1) What is Angular? / What do you know about Angular?

|  |  |  |
| --- | --- | --- |
| **Feature** | **AngularJS** | **Angular** |
| **Version** | AngularJS was the very first version initially released in 2010. It was a browser-side JavaScript used within HTML code and created a revolution in web application development. It is popularly known as AngularJS. | The later Angular versions were a complete rewrite of AngularJS. For example, Angular 2 was initially released in 2016. There is nothing common between Angular2 and AngularJS except the core developer's team. After that, Angular 6, Angular 7, Angular 8, Angular 9, and Angular 10 were released that are very similar to each other. These later versions are known as Angular. |
| **Architecture** | AngularJS supports the MVC design model. | Angular uses components and directives. |
| **Supported Language** | The recommended and supported language of AngularJS is JavaScript. | The recommended and supported language of Angular is TypeScript. |
| **Expression Syntax** | In AngularJS, a specific ng directive is required for the image/property and an event. | Angular uses () for event binding and [] for property binding. |
| **Mobile Support** | AngularJS doesn't provide any mobile support. | Angular provides mobile support. |
| **Dependency Injection** | There is no concept of Dependency Injection in AngularJS. | Angular supports hierarchical Dependency Injection with uni-directional tree-based change detection. |
| **Routing** | In AngularJS, $routeprovider.when() is used for routing configs. | In Angular, @RouteConfig{(?)} is used for the routing config. |
| **Structure** | It is the first and basic version, so it is very easy to manage. | It has a very simplified structure that makes the development and maintenance of large applications very easy. |
| **Speed** | It is slower because of its limited features. | It is faster than AngularJS because of its upgraded features. |
| **Support** | It doesn't provide support or new updates anymore. | It provides active support, and frequent new updates are made. |

Angular is one of the most popular JavaScript frameworks developed and maintained by Google. It is an open-source front-end web framework based on TypeScript. It is most suited for developing enterprise web applications because its code is reusable and maintainable.

2) What are some powerful features integrated into Angular?

Angular integrates some powerful features like declarative templates, end to end tooling, dependency injection and various other best practices that smoothens the development path.

3) What is the main purpose of Angular?

The main purpose of using Angular is to create fast, dynamic and scalable web applications. We can create these applications very easily with Angular using components and directives.

Angular was started as a SPA (Single-Page-Application) framework, and now it supports dynamic content based on different users through dependency injection. It provides a platform for easy development of web-based applications and empowers the front end developers in curating cross-platform applications. YouTubeTV is the most popular example that uses Angular.

### 4) What is the difference between AngularJS and Angular?

Let's compare the features of AngularJS and Angular in a tabular form:

**A list of differences between AngularJS and Angular-**

### 5) What are the biggest advantages of using Angular?

Following is the list of the biggest advantages of using the Angular framework:

* Angular supports two-way data-binding.
* It follows MVC pattern architecture.
* It supports static templates and Angular template.
* It facilitates you to add a custom directive.
* It also supports RESTfull services.
* Validations are supported in Angular.
* Angular provides client and server communication.
* It provides support for dependency injection.
* It provides powerful features like Event Handlers, Animation, etc.

### 6) What do you understand by Angular expressions?

Angular expressions are code snippets that are used to bind application data to HTML. Angular expressions are usually placed in binding such as {{ expression }} similar to JavaScript.\

Syntax: {{ expression }}

### 7) What are templates in Angular?

In Angular, templates contain Angular-specific elements and attributes. These are written with HTML and combined with information coming from the model and controller, which are further rendered to provide the user's dynamic view.

### 8) What is the difference between an Annotation and a Decorator in Angular?

In Angular, annotations are the "only" metadata set of the class using the Reflect Metadata library. They are used to create an "annotation" array. On the other hand, decorators are the design patterns used for separating decoration or modification of a class without actually altering the original source code.

### 9) Why was Angular introduced as a client-side framework?

Before the introduction of Angular, web developers used VanillaJS and jQuery to develop dynamic websites. Later, when the websites became more complex with added features and functionality, it was hard for them to maintain the code. Along with this, there were no provisions of data handling facilities across the views by jQuery. The need for a client-side framework like Angular was obvious that can make life easier for the developers by handling separation of concerns and dividing code into smaller bits of information (components).

Client-side frameworks like Angular facilitate developers to develop advanced web applications like Single-Page-Application. These applications can also be developed using VanillaJS, but the development process becomes slower by doing so.

10) How does an Angular application work?

