Installing angular CLI -> npm install –g @angular/cli  
To create new app -> ng new <app-name>  
To start angular app -> ng serve

To generate new component -> ng generate component <folder-name>/<component-name>   
 or ng g c <folder-name>/<component-name>

1. What is component

It is a typescript class decorated with @Component decorator and it contains methods & properties which we can use in HTML.

For example:-   
@Component({

selector: ‘app-root’, // value assigned to selector can be used as HTML tag.

templateUrl: ‘./app.component.html’ // this html file is also called as view template.

styleUrls: [‘./app.component.css’]

})  
export class AppComponent {

title = ‘AngularApp’;

message = ‘Angular is Javascript framework’

}

Note: If we place selector value within [] then it can be used as HTML attribute instead of html element.

So a component consists of   
a. Class: It contains code required for view template i.e. UI logic  
b. View Template: Defines User Interface, It contains HTML, data bindings & directives

c. Decorator: Adds meta data to the class, making it a component.

1. What is Data Binding?

It allows us to pass data between a component class to its corresponding view template and vice versa.

One way Binding: Data flows from either component class to view template (String Interpolation or Property Binding) or from View template to component class (event binding). This can be achieved using string interpolation, property binding & event binding.

String interpolation: {{data}}   
Property binding: [property] = data // this property will actually be an html attribute.

Event Binding: (data) = ’expression’

Two Way Binding: Data flows both ways. We use ngModel enclosed between [()]. For using ngModel we need to import FormsModule  
[(ngModel)] = “searchValue”

1. What is @NgModule?

@NgModule is a decorator that is used to create a module. A module is a way of organizing components, services, directives, and pipes into a cohesive unit. Modules are used to break down large applications into smaller, more manageable pieces.

The @NgModule decorator takes a configuration object that specifies the following:

* **declarations**: This is an array of components, directives, and pipes that belong to the module.
* **imports**: This is an array of other modules that the module depends on. This allows for the module to use components, directives, and pipes from other modules.
* **exports**: This is an array of components, directives, and pipes that the module makes available for other modules to use.
* **providers**: This is an array of services that the module provides. Services are singleton objects that can be shared across components.
* **bootstrap**: This is an array of components that are the starting point of the application. Typically, there is only one bootstrap component.

import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
import { AppComponent } from './app.component';

@NgModule({  
 declarations: [AppComponent],  
 imports: [BrowserModule],  
 providers: [],  
 bootstrap: [AppComponent]  
 })

export class AppModule { }

1. What is ngFor directive?

<div \*ngFor= “let item of itemList; let i = index”>  
 <p>{{item}}</p>

</div>

1. What is ngStyle directive?

<div [ngStyle] = “{color: available?‘Red’: ‘Green’}”

1. What is ngIf directive ?

<span \*ngIf= “text != ‘available’”>{{available}} </span>

Ng if with else

<span \*ngIf= “text != ‘available’; else notAvailable”>{{available}}</span>

<ng-template #notAvailable>{{notAvailable}}</ng-template> // #notAvailable is template reference variable

1. What is ngClass directive?

<div [ngClass] = “{changeBackground: value!= ‘’}”>

//changeBackground class will be applied when condition(value!= ‘’) corresponding to that class gives true.

</div>

1. @Input decorator

We can pass data from parent component to child component. We also call it custom property binding because here we bind custom properties of child component class with the property or method of parent component class

1. @Output decorator

We can pass data from child component to parent component

1. Template Reference Variable

It is a variable which stores a reference to DOM Element or a component or a directive. They can be identified by # followed by variable name.

1. @ViewChild

@ViewChild decorator is used to obtain a reference to a child component or DOM element within a parent component. This decorator allows a parent component to access and interact with a child component or DOM element directly from the component class.

1. Ng-content

It is used when we want to insert the content dynamically. Using ng-content we can pass content inside the component selector and when angular parses that content it appears at the place of ng-content. It is analogues to children prop in react.

1. Change Detection Cycle

On every event, Angular runs change detection cycle. The event can be any input change, DOM Events, timer events(setTimeOut, setInterval), HttpRequest, etc. During each change detection cycle Angular checks each and every property in the template with the component class. If it detects ant change it will update the DOM.



1. Angular Lifecycle methods

ngOnChanges, ngOnInit, ngDoCheck, ngAfterContentInit, ngAfterContentChecked, ngAfterViewInit, ngAfterViewChecked, ngOnDestroy

1. Services in Angular

It is a re-useable typescript class that can be used in multiple components across our Angular Application. Services can be injected into components or other services by using the @Injectable decorator. Use the 'ng generate service' command to create service

1. What is Observables

Observables converts ordinary stream of data into an Observable stream of data. We can think of Observable as wrapper around ordinary stream of data. It is provided by RxJs Library. RxJS allow us to work with asynchronous data stream.

Creating an Observable  
There are multiple ways of creating and Observable

**First**

myObservable = **new Observable**((observer)=>{

observer.next(“1”); // here next is emitting a value so that it can be caught in subscribe

observer.next(“2”);

observer.error(new Error(“something went wrong”))

observer.next(“3”); // 3 will not be emitted

observer.complete(); // no value will be emitted after completion

observer.next(“4”);

});

**Second**

myObserver = Observable.**create**((observer)=>{

observer.next(“1”); // here next is emitting a value so that it can be caught in subscribe

observer.next(“2”);

observer.error(new Error(“something went wrong”))

observer.next(“3”); // 3 will not be emitted

observer.complete(); // no value will be emitted after completion

observer.next(“4”);

});

**Third**

Of operator

**Fourth**

From operator

.next(value) 🡪 value will be emitted and this value will be caught in .subscribe() method

1. **Subjects**

We use subjects for cross component communication. There are 2 types of subjects in Angular

1. Behaviour Subject
2. Replay Subject

Difference between Subjects and Observable

1. A Subject is both ‘Observer’ & ‘Observable’. Which means it can both emit new value and subscribe to receive values.  
   An Observable is read only. Which means it can only be subscribed to receive values.
2. A Subject can have multiple subscribers.  
   An observable can have one subscriber at a time.

**How to create a subject**