**­­­Angular Topics**

**What is Angular?**

**Answer:**Angular is a Typescript-based open-source web application framework, developed and maintained by Google. It offers an easy and powerful way of building front end web-based applications.

**Angular Folder Structure:**

* **e2e**
* **node\_modules**
* **Src**
* **Package.json**
* **Angular.json**
* **tsconfig.json**

**Src 🡺**

* + - **App**
    - **assets**
    - **Index.html**
    - **polyfills.ts**
    - **Main.ts**

**e2e :** e2e stands for "end to end", this is the place where we can write the end to end test.

**Package.json:**This file has the list of node dependencies which are needed.

**tsConfig.json:**

The presence of a tsconfig.json file indicates that the directory is the root of a TypeScript project.

The tsconfig.json file specifies the root files and the compiler options required to compile the project.

**angular.json**: It  is standard configuration file of your application. .angular.json file when generating a project using Angular CLI.

**src/main.ts**: This is the main file which starts the Angular Application (AppModule is bootstrapped here as seen in the code ). Here the Extension .ts stands for Typescript. We can bootstrap our modules in this files.

**Pollyfills.ts:  Polyfills** in angular are few lines of code which make our application compatible for different browsers. The code we write is mostly in ES6 and typescript and is not compatible with IE or Firefox and needs some environment setups before being able to be viewed or used in these browsers.

## Why not Angular 3?

Angular 2 has been a single repository. Each package downloadable with the @angular/package-name convention. For example @angular/HTTP, @angular/router

All package names were assigned version 2, but router package by mistaken was given version 3. Therefore, the development team skipped Angular Version 3 and directly named it version 4 to maintain compatibility with Angular Router's version.

**What is string interpolation in Angular?**

Ans: [**Angular**](https://howtodoinjava.com/angular/dev-workspace-setup/)**interpolation** is used display a [component](https://howtodoinjava.com/angular/angular-component/) property in the respective view [template](https://howtodoinjava.com/angular/angular-templates-and-views/) with double curly braces syntax. We can display all kind of properties data into view e.g. [string](https://howtodoinjava.com/typescript/string-literal-types/), number, date, [arrays](https://howtodoinjava.com/typescript/arrays/), list or [map](https://howtodoinjava.com/typescript/maps/).

**How to generate a class in Angular using CLI?**

ng generate class Dummy

**Dependency Injection**

**1. What is Angular Dependency Injection?**

**Ans:** Dependency injection (DI) is an important application design pattern. Angular has its own DI framework, which is typically used in the design of Angular applications to increase their efficiency and modularity.

Dependency Injection is a technique in which we can provide instance to one object to another object which dependence on it. This technique also known as Inversion Of Control.

Angular makes use of DI to provide required dependencies to new components.

**Why Dependency Injection?**

**Watch: -** <https://www.youtube.com/watch?v=EzioZ9cww08&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=33>

**2. Parts Of the Angular Dependency Injection Frame Work?**

**Ans: 1.** Consumer

**2.** Dependency

**3.** DI Token

**4.** Provider

**5.** Injector

**3. What is Angular Injector?**

**Ans:**  The injector object that exposes APIs to us to create instances of dependencies. The angular injector is instantiating the dependency and injecting into Component or Service.

**Refer: Angular Tutorial Kud Venkat video No : 35 & 36.**

**What is <ng-template> ?**

 <ng-template> is an angular element for rendering HTML. It is never displayed directly. It can be displayed using structural directives.

Example:

h3>ng-template with ngFor</h3>

<ng-template ngFor let-person [ngForOf]= "allPersons" let-i="index">

<p> {{i + 1}}. {{person.name}} : {{person.age}} </p>

</ng-template>

**Ng-Directives**

1. **What are the different types of Directives?**

**Component directive**

**Attribute directive**

**Structural directive**

**Component directive:**  Component directive is used to create HTML template. This is most commonly used directive in angular project.

**Attribute directive:** Attribute directive changes the appearance or behaviour of DOM element.  Angular also provides in-built attribute directive such as NgStyle, NgClass.

Example: **[NgStyle]**

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

@Component({

  selector: 'my-app',

  template: `<h1 [ngStyle]="style">ng style</h1>

<h1 [ngStyle]=”{‘color’: ‘green’}”>Hello</h1>`,

})

export class AppComponent  {

  style = {color:'blue'}

 }

**Example: [NgCLass]**

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

@Component({

   selector: 'my-app',

   template: `<div [ngClass]=" flag ? 'primary' : 'secondary'"> sometext</div>`,

  styleUrls: [ './app.component.css' ]

})

export class AppComponent  {

  flag: boolean = true;

}

**Structural directive:**  Structural directive is used to change the DOM layout by adding and removing DOM elements. Angular provides in-built structural directive such as NgFor and NgIf

**Custom Attribute directive Example:**

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

@Directive({

selector: '[myHighlight]'

})

export class HighlightDirective {

constructor(el: ElementRef) {

el.nativeElement.style.backgroundColor = 'yellow';

}

}

Now this directive extends HTML element behavior with a yellow background as below

<p myHighlight>Highlight me!</p>

**Custom Structural Directive**

**Example ForLoop**

import { Directive, TemplateRef, ViewContainerRef, Input } from '@angular/core';

@Directive({

selector: '[cpLoop]'

})

export class CpLoopDecorator {

constructor( private templateRef: TemplateRef<any>,

private viewContainerRef: ViewContainerRef) { }

@Input('cpLoop') set loop(num: number) {

for(var i=0; i < num; i++) {

this.viewContainerRef.createEmbeddedView(this.templateRef);

}

}

}

**Example If:**

import { Directive, TemplateRef, ViewContainerRef, Input } from '@angular/core';

@Directive({

selector: '[cpIf]'

})

export class CpIfDirective {

constructor( private templateRef: TemplateRef<any>,

private viewContainer: ViewContainerRef) { }

@Input() set cpIf(condition: boolean) {

if (condition) {

this.viewContainer.createEmbeddedView(this.templateRef);

} else {

this.viewContainer.clear();

}

}

}

**What is TemplateRef**?

Represents an embedded template that can be used to instantiate embedded views. To instantiate embedded views based on a template, use the ViewContainerRef method with createEmbeddedView()

**What is ViewContainerRef?**

The ViewContainer provides an API that makes changes or dynamic updates to an [**View**](https://medium.com/@ole.ersoy/what-is-an-angular-view-4361594d4114) safe. It contains two types of Views:

***Embedded***: Views created from a TemplateRef instance.

***Host***: Views created using a component factory.

**Routing**

**What is Routing?**

**Ans:** Routing allows you to move from one part of the application to another part or one View to another View.

The Router Module provides the necessary service providers and directives for navigating through application views.