Every Angular app contains a file named angular.json. This file contains all the configurations of the app. While building the app, the builder looks at this file to find the application's entry point. See the structure of the angular.json file:

1. "build": {
2. "builder": "@angular-devkit/build-angular:browser",
3. "options": {
4. "outputPath": "dist/angular-starter",
5. "index": "src/index.html",
6. "main": "src/main.ts",
7. "polyfills": "src/polyfills.ts",
8. "tsConfig": "tsconfig.app.json",
9. "aot": false,
10. "assets": [
11. "src/favicon.ico",
12. "src/assets"
13. ],
14. "styles": [
15. "./node\_modules/@angular/material/prebuilt-themes/deeppurple-amber.css",
16. "src/style.css"
17. ]
18. }
19. }

When the application enters the build section, the options object's main property defines the entry point of the application. The application's entry point is main.ts, which creates a browser environment for the application to run and calls a function called bootstrapModule, which bootstraps the application.

These two steps are performed in the following order inside the main.ts file:

1. **import** { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2. platformBrowserDynamic().bootstrapModule(AppModule)

In the above line of code, AppModule is getting bootstrapped.

The AppModule is declared in the app.module.ts file. This module contains declarations of all the components.

**Below is an example of app.module.ts file:**

1. **import** { BrowserModule } from '@angular/platform-browser';
2. **import** { NgModule } from '@angular/core';
3. **import** { AppComponent } from './app.component';
4. @NgModule({
5. declarations: [
6. AppComponent
7. ],
8. imports: [
9. BrowserModule
10. ],
11. providers: [],
12. entryComponents: [],
13. bootstrap: [AppComponent]
14. })
15. export **class** AppModule { }

In the above file, you can see that AppComponent is getting bootstrapped. It is defined in app.component.ts file. This file interacts with the webpage and serves data to it.

**Below is an example of app.component.ts file:**

1. **import** { Component } from '@angular/core';
2. @Component({
3. selector: 'app-root',
4. templateUrl: './app.component.html',
5. styleUrls: ['./app.component.css']
6. })
7. export **class** AppComponent {
8. title = 'angular';
9. }

Each component is declared with three properties:

1. **Selector -** It is used to access the component.
2. **Template/TemplateURL -** It contains HTML of the component.
3. **StylesURL -** It contains component-specific stylesheets.

Now, Angular calls the index.html file. This file consequently calls the root component that is app-root. The root component is defined in app.component.ts.

**See how the index.html file looks like:**

1. <!doctype html**>**
2. **<html** lang="en"**>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<title>**Angular**</title>**
6. **<base** href="/"**>**
7. **<meta** name="viewport" content="width=device-width, initial-scale=1"**>**
8. **</head>**
9. **<body>**
10. **<app-root></app-root>**
11. **</body>**
12. **</html>**

The HTML template of the root component is displayed inside the <app-root> tags.This is the way how every angular application works.

### 11) Why is Angular preferred over other frameworks? / What are some advantages of Angular over other frameworks?

Due to the following features, Angular is preferred over other frameworks:

**Extraordinary Built-in Features:** Angular provides several out of the box built-in features like routing, state management, RxJS library, Dependency Injection, HTTP services, etc. That's why the developers do not need to look for the above-stated features separately.

**Declarative UI:** Angular has declarative UI. It uses HTML to render the UI of an application as it is a declarative language. It is much easier to use than JavaScript.

**Long-term Google Support:** Angular is developed and maintained by Google. Google has a long term plan to stick with Angular and provide support.

### 12) What are the different Lifecycle hooks of Angular?

When the Angular components are created, they enter their lifecycle and remain when they are destroyed. Angular Lifecycle hooks are used to check the phases and trigger changes at specific phases during the entire duration.

**ngOnChanges( ):** This method is called when one or more input properties of the component are changed. The hook receives a SimpleChanges object containing the previous and current values of the property.

**ngOnInit( ):** This is the second lifecycle hook. It is called once, after the ngOnChanges hook. It is used to initialize the component and sets the input properties of the component.

**ngDoCheck( ):** This hook is called after ngOnChanges and ngOnInit and is used to detect and act on changes that cannot be detected by Angular. In this hook, we can implement our change detection algorithm.

**ngAfterContentInit( ):** This hook is called after the first ngDoCheck hook. This hook responds after the content gets projected inside the component.

**ngAfterContentChecked( ):** This hook is called after ngAfterContentInit and every subsequent ngDoCheck. It responds after the projected content is checked.

