Angular 7 Interview Question and Answers

150+ Q&A

Inclues Angular 7, 6, 5, 4 and 2

Pratik Bandal

Angular 7 Interview Questions and Answers : Includes Angular 7, 6, 5, 4 and 2

Pratik Bandal

Contents

1. Fundamentals

1. What is Angular?
2. What is difference between Angular JS and Angular 2?
3. What is AOT in angular?
4. What is difference between Angular 2 and Angular 4?
5. Why there is no Angular 3?
6. What is Angular CLI?
7. How to install angular CLI?
8. What are IDEs that can be used to develop angular application?
9. What is src folder?
10. What is e2e folder?
11. How to use bootstrap with angular 4?
12. Difference between npm start and ng serve?
13. Explain Angular application flow.
14. Which browser does angular support?
15. What is server side rendering in angular?
16. What are new features in Angular 5?
17. What are new features in Angular 6?
18. What are new features in Angular 7?

2. Modules, Components and Templates

19. What is app-root?
20. What is Module?
21. How to create module from CLI?
22. Can you use component other than root component in index.html?
23. How to create component from CLI?
24. How to use inline template in component?
25. How to add style to your component in html which will be applicable to that particular component only?
26. How to use selector as attribute?
27. How to use selector as class?
28. Using Selector as id allowed in angular?
29. How to use selector as element?

3. Data Binding and Component Communication

30. What is databinding?
31. How to perform string interpolation?
32. How to perform property binding?

33. How to use event binding?
34. How to capture data emitted with event in event binding?
35. How to use two way binding?
36. How to access property of child component from parent component?
37. How to pass data from child component to parent component?
38. How to use event binding for child to parent communication?
39. How to use alias with event?
40. How to perform component communication?
41. What is view encapsulation?
42. How to disable view encapsulation?
43. What is use of local reference?
44. How to access local reference in Typescript code?
45. What is use of ng-content?

4. Directives

46. What are directives?
47. Explain different types of directives.
48. How to use ngIf directive?
49. How to use else with ngIf?
50. How to use ngStyle?
51. How to create directive from CLI?
52. How to use ngClass?
53. How to use ngFor directive?
54. Explain the difference between attribute and structural directive.
55. How to create custom directive?
56. How to create structural directive?
57. How to use ngSwitch?

5. Lifecycle Hooks

58. What is Lifecycle hook in angular?
59. What is ngOnChanges()?
60. What is ngOnInit()?
61. What is ngDoCheck()?
62. What is ngAfterContentInit()?
63. What is ngAfterContentChecked()?
64. What is ngAfterViewInit()?
65. What is ngAfterViewChecked()?
66. What is ngOnDestroy()?

6. Services

67. What are services?
68. How to create services in angular?
69. How to create service from angular CLI?

70. What is dependency injection in angular?
71. What is hierarchical dependency injection in angular?
72. How to inject service into other service?

7. Routing

73. What is Routing?
74. How to setup routes?
75. Which directive should be used for changing route?
76. What is difference between '/somepath' and 'somepath'?
77. How to make selected element active based on selected route?
78. How to programmatically trigger route?
79. How to pass parameters to route?
80. How to fetch parameters in Route?
81. How to add query params to url?
82. How to add query params programmatically?
83. How to retrieve query param and fragment from url?
84. How to add child routes?
85. What is use of Wildcard routes?
86. How to apply guard to route?
87. How to create guard?
88. How to create guard from CLI?
89. How to guard child routes?

8. Http

90. What is HttpClient?
91. Which module is required for HttpClient?
92. How to make get request using HttpClient?
93. How to pass headers to http request?
94. How to make post request using HttpClient?
95. How to make put request using HttpClient?
96. How to catch error of http request?
97. Which module is required for httpService?
98. How to make get request using alternative Http Service instead of HttpClient?
99. What is Observable?
100. What is use of map() operator in observable?
101. How to send post request in angular using alternative Http Service instead of HttpClient?

9. Pipes

102. What are pipes?
103. How to create pipe from CLI?
104. How to use pipe?
105. How to add parameters to a date pipe?

106. What are different built in pipes?
107. How to chain pipes?
108. How to create custom pipe?
109. How to pass parameters to custom pipe?
110. How to create pipe from angular CLI?
111. What is pure pipe and impure pipe?

10. Forms

112. What are approaches to handle form?
113. Which module is required for form?
114. How to access form input controls in Typescript class?
115. What are the important properties of ngForm?
116. How to access form without submit?
117. Why form validation is required?
118. What are different types of validator functions?
119. What are different built-in validators?
120. How to add validation messages for input controls in form?

11. Animation

121. What is animation in angular?
122. Which angular package is required to be installed for animation?
123. How to use animation in angular?

12. Cookies

124. What are Cookies?
125. What is command to install cookies from node?
126. How to use cookies in angular service?

13. Testing

127. What is Jasmine and Karma?
128. Which angular CLI commands is required to run all tests?
129. Which file is used for unit testing in angular?
130. What is TestBed in angular?
131. How to Test service in angular?
132. How to test input property of component in angular?
133. How to test output property of component in angular?
134. Which method is necessary to be called when testing angular pipe?

14. Debugging

135. What is debugging?
136. What is debugger keyword?
137. How to activate debugging in browser?

138. How to get mobile devices view of webpage in desktop browser?
139. How to deactivate breakpoint in browser?
140. How to pause script execution?
141. How to execute function line by line while debugging?
142. How to execute function without stepping into it while debugging?
143. What is code smell?

15. Typescript Basics

144. What is Typescript?
145. Who developed Typescript?
146. Difference between Typescript and Javascript.
147. How to install Typescript?
148. What are built-in datatypes in Typescript?
149. How to create variable in Typescript?
150. What are the variable scopes available in TypeScript?
151. What are Modules in Typescript?
152. Explain interface in Typescript.
153. What is class in Typescript?
154. What are the types of access modifiers supported by TypeScript?
155. What is namespace in Typescript?
156. What is use of question mark (?) with variable in Typescript?
157. What are decorators in Typescript?

1. Fundamentals

1. What is Angular?

Angular is a platform that makes it easy to build applications with the web.

Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges.

Angular empowers developers to build applications that live on the web, mobile, or the desktop.

Angular is Javascript based framework which allows you to create reactive single page application, rendering parts of DOM.

2. What is difference between Angular JS and Angular 2?

Angular JS	Angular 2
Was released in 2010	Was released in 2016
AngularJS is an open-source, JavaScript-based, front-end web application framework for dynamic web app development.	Both Angular 2 and 4 are open-source, TypeScript-based front-end web application platforms.
The architecture of AngularJS is based on model-view-controller (MVC) design. The model is the central component that expresses the application's behaviour and manages its data, logic, and rules.	In Angular 2, controllers and $scope were replaced by components and directives. Components are directives with a template.
AngularJS is written in JavaScript.	Angular uses Microsoft's TypeScript language, which is a superset of ECMAScript 6 (ES6).

Angular JS was not built for mobile devices.	Angular 2 is designed with mobile support in mind.

3. What is AOT in angular?

The Angular Ahead-of-Time (AOT) compiler converts your Angular HTML and TypeScript code into efficient JavaScript code during the build phase before the browser downloads and runs that code. Compiling your application during the build process provides a faster rendering in the browser.