**Using Angular Router you can:**

* Navigate to a specific view by typing a URL in the address bar
* Pass optional parameters to the View
* Bind the clickable elements to the View and load the view when user performs application tasks
* Handles back and forward buttons of the browser
* Allows you to dynamically load the view
* Protect the routes from unauthorized users using Guards

### Router

* The Angular Router is an object that enables navigation from one component to the next component as users perform application tasks like clicking on menus links, buttons or clicking on back/forward button on the browser.
* We can access the router object and use its methods like navigate() or navigateByUrl(), to navigate to a route.

**Route**

Route tells the Angular Router which view to display when a user clicks a link or pastes a URL into the browser address bar. Every Route consists of a path and a component it is mapped to. The Router object parses and builds the final URL using the Route.

**Example:-**

{path: 'home', component: HomeComponent},

{path:'data', component: CrudComponent},

{path:'admin', component: CountComponent},

{path:'formcontrol', component: FormControlComponent}

### Routes

Routes is an array of Route objects our application supports.

**Example:-**

const routes: Routes = [

{path: 'home', component: HomeComponent},

{ path: '', redirectTo: 'home', pathMatch: 'full' }

];

**RouterOutlet**

The RouterOutlet is a directive (<router-outlet>) that serves as a placeholder, where the Router should display the view.

**Router-outlet directive: -**

Router-outlet directive is used to render the components for specific location of your applications. Both the template and **templateUrl**render the components where you use this directive.

### RouterLink

The RouterLink is a directive that binds the HTML element to a Route. Clicking on the HTML element, which is bound to a RouterLink, will result in navigation to the Route. The RouterLink may contain parameters to be passed to the route’s component.

**Example:-**

<a [**routerLink**]**="['data']"** class="nav-link font-weight-bold fixr" href="#">Crud Data</a>

<a **routerLink="admin"** class="nav-link font-weight-bold fixr" href="#">Administration</a>

### What is the purpose of base href tag?

### Angular makes use of the base href to tell router how to compose navigation URLs

The routing application should add element to the index.html as the first child in the tag in order to indicate how to compose navigation URLs. If app folder is the application root then you can set the href value as below

<base href="/">

**Difference B/W forRoot and forChild**

forRoot creates a module that contains all the directives, the given routes, and the router service itself.

forChild creates a module that contains all the directives and the given routes, **but does not include the router service.**

### Client Side Routing

In a Multi-page web application, Every time the application needs to a display a page it has to send a request to the web server. You can do that by either typing the URL in the address bar, clicking on the Menu link/ button. Every such action results in a new request being sent to the Web server.

The Client side routing is routing is handled in two ways

1. Hashstyle Routing
2. HTML 5 Routing

## Hashstyle Routing:

When the requested anchor tag is on the current page, then the browser does not send the request to the Web server.

The Hashstyle Routing uses this technique to create the URL

The URL would look like something like

http://www.example.com  
http://www.example.com/#/about

In the above example, the URL’s  “#/about” and “#/contact” is never sent to the server.

# What are Route Guards?

Angular’s route guards are interfaces which can tell the router whether or not it should allow navigation to a requested route. They make this decision by looking for a true or false return value from a class which implements the given guard interface.

* CanActivate
* CanActivateChild
* CanDeactivate
* CanLoad
* Resolve

**What is CanLoad?**

CanLoad guard is used to decide if a module can be loaded or not configured with loadChildren property.

CanLoad is an interface with canLoad method. To use CanLoad guard we need to create a service by implementing CanLoad interface and override its canLoad method.

When we use canLoad in our service it returns true then the feature module protected by canLoad it will lazy loaded otherwise not loaded.

loadChildren property is a reference to lazy loaded routes.

Example:

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

import { CanLoad, CanActivate, Route, Router } from '@angular/router';

import { AuthService } from './authentication/services/auth.service';

@Injectable()

export class AuthGuardService implements CanLoad {

constructor(private authService: AuthService, private router: Router) {

}

canLoad(route: Route): boolean {

if (this.authService.isUserLoggedIn()) {

return true;

}

this.authService.setRedirectUrl(url);

this.router.navigate([ this.authService.getLoginUrl() ]);

return false;

}

}

Path:

**{**

**path: 'admin',**

**loadChildren: 'app/admin/admin.module#AdminModule',**

**canLoad: [ AuthGuardService ]**

**}**

**CanActivate:**  Checks route navigation before the component is loaded.

CanActivate is an Angular interface. It is used to force user to login into application before navigating to the route.

Example:

export class AuthGuard implements CanActivate {

  constructor(private \_authService: AuthService, private \_router: Router) {

  }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean>  {

    if (this.\_authService.isAuthenticated()) {

        return true;

    }

    this.\_router.navigate(['/login']);

        return false;

  }

**Routing file**

**{**

**path: '',**

**component: DashboardLayoutComponent,**

**canActivate: [AuthGuard],**

**}**

**CanActivateChild**: Checks route children navigation before the

component is loaded.

CanActivateChild is an Angular interface to guard child routes. Suppose a user has been authenticated but not authorized to visit the child routes, so child routes can be guarded using CanActivateChild

**Resolve**: Resolve loads/ retrieves data before the route is activated.

**CanDeactivate:**  The Angular CanDeactivate guard is called, whenever we navigate away from the route before the current component gets deactivated.