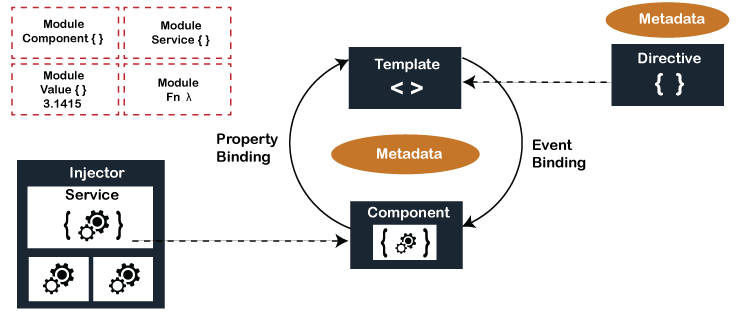
**ngAfterViewInit( ):** This hook is called after a component's view or the initialization of a child component's view.

**ngAfterViewChecked( ):** This hook is called after ngAfterViewInit. It responds after the component's view or when the child component's view is checked.

**ngOnDestroy( ):** This hook is called just before Angular destroys the component. This is used to clean up the code and detach event handlers.

### 19) What are the main building blocks of an Angular application? Explain with the pictorial diagram of Angular architecture.

Following are the main building blocks of an Angular application. You can see them in the following picture:



20) What is the difference between Observables and Promises in Angular?

In Angular, as soon as we make a promise, the execution takes place, but this is not the case with observables because they are lazy. It means nothing happens until a subscription is made.

21) What is the difference between Observables and Promises in Angular?

Let's see the differences between them in the tabular form:

|  |  |
| --- | --- |
| **Promise** | **Observable** |
| It emits a single value. | It emits multiple values over a period of time. |
| Not Lazy | Lazy. An observable is not called until we subscribe to the observable. |
| We can not cancel it. | We can cancel it by using the unsubscribe() method. |
|  | Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc. |

**Let's understand it by an example:**

1. const observable = rxjs.Observable.create(observer =**>** {
2. console.log('This is what inside an observable');
3. observer.next('Hello JavaTpoint');
4. observer.complete();
5. });
6. console.log('Before subscribing an Observable');
7. observable.subscribe((message)=**>** console.log(message));

When you run the above Observable, you can see the following messages displayed in the following order:

1. Before subscribing an Observable
2. This is what inside an observable
3. Hello JavaTpoint

Here, you can see that observables are lazy. Observable runs only when someone subscribes to them. That's why the message "Before subscribing an Observable" is displayed ahead of the message inside the observable.

**Now see the example of a Promise:**

1. const promise = new Promise((resolve, reject) =**>** {
2. console.log('This is what written inside promise');
3. resolve('Hello JavaTpoint');
4. });
5. console.log('Before calling then method on Promise');
6. greetingPoster.then(message =**>** console.log(message));

When you run the above Promise, you will see the messages displayed in the following order:

1. This is what written inside Promise
2. Before calling then method on Promise
3. Hello JavaTpoint

Here, you can see that the message inside Promise is displayed first. This means that the Promise runs first, and then the method is called.

The next difference between them is that Promises are always asynchronous; even when the Promise is immediately resolved. On the other hand, an Observable can be both synchronous and asynchronous.

In the case of the above example, observable is synchronous. Let's see the case where an observable can be asynchronous:

1. const observable = rxjs.Observable.create(observer =**>** {
2. setTimeout(()=**>**{
3. observer.next('Hello JavaTpoint');
4. observer.complete();
5. },3000)
6. });
7. console.log('Before calling subscribe on an Observable');
8. observable.subscribe((data)=**>** console.log(data));
9. console.log('After calling subscribe on an Observable');

When you run the above observable, you will see the messages in the following order:

1. Before calling subscribe on an Observable
2. After calling subscribe on an Observable
3. Hello JavaTpoint

### 22) What are directives in Angular?

A directive is a class in Angular that is declared with a @Directive decorator. Every directive has its own behavior, and you can import them into various components of an application.

### 23) When do we use a directive in Angular?

If you create an Angular application where multiple components need to have similar functionalities, you have to do it by adding this functionality individually to every component. This is not a very easy task. Directives are used to cope up with this situation. Here, we can create a directive with the required functionality and then import the directive to components that require this functionality.

### 24) What are the different types of directives in Angular?

There are mainly three types of directives in Angular:

**Component Directives:** The component directives are used to form the main class in directives. To declare these directives, we have to use the @Component decorator instead of @Directive decorator. These directives have a view, a stylesheet and a selector property.

**Structural directives:** These directives are generally used to manipulate DOM elements. The structural directive has a ' \* ' sign before them. We can apply these directives to any DOM element.

**Following are some example of built-in structural directives:**

**\*ngIf Structural Directive:** \*ngIf is used to check a Boolean value and if it's truthy, the div element will be displayed.