4. What is difference between Angular 2 and Angular 4?

Angular 2	Angular 4
Size of AOT(ahead-of-time) compiler generated code is larger than Angular 4	Size of AOT(ahead-of-time) compiler generated code is smaller than Angular 4
Not compatible with TypeScript 2.1 and 2.2	TypeScript 2.1 and 2.2 Compatibility
Animations package is present in Angular core package.	Animations package is pulled out of Angular core package. This means that in case you don't use animations, this excess code won't end up on your creation packages.
ngIf is uses ! operator to implement else syntax.	ngIf is improved with local reference for implementing else syntax.

5. Why there is no 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.

6. What is Angular CLI?

The Angular CLI is a command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications. You can use the tool directly in a command shell, or indirectly through an interactive UI such as Angular Console.

Angular CLI makes it easy to create an application that already works. It also provides commands to build that project for production and so on,

With angular CLI you can execute commands and few of them are as below,

> ng new: To generate angular application
>
> ng generate: To Generate components, routes, services and pipes with simple command.
>
> ng serve: Easily test your app locally while development.

7. How to install angular CLI?

First install node.js

Then, open node.js command prompt and issue command : npm install –g @angular/cli

8. What are IDEs that can be used to develop angular application?

Webstorm, Visual Studio Code, Sublime are few IDEs that can be used to develop angular application.

9. What is src folder?

In src folder your project is created,

In app folder you can write your code using components, routes, directives, services and pipes.

10. What is e2e folder?

It is required for end to end testing of angular application.

11. How to use bootstrap with angular 4?

Open terminal, navigate to project folder.

Execute npm command: npm install - - save bootstrap

Now verify if node_modules folder contain third party package bootstrap that we have installed.

In angular.json you can see styles array.

Add below line to styles array so that bootstrap can be used in project,

```
"styles":[
        "../node_modules/bootstrap/dist/css/
bootstrap.css",
        .
        .
]
```

12. Difference between npm start and ng serve?

When you run "npm start" it will call whatever command written inside "start" in package.json.

```
"scripts": {
        "ng": "ng",
        "start": "ng serves"
        .
        .
}
```

For example: Above content in package.json will call "ng serve" when npm start command is executed.

13. Explain Angular application flow.

The main.ts is the main file of angular application which bootstraps the app. This main file bootstraps ap-

plication with default module as below:

platformBrowserDynamic().bootstrapModule(AppModule);

As you can see above "AppModule" is the default module which will be picked up during application bootstrap.

The app.module.ts is the default module or you can create your own modules. The AppModule is Typescript class with special decorator or annotation @NgModule which tells angular CLI on how to process this file. So this will be processed as module file which will contain all of your Components, Services etc.

Bootstrap: [AppComponent]

Above property basically tells angular to bootstrap application with the AppComponent which becomes default component.

The AppComponent is picked up during application bootstrap which is basically the default component of angular application. Component is basically view which you see in the browser. AppComponent is Typescript class with decorator "@Component" which contain information about view and how to render component. You can associate component with view using "templateUrl" property of @Component decorator.

14. Which browser does angular support?

Angular supports most recent browsers, like:
Chrome, Firefox, Edge, IE, IE Mobile, Safari, iOS, Android

15. What is server side rendering in angular?

A normal Angular application executes in the *browser*, rendering pages in the DOM in response to user actions. Angular Universal executes on the server, generating static application pages that later get bootstrapped on the client. This means that the application generally renders more quickly, giving users a chance to view the application layout before it becomes fully interactive.

16. What are new features in Angular 5?

Below are major features introduced in Angular 5:

Build optimizer: Production builds created with the CLI in angular 5 will apply the build optimizer by default. The build optimizer is a tool included in our CLI for making your bundles smaller using our semantic understanding of your Angular application.

Angular Universal State Transfer API: ServerTransferState Module and the corresponding BrowserTransferStateModule module allows you to generate information as part of your rendering with platform-server, and then transfer it to the client side so that this information does not need to be regenerated. This is useful for cases such as when your application fetches data over HTTP. By transferring state from the server, this means developers do not need to make a second HTTP request once the application makes it to the client.

Compiler Improvements: This new version brought a lot of improvements in Angular Compiler to make rebuilds of the applications faster, mainly for AOT and production builds. And the TypeScript is also upgraded to the latest version i.e. TypeScript 2.4, which allows connecting to the standard TypeScript compilation pipeline. Tabs, newlines, and spaces in your templates have been faithfully recreated and included in your build by the compiler. You can now choose whether or not to preserve whitespace coming from your components and your application.

Service Workers and PWA: A service worker is a script that runs in the web browser and manages caching for an application. From Angular 5, Angular ships with a service worker implementation. Service workers have made it easier to build Progressive Web Application (PWA), it is now possible to get the feel like native apps with mobile web apps along with add-ons like offline experience and push notifications.

HttpClient: From Angular 5.0 version, developers are recommending HttpClient for all applications, and stop using the previous @angular/http library. To use the updated HttpClient, you should replace the HttpModule with HttpClientModule from @angular/common/http, inject the HttpClient service, and remove

the map(res => rex.json()) calls that is no longer required.

exportAs: Support for multiple names for components / directives has been added.

e.g.

```
@Component({
    moduleId: module.id,
    selector: 'a[mat-button], a[mat-raised-button], a[mat-icon-button], a[mat-fab], a[mat-mini-fab]',
    exportAs: 'matButton, matAnchor',
}
```

Internationalized Number, Date, and Currency Pipes: Number, date, and currency pipes have been built that increase standardization across browsers and eliminate the need for i18n polyfills.

Router Events: The Angular router in version 5 exposes more events during its lifecycle. These events can be used to further hook deeply into the Angular core and thus have more control of your application. The new events are that are available to the Router lifecycle are: ActivationStart, ActivationEnd, ChildActivationStart, ChildActivationEnd, GuardsCheckStart, GuardsCheckEnd, ResolveStart, ResolveEnd

RxJS Update: Angular 5 supports RxJS 5.5.2 or later, due to which way we import RxJS operators has changed.

CLI v1.5: With this version of the Angular CLI, all the projects are generated by default on Angular 5.

Forms Update: In Angular 5 it is possible run validation and value updates on `blur` or on `submit`, instead of on every input event.

e.g.

```
<input name="firstName" ngModel [ngModelOptions]="{updateOn: 'blur'}">
```

or

```
<form [ngFormOptions]="{updateOn: 'submit'}">
```

17. What are new features in Angular 6?

Below are major features introduced in Angular 6:

Angular Elements: Angular is a perfect framework for developing Single Page Applications. The Angular Ele-

ments package will give us the ability to create an Angular component and then publish that component as a web component which can be used in any HTML page (even if that page is not using the Angular framework) in other environments. It actually takes an Angular component and then wraps it within a custom element, such as a DOM element, so that we can use our favourite Angular component in other projects which do not use Angular.

Service Worker Support: Service workers are basically scripts which run in the web browser and manage to cache an application. Service workers were first introduced in Angular 5. In Angular 6, service workers come with some bug fixes, including some new functionalities. Angular 6 now supports the configuration of navigation URLs within the Service Workers. The service worker will redirect navigation requests that don't match any source or data group to the specified index file. Now, we can mention an optional navigationUrls list in ngsw-config.json files which contain the desired URLs. If, for example, a request's URLs match any of the positive patterns and none of the navigate patterns, then it will be considered a navigation request and handled the right way by the service worker. In Angular 6, the service worker remains in the current mode unless the server is reconnected and updates the work.

