**Define the ng-content Directive?**

Conventional HTML elements have some content between the tags. For instance:

<p>Put your paragraph here</p>

Now consider the following example of having custom text between angular tags:

++++++++++++++++++++

<app-work>This won’t work like HTML until you use ng-content Directive</app-work>

However, doing so won’t work the way it worked for HTML elements. In order to make it work just like the HTML example mentioned above, we need to use the ng-content Directive. Moreover, it is helpful in building reusable components.

Jtijrkotlkjly

Djfjglhllh

Djdjf hi git so confudinhgjrjkekrelkke

Jefkelkf,lef

Know more about the [ng-content directive](https://blog.angular-university.io/angular-ng-content/).

**Demonstrate navigating between different routes in an Angular application.**  
 Following code demonstrates how to navigate between different routes in an Angular app dubbed “Some Search App”:

import {Router} from "@angular/router";

.

.

.

@Component({

 selector: 'app-header',

 template: `

<nav class="navbarnavbar-light bg-faded">

 <a class="navbar-brand" (click)="goHome()">Some Search App</a>

 <ul class="navnavbar-nav">

   <li class="nav-item">

     <a class="nav-link" (click)="goHome()">Home</a>

   </li>

   <li class="nav-item">

     <a class="nav-link" (click)="goSearch()">Search</a>

   </li>

 </ul>

</nav>

`

})

classHeaderComponent {

 constructor(private router: Router) {}

 goHome() {

   this.router.navigate(['']);

 }

 goSearch() {

   this.router.navigate(['search']);

 }

}

**Enumerate some salient features of Angular 7.**  
 Unlike the previous versions of Angular, the 7th major release comes with splitting in @angular/core. This is done in order to reduce the size of the same. Typically, not each and every module is required by an Angular developer. Therefore, in Angular 7 each split of the @angular/core will have no more than 418 modules.

Also, Angular 7 brings drag-and-drop and virtual scrolling into play. The latter enables loading as well as unloading elements from the DOM. For virtual scrolling, the latest version of Angular comes with the <cdk-virtual-scroll-viewport> package. Furthermore, Angular 7 comes with a new and enhanced version of the ng-compiler.

**Explain Angular Authentication and Authorization.**  
 The user login credentials are passed to an authenticate API, which is present on the server. Post server-side validation of the credentials, a JWT (JSON Web Token) is returned. The JWT has information or attributes regarding the current user. The user is then identified with the given JWT. This is called authentication.

Post logging-in successfully, different users have a different level of access. While some may access everything, access for others might be restricted to only some resources. The level of access is authorization.

Here is a detailed post on Angular 7 – JWT Authentication Example & Tutorial: http://jasonwatmore.com/post/2018/11/16/angular-7-jwt-authentication-example-tutorial

**How to generate a class in Angular 7 using CLI?**  
ng generate class Dummy [options]

This will generate a class named Dummy.

**How do Observables differ from Promises?**  
 As soon as a [promise](http://andyshora.com/promises-angularjs-explained-as-cartoon.html) is made, the execution takes place. However, this is not the case with observables because they are lazy. This means that nothing happens until a subscription is made. While promises handle a single event, observable is a stream that allows passing of more than one event. A callback is made for each event in an observable.

**Observe the following image:**

  
**What should replace the “?”?**  
 Directives. The image represents the types of directives in Angular; Attribute, structural, and custom.

**What are the building blocks of Angular?**  
 There are essentially 9 building blocks of an Angular application. These are:

