Angular Questions

1. What is the difference between Angular-1, Angular-2?

|  |  |  |
| --- | --- | --- |
| S.No | Angular 2.x | Angular 1.x |
| 1. | 2016 | 2010 |
| 2. | Components based.  Controllers and $scope are removed. | Controller and $scope based |
| 3. | Improved DI(Hierarchical dependencies) | Supports DI(We need to dependencies for Parent and child controller). |
| 4. | Moblie first(Ionic, NativeScript) | Not built with mobile first. |
| 5. | Supports ES5/6, TypeScript, Dart | Supports ES5/6 (JavaScript) |
| 6. | Angular CLI | Doesn’t have CLI |
| 7. | Class is only way to define services | Factory, service, provider, value and constant are used for services. |
| 8. | Runs on client-side and server-side | Runs only on client-side |
| 9. | bootstrapModule() function is used to initialize | ng-app and angular.bootstrap() function are used to initialize |
| 10. | Supports pipe 🡪 text transformation means lowercase to uppercase | Supports Filter 🡪 text transformation means lowercase to uppercase |
| 11. | Supports CamelCase and PascalCase syntaxes like ngModel, ngForm and NGForm | Supports spinal-case and camelCase syntaxes like ng-model, ng-calss and ngModel |
| 12. | Use () parenthesis for events and [] for attributes | Doesn’t support () and [] |

1. What is difference between Angular 2 and 4 and 5?

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Angular 2.x | Angular 4.x | Angular 5.x |
|  |  |  |  |

1. What is Angular 2?
   * 1. It is more advance version of angularjs
     2. Angular 2 is completely revived component based **JavaScript framework.**
2. List the Angular Commands

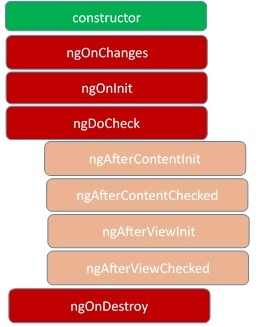
|  |  |  |
| --- | --- | --- |
| * 1. S.No | * 1. Command | * 1. Description |
| * 1. 1. | add | * 1. Adds support for an external library to your project.   2. **arguments:**   3. collection   4. The package to be added.   5. **options:**   6. --defaults   7. When true, disables interactive input prompts for options with a default.   8. --dry-run (-d)   9. When true, runs through and reports activity without writing out results.   10. --force (-f)   11. When true, forces overwriting of existing files.   12. --help   13. Shows a help message for this command in the console.   14. --interactive   15. When false, disables interactive input prompts. |
| * 1. 2. | build | * 1. Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory.   2. usage: ng build <**project**> [**options**]   3. **arguments:**   4. **project**   5. The name of the project to build. Can be an app or a library.   6. **options:**   7. --aot   8. Build using Ahead of Time compilation.   9. --base-href   10. Base url for the application being built.   11. --build-optimizer   12. Enables '@angular-devkit/build-optimizer' optimizations when using the 'aot' option.   13. --common-chunk   14. Use a separate bundle containing code used across multiple bundles.   15. --configuration (-c)   16. A named build target, as specified in the "configurations" section of angular.json.   17. Each named target is accompanied by a configuration of option defaults for that target.   18. --delete-output-path   19. Delete the output path before building.   20. --deploy-url   21. URL where files will be deployed.   22. --eval-source-map   23. Output in-file eval sourcemaps.   24. --extract-css   25. Extract css from global styles into css files instead of js ones.   26. --extract-licenses   27. Extract all licenses in a separate file.   28. --fork-type-checker   29. Run the TypeScript type checker in a forked process.   30. --help   31. Shows a help message for this command in the console.   32. --i18n-file   33. Localization file to use for i18n.   34. --i18n-format   35. Format of the localization file specified with --i18n-file.   36. --i18n-locale   37. Locale to use for i18n.   38. --i18n-missing-translation   39. How to handle missing translations for i18n.   40. --index   41. The name of the index HTML file.   42. --lazy-modules   43. List of additional NgModule files that will be lazy loaded. Lazy router modules will be discovered automatically.   44. --main   45. The full path for the main entry point to the app, relative to the current workspace.   46. --named-chunks   47. Use file name for lazy loaded chunks.   48. --ngsw-config-path   49. Path to ngsw-config.json.   50. --optimization   51. When true, uses optimization for the app build.   52. --output-hashing   53. Define the output filename cache-busting hashing mode.   54. --output-path   55. The full path for the new output directory, relative to the current workspace.   56. By default, writes output to a folder named dist/ in the current project.   57. --poll   58. Enable and define the file watching poll time period in milliseconds.   59. --polyfills   60. The full path for the polyfills file, relative to the current workspace.   61. --preserve-symlinks   62. Do not use the real path when resolving modules.   63. --prod   64. When true, sets the build configuration to the production target.   65. All builds make use of bundling and limited tree-shaking. A production build also runs limited dead code elimination.   66. --profile   67. Output profile events for Chrome profiler.   68. --progress   69. Log progress to the console while building.   70. --service-worker   71. Generates a service worker config for production builds.   72. --show-circular-dependencies   73. Show circular dependency warnings on builds.   74. --skip-app-shell   75. Flag to prevent building an app shell.   76. --source-map   77. Output sourcemaps.   78. --stats-json   79. Generates a 'stats.json' file which can be analyzed using tools such as: 'webpack-bundle-analyzer' or https://webpack.github.io/analyse.   80. --subresource-integrity   81. Enables the use of subresource integrity validation.   82. --ts-config   83. The full path for the TypeScript configuration file, relative to the current workspace.   84. --vendor-chunk   85. Use a separate bundle containing only vendor libraries.   86. --vendor-source-map   87. Resolve vendor packages sourcemaps.   88. --verbose   89. Adds more details to output logging.   90. --watch   91. Run build when files change. |
| * 1. 3. | config | * 1. Retrieves or sets Angular configuration values in the angular.json file for the workspace.   2. **usage: ng config <jsonPath> <value> [options]** |
| * 1. 4. | doc | * 1. (d) Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword. |
| * 1. 5. | e2e | * 1. (e) Builds and serves an Angular app, then runs end-to-end tests using Protractor. |
| * 1. 6. | generate | * 1. (g) Generates and/or modifies files based on a schematic. |
| * 1. 7. | help | * 1. Lists available commands and their short descriptions. |
| * 1. 8. | lint | * 1. (l) Runs linting tools on Angular app code in a given project folder. |
| * 1. 9. | new | * 1. (n) Creates a new workspace and an initial Angular app. |
| * 1. 10. | run | * 1. Runs an Architect target with an optional custom builder configuration defined in your project. |
| * 1. 11. | serve | * 1. (s) Builds and serves your app, rebuilding on file changes. |
| * 1. 12. | test | * 1. (t) Runs unit tests in a project. |
| * 1. 13. | update | * 1. Updates your application and its dependencies. See <https://update.angular.io/> |
| * 1. 14. | version | * 1. (v) Outputs Angular CLI version. |
| * 1. 15. | xi18n | * 1. Extracts i18n messages from source code. |