Template Element: <template> is removed from Angular 6 framework. Instead of using <template>, we need to use <ng-template>.

Tree Shakeable Providers: Angular version 6 introduced a new feature, Tree Shakeable Providers. In the @Injectable decorator we have a new property called providedIn. With this property we can tell Angular which module to register our service to instead of having to import the module and register it to the providers of a NgModule. By default, this syntax registers it to the root injector which will make our service an application wide singleton. The root provider is a reasonable default for most use cases. If you still need to control the number of instance of a Service the regular providers API is still available on Angular Modules and Components.

ngModelChange: Before Angular 6, the ngModelChange event was emitted before the said form

control updating. If we have an event handler for the ngModelChange event that checked the value through the control, the old value will be returned instead of the changed value. Now, in Angular 6, ngModelChange has emitted the value after the value is updated in the form control.

ng update & ng add: The Angular CLI gets two major new commands: update and add. With ng update, updating an Angular app is now just a command away. 3rd-party libraries can also hook into the new command, thanks to schematics, to make it easy to update those libraries as well. Under the hood the command will work with the project's package.json file and use the project's package manager (npm or Yarn) as well as apply code transforms where needed to update the code or configuration files where necessary. Similar to ng update, the new ng add command allows to add and configure new libraries into a project with a simple command. For example, installing and setting up Angular Material used to require quite a few steps, but now it can all be done simply with one command.
e.g.

```
ng add @angular/material
```

Other CLI changes: Angular projects that use the CLI had a configuration file called .angular-cli.json previously. With version 6 of the CLI, this file is now renamed to angular.json and its structure also changes. ng new now generates a workspace with a default application. Additional apps can be added to a workspace, making it possible to have multiple apps under one project.

Materials: Angular 6 has come up with a separate module @angular/materials.It offer features such as autocomplete, datepicker, slider, menus, grids, toolbar, sidenav, dashboard and datatable are available for use with materials in Angular 6. To start using materials, you need to install two packages - @angular/material and @angular/cdk.

RxJS 6.0: Angular 6 now used RxJS 6 internally. So we need to update our application accordingly. These changes provide developers an increase in performance and are easier to debug AJAX call stacks and im-

prove modularity also, making it as backward compatible as possible. But RxJS changed the way we import things.

18. What are new features in Angular 7?

Below are major features introduced in Angular 7 :

CLI prompts: In Angular 7, the CLI prompts have been updated to v7.0.2 with new features. The CLI will now prompt users when running common commands like ng new or ng add @angular/material to help you discover built-in features like routing or SCSS support.

Angular material & component dev kit (CDK): Angular Material users updating to v7 should expect minor visual differences reflecting the updates to the Material Design specification. Virtual Scrolling and Drag and Drop are now available.

Virtual Scrolling: Virtual Scrolling in Angular 7 basically loads and unloads items from the DOM depending upon visible parts of lists, Simply put, it activates a high-performant way by making the height of container element exactly same as the height of total number of remaining elements to be rendered. This, in turn, then renders the only visible items in view, creating faster experiences for the end-users.

Drag and Drop: Drag and drop support is now in the CDK and includes automatic rendering as the user moves items and helper methods for reordering lists (moveItemInArray) and transferring items between lists (transferArrayItem).

2. Modules, Components and Templates

19. What is app-root?

app-root is selector of app component which is placed as tag in index.html.

20. What is Module?

Angular uses module to bundle pieces i.e. components to packages. It gives angular information about functionalities i.e. components, pipes that application has.

We use @NgModule to decorate module. We import @NgModule from @angular/core.

To register component in module we can use declarations property of @NgModule and import it to tell Typescript its location.

e.g.

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

Imports property is used to import other modules to current module.

21. How to create module from CLI?

Below is angular CLI command which creates angular module :

ng generate module <name>

22. Can you use component other than root component in index.html?

No.

23. How to create component from CLI?

ng generate component <componentname>
Above command will create folder with <componentname> in app folder with ts, html, css, spec file.It will also add it to declarations property in AppModule and import it to the AppModule.

24. How to use inline template in component?

Use "template" instead of "templateUrl" in @Component decorator. If you want to write multiline html code then use ` backtick with "template" property.

25. How to add style to your component in html which will be applicable to that particular component only?

This can be achieved by using "styles" instead of "styleUrls" property of @Component decorator, both properties accept array.
e.g.

```
@Component({
            Selector: 'search',
            stylesUrls: ['./search.component.less'],
            .
            .
})
```

26. How to use selector as attribute?

Enclose selector with square braces.
e.g.

```
@Component({
            selector: '[search]',
            .
            .
})
```

And then use it as html attribute <div search> .

27. How to use selector as class?

Add dot (.) at the start of selector name.

```
@Component({
            selector: '.search',
            . .
```

```
})
```

And then use it in html as class .

28. Using Selector as id allowed in angular?

No.

29. How to use selector as element?

Use selector property of @Component decorator.

e.g.

```
@Component({
            selector: 'search',
            .
            .
})
```

And then use it as element in html <search></search>.

3. Data Binding and Component Communication

30. What is databinding?

Databinding is basically communication between Typescript code and template.

Below are different types of databinding's.

To output data from Typescript to html.

- String Interpolation.
- Property Binding.

To react to events from templates

- Event binding.

Combination of both

- Two way binding.

31. How to perform string interpolation?

To perform string interpolation write {{ expression }} in html. The expression can be Typescript code property or it can be hardcoded string or it can be method which returns string. You can use ternary expression as well, but you can't write multiline expression inside {{ }}. Number is also allowed inside {{}} as number can be resolved to string.

32. How to perform property binding?

Property of element in html is enclosed by [] & assign it to expression or property in Typescript component, then based on property value in Typescript code element property changes.

e.g.

```
<button [disabled]="!disabledVar"></button>
```

In Above example, "disabledVar" is variable in Typescript class.

33. How to use event binding?

Event binding uses () parenthesis and specify the name of event within parenthesis, you can bind all events made available by html. Assign event to the

method which you want to execute when event occurs. You can also assign event to inline code.

e.g.

```
<button [disabled]="!disabledVar" (click)="onClickMethod()"></button>
```

34. How to capture data emitted with event in event binding?

We can capture data emitted with event in event binding using $event as method input,

e.g.

```
<input type="text" (input)="onUpdate($event)"></input>
```

Through $event data will be received in method with data type as "Event".

35. How to use two way binding?

Use [(ngModel)] where ngModel is the special directive and assign it to property in Typescript and display same property on html using string interpolation.

e.g.

```
<input type="text" [(ngModel)]="textTypedInTextBox"></input>
<div>{{textTypedInTextBox}}</div>
```

36. How to access property of child component from parent component?

For this you can use custom property binding. In child component element tag use property to be used in square brackets[] as property binding assign value to it.

e.g.

```
<app-child-element [someproperty]="true"></app-child-element>
```

Then add decorator @Input to the child component property in Typescript file, otherwise property will not be accessible to outside components. @Input is part of angular core. This is how custom property binding is done.

