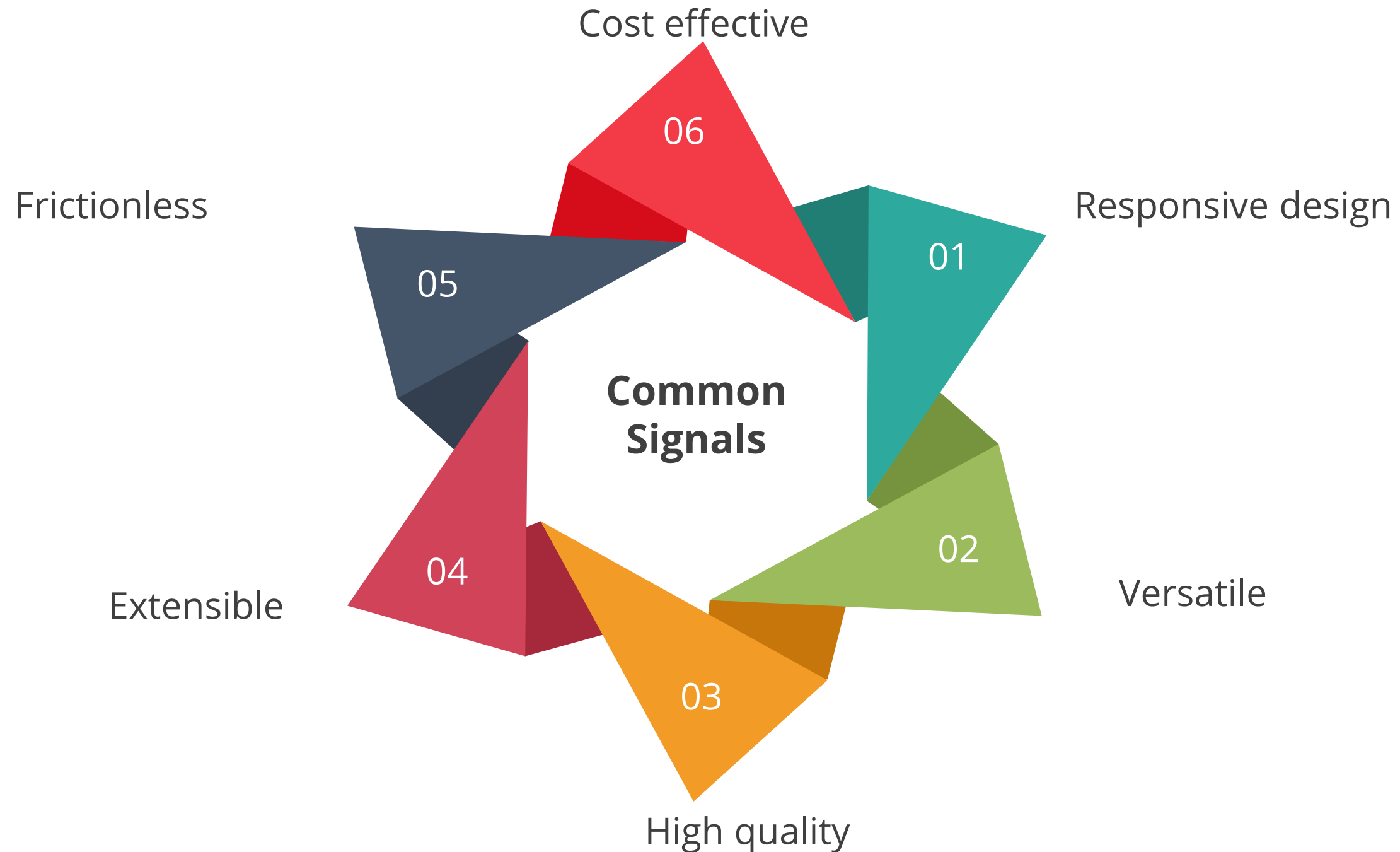
Browser Support

AngularJS Material supports the following types of browsers:

01   Evergreen browsers' two most recent major versions

02   Firefox ESR

03   Not considered "dead" browsers

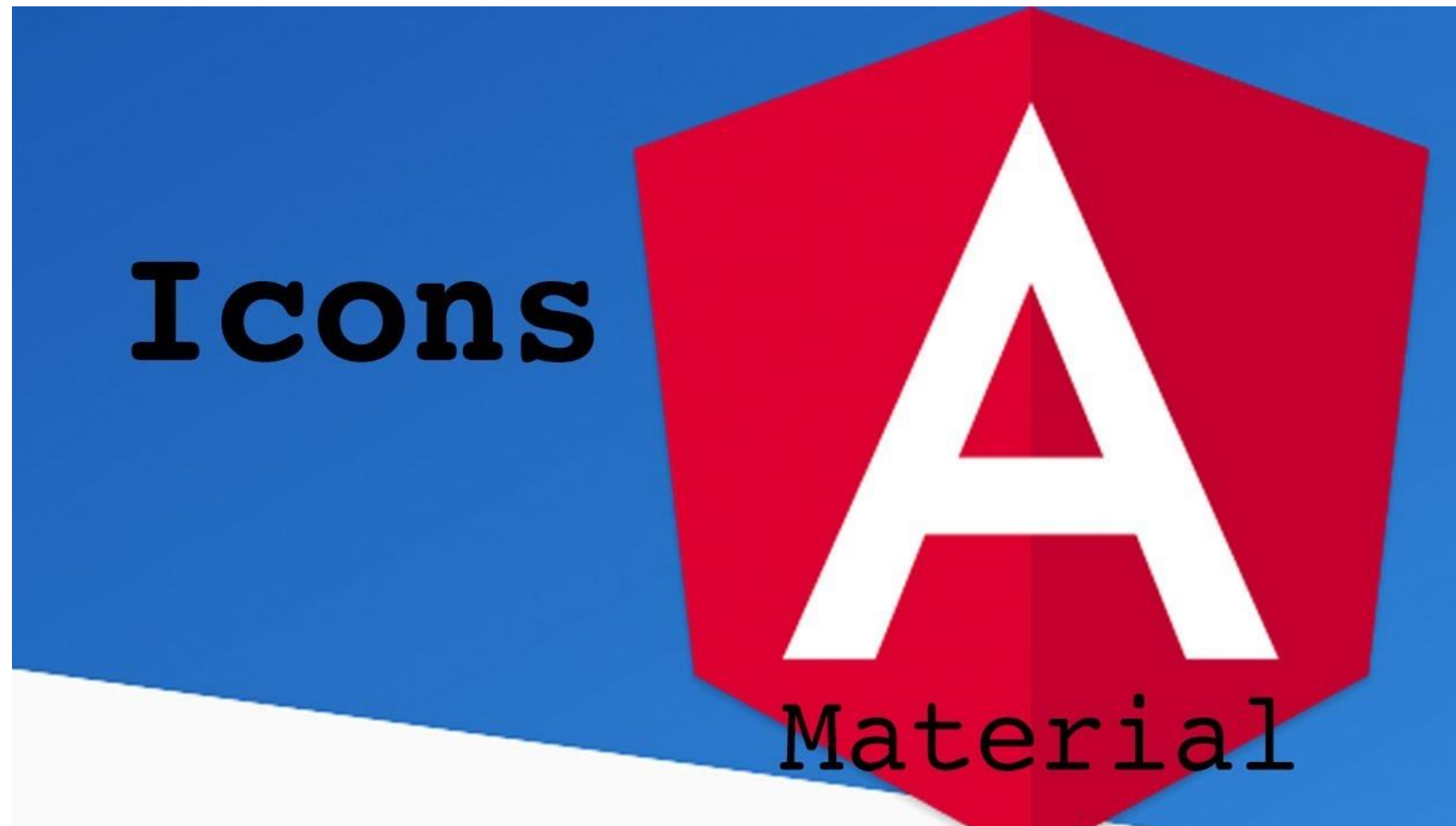Caltech | Center for Technology & Management Education

simplilearn

Angular Material Installation

These are the following requirements to install Angular Material:

⚙ Node.js greater than 8.11

⚙ npm greater than 5.6
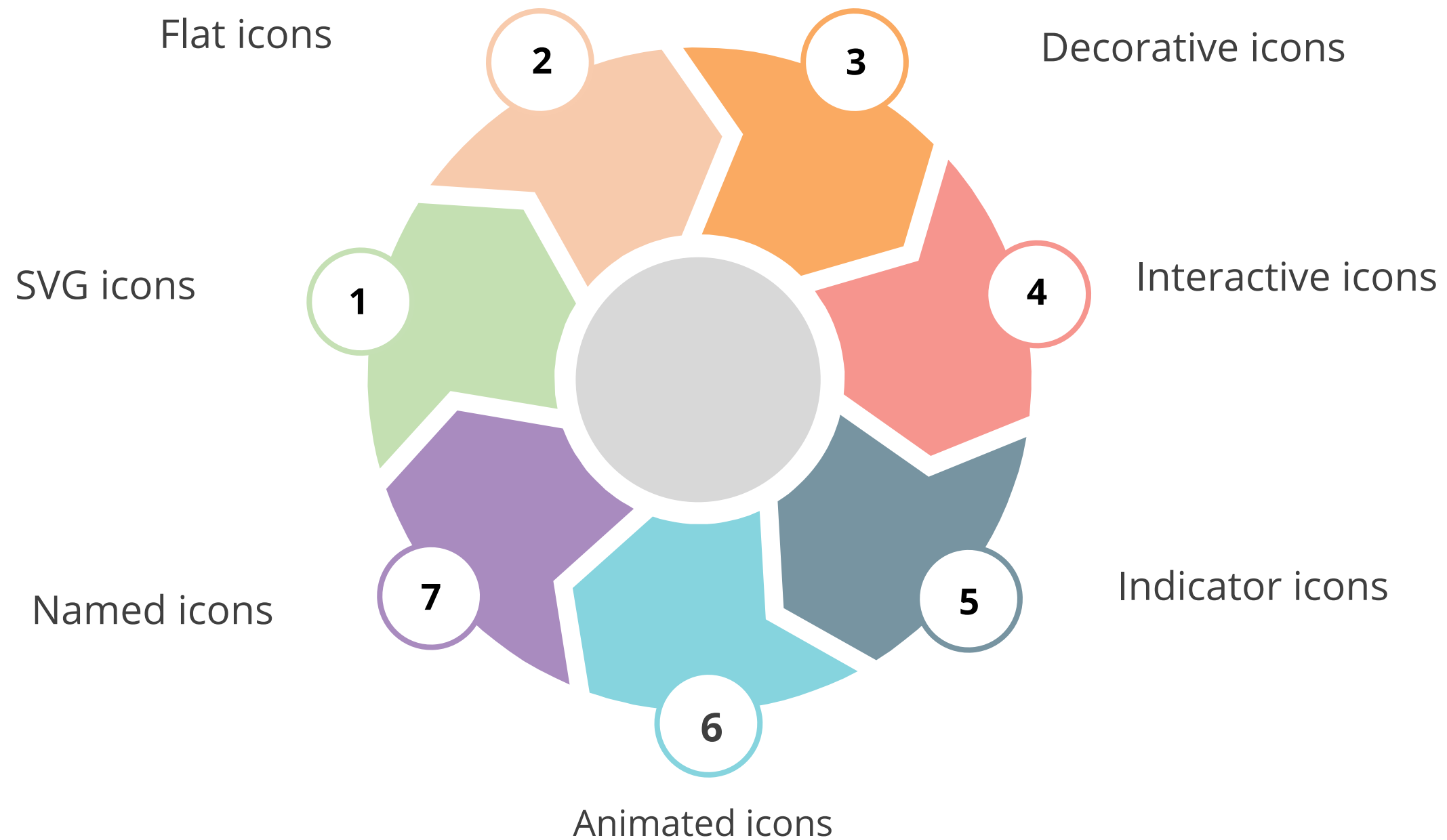
⚙ Angular CLI

⚙ IDE for writing the code

Angular Material Icons

An Angular Material icon is a component used in applications to display vector-based icons. In addition to using Google content icons, it also supports icon fonts and SVG icons.
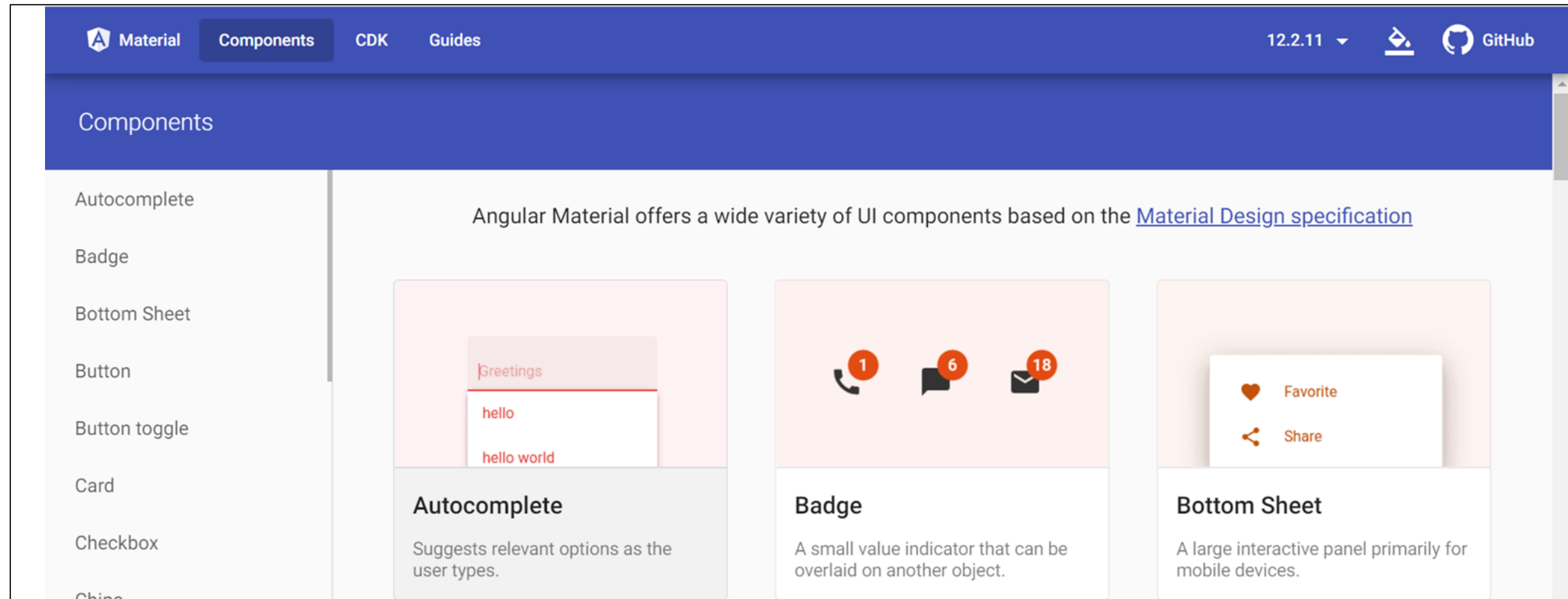
Angular Material Icons
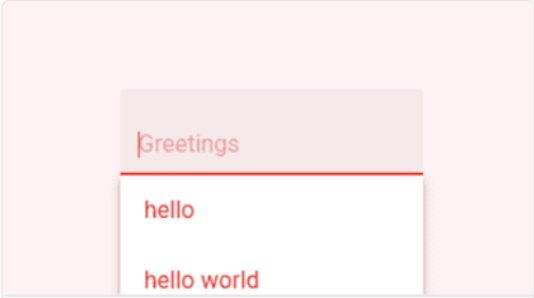
The different types of Angular Material icons are:

Flat icons

Decorative icons

SVG icons

Interactive icons

Named icons

Indicator icons

Animated icons

Caltech | Center for Technology & Management Education
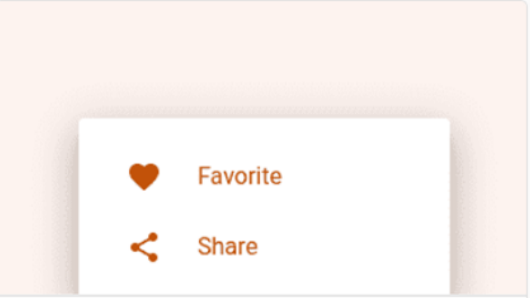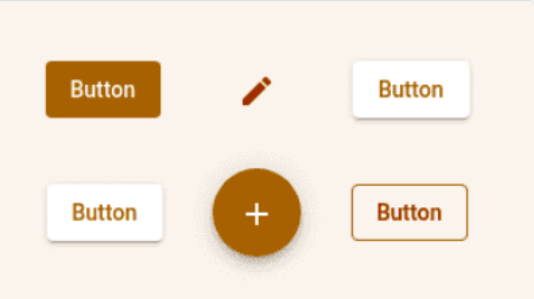
simpl**i**learn
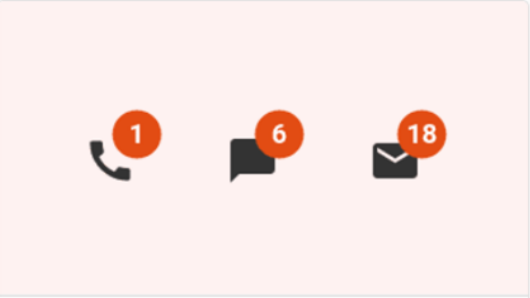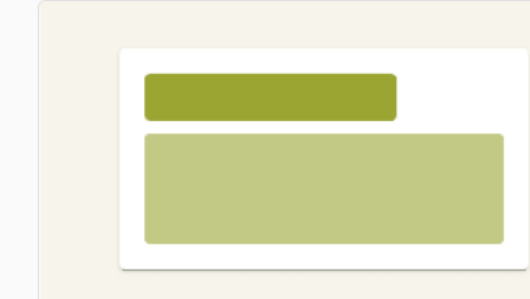
Angular Material Components

Material Components are interactive elements that can be used to create a user interface.

