Angular Top Interview Questions and And Answers By Pradip:

- Question No:- 0 :- **What is the difference between framework and library?**
    - A framework provides a structure for building software applications, while a library is a collection of code that can be used to perform specific tasks.
- Question No:- 1 :- **What's the use of Angular ?**
    - Angular is a UI binding framework which binds the HTML UI and JS model.
    - This helps you to reduce your effort on writing those lengthy lines of code for binding.
    - adding to it , it also helps you to build SPA by using the concept of routing.
    - It also has a lot of other features like HTTP, DI, Input output , because of which you do not need another framework.
- Question No:- 1.1 :**- What is SPA in Angular ?**
    - SPA stands for single page application.
    - SPAs are applications where the main UI gets loaded once and then the needed UI is loaded on demand.
    - In SPAs, the initial HTML, CSS, and JavaScript resources are loaded when the user first accesses the application.
    - Unlike traditional multi-page applications, SPAs do not require a full page reload with each user interaction. Instead, subsequent interactions or changes in content are handled by dynamically updating the existing page through asynchronous requests.
- Question No:- 2 **:- What are directives in Angular ?**
    - Directives are markers on HTML DOM file elements that tell angular to attach a behaviour to that element.
    - Angular comes with built-in directives like ngif for conditional rendering and ngfor for iterating over lists.
    - They are a way to extend, transform, and manipulate the DOM and its behavior within Angular applications.
- Question No:- 2.1 **:- What is DOM ?**
    - DOM stands for Document Object Model.
    - The Document Object Model (DOM) is a programming interface for web documents.
    - It represents the page so that programs can change the document structure, style, and content.
    - The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.
    - HTML is completely responsible for managing the concept of DOM.
- Question No:- 3 **:- Explain the different types of Angular directives ?**
    - Directives can be classified into two main types: structural directives and attribute directives.
    - Structural Directives
    - Structural directives are responsible for manipulating the structure of the DOM by adding, removing, or replacing elements. They are prefixed with an asterisk (*) in the template syntax.
    - Angular provides built-in directives, such as ngIf, ngFor, and ngSwitch, which are used for conditional rendering, looping through arrays, and switching between multiple views, respectively.
    - Additionally, you can also create custom directives to encapsulate and reuse behavior across components.

    - Attribute Directives
- Question No:- 4 :- **Explain the importance of NPM and Node_Modules folder ?**
    - Node package manager, which makes installation of javascript framework easy.
    - node_modules is the folder where all the packages are installed.
- Question No:- 5 :- **Explain the importance of Package.json file in Angular ?**
    - It has all the javascript references needed for the project.
    - So rather than installing one package at a time we can install all packages in one go.
    - Whatever js framework you are using in a project they are all listed in package.json.
- Question No:- 6 :- **What is typescript and why do we need it ?**
    - As the name says it adds types to javascript. Typescript is a superset of JS.
    - It gives a nice object-oriented programming environment which transpiles / converts to JS.
    - So as it is strongly typed we will have less errors and because we can do OOP with JS our productivity and quality also increases.
- Question No:- 7 :- **Explain importance of Angular CLI ?**
    - It stands for Command line interface
    - We can create an initial angular project template. So rather than starting from scratch we have some boiler plate code.
    - to install cli the command is - npm install @angular/cli
- Question No:- 8 :- **Explain the importance of Components and Modules ?**
    - Components is where you write your binding code. Module logically groups of components.
- Question No:- 9 :- **What is a decorator in Angular ?**
    - Decorators are special types of declarations in TypeScript that are used to modify the structure or behavior of classes or class members. Angular comes with a set of built-in decorators that are used to configure and enhance various elements in an Angular application.
    - Decorators define what kind of class it is.
    - @Component: Used to define a component and its metadata.
    - @Directive: Used to define a directive and its metadata.
    - @Injectable: Used to define a service and its metadata for dependency injection.
    - @NgModule: Used to define a module and its metadata.
    - The decorator function receives information about the decorated item and can modify its behavior, add metadata, or perform other tasks.
    - The @Component decorator is commonly used to define Angular components. It takes a metadata object as an argument, which provides information about the component.
- Question No:- 10 :- What are Annotationa or MetaData ?
    -
- Question No:- 11 :- **What is a template ?**
    - Template is a HTML view of angular in which we can write directives.
    - There are two ways of defining template one is inline and other is separate HTML file.
- Question No:- 12 :- Explain the four types of Data bindings in Angular ?
    - Data binding defines how the view and component communicate with each other. There are four types of the data binding in angular,
    - Interpolation binding {{}}- data flows from the component to the view and we can mix the same with html tags
    - property binding [] - data flows from the component to the view
    - event binding () - when you want to send event to the component
    - Two way binding [()] - data flows from the component to the view and vice-versa