The best use case for CanDectivate guard is the data entry component. The user may have filled the data entry and tries to leave that component without saving his work. The CanDeactivate guard gives us a chance to warn the user that he has not saved his work and give him a chance to cancel the navigation.

**Example**: **CanDeactivate**

Register.component.ts

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

@Component({

  templateUrl: "register.component.html",

})

export class RegisterComponent

   canExit() : boolean {

   if (confirm("Do you wish to Please confirm")) {

       return true

     } else {

       return false

     }

   }

}

deactivateGuard.service.ts

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

import { CanDeactivate } from '@angular/router';

import { ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

import { Observable } from 'rxjs';

import { RegisterComponent } from './register.component';

@Injectable()

export class DeactivateGuard implements CanDeactivate

{

    component: Object;

    route: ActivatedRouteSnapshot;

   constructor(){

   }

   canDeactivate(component:RegisterComponent,

                route: ActivatedRouteSnapshot,

                state: RouterStateSnapshot,

                nextState: RouterStateSnapshot) : Observable<boolean> | Promise<boolean> | boolean {

        return component.canExit();

  }

}

**Path:**

  { path: 'register', component: RegisterComponent, canDeactivate:[DeactivateGuard] }

## What are query parameters?

Query parameters are optional parameters that we pass to a route. The query parameters are added to the end of the URL Separated by Question Mark.

For Example: **/products?page=2**

**Example:**

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

import { ActivatedRoute, Router } from '@angular/router';

@Component({

selector: 'product-list',

template: `<!-- Show product list -->`

})

export default class ProductList {

sub: any;

page: number;

constructor( private route: ActivatedRoute,

private router: Router) {}

ngOnInit() {

this.sub = this.route.queryParams.subscribe(params => {

this.page = +params['page'] || 0;

});

}

ngOnDestroy() {

this.sub.unsubscribe();

}

nextPage() {

this.router.navigate(['product-list'], { queryParams: { page: this.page + 1 } });

}

}

**Path:**

**<a [routerLink]="['product-list']" [queryParams]="{ page: 99 }">Go to Page 99</a>**

**Basic Questions**

**Question: Can you give us an overview of Angular architecture?**  
**Answer**: You can draw some like this

### What is a bootstrapping module?

Every application has at least one Angular module, the root module that you bootstrap to launch the application is called as bootstrapping module. It is commonly known as AppModule. The default structure of AppModule generated by AngularCLI would be as follows,

import { BrowserModule } from '@angular/platform-browser';

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

import { FormsModule } from '@angular/forms';

import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

FormsModule,

HttpClientModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**2.** [**What Are The New Features Of Angular 2? Why You Used Angular 2?**](https://www.code-sample.com/2017/06/new-features-of-angular-2.html)

**Angular 2 Features –**

Angular 2 is Entirely Component Based

Directives

Dependency Injection

Used of Typescript

Used of Lambdas or Arrow functions

Generics

Forms and Validations

**3.** [**What are advantages of Angular and Why we Use Angular?**](https://www.code-sample.com/2016/06/angular-2-vs-angular-1-performance.html)

The core differences and many more advantages on Angular 2 vs. Angular 1 as following,

1.     It is entirely component based.

2.     Better change detection

3.     Angular2 has better performance.

4.     Angular2 has more powerful template system.

5.     Angular2 provide simpler APIs, lazy loading and easier to application debugging.

6.     Angular2 much more testable.

7.     Angular2 provides to nested level components.

8.     Ahead of Time compilation (AOT) improves rendering speed

9.     Angular2 execute run more than two programs at the same time.

10.  Angular1 is controllers and $scope based but Angular2 is component based.

11.  The Angular2 structural directives syntax is changed like ng-repeat is replaced with \*ngFor etc.

12.  In Angular2, local variables are defined using prefix (#) hash

13.  TypeScript can be used for developing Angular 2 applications

14.  Better syntax and application structure.

1. **What is AOT compilation? Why Use in Angular ?**

AOT compilation stands for “Ahead of Time compilation” and it is used to compiles the angular components and templates to native JavaScript and HTML during the build time.

The compiled HTML and JavaScript are deployed to the web server so that the compilation and render time can be saved by the browser. It is the big advantage to improve the performance of applications.

**Advantages of AOT -**

1.   **Faster download**: - The Angular 2 app is already compiled so it is faster.

2.   **Faster Rendering**: - If the app is not AOT compiled and the compilation process happens in the browser once the application is fully loaded. This has a wait time for all necessary components to be downloaded and then the time taken by the compiler to compile the app. With AOT compilation, this is optimized.

3.    **Lesser Http Requests**: - It is supporting to the lazy loading. Actually, lazy loading is great concepts for sending HTTP request to the

server. It is minimise the multiple requests for each associated html and css, there is a separate request goes to the server.

4.    **Detect error at build time**: - In Angular 2, the compilation happens beforehand and most of the errors can be detected at the compile time and this process providing us a better application’s stability.

**Disadvantages of AOT -**

1.  AOT only works only with HTML and CSS and not for other file types. If required other file types that time we will need to follow the previous build step.

2.   We need to maintain AOT version of bootstrap file.

1. We need to clean-up step before compiling.

### What is JIT?

Just-in-Time (JIT) is a type of compilation that compiles your app in the browser at runtime. JIT compilation is the default when you run the ng build (build only) or ng serve (build and serve locally) CLI commands. i.e, the below commands used for JIT compilation,

**ng build**

**ng serve**

**How do you categorize data binding types?**

|  |  |  |
| --- | --- | --- |
| **Data direction** | **Syntax** | **Type** |
| From the source-to-view(One-way) | 1. {{expression}} 2. [target]="expression" 3. bind-target="expression" | Interpolation, Property, Attribute, Class, Style |
| From view-to-source(One-way) | 1. (target)="statement" 2. on-target="statement" | Event |
| View-to-source-to-view(Two-way) | 1.[(target)]="expression" 2. bindon-target="expression" | Two-way |

1. [**What is Lazy Loading and How to enable Lazy Loading?**](https://www.code-sample.com/2017/06/angular-2-lazy-loading.html)

* **Lazy Loading** - Lazy Loading is the technique of loading the module or data on demand.
* It helps us to better the application performance
* Reduce the initial bundle size of our files.
* which in turn helps decrease load times.
* The initial page loads faster and we can also split.
* loadChildren property is a reference to lazy loaded child routes.