Angular Material Components

These are the different types of components:

Autocomplete

Suggests relevant options as the user types.

Badge

A small value indicator that can be overlaid on another object.

Bottom Sheet

A large interactive panel primarily for mobile devices.

Button

An interactive button with a range of presentation options.

Button toggle

A groupable on/off toggle for enabling and disabling options.

Card

A styled container for pieces of itemized content.

Caltech | Center for Technology & Management Education

simplilearn

Angular Material Components

These are the different types of components:

Checkbox

Captures boolean input with an optional indeterminate mode.

Chips

Presents a list of items as a set of small, tactile entities.

Core

Reusable parts used by other components in the library.

Datepicker

Captures dates, agnostic about their internal representation.

Dialog

A configurable modal that displays dynamic content.

Divider

A vertical or horizontal visual divider.

Angular Material CDK

The Component Dev Kit (CDK) is a set of behavior primitives for building UI components. It is a collection of tools that implement common interaction patterns while remaining agnostic about their presentation.
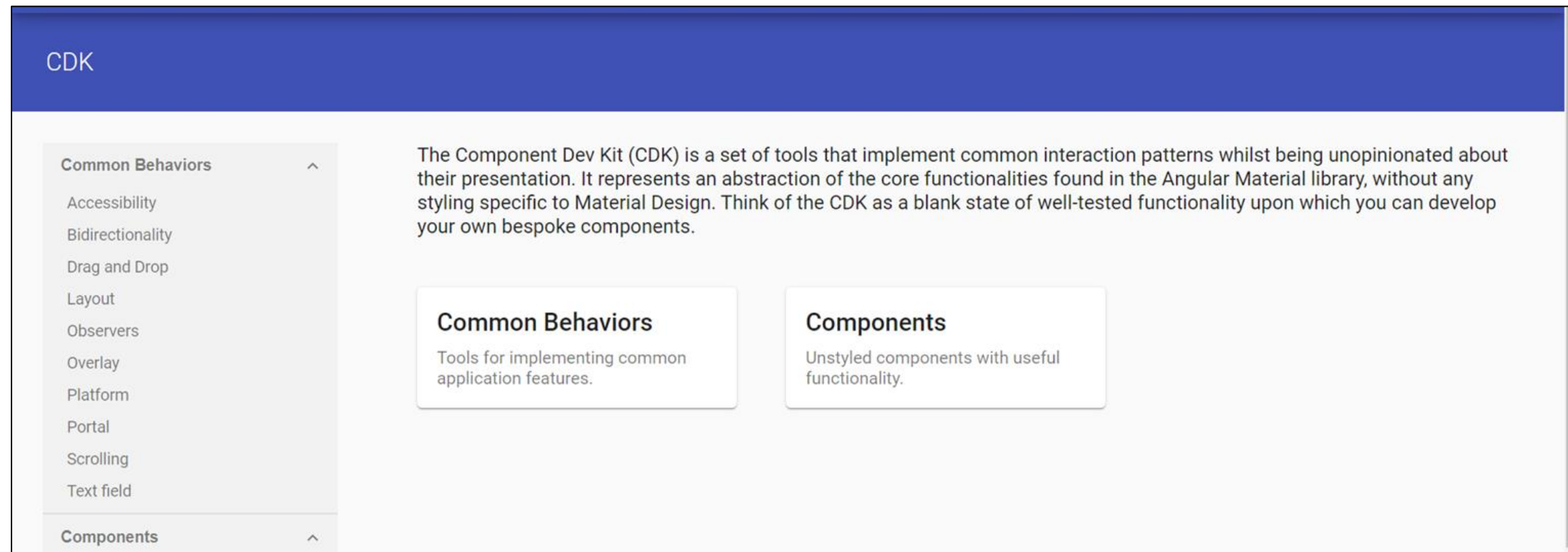
Angular Material CDK

The Angular Material CDK contains the following:

01 Accessibility

02 Accordion

03 Bidirectionality

04 Clipboard

05 Collections

06 Layout

Angular Material CDK

07 Drag and Drop

08 Observers

09 Portal

10 Table

11 Tree

12 Text field

Assisted Practice

Angular Material Installation   **Duration: 30 Min.**

Problem Statement:

Install Angular Material and use its components.

Assisted Practice: Guidelines

Steps to install Angular Material:

1. Create an angular application

2. Install Angular Material

3. Use the material UI components

PrimeNG

PrimeNG is a set of Angular components that provide a rich user interface.

PrimeNG

PrimeNG is accessible at npm. If you already have an application, use the following command to add it to it:

npm install primeng --save
npm install primeicons --save

Why PrimeNG ?

These are the reasons to use PrimeNG for Angular UI components:

Single library

PrimeNG includes a large set of components that will satisfy the majority of application's UI requirements, eliminating the need to add multiple libraries for diverse UI requirements.

Templating

PrimeNG components offer templating, which allows us to customize the content using the template property.

Caltech | Center for Technology & Management Education

simpli learn

Why PrimeNG ?

Active development

PrimeNG library is still under active development. So, most of the issues raised will be addressed quickly, and new versions of the library will be released frequently with improvements.

Themes

PrimeNG provides a variety of UI component themes from which to choose. This aids us in selecting the components' appearance and feel.

Limitations of PrimeNG
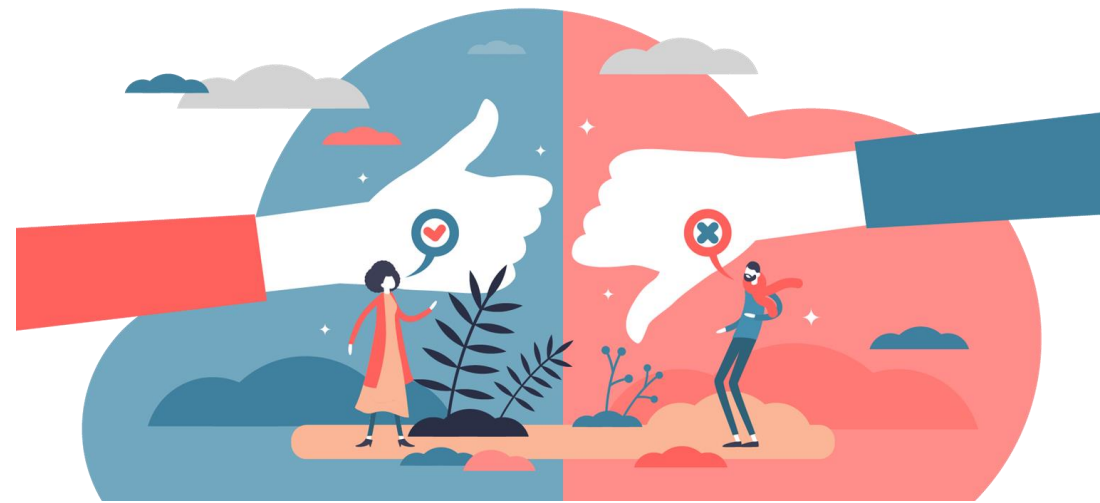
- **Inconsistent documentation:** PrimeNG component documentation does not always list all of its characteristics and events.

- **Only bug fixes:** The PrimeNG team only accepts pull requests from the community for bug fixes and exposes new properties in a component. They don't encourage people to submit pull requests for new component features.

Caltech | Center for Technology & Management Education

simplilearn

Dependencies

The majority of PrimeNG components (95%) are native, with a few exceptions requiring third-party dependencies. Components also need PrimeIcons for their icons.

Components	Dependency
Schedule	FullCalendar 4.1.0
Editor	Quill Editor
GMap	Google Maps
Charts	Charts.js 3.3.2+
Captcha	Google Recaptcha

Browser Support

PrimeNG supports these browsers:

Theme Designer API

Designer API is the most powerful tool for creating PrimeNG experience, with a SASS-based theme engine, 500+ variables, and a demo application.
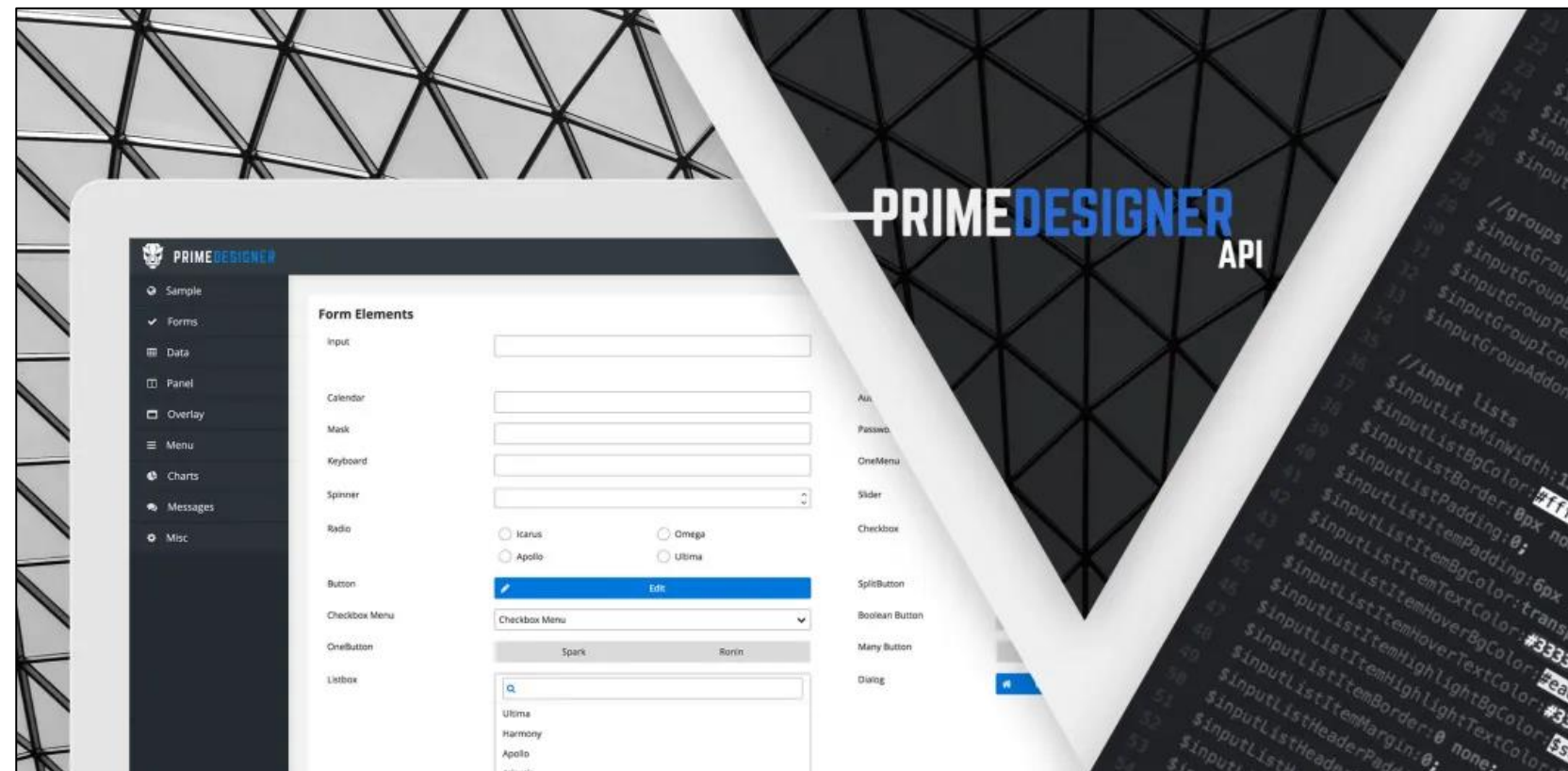
PrimeNG LTS

PrimeNG LTS is a yearly subscription-based service that provides users with a license for the most suitable version available.
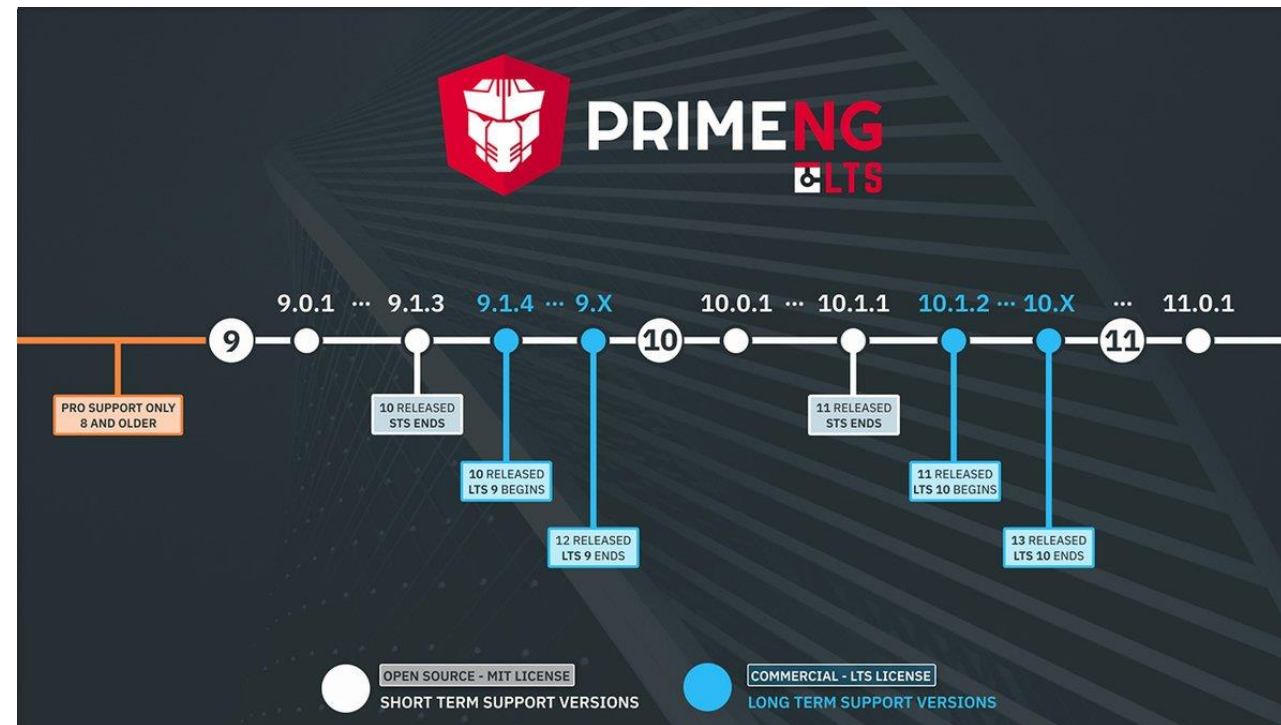
PrimeNG is a commercial software licensed under the LTS License, but PrimeNG is an open-source and free to use library released under the MIT license.

Assisted Practice

PrimeNGI Installation

Problem Statement:

Install PrimeNG

Assisted Practice: Guidelines

Steps to install PrimeNG:

1. Create an angular application

2. Install PrimeNG in it

3. Add PrimeNG CSS to the application

Components

Components

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

Components are also known as View Components.