- Question No:- 13 :- Explain architecture of Angular ?
    -
- Question No:- 15 :- How to implement SPA in Angular ?
    -
- Question No:- 16 :- How to implement routing in Angular ?
    - Routing is a simple collection which has two URLs and when this URL calls which component to load.
    - So routing helps you to define the navigation for your angular application.
    - So if you want to move from one screen to another screen and you want to respect SPA that means not loading and refreshing the whole UI routing is needed.
- Question No:- 17 :- Explain Lazy Loading ?
    - Lazy loading is a technique used in web development to optimize the loading time of a web page by deferring the loading of certain resources until they are actually needed.
    - In the context of routing in a web application, lazy loading refers to loading specific modules or components only when the user navigates to a particular route, rather than loading all modules and components at the initial page load.
    - Lazy loading routing is commonly used in Single Page Applications (SPAs) where the entire application is loaded initially, but certain parts of the application are loaded on-demand as the user interacts with the application.
- Question No:- 18 :- How to implement Lazy Loading in Angular ?
    - Lazy loading involves importing the component only when it's needed. In a lazy-loaded setup, the About Us component would be loaded only when the user navigates to the '/about-us' route.
    - When lazy loading a component, you typically use dynamic imports. Dynamic imports return a Promise, and the component is loaded when the Promise is resolved.
    - Note: If you see import() or something similar in your code, it's a good indication that lazy loading is being used.

```
{
        path: 'about-us',
        loadChildren: () => import('./features/about-us/about-us.module').then(m
        => m.AboutUsModule)
}
```

- path: 'about-us':

This specifies the route path. In this case, it's 'about-us'. This means that when the user navigates to the 'about-us' route, the specified module (lazy-loaded) will be loaded and associated with this route.

- loadChildren: () => import('./features/about-us/about-us.module').then(m => m.AboutUsModule):

This is the key part that enables lazy loading. Instead of directly importing the module at the time the application loads, it uses the loadChildren property to specify a function that will be called when the module is needed.

- import('./features/about-us/about-us.module') is a dynamic import statement that returns a Promise. The specified module is not loaded immediately; it will be loaded asynchronously when the Promise is resolved.
- The .then(m => m.AboutUsModule) part ensures that the module is loaded and the specific module class (AboutUsModule) is retrieved. This is essential because Angular expects a

dynamically loaded module to have a certain structure, and this structure is defined by the module class.