1. What is Angular life cycle hooks?

* In Angular, every component has a life-cycle, a number of different stages it goes through. There are 8 different stages in the component life cycle.

Every stage is called as life cycle hook event. So, we can use these hook events in different phases of our application to obtain control of the components.

* Since a component is a TypeScript class, every component must have a constructor method. The constructor of the component class executes first, before the execution of any other lifecycle hook events.
* If we need to inject dependencies into that component, then the constructor is the best place to inject those dependencies.
* After executing the constructor, Angular executes its lifecycle hook methods in a specific order.



* These stages are divided into two phases – one is linked to the component and another is linked to the children of that component.

**ngOnChanges:**

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

**ngOnInit:**

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

**ngDoCheck:**

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

**ngAfterContentInit:**

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

**ngAfterContentChecked:**

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

**ngAfterViewInit:**

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

**ngAfterViewChecked:**

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

**ngOnDestroy:**

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

1. What is difference between **let** and **var**
   1. 🡪 **let** gives you the privilege to declare variables that are limited in scope to the block, statement of expression unlike **var**.
   2. 🡪 **var** is rather then a keyword which defines a variable globally regardless of block scope.

|  |  |  |
| --- | --- | --- |
|  | * 1. **let** | * 1. **var** |
| * 1. Global Window Object |  |  |
| * 1. Block |  |  |
| * 1. Redeclaration |  |  |
| * 1. Function |  |  |

1. What is the use of package.json in the
2. Difference between constructor() and ngOninit()

**Constructor():**

* The constructor is a special method comes with every class.
* This method will be called whenever we create a new Objects.
* Generally, it is used for initializing the class members.
* constructors are not derived from Angular but are concepts derived from Object Oriented Designs.
* The constructor creates an instance of the component class.

**ngOninit():**

* It is a life cycle hook managed by Angular
* It is being added to prototype of the class created.
* This method is called by Angular when component is initialized.
* ngOninit() called by the Angular to indicate that Angular is done with initializing the component.
* To use ngOninit() we have to import OnInit from **@angular/core(**Actually it is not required, but as a good programming practice import OnInit**)**

1. **When is ngOnInit Executed?**

* ngOnInit will be executed, when Angular done with creating of component DOM.
* ngOnInit will be called after the constructor execution and after first ngOnChanges.

1. **When to use constructor and when to use ngOnInit in Angular?**

* I n object-oriented programming world, constructor is used to initialize the class members and use for dependency injection.

**Constructor usage in Angular:**

* In Angular, the constructor is used for injecting the dependencies into the component class.

**ngOnInit method usage in Angualr:**

* ngOnInit method will called after the constructor method completes the execution.
  + Constructor should finishes creating a component DOM.
  + Constructor will Inject all required dependencies and processed input bindings.
* If the logic doesn’t depend on DI, Component DOM or Input bindings then use ngOnInt method.