Every app has at least one component called the Root Component.
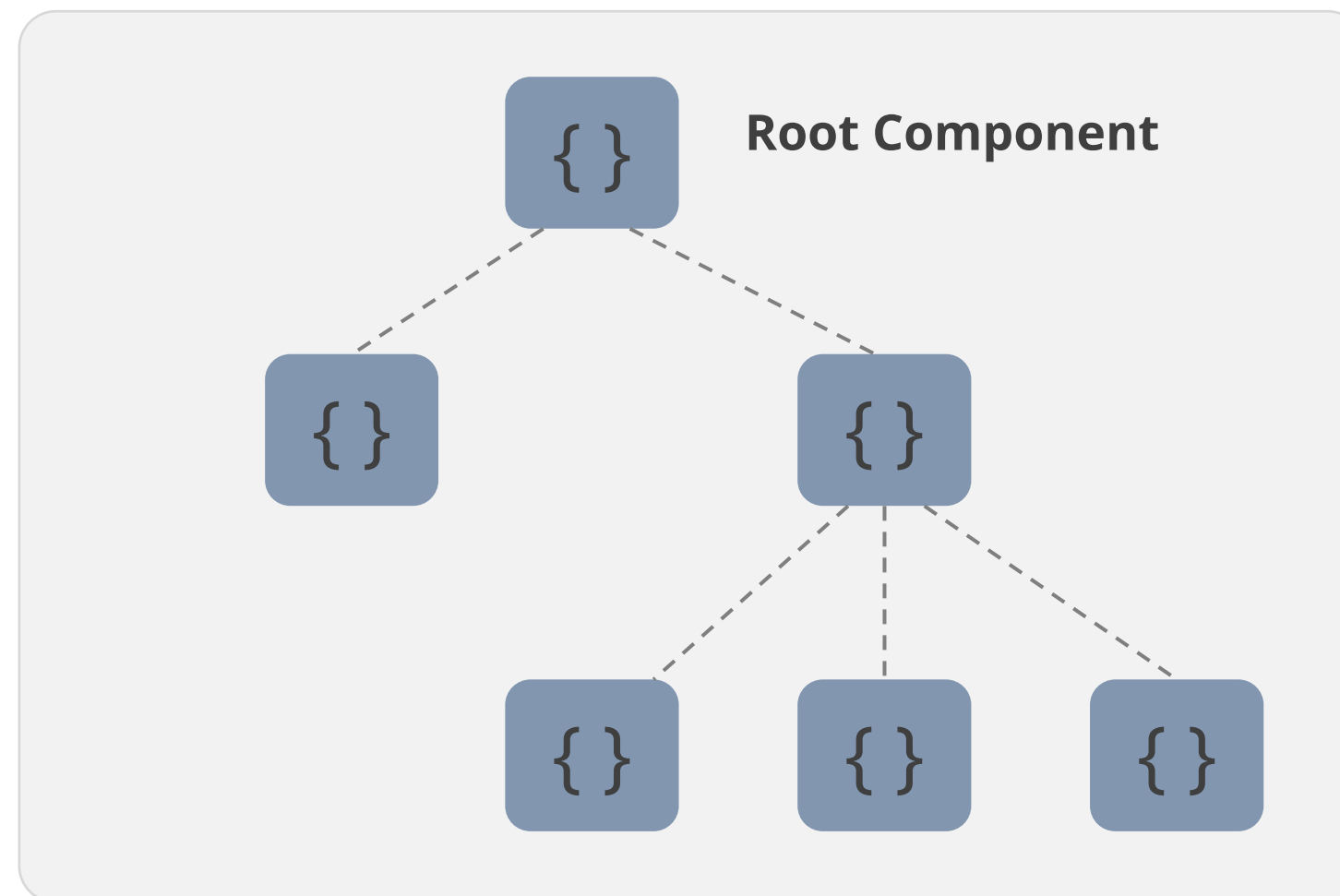
Components

However, in real world an application encapsulates many components.

Here is an example:

{ } **Navbar**

{ } **Sidebar**
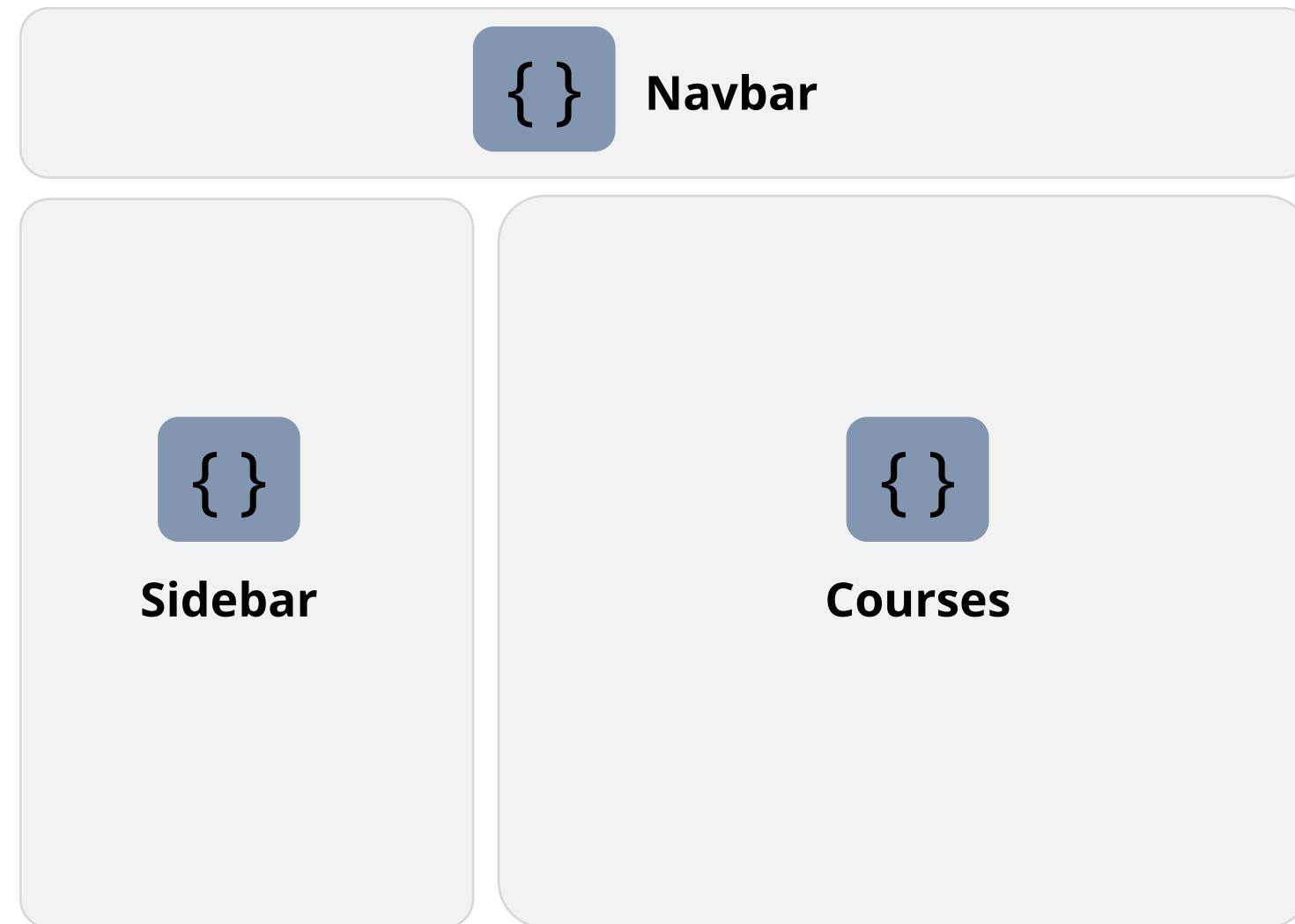
{ } **Courses**

Nested Components

- Each component has a template for the view and data or logic behind the view.

- Components can also contain other components called nested components.

In the diagram, courses are components.
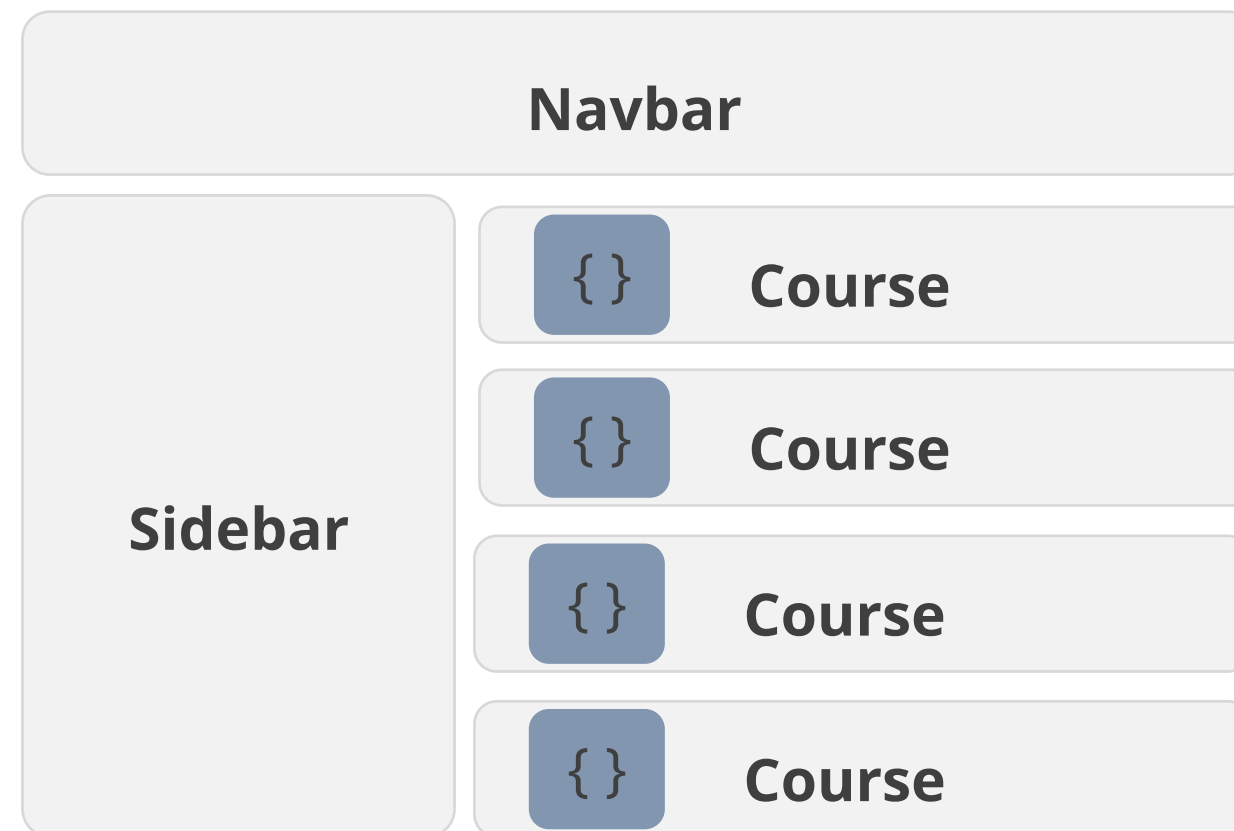
Deeper Nesting

The diagram shows the communication between deeply nested components in Angular.
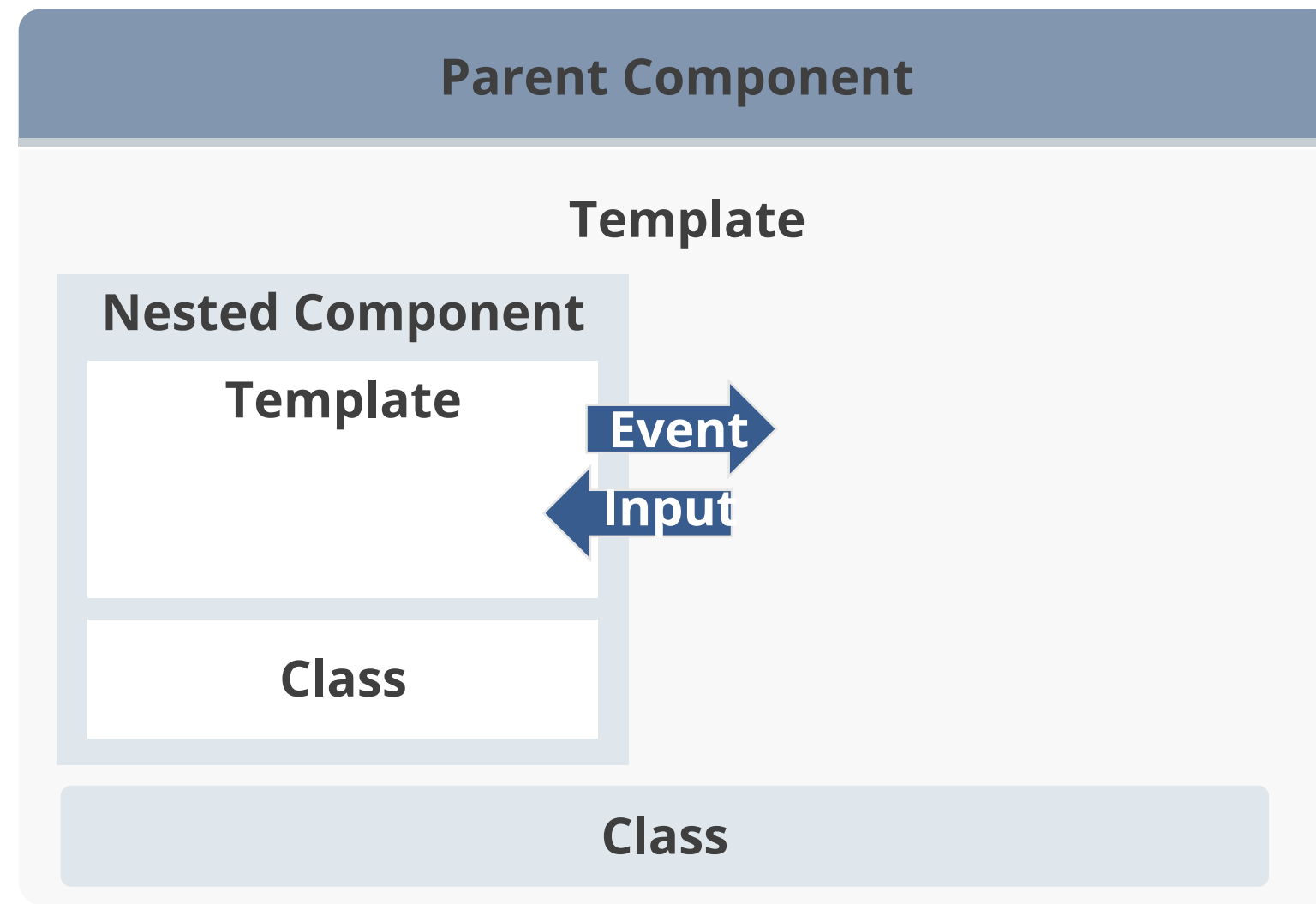
Deeper Nesting

As you write a complex app and reuse more components, the components get smaller and more nested as part of the process. So it is important to understand the working of nested components.
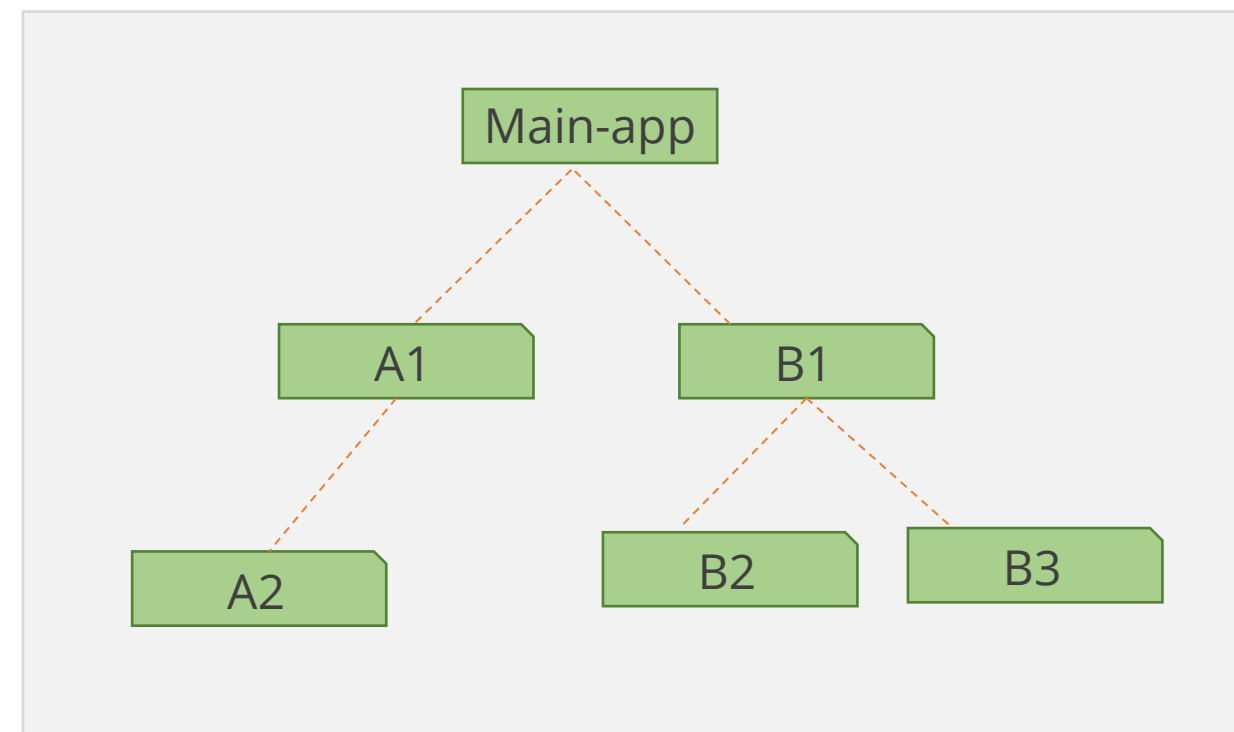
Angular Component's Lifecycle

constructor

ngOnChanges

ngOnInit

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

ngOnDestroy

A component has a lifecycle managed by Angular.

- Angular creates the component and renders it, creates and renders its children, checks when its data-bound properties change, and destroys it before removing it from the DOM.

- Angular offers lifecycle hooks that provide visibility to the key lifecycle moments of the component. It also provides the ability to act when these moments occur.