In this case, we are passing data from parent component to child component.

37. How to pass data from child component to parent component?

This can be achieved using event binding.

38. How to use event binding for child to parent communication?

Create properties in child component and assign them EventEmitter.

e.g.

```
onUpdate = new EventEmitter<>();
```

Setup what event should emit using methods in child class.

e.g.

```
onUpdateMethod() {
    this.onUpdate.emit({
        propertyToBeEmitted:    this.propertyTo-
    BeEmitted
    });
}
```

Add decorator @Output against property in child component.

e.g.

```
@Output
onUpdate = new EventEmitter<>();
```

Use this event in parent html with child component tag.

e.g.

```
<app-child-element                (onUpdate)="onUp-
dateMethod($event)"></ app-child-element>
```

39. How to use alias with event?

Pass argument to @output as

```
@Output('someArgument')
```

40. How to perform component communication?

Component communication can be performed through property binding, event binding and services dependency injection.

41. What is view encapsulation?

Styles in css files written in component are applied to elements of that particular component only not to its child component, this is called view encapsulation.

42. How to disable view encapsulation?

View encapsulation can be disabled by adding encapsulation : ViewEncapsulation.None in the @Component decorator.

This will apply styles of that component to entire application.

43. What is use of local reference?

Local reference passes element with all its properties to method.

e.g.

```
<input type="text" #inputLocalReference/>
<button
(click)="onAddServer(inputLocalReference)"></button>
```

44. How to access local reference in Typescript code?

Local reference can be accessed in Typescript code by using @ViewChild decorator.

e.g.

```
@ViewChild('someLocalReference') someLocalReference;
```

Here, someLocalReference is local reference. This property can now be used in Typescript code as:

```
var abc = this.someLocalReference.nativeElement.value;
```

45. What is use of ng-content?

By default anything placed inside opening and closing tag of component is last to be displayed. To display such content you should use <ng-content>.

Consider a simple <component-content> component: e.g.

```
<div>
    Hi, This is components html !!!
    <ng-content></ ng-content >
</div>

<component-content>
    <div>This text will be displayed inside ng-content</div>
</component-content>
```

The HTML content passed within the opening and closing tags of <component-content> component is the content to be projected. This is what we call Content Projection. The content will be rendered inside the <ng-content>within the component.

4. Directives

46. What are directives?

Directives are instructions in DOM, Components are directives as well.

Directive has selector which can be used as attribute as well as element in html.

Directives are more generally used as attribute.
e.g.

```
@Directive({
        selector:'[makeAppFancy]'
})

<p makeAppFancy></p>
```

Above example shows how to create and use custom directive. Angular provides few build in directives as well like ngModel, ngIf etc.

There are three kinds of directives:

- Component directives
- Structural directives
- Attribute directives

47. Explain different types of directives.

There are three kinds of directives in Angular:

Component directives: Directives with html template.

Structural directives: Change the DOM layout by adding and removing DOM elements. e.g. ngFor, ngIf.

Attribute directives: Change the appearance or behaviour of an element, component, or another directive. e.g. ngStyle.

48. How to use ngIf directive?

ngIf is structural directive. For structural directives * is used before directive.

```
<div *ngIf="isRole">
        <h3>Hi User</h3>
</div>
```

In above example only if isRole variable is true then only "Hi User" text will be displayed on UI.

49. How to use else with ngIf?

Create <ng-template> with local reference and in else place that local reference.

e.g.

```
<div *ngIf="isRole";else noRole>
            <h3>Hi User</h3>
</div>
<ng-template #noRole>
            <h3>Hi Unknown</h3>
</ng-template >
```

Another alternative could be to use ngIf with ! Expression.

e.g.

```
<div *ngIf="isRole >
            <h3>Hi User</h3>
</div>
<div *ngIf="!isRole" >
            <h3>Hi Unknown</h3>
</div>
```

50. How to use ngStyle?

ngStyle is used as property of html element. It is attribute directive. It is used with square brackets []. [] are not part of directive. ngStyle is then style. The value to style can be string or method that return string. The assigned value is Javascript object with key value pair.

e.g.

```
<p
[ngStyle="backgroundColor:getColor()"]>ExampleTex
t</p>
```

51. How to create directive from CLI?

Below angular command is used to create a new generic directive definition in the given or default project.

ng generate directive <name> or

ng g d <directivename>.

52. How to use ngClass?

ngClass is used as property of html element. It is attribute directive. It is used to dynamically add css class to element. ngClass is used with with property binding with square brackets [].The assigned value is Javascript object with key value pair.

e.g.

```
<p [ngClass]="{'classname': person.country === 'UK'}">ExampleText</p>
```

53. How to use ngFor directive?

ngFor is structural directive. . For structural directives * is used before directive.

It is used with 'let'.

e.g.

```
<li *ngFor="let country of countryListArray">{{ country}}</li>
```

It will loop "li" element for all countryListArray. You can obtain current index of iteration using "index" var.

e.g.

```
<li *ngFor="let country of countryListArray; let i = index">{{country}}</li>
```

54. Explain the difference between attribute and structural directive.

Attribute	Structural
Modifies the appearance or behaviour of an element	Modifies the DOM layout
Examples: ngStyle , ngClass	Examples: ngIf, ngFor

55. How to create custom directive?

Create transcript class.

Use @Directive on class.

Use selector inside @Directive, give appropriate name to selector.

Implement onInit interface.

In constructor of class pass elementRef.

In ngOnInit change style of elementRef, this can be done in constructor as well.

e.g.

```
@Directive({
    selector: '[appCustomDirectiveExample]'
})
Export class CustomDirectiveExample implements
OnInit {
    Constructor(private elementRef : ElementRef)
{}
    ngOnInit() {
        this.elementRef.nativeElement.style.color
    = 'red';
    }
}
```

56. How to create structural directive?

Create directive using angular CLI.

Use @Input decorator with property, this property will receive value which will be treated as condition.

Pass templateRef and viewContainerRef to constructor.

Use templateRef and viewContainerRef to modify DOM based on condition.

e.g.

```
@Directive({
  selector: '[cpLoop]'
})
export class CpLoopDecorator {
  constructor( private templateRef: TemplateRe-
f<any>,
  private viewContainer: ViewContainerRef) { }
  @Input('cpLoop') set loop(num: number) {
    for(var i=0; i < num; i++) {
        this.viewContainer.createEmbedded-
      View(this.templateRef);
```

```
        }
    }
}
```

Place directive selector on DOM element.

```
<ul>
    <li *cpLoop="5" >
        Hello World!
    </li>
</ul>
```

57. How to use ngSwitch?

```
<div [ngSwitch]="value">
        <p *ngSwitchCase="1">Value is 1</p>
        <p *ngSwitchCase="2">Value is 1</p>
    <p *ngSwitchDefault>Value is Default</p>
</div>
```

5. Lifecycle Hooks

58. What is Lifecycle hook in angular?

A component has a lifecycle managed by Angular.

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

Angular offers lifecycle hooks that provide visibility into these key life moments and the ability to act when they occur.

59. What is ngOnChanges()?

It is lifecycle hook method. It is executed multiple times. It is executed when component is created. It is also called when one of the input property changes i.e. properties decorated with @Input.