- In summary, when a user navigates to the 'about-us' route, the associated module (AboutUsModule) will be loaded asynchronously, reducing the initial bundle size and improving the application's loading performance. Lazy loading is especially useful in large applications where loading all modules at once would lead to slower initial load times.
- Question No:- 19 :- **Define Services ?**
    - Services are used for organizing and sharing code across components.
    - They are **singleton objects** that can be injected into components, providing a way to encapsulate and share functionality.
- Question No:- 19.1 :- **What are singleton objects?**
    - A Singleton object is an object for which there is only a single instance for a given class.
    - The Singleton pattern is used when you want to guarantee that a single instance of a class will be shared across clients in a system.
- Question No:- 20 :- **What is Dependency Injection ?**
    - DI is an application design pattern where rather than creating object instances from within the component, Angular injects it via the constructor.
    - Dependency injection in Angular allows you to declare the dependencies of a class (like services) in its constructor, and Angular's dependency injection system will provide instances of those dependencies when the class is instantiated.
    - Angular's dependency injection system is used to inject services into the components or other services that need them. When a component or another service requests a dependency (like a service) in its constructor, Angular provides the instance of that dependency.
- Question No:- 21 :- **How to implement Dependency Injection ?**
    - Each component in Angular interacts with the services for authentication, data fetching and user management.
    - DI is like a core concept in Angular that injects the dependencies a component needs as a function.
    - It's like having a personal assistant for your components, delivering the exact services they require at the right moment.
    - 1. We need service
    - 2. provide service in provider array in component
    - 3. Inject service in component through constructor. ex: constructor(private: ser: Service)
- Question No:- 23 :**- What's the benefit of Dependency Injection ?**
    - This makes the code more modular, testable, and maintainable, as dependencies can be easily swapped or mocked during testing. The @Injectable decorator is an essential part of this mechanism, allowing Angular to understand how to create and manage instances of services.
    - DI helps to decouple class dependencies, so that when you add new dependencies do not have change everywhere.
- Question No:- 24 :- **Differentiate between ng serve and ng build ?**
    - ng serve builds angular applications in memory or in ram, while ng build builds the applications on the hard disk.
    - So when you want to go for the production ‘ng build’ command is used.
- Question No:- 25 :**- Explain the --prod parameter in ng build ?**
    - Ng build —prod flag compresses your JS file, removes comments, creates GUIDs of your JS files and makes your application ready for production**.**

- Question No:- 26 :- **Explain ViewChild and ViewChildren?**
    - If we want to access DOM elements that are present inside the HTML Template of the same or child component. In that case, we can use View Child and View Children Decorators.
    - Also, if we want to access child component properties and methods, that is possible by using View Child and View Children Decorators.
    - When to use ViewChild - 1. Accessing template of same component. 2. Accessing template of child component
    - Definition of @ViewChild decorator- @ViewChild decorator provides us with a easy and simple way to access and manipulate the properties and methods of child component from a container or root component
    - Syntax of @ViewChild decorator - @ViewChild(selector)variableName: type
    - If you want to get a reference of first matching element then use @ViewChild or If you want to get reference of multiple matching element then use @ViewChildren

  SOURCE: https://www.youtube.com/watch?v=NJFIEp2RDBM

- Question No:- 27 :- **Why do we need Template reference variables?**
    - Template reference variables give you a way to declare a variable that references a value from an element in your template.
    - A template reference variable can refer to the following: a DOM element within a template (including custom elements) an Angular component or directive.
- Question No:- 28 :- **What is ContentProjection?**
    - Content projection is a pattern in which you insert, or project, the content you want to use inside another component. For example, you could have a Card component that accepts content provided by another component.
    - using ng-content we can achieve ContentProjection
    - The ng-content is used when we want to insert the content dynamically inside the component that helps to increase component reusability.
    - Using ng-content we can pass content inside the component selector and when angular parse that content that appears at the place of ng-content.

  SOURCE: https://www.youtube.com/watch?v=1Gkiq1u2aQc

- Question No:- 29 :- **Explain Content projection Slot?**
    - **Content Projection** has three different methods or use cases that we will be covering:
    - 1. Single Slot Content Projection
    - 2. Multi Slot Content Projection
    - 3. Conditional Content Projection

  SOURCE: https://medium.com/dvt-engineering/content-projection-in-angular-c23320ba3a70

- Question No:- 30 :- **What is ContentChild and ContentChildren?**
    - If you have any HTML element or component which is projected with the help of content projection or ng-content so to access those DOM elements inside the component class there we can use ContentChild and ContentChildren.