- A directive has the same set of lifecycle hooks, excluding the hooks that are specific to component content and views.

Assisted Practice

Components

Problem Statement:

Create Angular components.

Assisted Practice: Guidelines

Steps to demonstrate components:

1. Install Angular and create a new project

2. Create Angular components

3. Push the code to GitHub repositories

Data Interactions

Binding

Binding is a mechanism for coordinating between what users see and data values in the application.

Binding

While pushing and pulling values to and from HTML, it will be easier to write, read, and maintain the application if you turn these chores over to a binding framework

Binding between Source and HTML

To let the framework perform the task, we need to declare the bindings between sources and target HTML elements.

Source Binding → **Declare Bindings** → **HTML** Target

Binding

Angular provides different types of data binding.

{{Data binding}}

Binding types can be grouped into three categories that are classified based on the direction of data flow between:

- Source-to-view

- View-to-source

- In the two-way sequence it is "view-to-source-to-view"

Types of Binding Supported by Angular

Property Binding

Class Binding

Style Binding

Event Binding

Two-Way Binding

Component Binding

Property Binding

Property Binding

Every DOM property can be written via special attributes on HTML elements using square brackets []. An HTML attribute can start with anything.

Angular maintains the properties and attributes in sync when you use them, so if you are familiar with Angular 1.x directives, such as ng-hide, you can work directly with the hidden property and ng-hide is no more needed.

```
<div ng-hide="isHidden">Hidden element or not?</div>
```

The most common property binding sets an element property to a component property value.

```
> training@localhost:~

<img [src]="heroImageUrl">

<button [disabled]="isUnchanged">Cancel is disabled</button>
```

One-Time String Initialization

One should omit the bracket only when all the below criteria are met:

Target	If target property accepts string value
Value	String has fixed value once initialized
Template	If string is fixed and can be directly put into the template

Below is an example of one-time string initialization:

```
src/app/app.component.html

<app-string-init prefix="This is a one-time initialized string."></app-string-init>
```

Property Binding vs. Interpolation

The below example clearly explains the difference in the syntax and representation of property binding and interpolation:

src/app/app.component.html

```
<p><img src="{{itemImageUrl}}"> is the <i>interpolated</i> image.</p>
<p><img [src]="itemImageUrl"> is the <i>property bound</i> image.</p>


<p><span>"{{interpolationTitle}}" is the <i>interpolated</i> title.</span></p>
<p>"<span [innerHTML]="propertyTitle"></span>" is the <i>property bound</i> title.</p>
```

Assisted Practice

Property Binding

Problem Statement:

Create an Angular application to demonstrate property binding and interpolation.

Assisted Practice: Guidelines

Steps to demonstrate property binding:

1. Implement string interpolation

2. Implement property binding

3. Push the code to GitHub repositories

Class and Style Binding

Class Binding

You can add and remove CSS class names from an element's class attribute with class binding.

The following example shows how to add and remove the application's **special** class with class binding and how to set the attribute without binding:

```
<!-- standard class attribute setting-->
<div class="bad curly special">Bad curly special</div>
```

Class binding syntax resembles property binding. Instead of an element property between brackets, start with the prefix class, optionally followed by a dot (.) and the name of a CSS class: [class.class-na

```
<!-- reset/override all class names with a binding-->
<div class="bad curly special"[class]="badCurly">Bad curly</div>
```

Style Binding

You can set inline styles with style binding.

Style binding syntax resembles property binding. Instead of an element property between brackets, start with the prefix style, followed by a dot (.) and the name of a CSS style property: [style.style-property].

```
<button [style.color]="isSpecial ? 'red': 'green'">Red</button>
<button [style.background-color]="canSave ? 'cyan': 'grey'" >Save</button>
```

Some style binding styles have a unit extension. The following example conditionally sets the font size in "em" and "%" units.

```
<button [style.font-size.em]="isSpecial ? 3 : 1" >Big</button>
<button [style.font-size.%]="!isSpecial ? 150 : 50" >Small</button>
```

Caltech | Center for Technology & Management Education   simplilearn

Assisted Practice

Class And Style Binding

Problem Statement:

Create an Angular application to demonstrate class and style binding.

Assisted Practice: Guidelines

Steps to demonstrate class and style binding:

1. Disable a button using attribute binding

2. Implement class binding

3. Implement style binding

4. Push the code to GitHub repositories

Event Binding

Event Binding

The previous bindings directives allow data to flow in one direction: from a component to an element.

Users don't just stare at the screen. They enter text into input boxes. They pick items from lists. They click on buttons. Such user actions may result in a flow of data in the opposite direction: from an element to a component.

The only way to know about a user action is to listen for certain events such as keystrokes, mouse movements, clicks, and touches. You declare your interest in user actions through Angular event binding.

Event binding syntax consists of a target event name within parentheses on the left of an equal sign, and a quoted template statement on the right.

```
<button (click)="onSave()">Save</button>
```

The below image explains the basic syntax of event binding, where **(click)** is the type of the event and **onSave()** is the method to be executed as the button is clicked.

Target Event

Below are different ways to implement event binding:

src/app/app.component.html

```html
<button (click)="onSave($event)">Save</button>
```

src/app/app.component.html

```html
<button on-click="onSave($event)">on-click Save</button>
```

src/app/app.component.html

```html
<h4>myClick is an event on the custom ClickDirective:</h4>
<button (myClick)="clickMessage=$event" clickable>click with myClick</button>
{{clickMessage}}
```

$event And Event Handler

The flow of an event binding is given below:

An event handler is set up when an event is raised that executes the template statements.

Specific operation related to the event is executed in the backend code.

Event handler carries the data throughout the process.

1

2

3

Target Event

The reasons to use $event and EventEmitter are:

Custom events in Angular are raised by directives using **EventEmitter**.

EventEmitter executes **EventEmitter.emit(payload)** method and passes your operation object in the payload.

This request is listened by parent directive through **$event.**

One can implement component binding with the help of event binding.

Caltech | Center for Technology & Management Education

simplilearn

Assisted Practice

Duration: 20 Min.

Problem Statement:

Create an Angular application to demonstrate custom event binding and interacting the UI with user inputs and operations.

Assisted Practice: Guidelines

Steps to demonstrate event binding:

1. Implement event binding

2. Push the code to GitHub repositories

Two-Way Binding

Two-Way Binding

It is recommended to display both a data property and its updates when the user makes changes.

On the element side, it takes a combination of setting a specific element property and listening for an element change event.

Angular offers a special two-way data binding syntax for this purpose, [(x)]. The [(x)] syntax combines the brackets of property binding, [x], with the parenthesis of event binding, (x).

The two-way binding syntax is really just syntactic sugar for a property binding and an event binding. Angular desugars the SizerComponent binding into this:

```
<my-sizer [size]="fontSizePx" (sizeChange)="fontSizePx=$event"></my-sizer>
```

Component-Level Interactions

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

- Data Sharing Between Angular Components:
1. Parent to Child: via Input
2. Child to Parent: via Output() and EventEmitter
3. Child to Parent: via ViewChild
4. Unrelated Components: via a Service

- This is an important concept that one should practice as it facilitates a large amount of Angular functionalities.

Caltech | Center for Technology & Management Education

simplilearn
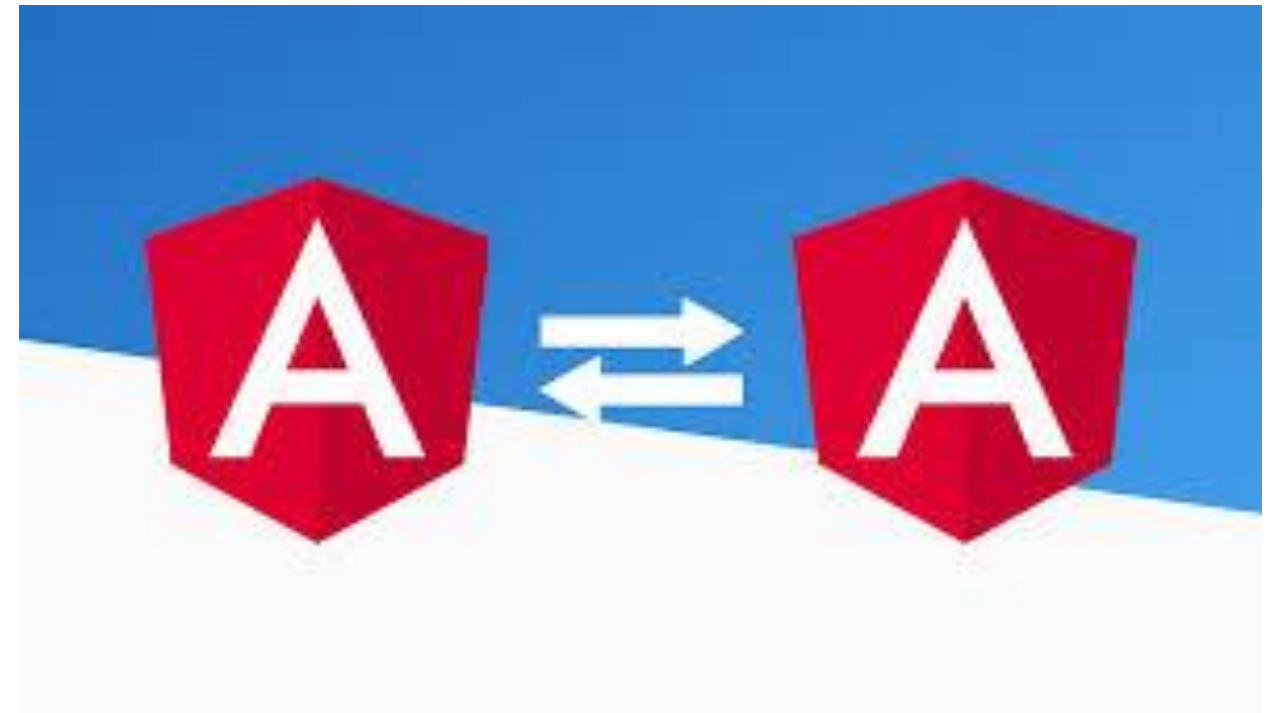
@Input and @Output

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

Example:

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

export class addTask{

@Input() item_name
@Output() onNameChanger = new
EventEmitter()

}
```

- The **@Input** decorator indicates that a data will be received from a component. The received data will be stored in **item_name**.

- The **@Output** decorator indicates that a data will be sent to another component via **onNameChanger** property.

EventEmitter and ViewChild

• EventEmitter is something which has the capability to propagate the event from a child component to a parent component.

•EventEmitter and @Output complement each other to navigate the requests from child to parent component.

•The major reason why ViewChild is needed to transfer data from child to parent component is that it gives extra control to parent component to access child events.

•Use the below code to import ViewChild:
import { Component, OnInit,ViewChild } from '@angular/core';

•Supply the child component as view child as done in the below lines:
@ViewChild(EcchildComponent)
private counterComponent: EcchildComponent;

Caltech | Center for Technology & Management Education

simplilearn

Assisted Practice

Two-Way Binding

Problem Statement:

You are given a project to establish communication between parent and child components.

Assisted Practice: Guidelines
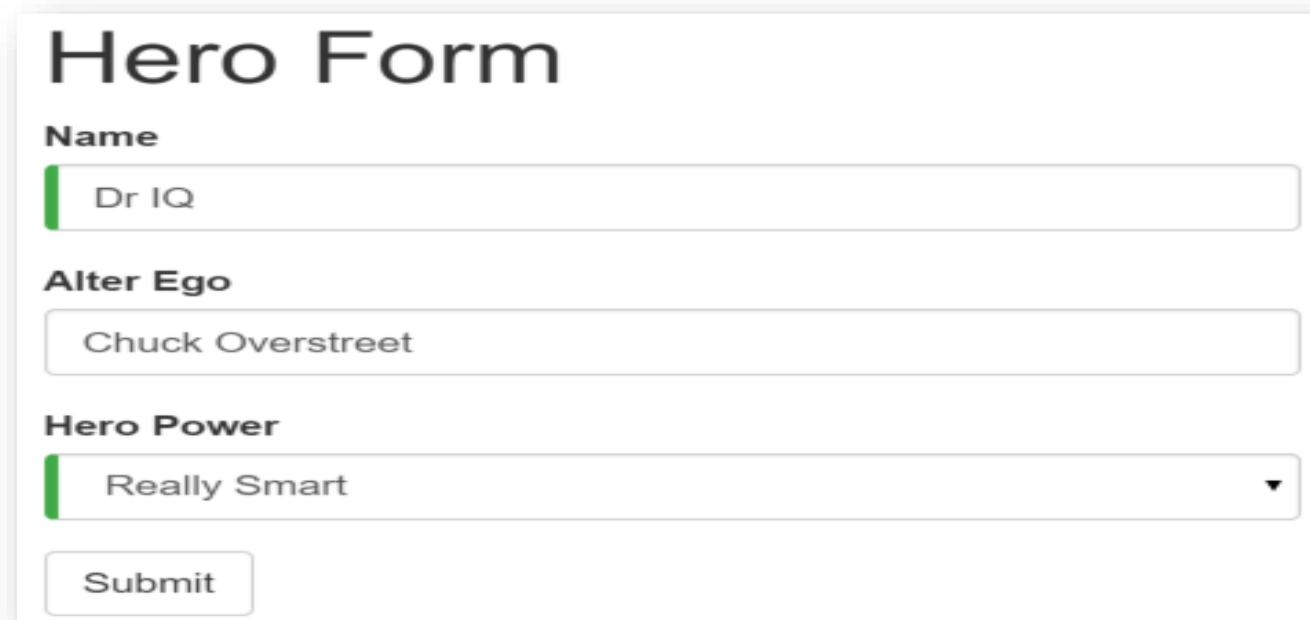
Steps to demonstrate two-way binding:

1. Configure the Angular application

2. Create parent and child components

3. Transfer data from a parent to the child component and vice versa

4. Push the code to GitHub repositories

Forms and Validations

Angular Form Benefits

A form creates a cohesive, effective, and compelling data entry experience.

An Angular form coordinates a set of data-bound user controls, tracks changes, validates input,

and presents errors.

Forms are probably the most crucial aspect of your web application.

You can often get events by clicking on links or from the movement of the mouse, but it's through

forms that you get the majority of your rich data input from users.

Angular Form Tools

Angular has these tools to help with all the challenges:

- **FormControls**
  - Encapsulate the inputs in your forms and give objects to them to work with

- **Validators**
  - Give you the ability to validate inputs any way you like

- **Observers**
  - Allow you to watch your form for changes and respond accordingly

Caltech | Center for Technology & Management Education

simpl**i**learn

Types of Angular Forms

Reactive and template-driven forms process and manage form data differently.
Each offers different advantages.
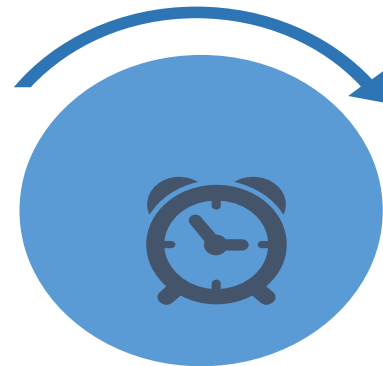
Reactive Forms
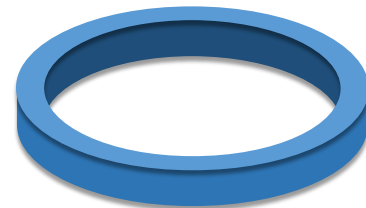
Reactive forms are more robust, scalable, reusable, and testable

Template-driven forms
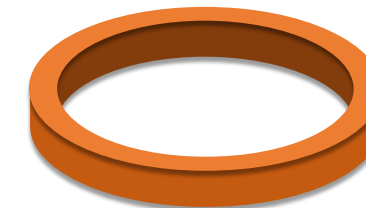
Template-driven forms are used for adding a basic forms in your app, such as an email list signup form

Caltech | Center for Technology & Management Education

simplilearn

Template-Driven Approach

Angular uses built-in directives to build forms such as **ngModel**, **ngModelGroup**, and **ngForm** available in **FormsModule** module.

With template-driven forms, you can essentially leave a component class empty until you need to read or write values (such as submitting and setting initial or future data).

```
training@localhost:~

Import {component} from '@angular/core' 1;

@Component ({
     selector: 'signup - form',
     template: '
          <form novalidate>…</form>
     '
})
export class signupFormComponent {
     constructor()_ {}
}
```

Model-Driven Approach

A lot of features and concerns were improved in Angular, such as communication between components, code reuse, separation of concerns, and unit tests.

To promote some of those benefits in the context of forms, Angular uses a model-driven or reactive technique of development.

Model-Driven Approach: Front-End

```
training@localhost:~

<form [ngFormModel]='form' (ngSubmit)="onSubmit()">
     <div class="form-group" [ngclass]="{'has-error': !name.valid}">
          <label for="name">Name</label>
          <input type="text" class="form-control" id="name"
                 ngcontrol='name'
                 #name="ngForm">
          <p *ngIf="!name.valid" class="help-block">
               Name is required
          </p>
     </div>
     <button type="submit" class="btn btn-primary"
          [disabled]="!form.valid">Add user</button>
</form>
```

Model-Driven Approach: Component

```
import {Component} from 'angular2/core';
import {Control, ControlGroup, Validators} from 'angular2/common';

@Component({
    selector: 'form',
    templateUrl: './dev/shared/form.component.html'
})
Export class FormComponent {
    form = new ControlGroup({
        name: new Control('', Validators.required);
    });

    onSubmit(){
        console.log(this.form.value);
    }
}
```

Angular Validation

Angular comes with two types of validation:

Built-in form validation

Custom form validation

Built-In Form Validation

Angular provides three out-of-the-box validators that can either be applied using the control class or using HTML properties.

For example: *<input required>* will automatically apply the required validator

- Required
- minLength
- maxLength

Apply them to the Control using the second parameter.