{

path: 'orders',

loadChildren: () => import('./orders/orders.module').then(mod => mod.OrdersModule)

},

1.         Each and every Angular application must have one main module that is called “AppModule” and your code should be splitted into various child modules based on your applications.

2.         We do not require to import or declare lazily loading module in root module.

3.         Add the route to top level routing and takes routes array and configures the router.

4.         Import module specific routing in the child module.

5.         And so on.

**Components**

**What are the building blocks of Angular?**

1. **Components**
2. **Directives**
3. **Data-Binding**
4. **Meta-data**
5. **Dependency-Injection**
6. **Modules**
7. **Routing**
8. **Services**
9. **Template**

**6. What is Component in angular?**

Components are the most basic building block of a UI in Angular applications and it controls views (HTML/CSS). They also communicate with other components and services to bring functionality to your applications.

The component is the core functionality of Angular app but we need to know to pass the data in to the components to configure them.

* Angular Component meta-data annotation is used to register the components.
* Angular components are used to create UI widgets.
* Angular components are used to split to application into smaller parts.
* Only one component is used per DOM element.
* In the Angular components, @View, template and templateUrl are mandatory in the components.

**7. Component Vs Directive?**

|  |  |
| --- | --- |
| [**@Components**](https://www.code-sample.com/2016/04/angular-2-components-vs-directives.html) | [**@Directive**](https://www.code-sample.com/2016/04/angular-2-components-vs-directives.html) |
| 1.       @Component meta-data annotation is used to register the components. | @Directive meta-data annotation is used to register the directives. |
| 2.       The components are used to create UI widgets. | The directives are used to add behaviour to existing DOM elements. |
| 3.       The components are used to split to application into smaller parts. | The directives are used to design a reusable components. |
| 4.       Only one component is used per DOM element for Rendering. | More than one directive are used per DOM element. |
| 5.       In the components, @View, template and templateUrl are mandatory in the components. | The directive do not have @View etc. |

**What Is Module ?**

**Modules:** An angular module is set of angular basic building blocks like component, directives, services etc. An application is divided into logical pieces and each piece of code is called as "module" which perform a single task.

**What is metadata?**

Metadata is used to decorate a class so that it can configure the expected behaviour of the class. The metadata is represented by decorators

Class decorators, e.g. @Component and @NgModule

**8. Constructor Vs NgOnonit?**

**Angular Constructors:-**

The constructor is a default method runs when component is being constructed.

The constructor is a typescript feature and it is used only for a class instantiations and nothing to do with Angular.

The constructor called first time before the ngOnInit().

**9. What is the use @Input()?**

**@Input { From parent to Child}**

@Input decorator binds a property within one component (child component) to receive a value from another component (parent component). This is one way communication from parent to child. The component property should be declared with @Input decorator to act as input property.

We can declare @Input decorator in child Component.

**Import {Input} from '@angular/core';**

**@Input()** msg: string= "Hello World”;

To use alias for the binding property name we need to assign an alias name as @Input(alias).

**@Input(stdMsg) msg = “Hello World”**

It can be invoked at any type of property such as number, string, array or user defined class.

**@Input()** array: Array<string>

**Example of @Input Decorator:-**

**Input-Decorator.component.ts (Child Component)**

import { Component, Input } from '@angular/core';

@Component({

selector: 'app-input-decorator',

template:`<h1>{{msg}}</h1>`,

styleUrls: ['./input-decorator.component.css']

})

export class InputDecoratorComponent {

@Input() msg : string

}

**App.component.ts (Parent Component)**

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Hello @Input and @Output Decorator Example';

}

**App.component.html**

<app-input-decorator [msg]="title"></app-input-decorator>

**10. What is the use of @Output?**

**@Output() {from child to parent}**

**Import {Output} from '@angular/core';**

@Output decorator binds a property of a component to send data from one component (child component) to calling component (parent component). This is one way communication from **child to parent** component. @Output binds a property of the type of angular EventEmitter class.

@Output()

sendMsgEvent = new EventEmitter<string>();

Find the @Output decorator using aliasing.

**Example:- Output-decorator.component.ts (Child Component)**

import { Component, Output, EventEmitter } from '@angular/core';

@Component({

selector: 'app-output-decorator',

templateUrl: `

<button (click)= "counterIncrement()">Click here</button><br/><br/>

<h2>Counter: {{counter}}</h2>`,

styleUrls: ['./output-decorator.component.css']

})

export class OutputDecoratorComponent {

constructor(){}

counter= 0;

@Output() myOutput = new EventEmitter();

counterIncrement() {

this.counter = this.counter + 1

this.myOutput.emit(this.counter);

}

}

**App.component.ts (Parent Componet)**

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

displayCounter() {

console.log("Hello");

}

}

**App.component.html**

<app-output-decorator (myOutput)= "displayCounter()"></app-output-decorator>

# What is Event Emitter ?

Use in components with the @[Output](https://angular.io/api/core/Output) directive to emit custom events synchronously or asynchronously, and register handlers for those events by subscribing to an instance.

**11. Renderer Vs ElementRef**

**ElementRef:-**

Use ElementRef class to access DOM to change host element appearance and behaviour.