  SOURCE: https://www.youtube.com/watch?v=bt6d9RJJsi8&t=101s

- Question No:- 31 :- **ViewChild vs ViewChildren vs ContentChild vs ContentChildren?**
    -
- Question No:- 32 :- **Explain the importance of Component life cycle ?**
    - A component's lifecycle is the sequence of steps that happen between the component's creation and its destruction.
    - Each step represents a different part of Angular's process for rendering components and checking them for updates over time.
    - Component lifecycle is important because it allows developers to control and modify components throughout their life cycle. This helps manage state, perform side effects, optimize renders, and handle errors.
- Question No:- 33 :- **Explain events and sequence of component life cycle ?**
    - First, here's the sequence of lifecycle hooks when a component is created:
    - Constructor - the constructor of the component class is called first
    - 1. ngOnChanges

If the component has input/output bindings, ngOnChanges is called next.
This hook is called before ngOnInit and provides information about the changes in the component's input/output property when the binding value changes.

- 2. ngOnInit

After ngOnChanges, the ngOnInit lifecycle hook is called.
This is called once after the component is initialized. Ideal for initializing component properties.

```
ngOnInit() {
// Initialization logic after ngOnChanges
console.log('Component initialized');
}
```

- 3. ngDoCheck

This hook is called after ngOnInit.
It provides an opportunity for the developer to implement custom change detection logic. Called during every change detection cycle.

```
ngDoCheck() {
// Custom change detection logic
console.log('Custom change detection');
}
```

- 4. ngAfterContentInit

Called after the component's content has been initialized.
This hook is often used when you need to perform initialization logic related to content projection.

```
ngAfterContentInit() {
// Initialization logic after content is initialized
console.log('Content initialized');
}
```

- 5. ngAfterContentChecked

It is called after every check of the component's content.
It provides an opportunity to perform logic after the content has been checked during the change detection cycle.

```
ngAfterContentChecked() {
// Logic after checking component content
console.log('Content checked');
}
```

- 6. ngAfterViewInit

Called once after the component's view and child views have been initialized.
Useful for performing operations that require the view to be ready.

```
ngAfterViewInit() {
// Initialization logic after views are initialized
console.log('Views initialized');
}
```

- 7. ngAfterViewChecked

It is called after every check of the views of the component. It allows you to perform logic after the views have been checked during the change detection cycle.

```
ngAfterViewChecked() {
// Logic after checking component views
console.log('Views checked');
}
```

- 8. ngOnDestroy

This hook is called just before the directive (or component) is destroyed.
It provides an opportunity to perform cleanup logic, such as unsubscribing from observables or releasing resources.
The ngOnDestroy lifecycle hook won't be automatically called for a component unless it is actually destroyed. Components in Angular are typically destroyed when their associated views are removed from the DOM or when their parent component is destroyed.

- Question No:- 34 :**- Constructor vs ngOnInit() ?**
    - The constructor is a Typescript feature used to instantiate the Typescript class. In most Angular projects the only thing that should ever be done in the constructor is to inject services .
    - The ngOnInit function is specific to the Angular framework and is called when Angular is done creating the component
- Question No:- 35 :- **How to make HTTP calls using Angular ?**
    - In Angular, when you make HTTP requests, such as fetching data from a server, the response is often represented as an observable. An observable is an object that emits a sequence of values over time, and it can be used to handle asynchronous operations like HTTP requests.

    - Observables are part of the Reactive Extensions for JavaScript (RxJS) library.