```
this.name = new Control('', Validators.minLength(4));
```

Custom Form Validation

You can also write your own custom validators. Here is an example of a validator that checks if the first character is not a number:

```
Interface ValidationResult {
[key:string] :boolean;
 }

 class UsernameValidator {
 static startWithNumber (control: Control) :ValidationResult {
      if ( control.value ! "" && !isNaN
(control.valuel.charAt(0)) ) {
      }
      return null;

 }
 }
```

Assisted Practice

Form Validations

Problem Statement:

You are given a project to create a reactive-driven form with appropriate validators.

Assisted Practice: Guidelines

Steps to demonstrate form validations:

1. Configure the Angular application

2. Build a front-end form for sign-up

3. Complete the functionality by implementing the validators

4. Push the code to GitHub repositories

Caltech | Center for Technology & Management Education

simplilearn

Services and Injectables

Features of Dependency Injection
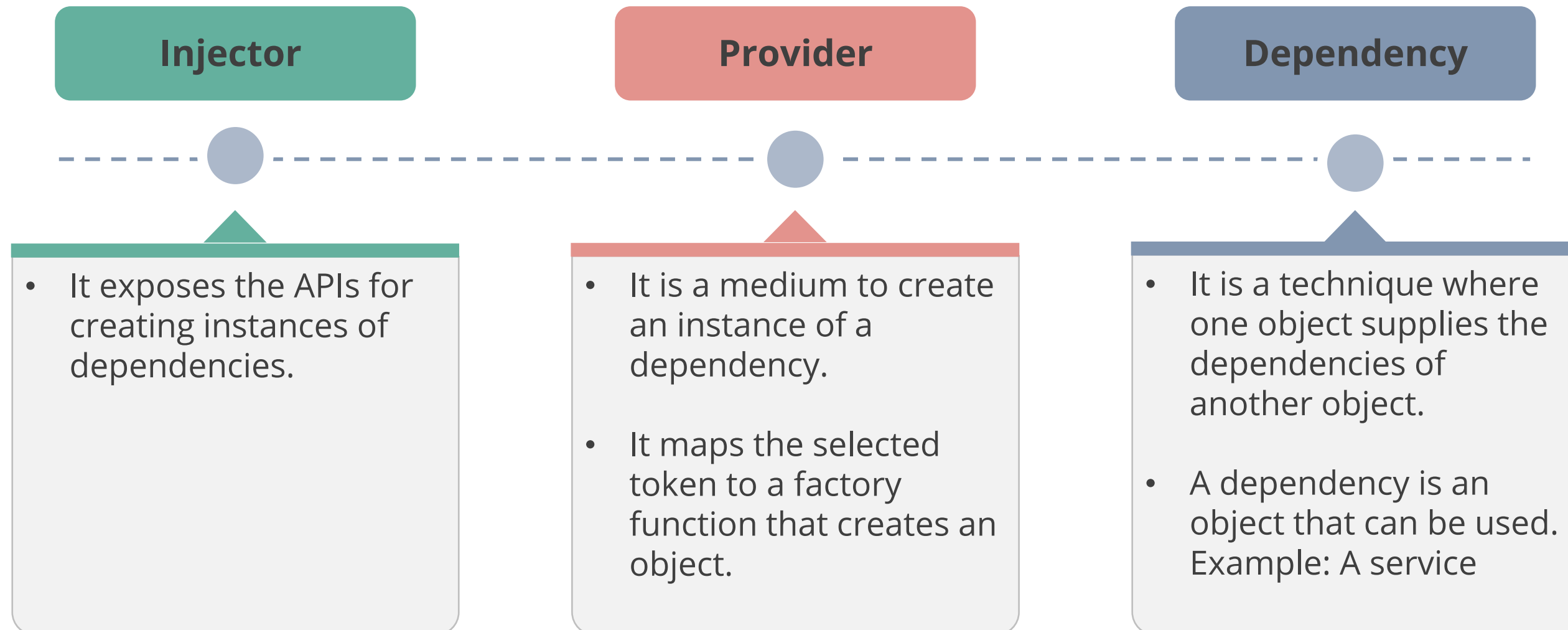
- An Injector provides a mechanism to create a service instance, and to create the instance, it needs a provider.

- Provider is a medium for creating a service.

- Providers can be registered along with the injectors.

Container

Client Object

Give Me an Iclub

Dependency Resolver

Have a Putter Club

Iclub Implementations:
- Wood Club
- Iron Club
- Wedge Club
- Putter Club

Importance of Dependency Injection API

Angular has its own dependency injection framework that can also be used as a standalone module by other applications and frameworks.

DI in Angular consists of:

Injector	Provider	Dependency
• It exposes the APIs for creating instances of dependencies.	• It is a medium to create an instance of a dependency. • It maps the selected token to a factory function that creates an object.	• It is a technique where one object supplies the dependencies of another object. • A dependency is an object that can be used. Example: A service

Creation of Service

Import the Angular Injectable function and apply that function as an @Injectable() decorator.

```
> training@localhost:~

import { Injectable } from '@angular/core';

@Injectable()
export class HeroService {
}
```

TypeScript looks at the @Injectable() decorator and emits metadata about the service.

Metadata in Angular may need to inject other dependencies into this service.

Injecting a Service

Step 1: Import the service you want to use so that you can refer to it in the code.

```
training@localhost:~

import { HeroService } from './hero.service';
```

Injecting a Service

Step 2: Inject the HeroService.

Note: You can create a new instance of the HeroService with *new*:

```
training@localhost
:~

heroService = new HeroService(); // don't do this
```

The one line of code (with new) is replaced with two lines:

- Add a constructor that also defines a private property.

- Add to the component provider's metadata.

```
training@localhost:
~

constructor(private heroService: HeroService) { }
```

Injecting a Service

Step 3: Add the providers array property to the bottom of the component metadata in the @Component call to teach the injector the process to make a HeroService.

```
training@localhost:~

providers: [HeroService]
```

Directives

What Are Directives?

Directives are used to separate the reusable functions and features of Angular. This is an angular decorator. We use this decorator to define a directive.

Note Unlike components, directives don't have HTML templates.

They add behavior to an existing DOM element.

Directives: Example

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

Angular

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

Types of Directives

There are three types of directives:

Components:	Structural Directives:	Attributes Directives:
The component directive includes directives with the template.	This directive changes the behavior of a component or an element by affecting how the template is rendered.	This type of directive changes the behavior of a calling component or an element but doesn't affect the template.

Component: Directive

Component Directive

The naming convention for components is cname.component.ts.
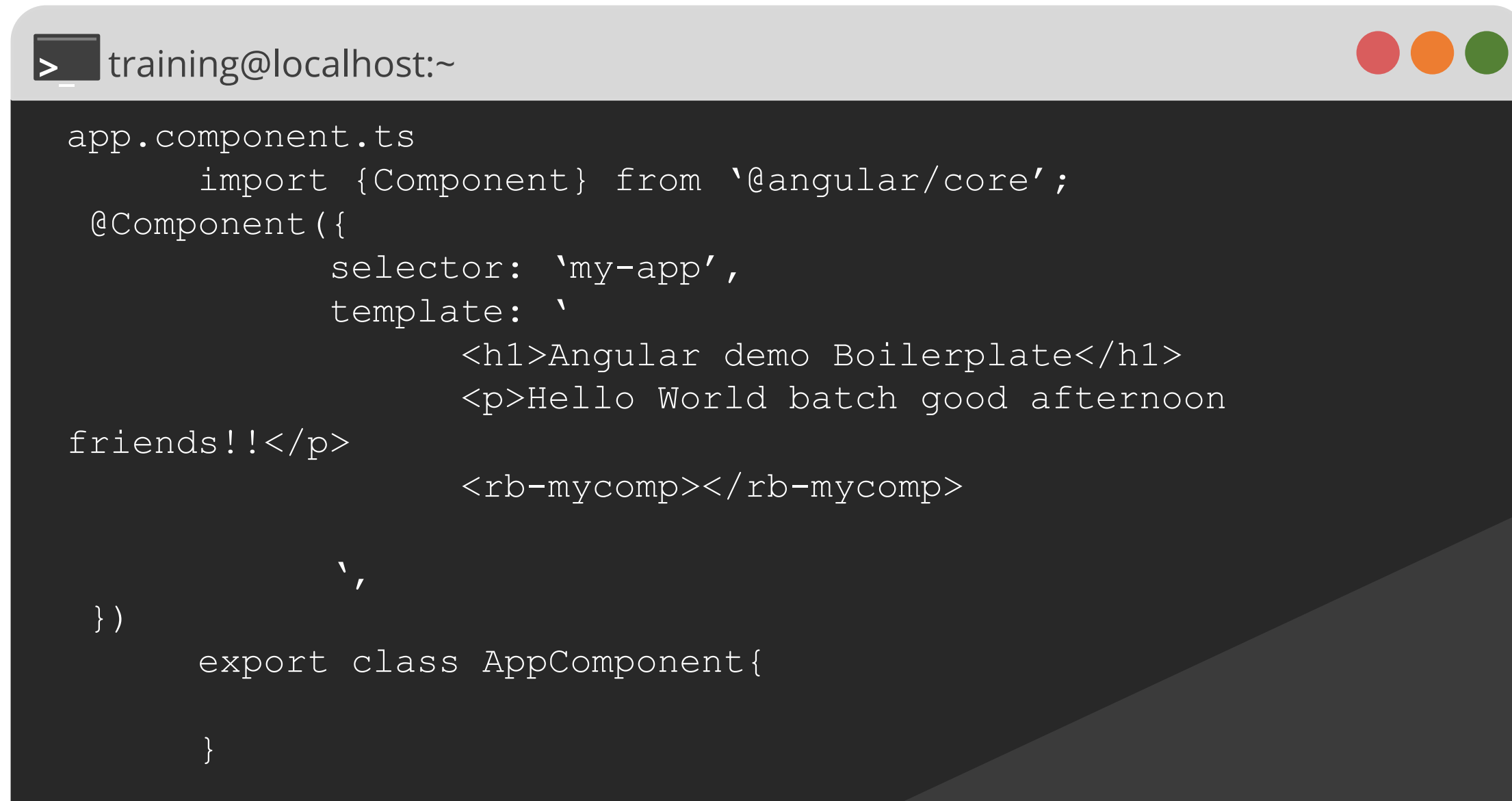
An angular application cannot be created without this.

This type of directive includes DOM elements.

It requires a template along with its attached behavior.

Caltech | Center for Technology & Management Education

simplilearn

Component Directive: Example

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{

    }
```

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**.

```
<divcolorChange ><div>
```

```
<pcolorChange ></p>
```

Attribute Directive

There are predefined attribute directives.

The import statement specifies symbols from the Angular core.

Directive	ElementRef	Input

Provides the functionality of the @Directive decorator

Injects elements into the directive's constructor so that the code can access the DOM element

Allows data to flow from the binding expression into the directive
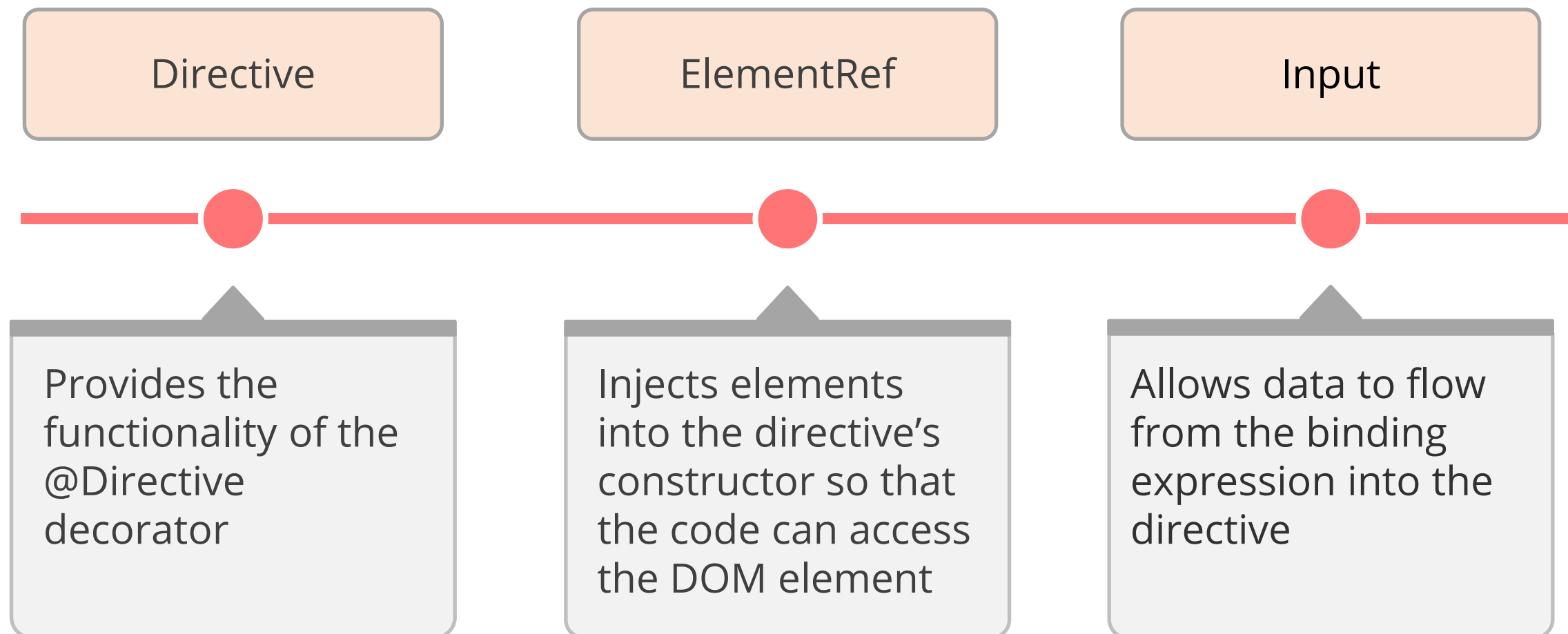
Attribute Directive: Example

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';
}
}
```

ngStyle Attribute Directive

- Creating dynamic styles in web applications can be difficult.

- In Angular, 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>
```

ngClass Attribute Directive

- The ngClass directive changes the class attribute that is bound to the component or attached element.

- It is an Angular 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.

- It is applied to a DOM element and then bound to an expression.

- It evaluates the expression and changes the class attribute of the element to which it is applied.
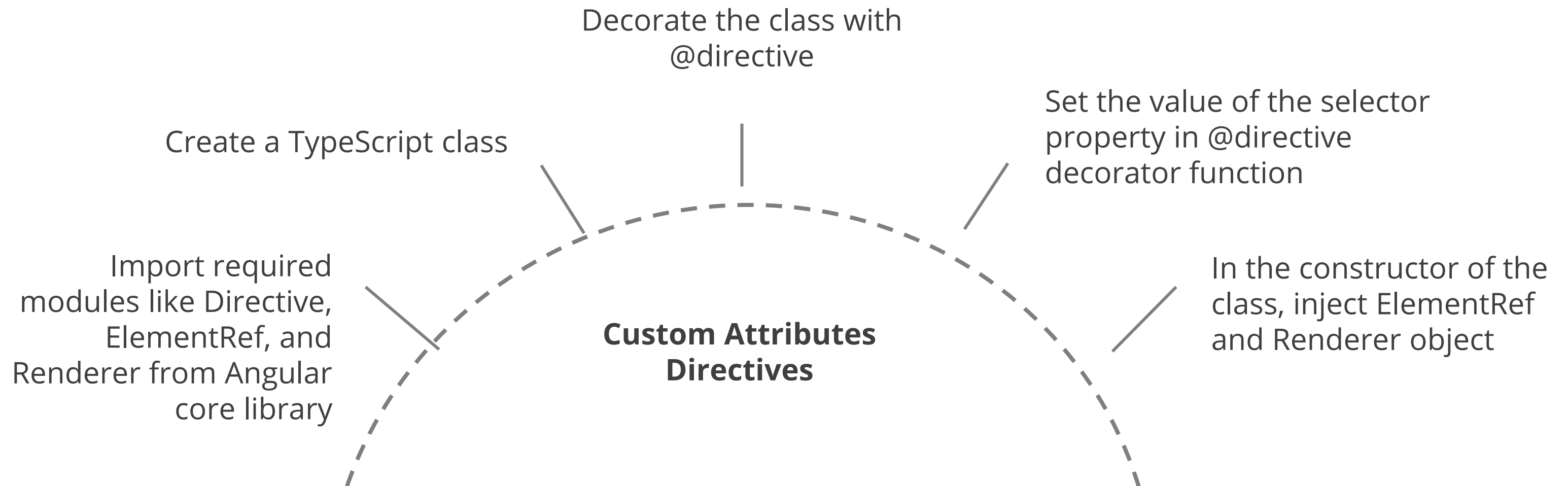
ngClass Attribute Directive

- It 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.