1. **Components –** A component controls one or more views. Each view is some specific section of the screen. Every Angular application has at least one component, known as the [root component](https://www.learnhowtoprogram.com/javascript/angular/angular-2-setup-root-component-root-module-and-more). It is bootstrapped inside the main module, known as the root module. A component contains application logic defined inside a class. This class is responsible for interacting with the view via an API of properties and methods.
2. **Data Binding –** The mechanism by which parts of a template coordinates with parts of a component is known as data binding. In order to let Angular know how to connect both sides (template and its component), the binding markup is added to the template HTML.
3. **Dependency Injection (DI) –** Angular makes use of DI to provide required dependencies to new components. Typically, dependencies required by a component are services. A component’s constructor parameters tell Angular about the services that a component requires. So, a dependency injection offers a way to supply fully-formed dependencies required by a new instance of a class.
4. **Directives –** The templates used by Angular are dynamic in nature. Directives are responsible for instructing Angular about how to transform the DOM when rendering a template. Actually, components are directives with a template. Other [types of directives](https://angular.io/guide/attribute-directives) are attribute and structural directives.
5. **Metadata –** In order to let Angular know how to process a class, metadata is attached to the class. For doing so decorators are used.
6. **Modules –** Also known as NgModules, a module is an organized block of code with a specific set of capabilities. It has a specific application domain or a workflow. Like components, any Angular application has at least one module. This is known as the root module. Typically, an Angular application has several modules.
7. **Routing –** An Angular router is responsible for interpreting a browser URL as an instruction to navigate to a client-generated view. The router is bound to links on a page to tell Angular to navigate the application view when a user clicks on it.
8. **Services –** A very broad category, a service can be anything ranging from a value and function to a feature that is required by an Angular app. technically, a service is a class with a well-defined purpose.
9. **Template –** Each component’s view is associated with its companion template. A template in Angular is a form of HTML tags that lets Angular know that how it is meant to render the component.

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



Here is Angular Architecture in detail: https://angular.io/guide/architecture

**What is Angular Material?**  
 It is a UI component library. [Angular Material](https://material.angular.io/) helps in creating attractive, consistent, and fully functional web pages as well as web applications. It does so while following modern web design principles, including browser portability and graceful degradation.

**What is AOT (Ahead-Of-Time) Compilation?**  
 Each Angular app gets compiled internally. The Angular compiler takes in the JS code, compiles it and then produces some JS code. This happens only once per occasion per user. It is known as AOT (Ahead-Of-Time) compilation.

**What is Data Binding? How many ways it can be done?**  
 In order to connect application data with the DOM (Data Object Model), data binding is used. It happens between the template (HTML) and component (TypeScript). There are 3 ways to achieve data binding:

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

**What is demonstrated by the arrow in the following image?**

  
 This represents a dependency injection or DI.

**What is new in Angular 6?**  
 Here are some of the new aspects introduced in Angular 6:

* Angular Elements – It allows converting Angular components into web components and embeds the same in some non-Angular application
* Tree Shakeable Provider – Angular 6 introduces a new way of registering a provider directly inside the @Injectable() decorator. It is achieved by using the providedIn attribute
* RxJS 6 – Angular 6 makes use of RxJS 6 internally
* i18n (internationalization) – Without having to build the application once per locale, any Angular application can have “runtime i18n”

**What is ngOnInit ()? How to define it?**  
ngOnInit () is a lifecycle hook that is called after Angular has finished initializing all data-bound properties of a directive. It is defined as:

Interface OnInit {

          ngOnInit () : void

     }

**What is SPA** **(Single Page Application) in Angular? Contrast SPA technology with traditional web technology?**  
 In the SPA technology, only a single page, which is index.HTML, is maintained although the URL keeps on changing. Unlike the traditional web technology, SPA technology is faster and easy to develop as well.

In the conventional web technology, as soon as a client requests a webpage, the server sends the resource. However, when again the client requests for another page, the server responds again with sending the requested resource. The problem with this technology is that it requires a lot of time.

**What is the code for creating a decorator?**  
 We create a decorator called Dummy:

    function Dummy(target) {

       dummy.log('This decorator is Dummy', target); }

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

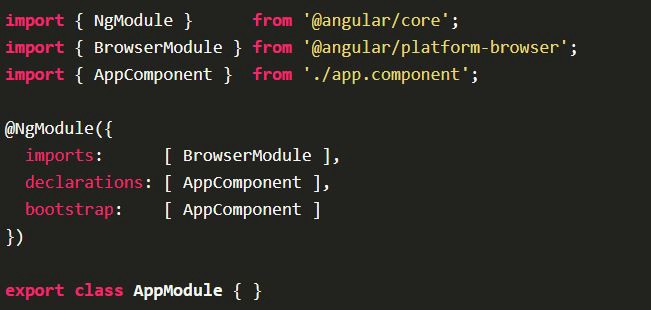
**What is ViewEncapsulation and how many ways are there do to do it in Angular?**  
 To put simply, ViewEncapsulation determines whether the styles defined in a particular component will affect the entire application or not. Angular supports 3 types of ViewEncapsulation:

* Emulated – Styles used in other HTML spread to the component
* Native – Styles used in other HTML doesn’t spread to the component
* None – Styles defined in a component are visible to all components of the application

**Why prioritize TypeScript over JavaScript in Angular?**  
TypeScript is developed by Microsoft and it is a superset of JavaScript. The issue with JS is that it isn’t a true OOP language. As the JS code doesn’t follow the Prototype Pattern, the bigger the size of the code the messier it gets. Hence, it leads to difficulties in maintainability as well as reusability. To offset this, TypeScript follows a strict OOP approach.