1. **<div** \*ngIf="isReady" class="display\_name"**>**
2. {{name}}
3. **</div>**

**\*ngFor Structural Directive:** \*ngFor is used to iterate over a list and display each item of the list.

1. **<div** class="details" \*ngFor="let x of details" **>**
2. **<p>**{{x.name}}**</p>**
3. **<p>** {{x.address}}**</p>**
4. **<p>**{{x.age}}**</p>**
5. **</div>**

**Attribute Directives:** The attribute directives are used to change the look and behavior of a DOM element. Let's create an attribute directive to understand it well:

**This is how we can create a custom directive:**

Go to the command terminal, navigate to the directory of the angular app and type the following command to generate a directive:

1. ng g directive yellowBackground

This will generate the following directive. Manipulate the directive to look like this:

1. **import** { Directive, ElementRef } from '@angular/core';
2. @Directive({
3. selector: '[appYellowBackground]'
4. })
5. export **class** YellowBackgroundDirective {
6. constructor(el:ElementRef) {
7. el.nativeElement.style.backgroundColor = "yellow";
8. }
9. }

Now, you can easily apply the above directive to any DOM element:

1. **<p** appYellowBackground**>**Hello JavaTpoint**</p>**

### 25) What are string interpolation and property binding in Angular?

String interpolation and property binding are parts of data-binding in Angular. Data-binding is a feature of Angular, which is used to provide a way to communicate between the component (Model) and its view (HTML template). There are two ways of data-binding, one-way data binding and two-way data binding. In Angular, data from the component can be inserted inside the HTML template. Any changes in the component will directly reflect inside the HTML template in one-way binding, but vice-versa is not possible. On the other hand, it is possible in two-way binding.

String interpolation and property binding both are examples of one-way data binding. They allow only one-way data binding.

**String Interpolation:** String interpolation uses the double curly braces {{ }} to display data from the component. Angular automatically runs the expression written inside the curly braces. For example, {{ 5+5 }} will be evaluated by Angular, and the output will be 10. This output will be displayed inside the HTML template.

**Property Binding:** Property binding is used to bind the DOM properties of an HTML element to a component's property. In property binding, we use the square brackets [ ] syntax.

### 26) Is it possible to make an angular application to render on the server-side?

Yes, we can make an angular application to render on the server-side. Angular provides a technology Angular Universal that makes you able to render applications on the server-side. **Following are the benefits of using Angular Universal:**

**Better User Experience:** It enables users to see the view of the application instantly.

**Better SEO:** Angular Universal ensures that the content is available on every search engine leading to better SEO.

**Load Faster:** Angular Universal ensures that the render pages available to the browsers sooner to make the loading faster server-side application loads faster.

27) What is Dependency Injection in Angular?

Dependency injection is an application design pattern that is implemented by Angular. It is used to form the core concepts of Angular. Dependencies are services in Angular which have some specific functionality. Various components and directives in an application can need these functionalities of the service. Angular provides a smooth mechanism by which these dependencies are injected into components and directives.

28) Can you demonstrate navigation between different routes in an Angular application?

You can demonstrate the navigation between different routes in an Angular app in the following way. See the following code to demonstrate navigation in an Angular app named "My First App."

1. **import** from "@angular/router";
2. .
3. .
4. .
5. @Component({
6. selector: 'app-header',
7. template: `
8. <nav **class**="navbar navbar-light bg-faded">
9. <a **class**="navbar-brand" (click)="goHome()">My First App</a>
10. <ul **class**="nav navbar-nav">
11. <li **class**="nav-item">
12. <a **class**="nav-link" (click)="goHome()">Home</a>
13. </li>
14. <li **class**="nav-item">
15. <a **class**="nav-link" (click)="goSearch()">Search</a>
16. </li>
17. </ul>
18. </nav>
19. })
20. **class** HeaderComponent {
21. constructor(**private** router: Router) {}
22. goHome() {
23. **this**.router.navigate(['']);
24. }
25. goSearch() {
26. **this**.router.navigate(['search']);
27. }
28. }

### Explain string interpolation and property binding in Angular.

String interpolation and property binding are parts of **data-binding** in Angular.  
Data-binding is a feature in angular, which provides a way to communicate between the component(Model) and its view(HTML template).  
Data-binding can be done in two ways, **one-way** binding and **two-way** binding.  
In Angular, data from the component can be inserted inside the HTML template. In one-way binding, any changes in the component will directly reflect inside the HTML template but, vice-versa is not possible. Whereas, it is possible in two-way binding.  
  
String interpolation and property binding allow only one-way data binding.  
String interpolation uses the double curly braces **{{ }}** to display data from the component. Angular automatically runs the expression written inside the curly braces, for example, {{ 2 + 2 }} will be evaluated by Angular and the output 4, will be displayed inside the HTML template. Using property binding, we can bind the DOM properties of an HTML element to a component's property. Property binding uses the square brackets **[ ]** syntax.