60. What is ngOnInit()?

It is lifecycle hook method which will be executed once component object is created. It is executed after constructor.

61. What is ngDoCheck()?

It is lifecycle hook method which will run on every change detection. e. g. any click, any input check.

62. What is ngAfterContentInit()?

It is lifecycle hook method which is called after ng-content has been projected in the view.

63. What is ngAfterContentChecked()?

It is lifecycle hook method which is called after Angular checks the content projected into the component.

64. What is ngAfterViewInit()?

It is lifecycle hook method which is called when components view has been initialized or rendered.

65. What is ngAfterViewChecked()?

It is a likecycle hook method which is called every time Angular has finished running change detection on a component.

66. What is ngOnDestroy()?

It is lifecycle hook method which is called once component is about to be destroyed.

This method is useful for clean-up work, unsubscribing Observables and detaching event handlers to avoid memory leaks.

6. Services

67. What are services?

Services are basically used to reduce duplication of code. E.g. for logging you can create service and use it everywhere. Services are used for component communication.

68. How to create services in angular?

Create .ts file.

Write Typescript class, no decorator is required.

Use constructor dependency injection in the class where we want to use this service.

e.g.

```
constructor(private customServiceExample : CustomServiceExample) {}
```

In @Component use providers property and specify type of service which component depends on.

e.g.

```
@Component({
  selector:
  templateUrl:
  providers: [CustomServiceExample]
})
```

Use this service as,

```
this.customServiceExample.doSomething("some action").
```

69. How to create service from angular CLI?

Below commands are used to creates a new, generic service definition in the given or default project.

ng generate service <name> or

ng g service <name>

70. What is dependency injection in angular?

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. The DI framework lets you supply data to a component from an injectable *service* class, defined in its own file.

e.g.

```
import { Injectable } from '@angular/core';
import { HEROES } from './mock-heroes';

@Injectable({
    // we declare that this service should be created
    // by the root application injector.
    providedIn: 'root',
    })
    export class HeroService {
    getHeroes() { return HEROES; }

}
```

The @Injectable() decorator marks it as a service that can be injected, root injector which will make our service an application wide singleton .

But Angular can't actually inject it anywhere until you configure an Angular dependency injector with a provider of that service.

You can tell Angular to inject a dependency in a component's constructor by specifying a constructor parameter with the dependency type.

e.g.

```
constructor(heroService: HeroService)
```

71. What is hierarchical dependency injection in angular?

The Angular dependency injection system is hierarchical. There is a tree of injectors that parallels an app's component tree. You can reconfigure the injectors at any level of that component tree.

If service is provided using providers array to AppModule then service is available application wide, which means it is available to all component, services and directives. Similarly When you use providedIn:'root' of @Injectable() decorator, you are configuring the root injector for the app, which is the injector for AppModule due to which service is available application wide

If service is injected into any component then it will be available to its child component and to that particular component. You can limit the scope of a provider to a component and its children by con-

figuring the provider at the component level using the @Component metadata.

You can configure a provider at the module level using the providedIn metadata option for a non-root NgModule, in order to limit the scope of the provider to that module. This is the equivalent of specifying the non-root module in the @Injectable() metadata. You generally don't need to specify AppModule with-providedIn, because the app's root injector is the AppModule injector. However, if you configure a app-wide provider in the@NgModule() metadata for AppModule, it overrides one configured for root in the @Injectable()metadata.

If service is provided in AppModule and AppComponent as well as particular component nearest dependency injection will override other dependency injections.

72. How to inject service into other service?

Use @Injectable() with service which you want to inject into other service.

7. Routing

73. What is Routing?

When url changes, based on url major part of DOM also changes. This is called routing.

74. How to setup routes?

Add constant array of type Routes in AppModule, add routes to it as Javascript object.

e.g.

```
const routes_config : Routes = [
        {path: 'roles', component: RoleComponent},
        { path: 'home', component: HomeComponent }
 ]
```

Add RouterModule to imports property of @NgModule as,

```
Imports: [
        RouterModule.forRoot(routes_config);
]
```

Pass routes_config array to forRoot method of RouterModule.

Use directive <router-outlet> in app.component.html where you want to render the component based on route.

e.g.

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

75. Which directive should be used for changing route?

routerLink should be used for changing route.

e.g.

```
<a routerLink='roles'></a>
<a [routerLink]="['users', 'something']" ></a>
```

routerLink will not reload page unlike href and also state of application will persist.

76. What is difference between '/somepath' and 'somepath'?

If '/somepath' is used with routerLink it will be absolute path whereas 'somepath' will be relative path.

77. How to make selected element active based on selected route?

routerLinkActive is used for this purpose.

```
<li routerLinkActive = "active">
<a routerLink='/users'></a></li>
[routerLinkActiveOptions]="{exact: true}"
```

routerLinkActiveOptions can be used with router-LinkActive can be used with routerLinkActive for exact match of path, this is required as '/' is part of all routes.

78. How to programmatically trigger route?

Using navigate method of Router.

First Inject router in constructor and then use navigate method of router.

e.g.

```
this.router.navigate(['somepath']);
```

To make this route relative you can use relativeTo property.

e.g.

```
this.router.navigate(['servers'], {relativeTo: this.route});
```

To get currently active route use active route service.

79. How to pass parameters to route?

By using colon ':' in route path.

e.g.

```
{path : 'user/:id', component : UserComponent}.
```

80. How to fetch parameters in Route?

In Typescript file inject ActivedRoute in constructor.

Access params from activatedRoute.

e.g.

```
this.activatedRoute.snapshot.params['id'];
```

'id' must be specified as dynamic param in route in AppModule.

81. How to add query params to url?

In <a> anchor tap use [queryparam] property of routerLink as property binding.

e.g.

```
<a [routerLink]="['/somepath,5,'update']"
[queryParams]="{allowUpdate: '/'}"

[fragment]="loading"]></a>
```

In Above case url will look like

localhost:4200/somepath/5/update/allowUpdate=1#loading

82. How to add query params programmatically?

By using navigate method of Router.

e.g.

```
this.router.navigate(['/somepath', id, 'update'],
{queryParams: {allowUpdate:'1'}, fragment: 'loading'});
```

In above case url will be

localhost:4200:/somepath/1/update?allowUpdate=1#loading

83. How to retrieve query param and fragment from url?

By using queryParam and fragment from snapshot property of ActivatedRoute.

e.g.

```
this.activatedRoute.snapshot.queryParams
this. activatedRoute.snapshot.fragment
```

By subscribing to queryParam and fragment of activated route.

e.g.

```
this.activatedRoute.queryParams.subscribe();
this.activatedRoute.fragment.subscribe();
```

84. How to add child routes?

By using 'children' property of Routes.

e.g.

```
const appRoutes : Routes = [
        { path : 'users', component: UserComponent,
        children:[
                        {path:'id', component: UserCom-
ponent },
                        {path:'id/update', component: Up-
dateUserComponent}
        ]}
]
```

After children routes are added add <router-outlet> in parent component html. In this <router-outlet> child's will be loaded.