- Expressions of ngClass can be evaluated using three options: String, Array, and Object.
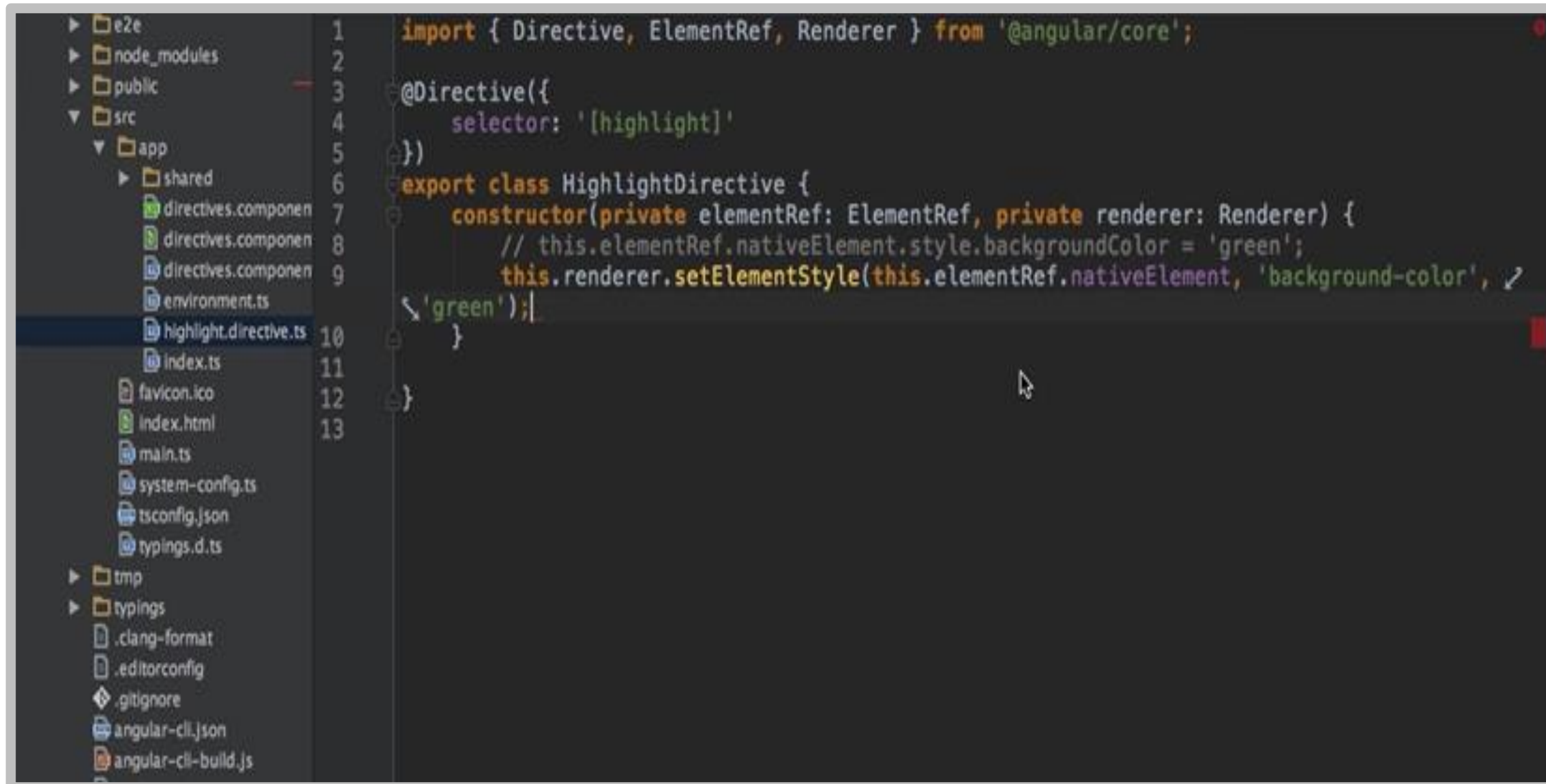
Custom Directives

To create an attribute directive, you need to create a class and decorate it with **@directive** decorators.

There are few important points to remember:

Decorate the class with @directive

Create a TypeScript class

Set the value of the selector property in @directive decorator function

Import required modules like Directive, ElementRef, and Renderer from Angular core library

Custom Attributes Directives

In the constructor of the class, inject ElementRef and Renderer object

Caltech | Center for Technology & Management Education   simplilearn

Creating Custom Directives
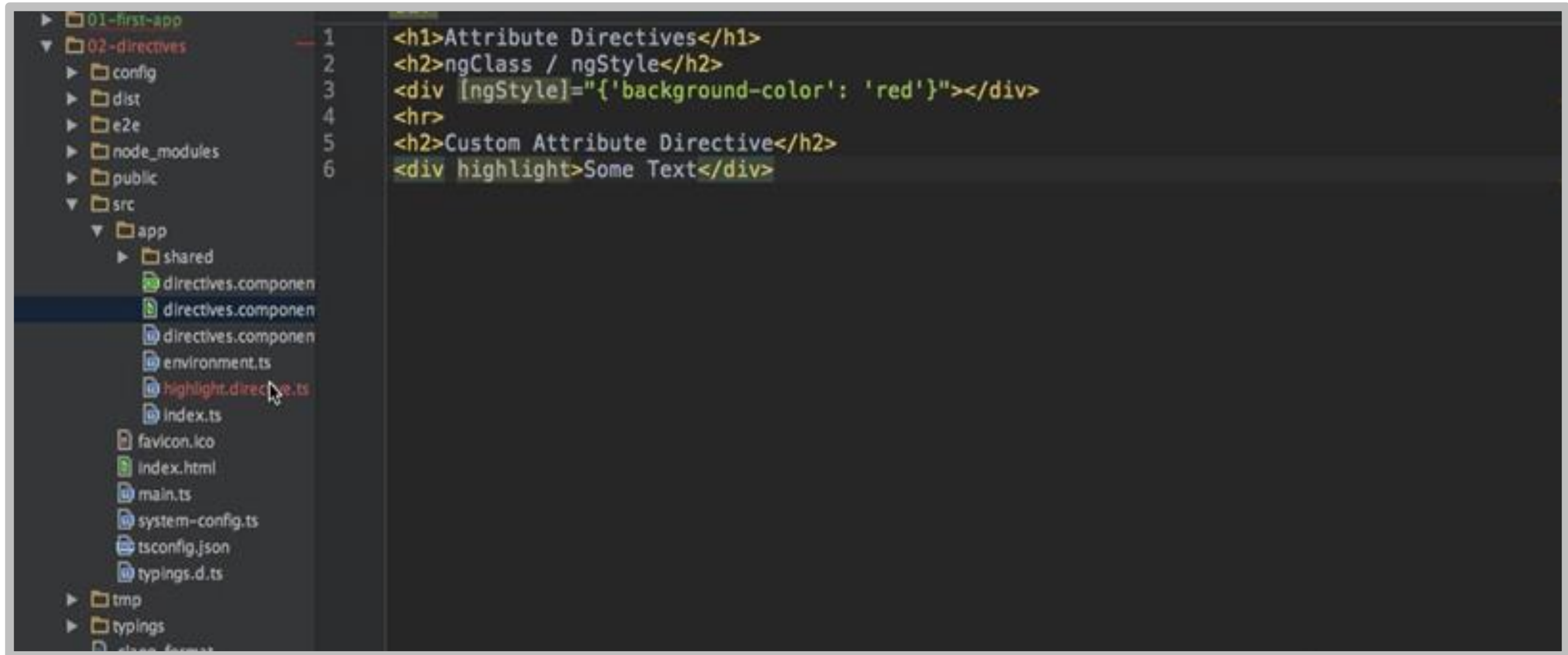
This class is created by overriding ElementRef and Renderer.

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

@Directive({
    selector: '[highlight]'
})
export class HighlightDirective {
    constructor(private elementRef: ElementRef, private renderer: Renderer) {
        // this.elementRef.nativeElement.style.backgroundColor = 'green';
        this.renderer.setElementStyle(this.elementRef.nativeElement, 'background-color',
'green');
    }
}
```

Creating Custom Directives

This method **highlights** a directive in a template.

Structural Directives

- 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 Directives

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

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

Angular has a few **built-in** structural directives:

ngIf

ngSwitch

ngFor

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.

- Browser developer tools can be used to inspect a 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   }
```

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.

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.

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>
```

Assisted Practice

Duration: 20 Min.

Problem Statement:

You are given a project to create a reactive-driven form with appropriate validators.

Assisted Practice: Guidelines

Steps to demonstrate directives and injectables:

1. Create a directive

2. Declare the directive

3. Add the directive as a property

4. Push the code to GitHub repositories

Pipes

What Are Pipes?

- With the data binding of Angular, you can just bind an element property to a class property and display data easily. But, sometimes, the data is not in an appropriate format to display. Therefore, we use pipes.

- Before pipes are displayed, they transform bound properties.

- Pipe is a way to write display-value transformations that you can declare in your HTML, as you can alter the property value to make them more user-friendly or more locale-appropriate.

Uses of Pipes

Pipes don't provide any additional features, however:

- Pipe is a good way to deal with functions and logics in templates

- Pipes make your code structured and clear

- A pipe takes in data as input and transforms it into the desired output

Uses of Pipes

- Built-in pipes can be used to format values such as date, number, decimal, percentage, currency, uppercase, and lowercase.

- A few pipes can be used for working with objects.

  Example:

  JSONPipe displays the content of an object as JSON string.

  SlicePipe selects a specific subset of elements from a list.

@angular/common

AsyncPipe	COMMON_PIPES	CurrencyPipe
DatePipe	DecimalPipe	I18nPluralPipe
I18nSelectPipe	JsonPipe	LowerCasePipe
PercentPipe	ReplacePipe	SlicePipe
UpperCasePipe		

@angular/core

PLATFORM_PIPES	Pipe	PipeMetadata
PipeMetadataFactory	PipeTransform	

Caltech | Center for Technology & Management Education   simplilearn
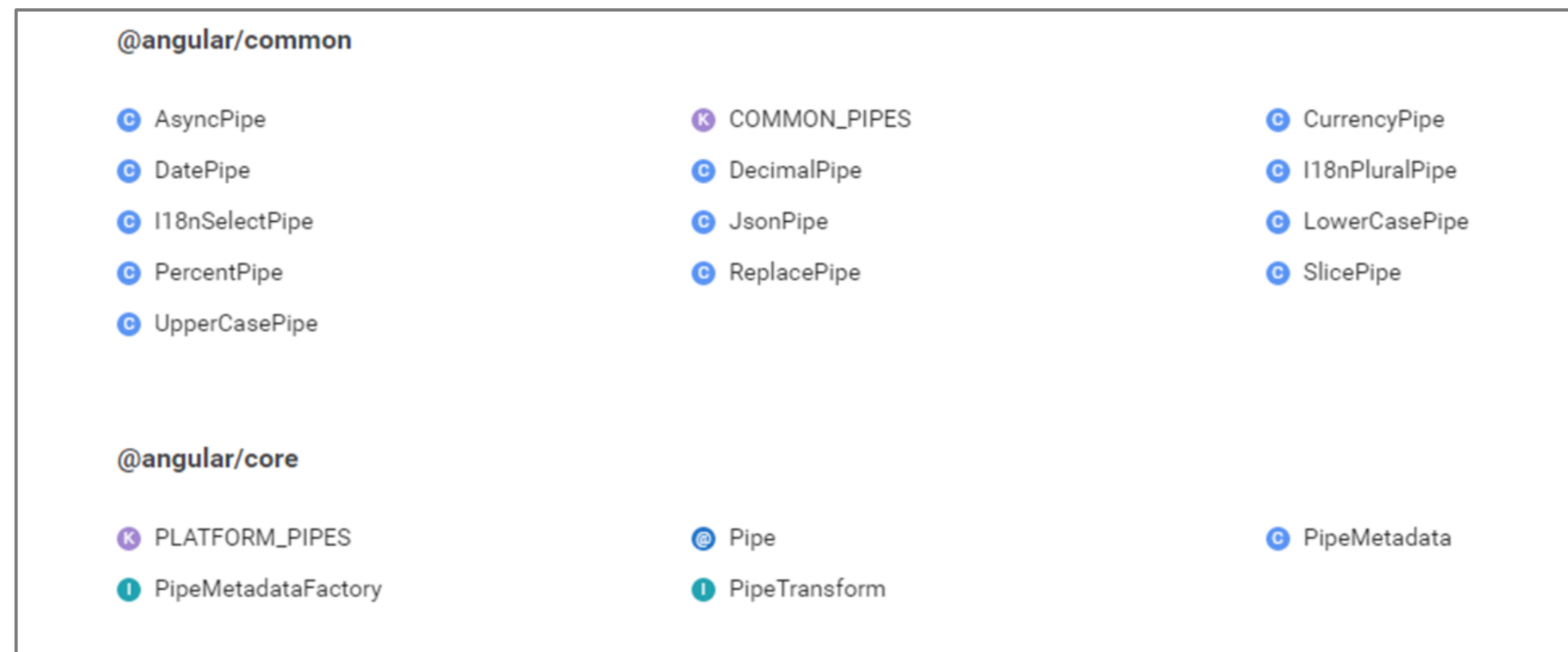
Types of Pipes
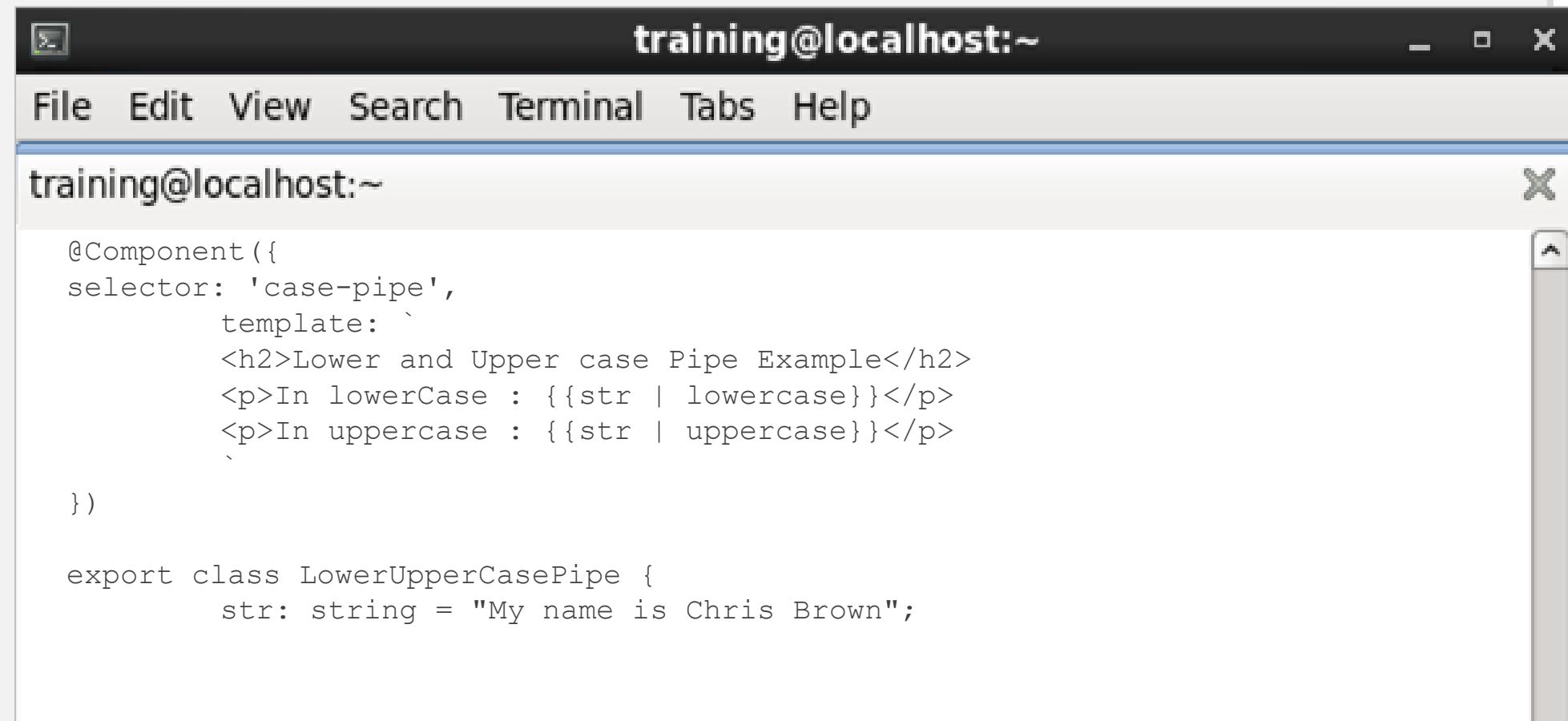
- UpperCase and LowerCase pipes change the case of the characters.

- Both these pipes do not accept any parameters.

UpperCase

LowerCase

Decimal

Currency

Date

JSON