ElementRef is a class that is a partial abstraction done the DOM Manipulations without breakable environments and it also can hold a reference to a DOM elements

The ways to get an ElementRef instance looks like,

* @ViewChild()
* @ViewChildren()
* @ContentChild()
* @ContentChildren()

**Example:-**

**Color-input.directive.ts**

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

@Directive({

selector: '[textSize]'

})

export class TextSizeDirective implements AfterViewInit {

@Input('textSize') tsize: string;

constructor(private elRef: **ElementRef**) {

    }

    ngAfterViewInit(): void {

     this.elRef.nativeElement.style.fontSize = this.tsize;

    }

}

**App.component.ts**

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html'

})

export class AppComponent {

txtsize = '25px';

}

**App.component.html**

<p [textSize]="txtsize"> textSize Directive Demo using Bracket []</p>

<p bind-textSize="txtsize"> textSize Directive Demo using bind- prefix </p>

<p textSize="{{txtsize}}"> textSize Directive Demo using Interpolation</p>

**Renderer**:-

Renderer is a class that is a partial abstraction done the DOM manipulations and the DOM manipulating is not breaking server side rendering or web workers.

**What is the Use of @ViewChild() ?**

* The **@ViewChild**  decorators in Angular provide a way to access and manipulate DOM elements, directives and components.
* **@ViewChild()** decorator configures a view query.
* **@ViewChild** provide the instance of another component in Parent Component.
* And it can be accessed the properties and methods from its Child Component.
* To use **@ViewChild()** we need to pass child component name or directive name or template variable as an argument.

Example:

**numberChild.component.ts**

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

@Component({

selector: 'app-number',

template: `<h1>{{count}}</h1>

<h1>{{message}}</h1>`

})

export class NumberChildComponent {

message:string ='';

count:number = 0;

increaseByOne() {

this.count = this.count + 1;

this.message = "Counter: " + this.count;

}

decreaseByOne() {

this.count = this.count - 1;

this.message = "Counter: " + this.count;

}

}

Now we will create the instance of NumberComponent in our parent component using @ViewChild().

**number-parent.component.ts**

import { Component, ViewChild } from '@angular/core';

import { NumberChildComponent } from './numberChild.component';

@Component({

selector: 'app-number-parent',

templateUrl: './number-parent.component.html'

})

export class NumberParentComponent {

@ViewChild(NumberChildComponent)

private numberChildComponent: NumberChildComponent;

increase() {

this. numberChildComponent.increaseByOne();

}

decrease() {

this. numberChildComponent.decreaseByOne();

}

}

We will observe that we are able to access the methods of NumberComponent in NumberParentComponent. We will use selector of NumberComponent in HTML template of NumberParentComponent.

**number-parent.component.html**

**<h3>@ViewChild using Component</h3>**

**Number Example:**

**<button type="button" (click)="increase()">Increase</button>**

**<button type="button" (click)="decrease()">Decrease</button>**

**<app-number></app-number>**

**What is the Use of @ViewChildren() ?**

Angular @ViewChildren Decorator is used to get the QueryList of multiple elements or directives from the view DOM and return a [QueryList](https://angular.io/api/core/QueryList).

**Example**

**Hello.component.ts**

import { Component, Input } from '@angular/core';

@Component({

selector: 'hello',

template: `<h1>Hello {{name}}!</h1>`,

styles: [`h1 { font-family: Lato; }`]

})

export class HelloComponent {

@Input() name: string;

}

**App.component.ts**

import { Component, ViewChildren, AfterViewInit, QueryList } from '@angular/core';

import { HelloComponent } from './hello.component';

@Component({

selector: 'my-app',

templateUrl: './app.component.html',

styleUrls: [ './app.component.css' ]

})

export class AppComponent implements AfterViewInit {

name = 'Angular';

@ViewChildren(HelloComponent) hellos: QueryList<any>;

ngAfterViewInit() {

this.hellos.forEach(hello => console.log(hello));

}

}

**App.component.html**

<hello name="Angular 6" ></hello>

<hello name="Angular 7" ></hello>

<hello name="Angular 8" ></hello>

Output 🡺 Hello Angular 6

Hello Angular 7

Hello Angular 8

**Services**

**Could you explain services in Angular?**

Ans: If We declare the services in angular application we can initialize the some methods and logic functionality to the web page. Then this logic and methods we can use the throughout the application. It is easy Process to share the data between the components.

1. **What is use of Service in angular?**

Service is a piece of reusable code with a focused Purpose. A code that you will use it in many components across our application.

Services are singleton objects.

**What services are used for?**

* Features that are independent of components such a logging services
* Share logic or data across components
* Encapsulate external interactions like data access

**Advantages of Service**

* Services are easier Test.
* Services are easier to Debug.
* You can reuse the service.
* Service class is decorated with Injectable decorator.

## Singleton service

There are two ways to make a service a singleton in Angular:

When we import the service within the component like app.component.ts then we can use that service in app.component itself. we can’t to use that service in throughout the application.

* Declare that the service should be provided in the application root.
* Include the service in the AppModule or in a module that is only imported by the AppModule

**What is Observable?**

Observables provide support for passing messages and any kind of data between publishers and subscribers in our application.

They are mainly used for event handling, asynchronous programming, and handling multiple values.

It Handling the multiple values over a time and it can return multiple values.

An observable can deliver multiple values of any type—literals, messages, or events, depending on the context.

2. **What are the differences between Observables & Promises?**

**Promise**:-

* Promises are only called once and It can return only a single value at a time .
* Promises are not cancellable.
* This is a JavaScript class.
* Promise has methods such as then(), catch() etc.

**Observables**:-

* Observables handle multiple values over time and it can return multiple values
* Observables are cancellable.
* The Observables are more advanced than Promises.
* Observable provides methods such as map(), catch() etc.

**PIPES Questions**

* 1. **What is pipe or Filters in angular?**

Angular Pipes are used to format the data, Simply transform the data into user Expected format.

These filters can be added to the templates, directives, controllers or services.