85. What is use of Wildcard routes?

Wild card route is basically used to handle all routes which are not present in Routes array, it should be at the bottom of Routes array.

e.g.

```
{path: 'error', component:ErrorPageComponent},
{path:'**', redirectTo:'/error'}
```

86. How to apply guard to route?

By using 'canActivate' property in routes.

e.g.

```
{
   path: 'users',
   canActivate:[RoleGuard],
   component: UserComponent
 }
```

87. How to create guard?

Create service with particular guard name.e.g. RoleGuard.

Implement CanActivate interface in this service.

Implement canActivate method. This method will return Observable<boolean> or Promise<boolean> or Boolean.

In canActivate method write business logic which will confirm if route should be accessed or not.

```
canActivate(route:ActivatedRouteSnapshot,state:RouterStateSnapshot) :Observable<boolean>|Promise<boolean>|boolean {
  return true;
}
```

88. How to create guard from CLI?

Below is command to create a new, generic route guard definition in the given or default project:

```
ng generate guard <name> [options] or
ng g guard <name> [options]
```

89. How to guard child routes?

Add 'canActivateChild' property in route instead of 'canActivate'.

Implement canActivateChild interface in guard.

Implement canActivateChild method, it has same return type as canActivate.

8. Http

90. What is HttpClient?

Most front-end applications communicate with back-end services over the HTTP protocol, HttpClient in @angular/common/http is used to send HTTP requests or make API calls to RESTful endpoints of remote servers in order to fetch data.

91. Which module is required for HttpClient?

HttpClientModule is required to be imported in AppModule before using HttpClient.

92. How to make get request using HttpClient?

import HttpClientModule into the AppModule.

inject the HttpClient into constructor of service where you want to use it.

e.g.

```
constructor(private httpClient: HttpClient) { }
```

Write a method in service and in this method, use get() method of HttpClient and subscribe to it.

e.g.

```
getSomeData() {
                this.httpClient.get('http://local-
host:4200/getService');
}
```

Call service method in component where data is needed to be displayed and subscribe to it.

```
showSomeData() {
  this.someDataService.getSomeData()
    .subscribe((data: SomeData) => this.component-
Data = {
        name: data['name'],
        address: data['address']
    });
}
```

subscription callback copies the data fields into the component's object, which is data-bound in the component template for display.

93. How to pass headers to http request?

Create header object

e.g.

```
const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type': 'application/json',
      'Authorization': 'my-auth-token'
  })
};
```

Add this headers to http request.

e.g.

```
this.http.post('http://localhost', data, {headers: headersObj });
```

94. How to make post request using HttpClient?

import HttpClientModule into the AppModule.

inject the HttpClient into constructor of service where you want to use it.

e.g.

```
constructor(private httpClient: HttpClient) { }
```

Create Header Options

e.g.

```
const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type': 'application/json',
      'Authorization': 'my-auth-token'
  })
};
```

Write a method in service and in this method use post() method of HttpClient passing url, data and http options to it, then you take the Observables returned by the HttpClient methods and pipe them through to the error handler.

e.g.

```
addSomeData(data: SomeData): Observable< SomeData > {
```

```
        this.httpClient.post('http://localhost:4200/get-
        Service', data, httpOptions).pipe(
            catchError(this.handleError);
        );
    }
```

The Component initiates the actual POST operation by subscribing to the Observable returned by this service method.

e.g.

```
this.someDataService.addSomeData(newData).sub-
scribe(data => this.dataArray.push(data));
```

95. How to make put request using HttpClient?

Create service.

Inject http service into constructor of service

e.g.

```
constructor(private http : Http){}
```

Make http request using post method of http.

e.g.

```
putMethod(users: User) {
                this.http.post('http://localhost:8080',
users);
}
```

Put method output is observable.

Create subscriber to this observable, else http request will not be sent.

e.g. `this.putMethodService.putMethod(users).subscribe((responce)=>console.log(responce));`

96. How to catch error of http request?

You could handle in the component by adding a second callback to the .subscribe():

e.g.

```
showConfig() {
  this.configService.getConfig()
    .subscribe(
      (data: Config) => this.config = { ...data }, // suc-
cess path
      error => this.error = error // error path
    );
```

```
}
```

97. Which module is required for httpService?

HttpModule is required to be imported in app module.

98. How to make get request using alternative Http Service instead of HttpClient?

Create service.

Inject http service into constructor of service

e.g.

```
constructor(private http : Http){}
```

Make http request using get method of http.

e.g.

```
getMethod() {
                this.http.get('http://localhost:4200/
getService');
}
```

Get method output is observable.

Create subscriber to this observable, else http request will not be sent.

e.g.

```
this.http. getMethod().subscribe((responce)=>{
      console.log(response.json());
},(error)=>console.log(error));
```

99. What is Observable?

Observable is basically datasource. In angular we import observable from third party package like 'rxjs'. It is used to handle asynchronous tasks e.g. observable is used with http service. Observable also has 'observer' which is implemented using subscribe method. Observer handles data in three ways : Handle normal data, handle error, handle completion. In the handling method you can perform your actions.

100. What is use of map() operator in observable?

map() operator is basically used to transform response data.

e.g.

```
this.http.get('http://localhost:4200/myser-
vice').map(
    (response : Responce) => {
        const formattedData = response.json();
        return formattedData;
    }
);
```

map is useful because it returns transformed observable to all observers and hence uniformity to all observers receiving data.

101. How to send post request in angular using alternative Http Service instead of HttpClient?

Create service.

Inject http service into constructor of service
e.g.

```
constructor(private http : Http){}
```

Make http request using post method of http.
e.g.

```
postMethod(users: User) {
            this.http.post('http://localhost:8080',
users);
}
```

Post method output is observable.

Create subscriber to this observable, else http request will not be sent.
e.g. this.postMethodService.postMethod(users).subscribe((responce)=>console.log(responce));

9. Pipes

102. What are pipes?

A pipe takes in data as input and transforms it to a desired output.

103. How to create pipe from CLI?

Below is the command that creates a new generic pipe definition in the given or default project:

```
ng generate pipe <name> [options]
ng g pipe <name> [options]
```

104. How to use pipe?

Pipe is used with string to be transformed after pipe char.

e.g.

```
<p>The hero's birthday is {{ birthday | date }}</p>
```

Inside the interpolation expression, you flow the component's birthday value through the pipe operator (|) to the Date pipe function on the right. All pipes work this way.

105. How to add parameters to a date pipe?

A pipe can accept any number of optional parameters to fine-tune its output. To add parameters to a pipe, follow the pipe name with a colon (:) and then the parameter value (such as date:"MM/dd/yy").

```
<p>The hero's birthday is {{ birthday | date:"MM/dd/yy" }} </p>
```

106. What are different built in pipes?

currency, date, decimal, json, lowercase, uppercase, percent, slice are few of the built in pipes.

107. How to chain pipes?

Pipes can be chained by repeatedly using pipes.

e.g.

```
<p>The chained hero's birthday is {{ birthday |
date | uppercase}}<p>
```

Pipes are applied left to right.

108. How to create custom pipe?

Create class and implement PipeTransform interface which will force you to use transform method.

In transform method write transformation logic and return transformed data.

Use @Pipe decorator on this class and give name to the pipe using name property of @Pipe.

e.g.

```
@Pipe({name: 'exponentialStrength'})
export class ExponentialStrengthPipe implements PipeTransform {
    transform(value: number, exponent: string): number {
        let exp = parseFloat(exponent);
        return Math.pow(value, isNaN(exp) ? 1 : exp);
    }
}
```

Use this pipe in template as, Super power boost: {{2 | exponentialStrength: 10}} .