```
training@localhost:~

File  Edit  View  Search  Terminal  Tabs  Help

training@localhost:~

@Component({
selector: 'case-pipe',
        template: `
        <h2>Lower and Upper case Pipe Example</h2>
        <p>In lowerCase : {{str | lowercase}}</p>
        <p>In uppercase : {{str | uppercase}}</p>
        `
})

export class LowerUpperCasePipe {
        str: string = "My name is Chris Brown";
```

Types of Pipes

- Decimal pipes are used to create numbers.
- Decimal pipes provide the option to select a minimum and maximum number length after the decimal point and fix the number of places before the decimal point.

UpperCase

LowerCase

Decimal

Currency

Date

JSON

training@localhost:~

File   Edit   View   Search   Terminal   Tabs   Help

training@localhost:~

```
@Component({
selector: 'decimal-pipe',
})
@View({
        template: `
        <h2>Decimal Pipe Example</h2>
        <p>pi (no formatting): {{pi}}</p>
        <p>pi (.5-5): {{pi | number:'.5-5'}}</p>
        <p>pi (2.10-10): {{pi | number:'2.10-10'}}</p>
        <p>pi (.3-3): {{pi | number:'.3-3'}}</p>


        `
})
export class DecimalPipe {
        pi: number = 3.1415927;
}
```
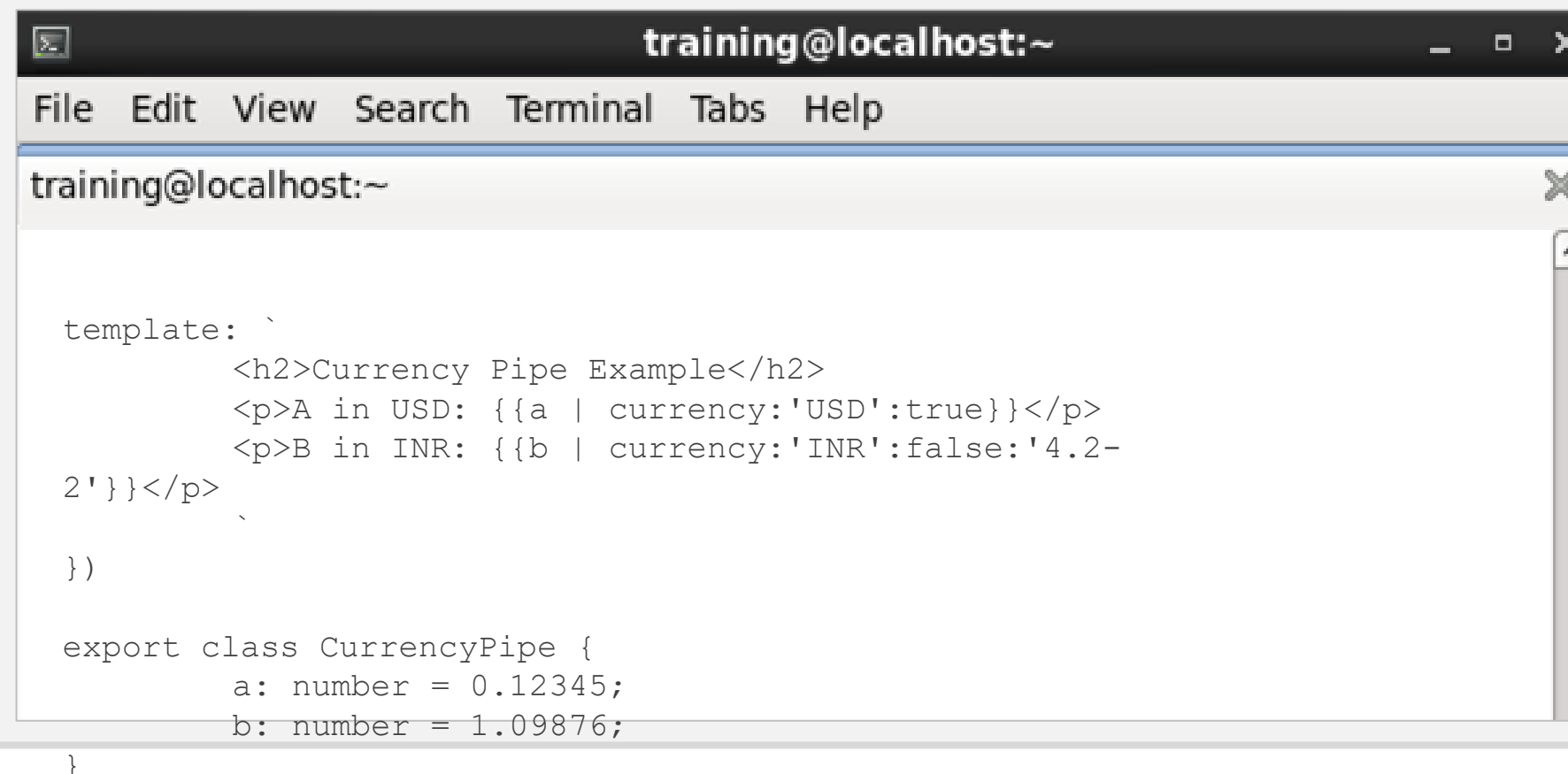
Types of Pipes

- Currency pipe helps you format and use symbols.

  Example: Per ISO 4217 currency code, **EUR** stands for euro and **USD** stands for the US dollar

- It takes *symbolDisplay* with a default value of false as the second parameter.

- The third parameter for the pipe is *digitInfo* and it works as a *DecimalPipe.*

UpperCase

LowerCase

Decimal

Currency

Date

JSON

```
                              training@localhost:~              _  □  ✕

File   Edit   View   Search   Terminal   Tabs   Help

training@localhost:~                                                    ✕


template: `
        <h2>Currency Pipe Example</h2>
        <p>A in USD: {{a | currency:'USD':true}}</p>
        <p>B in INR: {{b | currency:'INR':false:'4.2-
2'}}</p>
        `

})

export class CurrencyPipe {
        a: number = 0.12345;
        b: number = 1.09876;
}
```
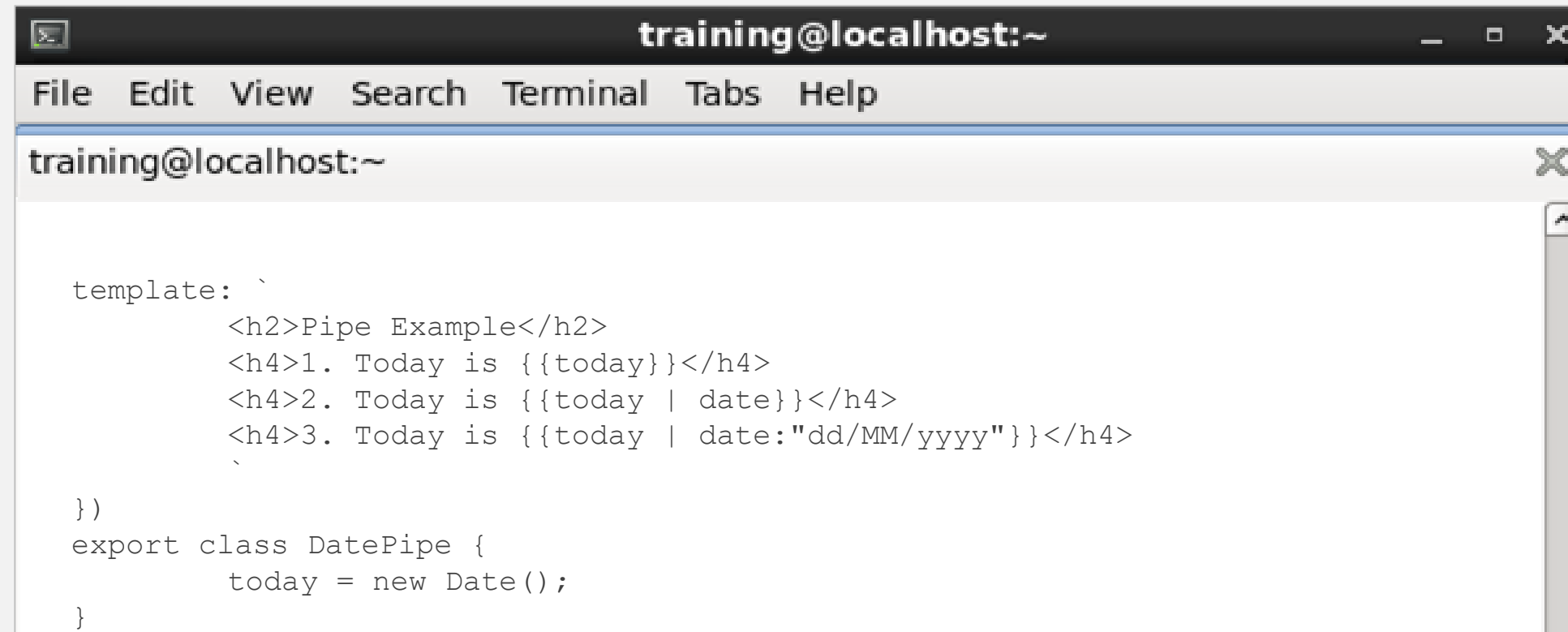
Types of Pipes

UpperCase

LowerCase

Decimal

Currency

Date

JSON

- With Date pipes, you can display date in a specific format.

- You can access the various predefined date formats.

  Examples: **medium**, **fullDate**

- You can also create a custom format. For example, use **yy** for year, **MM** for month, **dd** for date, **hh** for hour, **mm** for minutes, and **ss** for seconds.

```
training@localhost:~
File   Edit   View   Search   Terminal   Tabs   Help

training@localhost:~


template: `
        <h2>Pipe Example</h2>
        <h4>1. Today is {{today}}</h4>
        <h4>2. Today is {{today | date}}</h4>
        <h4>3. Today is {{today | date:"dd/MM/yyyy"}}</h4>
        `
})
export class DatePipe {
        today = new Date();
}
```
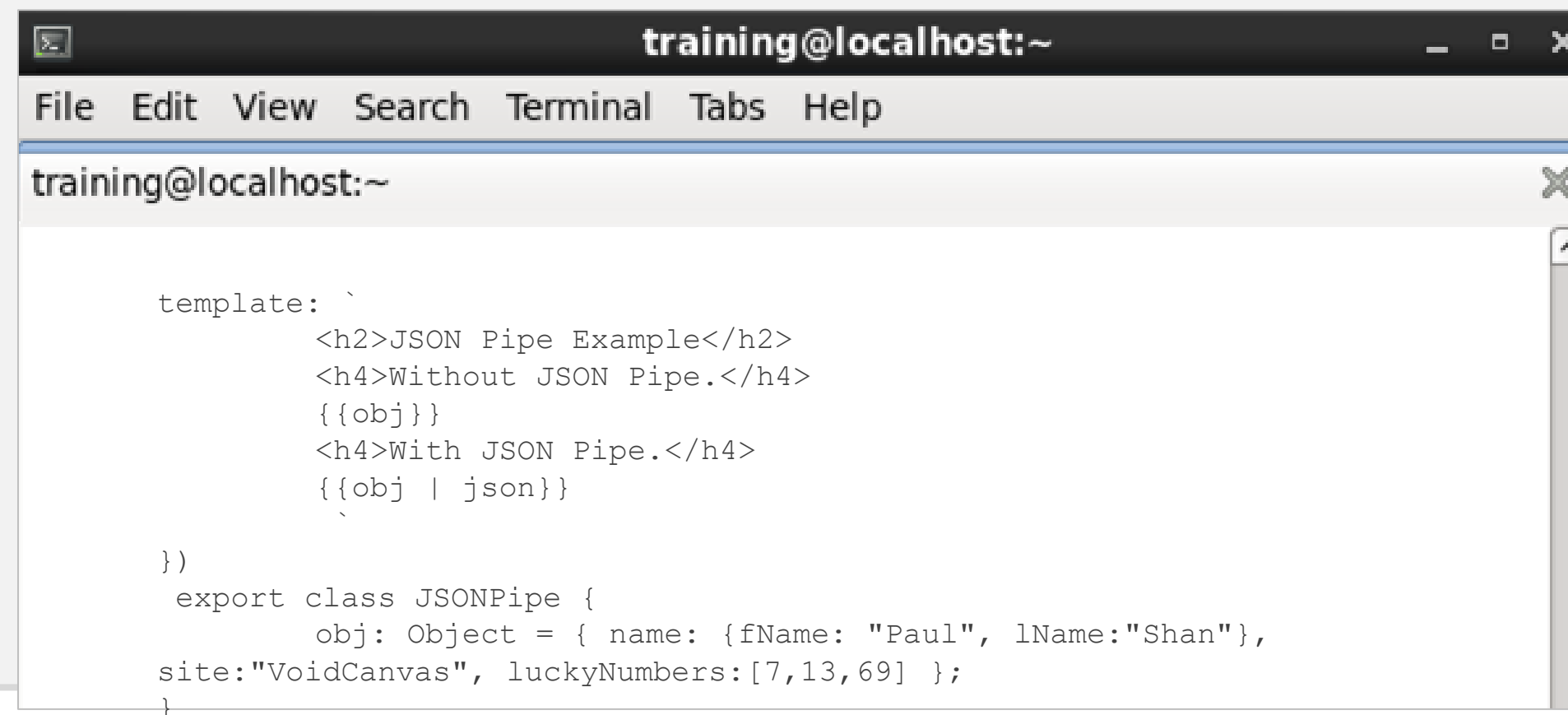
Types of Pipes

UpperCase

LowerCase

Decimal

Currency

Date

JSON

- You use double curly braces to print a value, but these cannot print the complete object.

- You can use *JSONPipe t*o print the JSON object.

```
training@localhost:~
File   Edit   View   Search   Terminal   Tabs   Help

training@localhost:~                                              ✕

        template: `
                <h2>JSON Pipe Example</h2>
                <h4>Without JSON Pipe.</h4>
                {{obj}}
                <h4>With JSON Pipe.</h4>
                {{obj | json}}
                `
        })
         export class JSONPipe {
                obj: Object = { name: {fName: "Paul", lName:"Shan"},
        site:"VoidCanvas", luckyNumbers:[7,13,69] };
        }
```

Custom Pipes

- To create a custom pipe, you give it a name and a transform function.
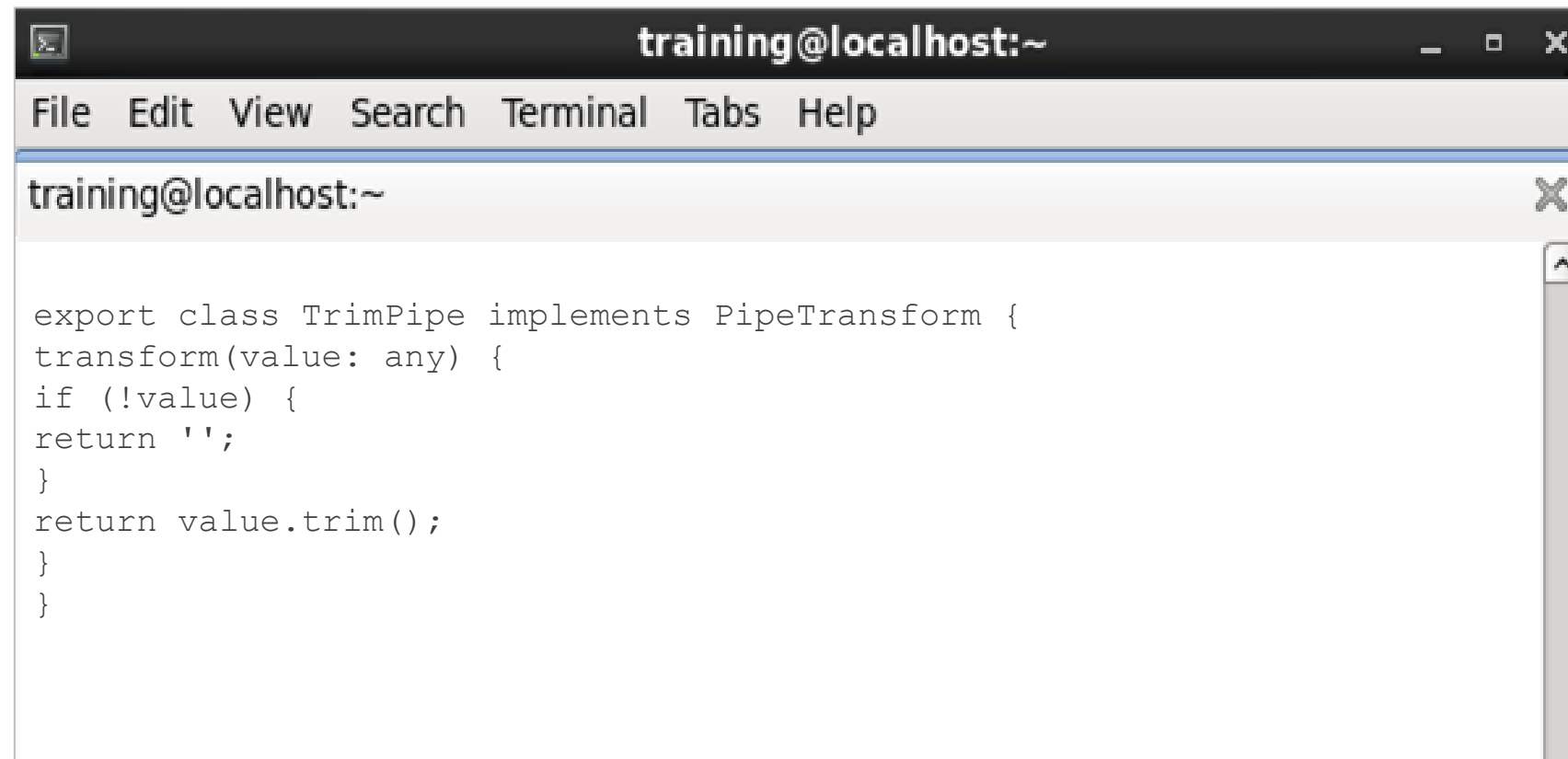
To create a pipe you need to:

- Create a new .ts file named trim.pipe.ts in demo/pipes folder.

- Import the module Pipe and PipeTransform from **@angular/core**. It tells Angular that this is the pipe which is imported from the core Angular library, by applying the **@Pipe** decorator.

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

Caltech | Center for Technology & Management Education   simplilearn

Structural Directives

Creating the pipe class implements the PipeTransform interface's transform method.

The transform method takes an input value and optional parameters and returns the transformed value.