 Filters are added to the expressions by using the pipe character |

**Angular Built-in Pipes Or Filters:-**

* **currency:** Format a number to a currency format.
* **date:** Format a date to a specified format.
* **filter:** Select a subset of items from an array.
* **json:** Format an object to a JSON string.
* **limit:**To Limits an array/string, into a specified number of elements/characters.
* **lowercase:** Format a string to lowercase.
* **number:** Format a number to a string.
* **orderBy:** Orders an array by an expression.
* **uppercase:** Format a string to upper case.

**Examples**

**{{message | uppercase}}**

Output will be as follows.

**HELLO WORLD!**

2. Example for **lowercase** pipe:

**{{message | lowercase}}**

Output will be as follows.

**hello world!**

3. Example for **date** pipe with **fullDate** format:

**{{person.dob | date:'fullDate'}}**

Find the output.

Saturday, April 12, 1980

**2. What is Async Pipe?**

Angular provides us special kinds of pipe that is called Async pipe and the Async pipe subscribes to an Observable or Promise and returns the latest value it has emitted.

# 3. How To Create and Use custom Pipes in Angular 2?

**Creating a Custom Pipe:-**

* Create a typescript class.
* Decorate the class using @Pipe.
* Implement PipeTransform interface.
* Override transform() method.
* Configure the class in application module with @NgModule.
* Ready to use our custom pipe anywhere in application.

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

@Pipe({

name: 'welcome'

})

export class WelcomePipe implements PipeTransform {

transform(value: string): string {

let message = "Welcome to " + value;

return message;

}

}

The name metadata of @Pipe decorator has the value welcome and that will be the name of our custom pipe. In our custompipe.component.ts file we are using **welcome** pipe as given below.

{{person.name | welcome}}

**Output**

Welcome to Ram

**Custom Pipe**

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

@Pipe({

name: 'filterPipe'

})

export class FilterPipePipe implements PipeTransform {

transform(value: any, app: string): any {

if (app) {

app = app.toLocaleLowerCase();

return value.filter(function (el) {

return el.name.toLocaleLowerCase().indexOf(app) > -1

})

}

return value;

}

}

[**Template Questions**](javascript:void(0);)

**1.** [**What is Template Angular ? Why Use?**](https://www.code-sample.com/2016/06/angular-2-template-components.html)

A template is a HTML view that tells Angular for render your components in the views.

**Two types of Template integration in component:-**

1. Template: Inline Html used in component

* 1. TemplateUrl: html defined in separate file we can import its own Component

1. **StyleUrl:** CSS defined in external file we can import its own Component through Array.

**EX:** @Component({

selector: 'users-app',

templateUrl: './user.component.html',

styleUrls: ['./user.component.css']

})

1. [**What is a template reference** **variable. How would you use it?**](https://www.code-sample.com/2016/06/angular-2-template-components.html)

A template reference variable (#var) is a reference to a DOM element within a template. We can declare the template reference variable we use the # Symbol.

**Ex: <input #name placeholder=”Enter your Name”> {{name.value}}**

Now we can access any property of the input in DOM, using this reference variable. For example, we can get the value of the input element as name.value

**What Is Subscribe()?**

## Observable. Subscribe()

The observable subscribe method is used by angular components to subscribe to messages that are sent to an observable.

In **Angular** .**subscribe**() is a method on the Observable type. The Observable type is a utility that asynchronously or synchronously streams data to a variety of components or services that have **subscribed** to the observable.

**Question: Explain the difference between an Annotation and a Decorator in Angular?  
Answer:** In Angular, annotations are used for creating an annotation array. They are only metadata set of the class using the Reflect Metadata library.

Decorators in Angular are design patterns used for separating decoration or modification of some class without changing the original source code.

## Decorators and Annotations in Angular

Both Decorators and Annotations are supported by Angular. This is a legacy thing because Angular 2 swapped from AtScript to TypeScript while it was still in development.

* Compiling with Traceur? You have annotations.
* Handcoding ES5 or ES6 with the annotations style? You have annotations.
* Compiling with TypeScript? You have decorators.
* Handcoding ES5 or ES6 with the Lucid API (my favourite)? You have decorators.

The main difference you will notice is your imports. Because annotations and decorators are different, you will need to import different objects.

If you are using decorators, your imports will look like normal TypeScript imports:

import {Component, View} from 'angular2/angular2';

If you are using annotations then you'll have to import the annotation version of the core angular components

import {ComponentAnnotation as Component, ViewAnnotation as View} from 'angular2/angular2';

Otherwise, your Angular code will remain unchanged.

**What is Data Binding? How many ways it can be done?**

**Answer**: Data binding is a core concept in Angular and allows to define communication between a component and the DOM, making it very easy to define interactive applications without worrying about pushing and pulling data.

Event Binding – Enables the application to respond to user input in the target environment

1. Property Binding – Enables interpolation of values computed from application data into the HTML
2. Two-way Binding – Changes made in the application state gets automatically reflected in the view and vice-versa. The ngModel directive is used for achieving this type of data binding.

**Could you explain the various types of filters in Angular.**  
**Answer:** In order to format the value of expression so that it can be displayed to the user, Angular has filters. It is possible to add these filters to the controllers, directives, services, or templates. Angular also provides support for creating custom filters.

***currency*** – Formats a number to the currency format

***date*** – Formats a data to some specific format

***filter*** – Selects a subset of items from an array

***json*** – Formats an object to a JSON string

***limitTo*** – Limits an array or string into a specified number of characters or elements

***lowercase*** – Formats a string to lowercase

***number*** – Formats a number to a string

***orderBy*** – Orders an array by an expression

**What is the process called by which Typescript code is converted into JavaScript code?**  
**Answer**: It is called Transpiling. Even though TypeScript is used for writing code in Angular applications, it gets internally transpiled into equivalent JavaScript.

**How to prevent security threads in Angular App? What are all the ways we could secure our App?**

* Avoid using/injecting dynamic HTML content to your component.
* If using external HTML which is coming from database or somewhere outside the application, sanitize it before using.
* Try not to put external urls in the application unless it is trusted. Avoid url re-direction unless it is trusted.
* Consider using AOT compilation or offline compilation.
* Try to prevent XSRF attack by restricting the api and use of the app for known or secure environment/browsers.