109. How to pass parameters to custom pipe?

Add argument to transform method.

transform(value: number, exponent: string)

In template pass parameter to pipe using colon ":".

Super power boost: {{2 | exponentialStrength: 10}} .

110. How to create pipe from angular CLI?

ng g p <pipename>

111. What is pure pipe and impure pipe?

There are two categories of pipes: pure and impure.

Pipes are pure by default.

Pure pipe: Angular executes a pure pipe only when it detects a pure change to the input value.

Impure pipe: Angular executes an impure pipe during every component change detection cycle. An impure pipe is called often, as often as every keystroke or mouse-move.

You make a pipe impure by setting its pure flag to false.

10. Forms

112. What are approaches to handle form?

Angular provides two different approaches to handling user input through forms: reactive and template-driven. Both capture user input events from the view, validate the user input, create a form model and data model to update, and provide a way to track changes.

Reactive forms are more robust: they're more scalable, reusable, and testable. If forms are a key part of your application, or you're already using reactive patterns for building your application, use reactive forms.

Template-driven forms are useful for adding a simple form to an app, such as an email list signup form. They're easy to add to an app, but they don't scale as well as reactive forms. If you have very basic form requirements and logic that can be managed solely in the template, use template-driven forms.

113. Which module is required for form?

FormModule.

114. How to access form input controls in Typescript class?

On form tag create ngSubmit event and write method against it.

Create local reference to form element and assign it values of form using ngForm and pass it to submit method.

e.g.

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">
```

Now if we submit form, form values will be available to onSubmit() method. All values will be available in "value" property of "ngForm".

115. What are the important properties of ngForm?

In Angular 4, the following statuses are commonly used by forms:

valid – state of the validity of all form controls, true if all controls are valid

invalid – inverse of valid; true if some control is invalid

pristine – gives a status about the "cleanness" of the form; true if no control was modified

dirty – inverse of pristine; true if some control was modified

disabled - Reports whether the control is disabled.

enabled - Reports whether the control is enabled.

touched - Reports whether the control is touched, meaning that the user has triggered a blur event on it.

116. How to access form without submit?

This can be done by using ViewChild decorator and using localReference of form argument to it.

e.g.

```
@ViewChild('someLocalReference') anyForm : NgForm;
```

Then access it in required method.

e.g.

```
this.anyForm;
```

117. Why form validation is required?

To improve overall data quality by validating user input for accuracy and completeness.

118. What are different types of validator functions?

There are two types of validator functions: sync validators and async validators.

Sync validators: functions that take a control instance and immediately return either a set of validation errors or null. You can pass these in as the second argument when you instantiate a FormControl.

Async validators: functions that take a control instance and return a Promise or Observable that later emits a set of validation errors or null. You can pass these in as the third argument when you instantiate a FormControl.

119. What are different built-in validators?

min: Validator that requires the control's value to be greater than or equal to the provided number.

max: Validator that requires the control's value to be less than or equal to the provided number.

required: Validator that requires the control have a non-empty value.

email: Validator that requires the control's value pass an email validation test.

minLength: Validator that requires the length of the control's value to be greater than or equal to the provided minimum length.

maxLength: Validator that requires the length of the control's value to be less than or equal to the provided maximum length.

pattern: Validator that requires the control's value to match a regex pattern.

120. How to add validation messages for input controls in form?

Use span which will display error message along with input control.

e.g.

```
<input name="email" required email #email="ngModel"/>
<span *ngIf="!email.valid && email.touched">Please enter valid email ID!!!</span>
```

11. Animation

121. What is animation in angular?

Animation provides the illusion of motion: HTML elements change styling over time. Well-designed animations can make your application more fun and easier to use

122. Which angular package is required to be installed for animation?

@angular/animations is required for animation. You can install it from angular CLI using command npm install @angular/animations@latest --save

123. How to use animation in angular?

If you plan to use specific animation functions in component files, import those functions from @angular/animations.

In the component file, add a metadata property called animations: within the @Component() decorator. You put the trigger that defines an animation within the animations metadata property.

Use the style() function to define a set of styles to associate with Angular's state() function to define different states to call at the end of each transition.

e.g.

```
state('open', style({
      height: '200px',
      opacity: 1,
      backgroundColor: 'yellow'
})),
state('closed', style({
      height: '100px',
      opacity: 0.5,
      backgroundColor: 'green'
})),
```

Then, use the transition() function which accepts two arguments: the first argument accepts an expression that defines the direction between two transi-

tion states, and the second argument accepts an animate() function.

The animate() function (second argument of the transition function) accepts the timings and styles as input parameters.

e.g.

```
transition('open => closed', [ animate('1s') ]),
```

Above example provides a state transition from open to closed with a one second transition between states.

An animation requires a trigger, so that it knows when to start. The trigger() function collects the states and transitions, and gives the animation a name, so that you can attach it to the triggering element in the HTML template.

e.g.

```
@Component({
  selector: 'app-open-close',
  animations: [
      trigger('openClose', [
      // ...
          state('open', style({
              height: '200px',
              opacity: 1,
              backgroundColor: 'yellow'
          })),
          state('closed', style({
              height: '100px',
              opacity: 0.5,
              backgroundColor: 'green'
          })),
          transition('open => closed', [
              animate('1s')
          ]),
          transition('closed => open', [
              animate('0.5s')
          ]),
      ]),
  ],
  templateUrl: 'open-close.component.html',
  styleUrls: ['open-close.component.css']
})
```

```
export class OpenCloseComponent {
  isOpen = true;
  toggle() {
      this.isOpen = !this.isOpen;
  }
}
```

The following code snippet binds the trigger to the value of the isOpen property.

```
<div [@openClose]="isOpen ? 'open' : 'closed'"
class="open-close-container"> <p>The box is now
{{ isOpen ? 'Open' : 'Closed' }}!</p> </div>
```

12. Cookies

124. What are Cookies?

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

- When a user visits a web page, his/her name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his/her name.

125. What is command to install cookies from node?

npm install –save ngx-cookie-service

126. How to use cookies in angular service?

Import CookieService in app module.

Add CookieService to providers property of @NgModule decorator of AppModule.

Inject cookie service in angular component using constructor.

e.g.

```
constructor(private cookie : CookieService) {}
```

Use event binding to fetch information coming from UI which is required to be set in cookie.

Set information coming from UI to cookie using set() method of cookie.

e.g.

```
this.cookie.set("username",username);
```

Get method can be used to fetch data stored in cookie.

e.g.

```
this.cookie.get("username");
```

13. Testing

127. What is Jasmine and Karma?

Jasmine is test framework for angular and karma is test runner.

128. Which angular CLI commands is required to run all tests?

ng test

129. Which file is used for unit testing in angular?

.spec.ts file is is used for unit testing of respective component in angular.

130. What is TestBed in angular?

TestBed configures and initializes environment for unit testing and provides methods for creating components and services in unit tests.

131. How to Test service in angular?

Import service which needs to be unit tested in .spec.ts file.

Then we declare describe() for the tests.

Then create variable which is going to hold the instance of our service class.

Then, we just create a new instance of our Service.

Then in it() we call method of service which needs to be tested and also set the expectation what should be returned.

e.g.