- Question No:- 36 :- **What is the need of the Subscribe function ?**
    - In easy way subscribe means Listening to Changes:
    - Now, imagine you want to know the temperature changes. You subscribe to this stream of temperature data. Whenever the temperature changes, you get notified. You subscribe at 9 AM. You receive 20°C. You subscribe at 10 AM. You receive 22°C. You are "subscribed" to the observable stream, and you get updates whenever there's a new value.
- Question No:- 37 :- **How to handle errors when HTTP fails ?**
    - SOURCE: https://www.youtube.com/watch?v=ThgS7WDgEZY
- Question No:- 38 :- **How to pass data between components ?**
    - **@Input() Decorator**
        - The @Input() decorator is probably one of the first things that comes to mind when it comes to passing data between components in Angular. It is a rather straightforward way to pass data from a parent component to its children.
        - @Input() is particularly useful when the components are directly related to each other in the component tree. If they are not, you might have to resort to the so-called "**prop drilling**" — passing the same data through a chain of multiple components. The term itself comes from React, where props serve a similar function to Angular's @Input().Prop drilling is considered a bad practice and tends to become cumbersome and error-prone as the application grows larger.
        - Another common use case for @Input() is building **"stupid" or "representational" components** that only handle displaying the data. In this case, the parent component is managing the state and updates its children via the @Input() decorator.
- **@Output() Decorator**
    - A polar opposite of @Input(), the @Output() decorator is used to emit events from a child component to its parent or any component that is situated higher in the component tree.
- **QueryParams**
    - But shifting gear back to Angular, let's talk about several things concerning the topic of navigation.
- **Route Params**
    - Similar to QueryParams, Route Params offer another way to pass data to the component via a URL. The difference is, while QueryParams are used to pass **optional parameters**, Route Params are **mandatory**.
- **Passing Data on Navigation**
    - Router.navigate() takes an optional parameter called state to which you can pass data that will be available at the route you navigate to. It is useful when you need to share the data when navigating to a component, but for some reason cannot do it via QueryParams.
- **Using a Service with a RxJS Subject**
    - When the components are further apart in the component tree, passing data around with @Input() and @Output() can quickly get out of hand and become too confusing and difficult to maintain. In these cases, using a shared service can provide a central location for managing and sharing data.
- Question No:- 39 :- **Explain importance of input, output & event emitters ?**

    - First of all, the idea of *Input* and *Output* is to exchange data between components. They are a mechanism to send/receive data from one component to another.
    - *Input* is used to receive data in whereas *Output* is used to send data out.
    - *Output* sends data out by exposing event producers, usually *EventEmitter* objects.
- Question No:- 40 :- **How to pass data during routing ?**
    -
- Question No:- 41 :- I**s it a good practice to pass data using services ?**
    - Using services with signals may significantly improve data management and data passing between components in current Angular apps.
    - Signals offer an agile approach to managing modifications, guaranteeing that your application stays responsive and effective.
- Question No:- 42:- **What is the need of Angular Pipes?**
    - In Angular, pipes are a way to transform data in the template before displaying it.
    - They are similar to filters in other frameworks and can be used for various tasks, such as formatting dates, converting text to uppercase, or filtering lists. Angular provides several built-in pipes, and you can also create custom pipes.
- Question No:- 43:- **Can you name some built-in Angular Pipes?**
    - 1. Uppercase and Lowercase: The uppercase and lowercase pipes transform the input string
    - 2. Date Pipe: The date pipe formats a date based on the provided format.
    - 3. Percent Pipe: The percent pipe multiplies the input by 100 and appends a percentage sign.
    - 4. Json Pipe: The json pipe converts an object to its JSON string representation. It's useful for debugging or displaying structured data.
    - 5. Keyvalue Pipe: The keyvalue pipe iterates over the key-value pairs of an object, allowing you to display them in a template.
    - 6. Slice Pipe: The slice pipe extracts a portion of a string or array. In this example, it extracts characters from index 0 to 6.
    - 7. Title Case Pipe: The titlecase pipe converts the input string to title case (capitalizing the first letter of each word).
- Question No:- 44:- **How to create Custom pipes in Angular?**
    -
- Question No:- 47 :- **What are observables and observers?**
    - .In Angular, observables are often used to handle asynchronous operations like HTTP requests or user
    - interactions. Observable is the result of the HTTP request. When you make an HTTP request, it returns an observable representing the stream of data that will come from the server.
    - You subscribe to this observable to listen for data changes or handle errors. Imagine a Stream Observable as a Stream: Think of an observable as a stream of events or data over time. Just like a river that flows continuously, an observable emits values or events continuously.
    - Data Over Time Values Over Time: Imagine you're tracking the temperature every hour. The temperature at each hour is valued in the stream of temperatures over time. At 9 AM: 20°C At 10 AM: 22°C At 11 AM: 25°C
    - This series of temperature values over time forms a stream, and an observable is like this stream of data.
- Question No:- 48 :- **Explain the use of Subscribe with sample code.**