**How to optimize Angular app?**

* Consider lazy loading instead of fully bundled app if the app size is more.
* Make sure that any 3rd party library, which is not used, is removed from the application.
* Have all dependencies and dev-dependencies are clearly separated.
* Make sure the application doesn’t have un-necessary import statements.
* Make sure the application is bundled, uglyified, and tree shaking is done.
* Consider AOT compilation.

**Q16. What is Traceur compiler?**

Traceur compiler is a Google project. It compiles ECMAScript Edition 6 (ES6) (including classes, generators and so on) code on the fly to regular Javascript (ECMAScript Edition 5 [ES5]) to make it compatible for the browser.

Traceur itself is written in ES6, compiled to ES5

**What are the Angular lifecycle hooks?**

**ngOnChanges**– This event executes every time when a value of an input control within the component has been changed. Actually, this event is fired first when a value of a bound property has been changed. It always receives a change data map, containing the current and previous value of the bound property wrapped in a SimpleChange.

**ngOnInit** – This event initializes after Angular first displays the data-bound properties or when the component has been initialized. This event is basically called only after the ngOnChanges()events. This event is mainly used for the initialize data in a component.

**ngDoCheck**– This event is triggered every time the input properties of a component are checked. We can use this hook method to implement the check with our own logic check. Basically, this method allows us to implement our own custom change detection logic or algorithm for any component.

**ngAfterContentInit**–  This lifecycle method is executed when Angular performs any content projection within the component views. This method executes when all the bindings of the component need to be checked for the first time. This event executes just after the ngDoCheck() method. This method is basically linked with the child component initializations.

**ngAfterContentChecked** – This lifecycle hook method executes every time the content of the component has been checked by the change detection mechanism of Angular. This method is called after the ngAfterContentInit() method. This method is also called on every subsequent execution of ngDoCheck(). This method is also mainly linked with the child component initializations.

**ngAfterViewInit** – This lifecycle hook method executes when the component’s view has been fully initialized. This method is initialized after Angular initializes the component’s view and child views. It is called after ngAfterContentChecked(). This lifecycle hook method only applies to components.

**ngAfterViewChecked**– This method is called after the ngAterViewInit() method. It is executed every time the view of the given component has been checked by the change detection algorithm of Angular. This method executes after every subsequent execution of the ngAfterContentChecked(). This method also executes when any binding of the children directives has been changed. So this method is very useful when the component waits for some value which is coming from its child components.

**ngOnDestroy** – This method will be executed just before Angular destroys the components. This method is very useful for unsubscribing from the observables and detaching the event handlers to avoid memory leaks. Actually, it is called just before the instance of the component is finally destroyed. This method is called just before the component is removed from the DOM.

**What is webpack?**

### Webpack is an open source JavaScript module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or asset.

### What is multicasting?

Multi-casting is the practice of broadcasting to a list of multiple subscribers in a single execution. If we don't register a couple of listeners on the document, however as an alternative re-use the first listener and send values out to every subscriber. Let's demonstrate the multi-casting feature.

var source = Rx.Observable.from([1, 2, 3]);

var subject = new Rx.Subject();

var multicasted = source.multicast(subject);

// These are, under the hood, `subject.subscribe({...})`:

multicasted.subscribe({

next: (v) => console.log('observerA: ' + v)

});

multicasted.subscribe({

next: (v) => console.log('observerB: ' + v)

});

### What is zone?

### NgZone is a wrapper around Zone.js which is a library that creates a context around asynchronous functions in order to make them trackable.

### Without zones, we don’t get any change detection, so we don’t get any of the nice UI updates that we’d expect!

### 

### How do you select an element with in a component template?

You can use @ViewChild directive to access elements in the view directly. Let's take input element with a reference,

<input #uname>

and define view child directive and access it in ngAfterViewInit lifecycle hook

@ViewChild('uname') input;

ngAfterViewInit() {

console.log(this.input.nativeElement.value);

}

**What is NgForm?**

NgForm directive is used with HTML form tag that can be exported in local template variable to access form values and validation status and to pass entire form to our class on form submit.

**What is NgModel?**

The **ng-model** directive binds the value of **HTML** controls (**input, select, textarea**) to application data.  Using the two-way binding, we can display a data property as well as an update that property when the user makes changes. We can merely achieve it in the component element as well as **HTML** element both.

## NgModule :

NgModule is most important in the Angular application. It is a **class** where all the parts of the angular application fit together. Every application has at-least one NgModule for bootstrapping the angular application.

**app.module.ts**

In the above code of snippet, we are using @NgModule decorator which tells to the angular AppComponent is an NgModule class. @NgModule takes an object which says as meta-data. It takes three things

1. The import Array
2. The declaration Array
3. The bootstrap Array

**Angular 6 new Features**

The <template> deprecated, Now Angular 6 introduce <ng-template> –

Now in Angular 6, you should use *<ng-template>* instead of *<template>*

Angular 6 uses RxJS 6 - this is the third-party library (RxJS) and introduces two important changes as compared to RxJS 5.

1.      RxJS 6 introduces a new internal package structure

import { Observable } from 'rxjs/Observable';

import { Subject } from 'rxjs/Subject';

**Instead Of these**

import { Observable, Subject } from 'rxjs';

1. Ng Update : This CLI commands will update your angular project dependencies to their latest versions.
2. Changed Operator Usage:

getMicroStoreData() {

return this.http.get('assets/data/data.json').map((response:Response) => response.json());

}

**Instead Of**

**we can use the new pipe () method**

getMicroStoreData() {

return this.http.get('assets/data/data.json').pipe(

map((response:Response) => response.json()));

}

**Angular 7 features**

* Introduce a new Pipe called — KeyValuePipe
* Added a new life cyclehook — DoBootstrap interface
* This is a major release and expanding to the entire platform including-  
  — Core framework,  
  — Angular Material,  
  — CLI
* Added a new interface — UrlSegment[] to CanLoad interface
* Angular 7 added a new compiler — Compatibility Compiler (ngcc)

**KeyValuePipe;** It introduces a new KeyValue pipe to help you iterate through objects, maps, and arrays. The ngFor directive doesn’t support iterations over objects or Maps. To fix this issue, Angular 7 introduces a new KeyValue pipe. The **KeyValue** pipe converts an **Object** or **Map** into an array of key-value pairs to use with ngFor.