```
training@localhost:~

File   Edit   View   Search   Terminal   Tabs   Help

training@localhost:~

export class TrimPipe implements PipeTransform {
transform(value: any) {
if (!value) {
return '';
}
return value.trim();
}
}
```

Assisted Practice

Pipes

Problem Statement:

You are asked to implement pipes in Angular.

Assisted Practice: Guidelines

Steps to demonstrate working of pipes:

1. Use built-in Angular pipes

2. Use custom Angular pipes
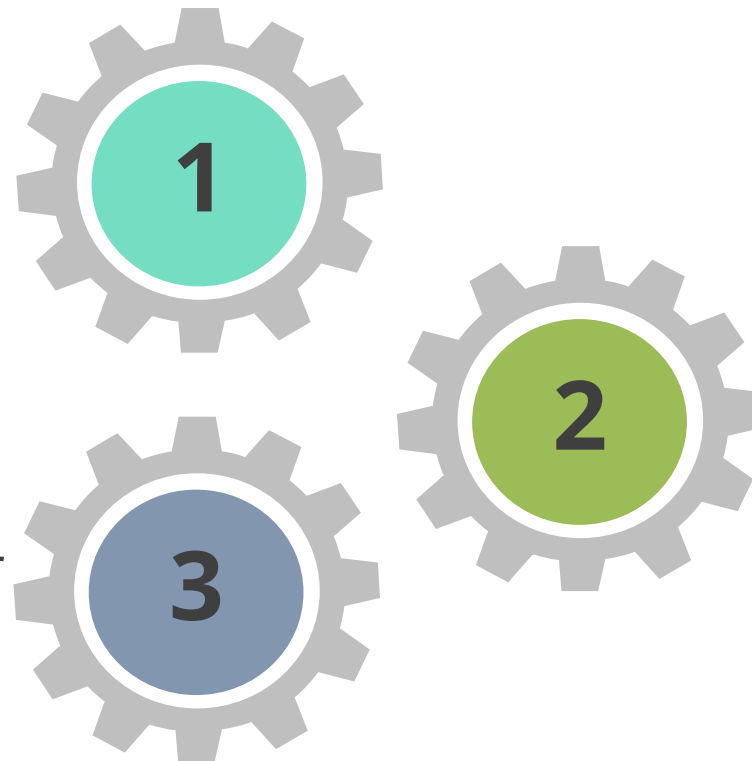
3. Push the code to GitHub repositories

Routing Mechanisms

Router Concepts

Main components used to configure routing:

Routers define the routes that your application supports

router-outlet is where the page content goes

routerLink directive is used for navigation

1

2

3

Caltech | Center for Technology & Management Education

simplilearn

Defining Routes
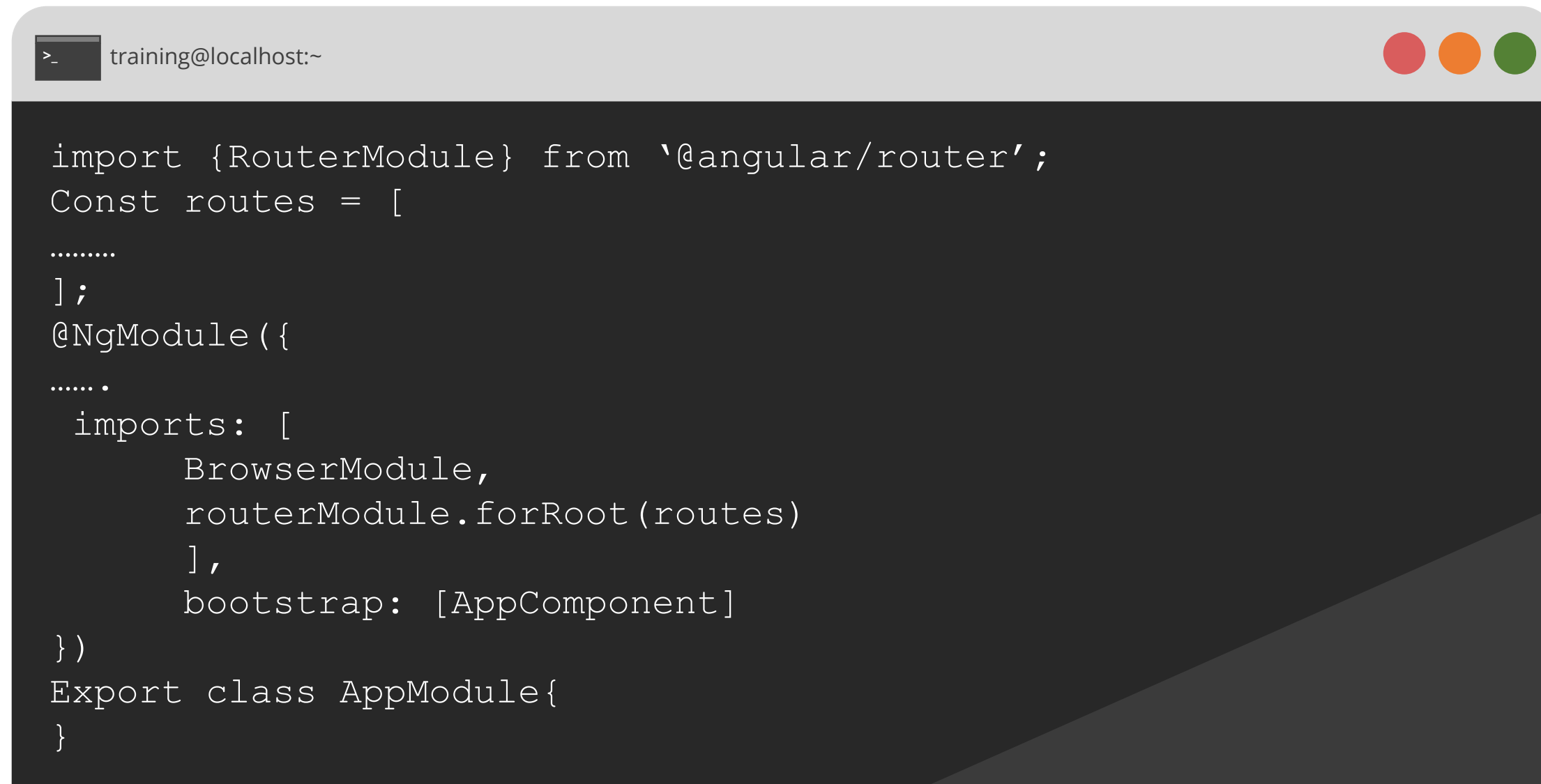
This is a mapping file where you can map components to their target URL:

```
training@localhost
:~

Const routes = [
        {path: '', redirectTo: 'home'}, //Index router
        {path: 'home', component: HomeComponent},
        {path: 'guards', component: GuardsComponent},
        {path: 'resolver', component: ResolverComponent},
        {path: 'preview', component: PreviewComponent},
        {path: '**', component: E404Component //Fallback
router
];
```

Registering Root Router

This is where the Root Router gets registered:

```
training@localhost:~

import {RouterModule} from '@angular/router';
Const routes = [
.........
];
@NgModule({
.......
 imports: [
      BrowserModule,
      routerModule.forRoot(routes)
      ],
      bootstrap: [AppComponent]
})
Export class AppModule{
}
```

Navigating to a Link

routerLink Directive:

- routerLink directive is used to bind a clickable HTML element to a route.

- Click on an element with a routerLink directive that is bound to a string or a link array which triggers a navigation.

```
>_    training@localhost:~

<a [routerLink]="['/home']">Home</a>
<a [routerLink]="['/guards']">Guards</a>
<a [routerLink]="['/resolver']">Resolver</a>
<a [routerLink]="['/preview']">Preview</a>
```

Rendering the Page

Router Outlet Directive:

- Marks where the router displays a view

- Acts as a placeholder that Angular dynamically fills based on the current router state

```
>_   training@localhost:~

<router-outlet></router-outlet>
```

Child Routes

Sometimes, when routes are accessible and viewed only within other routes, it is more relevant to create them as child routes.

Example :

- There are **tabbed navigation** sections in the product details page that present the product overview by default.

- When the user clicks on the **Technical Specs** tab, the section shows the specs instead.

Declaration of Child Route

Example :

- In case the user clicks on the link of ID 3, show the user details page with the **overview**: localhost:3000/user-details/3/overview

- When the user clicks on **user skills**: localhost:3000/user-details/3/skills

- Here, overview and skills are child routes of user-details/:id. They are reachable only within user details

Declaration of Child Route

Example :

- In case the user clicks on the link of ID 3, show the user details page with the **overview**: localhost:3000/user-details/3/overview

- When the user clicks on **user skills**: localhost:3000/user-details/3/skills

- Here, overview and skills are child routes of user-details/:id. They are reachable only within user details

Angular HTTP and Observables

Later, callbacks were considered a bad practice, and **promises** had to be used.

Earlier, **callbacks** were preferred for handling XHR.

Now, **observables,** rather than promises, can be used to handle HTTP requests.

Caltech | Center for Technology & Management Education

simplilearn
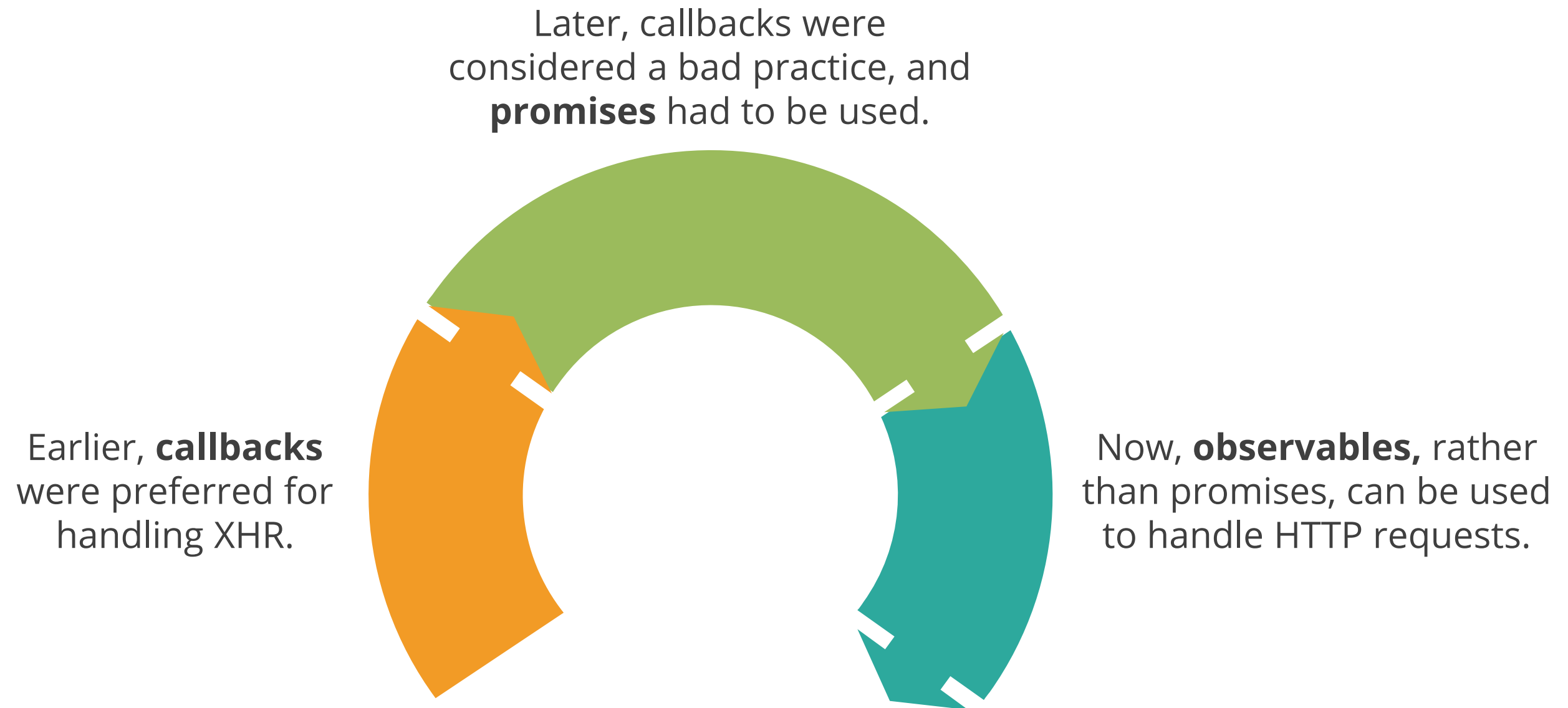
Using HTTP in Angular

HTTP is a separate model in Angular available under the **@angular/common/http** package.
To use HTTP in Angular:

01 Import the @angular/common/http package and inject it into the component or service

02 Use **get** method to send an HTTP request and subscribe to the response asynchronously

03 When the response arrives, map it to the desired object and display the result

Setting up HTTP in Angular

To use the new HTTP module in the components, you have to import HTTP. Then, inject it via Dependency Injection in the constructor.

Here is an example:

```
training@localhost:~

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class MyComponent {
  constructor(private http: Http) { }
}
```

Assisted Practice

Routing Mechanisms

Problem Statement:
You are given a project to implement the routing mechanism in your Angular application.

Assisted Practice: Guidelines

Steps to demonstrate routing mechanisms:

1. Configure the Angular application

2. Change the source code to implement the routers

3. Complete the functionality by implementing validators

4. Push the code to GitHub repositories

Key Takeaways

⦿ Angular was created by Google. Angular provides a way to quickly develop modern single-page web applications.

⦿ Angular directives are used to extend the power of HTML by giving a new stantx to it.

⦿ Pipes are the way to write display-value transformations that you can declare in your HTML.

myPortfolio: Easy Way to Create Your Amazing Portfolio

Problem Statement:

As a Full Stack Developer, you have to create a frontend for the myPortfolio application.

Before the next Class

You should know:

- Basics of Bootstrap
- Basics of JavaScript
- Basics of Angular

Phase-End Project

Duration: 240 min.

Kitchen Story

Project Objective:
As a Full Stack Developer, you are asked to develop an e-commerce portal that allows people to shop for basic food items.

Background of the Project Statement

Kitchen Story is an e-commerce portal that lets people shop for basic food items on their website. The website needs to have the following features:

- A search form on the home page to allow entry of the food items to be purchased by the customer.
- Based on the item details entered, it will show the available food items with their respective prices.
- Once a person selects an item to purchase, they will be redirected to the list of available items. On the next page, they are shown the complete breakout of the order and details of the payment to be made in the payment gateway. When payment is made, they are shown a confirmation page with details of the order.

You Are Asked to Do

For the above features to work, there will be an admin in the backend with the following features:

- Admin login page where the admin can change the password after login
- A master list of all the food items available for purchase
- A functionality to add or remove food items

You Must Use the Following
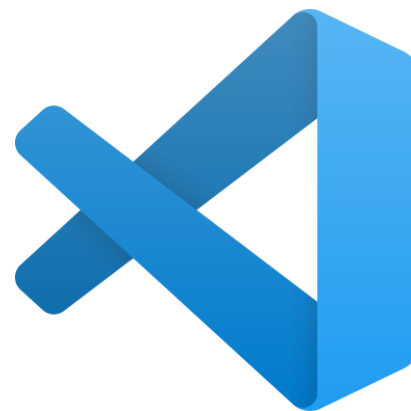
IDE: Eclipse or IntelliJ

Programming Language: Angular

Git and GitHub

MongoDB

Visual Studio Code

Specification Document