        - The subscribe method is part of the Observable pattern in JavaScript and is used to listen to events emitted by an observable.
        - In the context of Angular's Reactive Forms, when you call subscribe on the valueChanges observable of a form control, you're essentially saying, "I want to be notified whenever the value of this form control changes."
        - HTML–
        - <label for="gender" class="asterisk">Gender</label>
        - <select id="gender" formControlName="gender"
        - placeholder="Please select Gender">
        - <option value="" disabled selected>Select Gender</option>
        - <option value="male">Male</option>
        - <option value="female">Female</option>
        - <option value="other">Other</option>
        - </select>
        - TS–
        - ngOnInit() {
        - this.medicalForm.get('gender').valueChanges.subscribe((value:string)=>
        - {
        - this.selectedGender=value;
        - console.log("selectedGender",this.selectedGender);
        - })
        - }
- Question No:- 49 :- **Explain concept of operators with sample code.**
    - Operators are nothing but they are small piece of logic which actually converts an observable streams to another observable streams
    - If you want to go and plug in preprocessing logic before the data comes to the listener I will use operator.
- Question No:- 50 :- **What is Push/reactive vs Pull/Imperative?**
    - Imperative programming means listener code is responsible to pull a stream of data.
    - Reactive programming means you register a callback and the stream is responsible for pushing data.
    - Some developers also visualize it as a publisher and subscriber model as well.
- Question No:- 51 :- **What are Interceptors in Angular?**
    - HTTP Interceptors are a concept in web development and server-side programming, typically associated with web frameworks and libraries.
    - These interceptors allow developers to intercept and handle HTTP requests and responses globally within an application.
    - HTTP Interceptors in Angular are classes that implement the **HttpInterceptor** interface.
-
    - They can be used to perform various tasks related to HTTP requests and responses, such as adding headers, handling errors, modifying the request or response data, logging, authentication, etc.
    - **HttpInterceptor** defines a single method called **intercept**, which takes two parameters: the **HttpRequest** and the **HttpHandler**.
    - Best Resource(https://medium.com/@jaydeepvpatil225/http-interceptors-in-angular-6e9891ae0538)
- Question No:- 52 :- **How to implement Interceptors?**
    - Best Resource(https://medium.com/@jaydeepvpatil225/http-interceptors-in-angular-6e9891ae0538)

- Question No:- 53 :- **Give some use of Interceptors?**
    - Best Resource(https://medium.com/@jaydeepvpatil225/http-interceptors-in-angular-6e9891ae0538)
- Question No:- 54 :- **Can we provide multi-Interceptors?**
- Question 55 :- **What are two ways of doing validation in Angular?**
    -
- Question 56 :- **Template driven forms VS Reactive Forms?**
    -
- Question 57 :- **In what situations you will use what?**
    -
- Question 58 :- **Explain template reference variables ?**
-
    - A template reference variable is a variable which store reference of DOM element, component or directive on which it is used,

    SOURCE: https://www.youtube.com/watch?v=Vcax2zHZGD8

- Question 59 :- **How do we implement Template driven forms?**
    -
- Question 60 :- **How to check if overall validation and specific validations are good ?**
    -
- Question 61 :- **How do we implement Reactive forms ?**
    -
- Question 62 :- **How can we implement composite validations?**
    -
- Question 63 :- **How to create dynamic validation ?**
    -
- Question 64 :- **Can you talk about some inbuilt validators ?**
    -
- Question 66:- **How can you create your own custom validator ?**
    -
- Question 67:- **Can we implement angular validators without FORM tag ?**
    -
- Question 68:- **What is [ngModelOptions]="{standalone: true}" ?**
    - In angular 15 standalone components become stables.
    - Till angular 16 we can use —standalone flag during generating components.
    - But from angular 17 we don't need to add any flag while generating the component.
    - We should not declare one component into another, instead we should import
    - When the standalone concept was not there at that time, if we were required to add component specific dependencies then we needed to declare one component into another and we needed to use a module to import that component.
    - make some process simple
    - NgModule is not required
    - small bundle size

    SOURCE: https://www.youtube.com/watch?v=RPRbkTI38vw

- Question 68:-**Tell me about the event emitter?**