**DoBootstrap:** It is a life cycle hook using the manual bootstrapping the application using of this DoBootstrap lifecycle hook to bootstrapping the application.

**Example:**

class AppModule implements [DoBootstrap](https://angular.io/api/core/DoBootstrap) {

ngDoBootstrap(appRef: [ApplicationRef](https://angular.io/api/core/ApplicationRef)) {

appRef.bootstrap(AppComponent);

}

}

Using this Method to bootstrapping the angular Application

**TypeScript supports the following modifiers:**

1. **Public:**All the properties and methods could be accessed everywhere if they are declared as public.

2. **Private:**The private declared properties and methods can be accessed only within the class definition itself.

3. **Protected:**Properties and methods can be accessed from inside the class or any other class extending the one that owns the property or the method which are declared as protected.

**Difference between TypeScript and JavaScript:**

* Typescript is known as Object oriented programming language whereas JavaScript is a scripting language.
* TypeScript has a feature known as Static typing but JavaScript does not have this feature.
* TypeScript gives support for modules whereas JavaScript does not support modules.
* TypeScript has Interface but JavaScript does not have Interface.
* TypeScript support optional parameter function but JavaScript does not support optional parameter function.

**Reactive Form**: Creating form using FormControl, FormGroup and FormArray is said to be reactive form. They use ng module as ReactiveFormsModule.  
**Template-Driven Form**: Creating form using NgForm and NgModel, is said to be template-driven form. They use ng module as FormsModule.

ReactiveFormsModule and FormsModule both belong to @angular/forms library.

**What are Reactive forms ?**

**FormControl**: It is a class that is used to get and set values and validation of a form control such as <input> and <select> tag.

tracks the value and validation status  
**FormGroup**: It has the role to track value and validity state of group of FormControl.

The FormGroup is a group of FormControl instances, keeps track of the value and validation status of total form control.  
**FormArray**: It tracks the value and validity state of array of FormControl, FormGroup or FormArray instances.

**what is parameterized pipe in angular ?**

A pipe can take parameters as well. We can pass parameters along with the pipe. A parameter is separated with a colon symbol (:) after the pipe:

**Example:**

**{{salary | currency:'USD':true}}**

##### **Angular 7 has two core dependencies namely RxJS and TypeScript**

* + RxJS 6.3 – Angular 7 framework uses RxJS version 6.3.
  + TypeScript- Angular 7 framework uses TypeScript version 3.1

**What Is HttpClient in Angular?**

The **HttpClient** is an **Angular** module that allows your **application** to communicate with backend services over the HTTP protocol. You can perform all HTTP requests including GET, POST, PUT, PATCH and DELETE.

**Import { HttpClient } from “@angular/common/http”;**

**What is URL Segments in Angular 7?**

**UrlSegment Interface -**UrlSegment interface represents a single URL segment and the constructor, properties, and methods.

The UrlSegment is a part of a URL between the two slashes and it contains a path and matrix parameters associated with the segment.

**Example:**

classUrlSegment {

constructor(path: string, parameters: {...})

path: string

parameters: {...}

toString(): string

}

**What is the use of Rxjs In Angular?**

**RxJS** (Reactive Extensions for JavaScript) is a library for reactive programming using observables that makes it easier to compose asynchronous or callback-based code.

**What is Decorator?**

Decorators are a design pattern that is used to separate modification or decoration of a class without modifying the original source code. In AngularJS, decorators are **functions** that allow a service, directive or filter to be modified prior to its usage.

## Defining Single Page Application

**Single**-**Page Applications** (SPAs) are Web **apps** that load a **single** HTML **page** and dynamically update that **page** as the user interacts with the app. SPAs use AJAX and HTML5 to create fluid and responsive Web **apps**,

**What is Typescript?**

TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript, and adds optional static typing to the language. TypeScript is designed for development of large applications and transcompiles to JavaScript.

##### **Static Typing**

Static typed programming languages are those in which variables need not be defined before they’re used. This implies that static typing has to do with the explicit declaration (or initialization) of variables before they’re employed.

**Ivy render**

It is a next generation compilation and rendering pipeline and it redering pipe line is simple and faster to use.

**What is Obserable Pipe ?**

RxJS pipe is used to combine functional operators into a chain. pipe is an instance method of Observable as well as a standalone RxJS function.

pipe accepts operators as arguments such as filter, map, mergeScan etc

**What is Pure Pipes?**

* Angular executes a pure pipe only when it detects a pure change to the input value.
* A pure change is either a change to a primitive input value (String, Number, Boolean) or a changed object reference (Array, Date, Object).
* Pure pipes are Default In angular.
* Pure Pipe working is Fast.
* To make a pipe pure, set it's pure property to true.  
    
  @Pipe({  
      name: 'employeeFilter',  
      pure: true  
  })  
  export class EmployeeFilterPipe implements PipeTransform {

  transform(array: number[], args?: any): any {

     return array.sort((a: number, b: number) => b - a);

   }

}

**What is Impure Pipe?**

An **impure pipe** is called for every change detection cycle no matter whether the value or parameter(s) changes.

To make a pipe impure, set it's pure property to false.  
  
@Pipe ({  
    name: 'employeeFilter',  
    pure: false  
})  
export class EmployeeFilterPipe implements PipeTransform {

 transform(array: number[], args?: any): any {

    return array.sort((a: number, b: number) => b - a);

  }

}