**What is @NgModule?**

An NgModule class describes how the application parts fit together. Every application has at least one NgModule, the root module that we bootstrap to launch the application.



Here the AppComponent is the root module of our application that Angular creates and inserts it into the index.html page.

**What are all the *metadata* properties of NgModule? And what are they used for?**

@NgModule accepts a metadata object that tells Angular how to compile and launch the application. The properties are:

* **imports** – Modules that the application needs or depends on to run like, the BrowserModule, Routing module, ReactiveFormsModule, HttpClientModuleetc that every application needs to run in a browser.
* **declarations** – the application's components, which belongs to the NgModuleclass. We must declare every component in an NgModule class. If we use a component without declaring it, we'll see a clear error message in the browser console.
* **bootstrap** – the root component that Angular creates and inserts into the index.html host web page. The application will be launched by creating the components listed in this array.
* P**roviders** – When you add a service provider to the root application injector, it’s available throughout the app. additionally, these providers are also available to all the classes in the app as long they have the lookup token. You should always provide your service in the root injector unless there is a case where you want the service to be available only if the consumer imports a particular @[NgModule](https://angular.io/api/core/NgModule).
* **Exports** -It enables an **Angular** module to use functionality that was defined in another**Angular** module. An **export** what you put is the **exports** property of the @NgModule decorator. It enables an **Angular** module to expose some of its components/directives/pipes to the other modules in the applications.

@NgModule({

declarations: [

AppComponent,

LoginComponent,

ListUserComponent,

AddUserComponent,

EditUserComponent

],

imports: [

BrowserModule,

routing,

ReactiveFormsModule,

HttpClientModule

],

providers: [AuthenticationService, UserService],

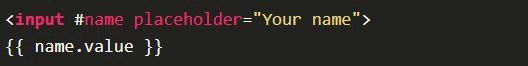
bootstrap: [AppComponent]

})

export classAppModule { }

**What is Template reference variables?**

A template reference variable (#var) is a reference to a DOM element within a template. We use hash symbol (#) to declare a reference variable in a template.

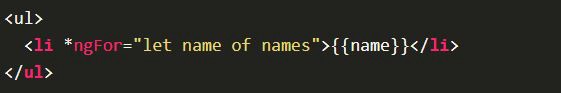


In the above code the #name declares a variable on the input element. Here the name refers to the *input* element. Now we can access any property of the inputDOM, using this reference variable. For example, we can get the value of the inputelement as name.value and the value of the placeholder property by name.placeholder anywhere in the template.

Finally, a Template reference variable refers to its attached element, component or directive. It can be accessed anywhere in the entire template. We can also use ref- instead of #. Thus we can also write the above code as ref-name.

**What are structural directives?**

Structural directives are responsible for HTML layout. They shape or reshape the DOM’s structure, typically by adding, removing, or manipulating elements. Structural directives are easy to recognize. An asterisk (\*) precedes the directive attribute name as in this example.



The ngFor directive iterates over the component's names array and renders an instance of this template for each name in that array.

Some of the other structural directives in Angular are ngIf and ngSwitch.

**What is Directive in Angular 4? How it differs from Components?**

Directives allow us to attach behavior to elements in the DOM, for example, doing something on mouse over or click. In Angular, a Directive decoraor (@Directive) is used to mark a class as an Angular directive and provides additional metadata that determines how the directive should be processed. Below are the metadata properties of a directive.

* selector - css selector that identifies this component in a template
* host - map of class property to host element bindings for events, properties and attributes
* inputs - list of class property names to data-bind as component inputs
* outputs - list of class property names that expose output events that others can subscribe to
* providers - list of providers available to this component and its children
* queries - configure queries that can be injected into the component
* exportAs - name under which the component instance is exported in a template

A Component is a directive with a template. So we should use a Component whenever we want reusable set of DOM elements with behaviors of UI. And we should use a Directive whenever we want reusable behavior to supplement the DOM.