```
import { UserService } from './user.service';

describe('UserService( no testbed)', () => {
    //variable declaration
    let service : UserService;
    beforeEach( ()=>{
        service = new UserService();
    })
```

```
    it('#getUsername should return Pratik Bandal', ()=>{
        expect(service.getUsername()).toBe('Pratik Bandal')
    })
});
```

132. How to test input property of component in angular?

To test inputs we need to do things:

- We need to be able to change the input property enabled on our component.
- We need to check that the button(element) is enabled or disabled depending on the value of our input property.

Just because it's an @Input doesn't change the fact it's a still just a simple property which we can change like any other property, like so:

e.g.

```
it('Setting enabled to false disables the submit button', () => {
    component.enabled = false;
});
```

For the second we need to check the disabled property value of the buttons DOM element like so:

e.g.

```
it('Setting enabled to false disables the submit button', () => {
    component.enabled = false;
    fixture.detectChanges();
    expect(submitEl.nativeElement.disabled).toBeTruthy();
});
```

133. How to test output property of component in angular?

Since the output event is actually an Observable we can subscribe to it and get a callback for every item emitted.

We store the emitted value to a component object and then add some expectations on the component object.

e.g.

```
it('Entering email and password emits loggedIn event', () => {
    let user: User;
    loginEl.nativeElement.value = "test@example.com";
    passwordEl.nativeElement.value = "123456";

    component.loggedIn.subscribe((value) => user = value);

    submitEl.triggerEventHandler('click', null);

    expect(user.email).toBe("test@example.com");
    expect(user.password).toBe("123456");
});
```

In above example we set email and password input controls in the view, then we subscribe to component.

Next we trigger a click on the submit button, this synchronously emits the user object in the subscribe callback!

Then we use expect() method to test values that we have received.

134. Which method is necessary to be called when testing angular pipe?

transform();

14. Debugging

135. What is debugging?

Debugging is the process of detecting and fixing existing and potential errors in software code that can cause it to behave unexpectedly.

To debug a program, programmer has to start with problem, identify source of the problem and then fix it.

Sometimes it takes more time to debug a program than to code it.

136. What is debugger keyword?

Debugger statement stops the execution. If no debugging functionality is available, this statement has no effect.

Debugger keyword is like breakpoint in source code.

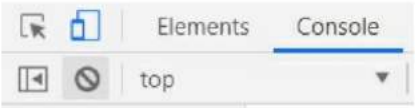
137. How to activate debugging in browser?

You can activate debugging in browser by pressing F12 and then select console in debugger menu.

138. How to get mobile devices view of webpage in desktop browser?

In browser press F12,

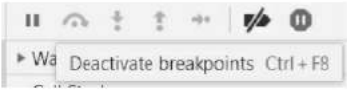
Then click on toggle device toolbar,



Then, select device for which you want webpage view.

139. How to deactivate breakpoint in browser?

This can be done by clicking on "deactivate breakpoints" icon in "Sources" tab of browser developer tool.



140. How to pause script execution?

This can be done by clicking on "pause script execution" icon in "Sources" tab of browser developer tool.

141. How to execute function line by line while debugging?

This can be done by clicking on "Step into next function call" icon in "Sources" tab of browser developer tool.

142. How to execute function without stepping into it while debugging?

This can be done by clicking on "Step over next function call" icon in "Sources" tab of browser developer tool.

143. What is code smell?

In computer programming, a code smell is any characteristic in the source code of a program that possibly indicates a deeper problem.

Determining what is and is not a code smell is subjective, and varies by language, developer, and development methodology.

15. Typescript Basics

144. What is Typescript?

TypeScript is a strongly typed, object oriented, compiled language.

Typescript is superset of Javascript.

It provides more features than Javascripts like classes, interfaces and types.

Unlike Javascript, if certain variable in Typescript is declared as number, string or some other type you cannot assign it to other type, for that you have to do type casting.

Typescript is compiled to Javascript using angular CLI.

145. Who developed Typescript?

Typescript was developed and maintained by Microsoft.

146. Difference between Typescript and Javascript.

Typescript	Javascript
It supports object-oriented programming concept like classes, interfaces, inheritance, generics, etc.	It is just a scripting language.
TypeScript file extension is ".ts".	JavaScript file extension in ".js".
Typescript generate compilation errors.	JavaScript reports error at runtime as it is an interpreted language.
TypeScript supports ES6.	JavaScript doesn't support ES6.
TypeScript gives support for modules	JavaScript doesn't support modules.

It supports strongly typed or static typing feature.	It doesn't support strongly typed or static typing.

147. How to install Typescript?

Typescript can be installed using node package manager i.e. npm.

Below is the npm command to install Typescript:

npm install -g Typescript

148. What are built-in datatypes in Typescript?

number: Represents both Integer as well as Floating Point numbers.

string: Represents a sequence of characters stored as Unicode UTF-16 code.

boolean: Represents true and false values.

void: Used as function return type when it does not return anything.

null: Represents variable/object whose value not set.

undefined: Represents uninitialized variables.

any: Represents variable that hold any value types.

149. How to create variable in Typescript?

When you declare a variable, you have four options – Declare its type and value in one statement.

e.g.

```
var name:string = "Pratik";
```

Declare its type but no value. In this case, the variable will be set to undefined.

e.g.

```
var name:string;
```

Declare its value but no type. The variable type will be set to the data type of the assigned value.

e.g.

```
var name = "Pratik";
```

Declare neither value not type. In this case, the data type of the variable will be any and will be initialized to undefined.

e.g.

```
var name;
```

150. What are the variable scopes available in TypeScript?

The variable scopes available in TypeScript are:

- Global Scope
- Class Scope
- Local Scope

151. What are Modules in Typescript?

A module is used to organize code written in TypeScript. Modules are of 2 types, Internal Modules i.e. namespaces and External Modules also referred just as modules simply.

152. Explain interface in Typescript.

Interface is a structure that defines the blueprint for classes to follow. Classes that are derived from an interface must follow the structure.

An Interface can define member properties and methods. Interface only declare the members and deriving class define the members.

e.g.

```
interface Person{
        firstName: string,
        lastName: string,
        address: string
}
```

153. What is class in Typescript?

A class is a template or blueprint for creating objects. A Class can have constructor, member variables, proper-

ties and methods. It is a logical entity. The "class" keyword is used to declare a class in Typescript.

154. What are the types of access modifiers supported by TypeScript?

The types of access modifiers supported by TypeScript are:

- Public
- Private
- Protected

155. What is namespace in Typescript?

A namespace is a way to logically group related classes or interfaces in a wrapper.

e.g.

```
export namespace Animals {
  export class Tiger { }
  export class Lion { }
}
```

Namespace is also known as internal modules.

156. What is use of question mark (?) with variable in Typescript?

Question mark on a Typescript variable marks it as optional. Any optional parameters must follow required parameters.

e.g.

```
function Demo(arg2: number, arg1? :number) {
}
```

In above case arg1 is always required, and arg2 is an optional parameter.

157. What are decorators in Typescript?

In Angular, decorators allow developers to configure classes as particular elements by setting metadata on them.

You may also like to read (available on amazon):

Javascript Interview Questions and Answers