1. **ngOnInit vs Constructor**

|  |  |  |
| --- | --- | --- |
| **S. No** | **ngOnInit** | **Constructor** |
| **1.** | One of the Angular life cycle hook method. | Typescript feature nothing to do with Angular. |
| **2.** | ngOnInit being added to prototype of the class created | Constructor is transformed to function with the same name as class created. |
| **3.** | Called by Angular | Called by Javascript engine. |
| **4.** | Invoked by Angular when component is initialized | Constructor is automatically called at the time of creating Object of the class. |
| **5.** | Actual business logic performed here | Used for Injecting dependencies. |
| **6.** | Everything is ready at the time of invocation | Not everything in component is initialized at the time of invocation |

1. **What is Component? Explain with Example.**

* A component is made up of three parts.
  + **Template** 
    - Which represents the View Part.
    - This is created using HTML and will be user interface for the Application.
  + **Class** 
    - Which is nothing but the code that supports the View.
    - This is created using the TypeScript.
    - It can have data members and methods to control the View.
  + **Metadata**
    - This is the information that Angular needs to decide if particular class is a component or just a regular class.
    - metadata is defined using decorator which is feature in Typescript.
    - Decorator is just a function that provides the information about the class attached to it.
    - For the Component we use @Component decorator.
* Put together Template + Class + Metadata we get an Angular Component.

Ex:

**app.component.ts**

import {Component} from @ ‘@angular/core’;

@Component({

selector: ‘app-root’,

templateUrl: ‘./app.component.html’,

styleUrls: [‘./app.component.css’]

})

export class AppComponent{

title = “Codevolution”;

}

**Class Name 🡪** Name of the class is AppComponent.

**Metadata 🡪** The class **AppComponent** attached the metadata in the form of decorator and to more specific the @Component({}) decorator.

**@Component({})** 🡪 component decorator is basically a function attaches to the class below it, and this Component decorator tells the Angular it is a Component class but not plain class. The component decorator contains both the metadata and the template which represents the View.

* So As part of metadata we have selector, templateUrl, and styleUrls.

**selector: ‘app-root’ 🡪** is basically a custom HTML tag that can be used to represent this component. When we specify the selector in our HTML, Angular does the component template in its place.

**templateUrl: ‘./app.component.html’ 🡪** ThetemplateUrl points to the **app.component.html** file that represents the view for this component.

**styleUrls: [‘./app.component.css’] 🡪** The styleUrls points to the **app.component.css** file that represents the style for this component.

1. **What are the ways we can specify the selector?**

* There are 3 ways to specify the selector for the component.
  + By using selector as a custom HTML tag
    - **Selector : ‘app-test’**
    - <app-test></app-test>
  + By Using selector as a class
    - **Selector : ‘.app-test’**
    - **<div class=”app-test”></div>**
  + By using selector as an Attribute
    - **Selector : ‘[app-test]’**
    - **<div app-test ></div>**

1. **What is the use of declarations array in app.module ?**

* In app.module.ts file contains the declarations[].
* Import all the components in this app.module.ts file for using this application
* Add all the imported components to this Array.
* This declaration[] contains all the components which is used by the application.

1. **What are the possible ways to specify the HTML content in @Component decorator?**

* There are 2 possible ways to specify the HTML content in component decorator.
  + In-line HTML Content
    - **template:** is the property tospecify the inline HTML
    - If HTML content is single line, then we can use single quotes
      * **Ex:- template: ‘<div>Inline Template</div>’**
    - If HTML content multiline, we can use of `**back tics`**
      * **Ex:- template: `<div>**

**Inline Template**

**</div>`**

* + External file HTML content by specifying **templateUrl**.

1. **What are the possible ways to specify the style in @Component decorator?**

* There are 2 possible ways to specify the styles in component decorator.
  + In-line style
  + External file style

1. **What is Interpolation?**

* Interpolation {{}}
* Interpolation refers to embedding expressions into marked up text.
* Interpolation uses as its delimiter the double curly braces, {{ and }}
* In the following snippet, {{ currentCustomer }} is an example of interpolation.
* <h3>Current customer: {{ currentCustomer }}</h3>
* The text between the braces is often the name of a component property. Angular replaces that name with the string value of the corresponding component property.
* <p>{{title}}</p> <div><img src="{{itemImageUrl}}"></div>

The above example, Angular evaluates the **title** and **itemImageUrl** and fills in the blanks, first displaying some title text and then an image.

* More generally, the text between the braces is a **template expression** that Angular first **evaluates** and then **converts to a string.** The following interpolation illustrates the point by adding two numbers.

<!-- "The sum of 1 + 1 is 2" -->

<p>The sum of 1 + 1 is {{1 + 1}}.</p>

* The expression can invoke methods of the host component such as **getVal()** in the following example.

<!-- "The sum of 1 + 1 is not 4" -->

<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}.</p>

* Angular evaluates all expressions in double curly braces, converts the expression results to string, and links them with neighboring literal strings. Finally, it assigns this composite interpolated result to an **element or directive property.**

1. **A**