**What are all the types of Directives?**

There are three types of directives in Angular. They are **attribute directives**, **structural directives**, and **components**.

* **Structural directives** change the DOM layout by adding and removing DOM elements. For example, \*ngIf and \*ngFor
* **Attribute directives** change the appearance or behavior of an element. . For example, \*ngStyle and \*ngClass
* **Components** are basically directives with a template.

**Q10. What are all the uses of a service?**

Services encapsulates business logic and separates them from UI concerns or the controller concerns, which governs them both.

Services focus on functionality thus benefits in maintainability. The separation of UI logic from business logic is intended to reduce the coupling between the UI layer and the Model layer, leading to a cleaner design that is easier to develop, test, and maintain.

**What is Pure and Impure Pipes?**

A pure pipe is only called when Angular detects a change in the value or the parameters passed to a pipe. An impure pipe is called for every change detection cycle no matter whether the value or parameters changes.

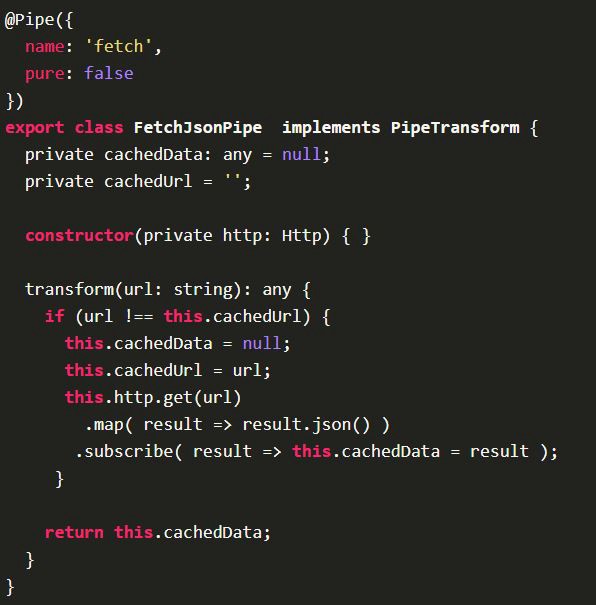
This is relevant for changes that are not detected by Angular

* when you pass an array or object that got the content changed (but is still the same instance)
* when the pipe injects a service to get access to other values, Angular doesn't recognize if they have changed.

I these cases you probably still want the pipe to be executed.

You should be aware that impure pipes are prone to be inefficient. For example when an array is passed into the pipe to filter, sort, ... then this work might be done every time change detection runs (which is quite often especially with the default ChangeDetectionStrategy setting) event though the array might not even have changed. Your pipe should try to recognize this and for example return cached results.

The code uses the Angular http client to retrieve data:



**What is Redux and @ngRx?**

Redux

* This is an application state manager for JavaScript applications, and keeps with the core principles of the Flux-architecture by having a unidirectional data flow in our application.
* Redux applications have only one global, read-only application state.
* This state is calculated by “reducing” over a collection or stream of actions that update it in controlled ways.

@ngrx

* This is a set of modules that implement the same way of managing state as well as some of the middleware and tools in the Redux ecosystem. In other way, ngrx is a collection of reactive libraries for angular, containing a redux implementation and many other useful libraries.
* Using this technique, we keep our application state in Store and everything saved in the store is read only.
* The only way to change the state is to emit an action, an object describing what happened.

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

Some of them are:

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

**How to optimize Angular app?**

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

**What is NgZone service? How Angular is notified about the changes?**

Zone.js is one of the Angular dependencies which provides a mechanism, called zones, for encapsulating and intercepting asynchronous activities in the browser (e.g.setTimeout,setInterval,promises). These zones are *execution contexts* that allow Angular to track the start and completion of asynchronous activities and perform tasks as required (e.g. change detection). Zone.js provides a global zone that can be forked and extended to further encapsulate/isolate asynchronous behaviour, which Angular does so in its NgZone service, by creating a fork and extending it with its own behaviours.

The NgZone service provides us with a number of Observables and methods for determining the state of Angular's zone and to execute code in different ways inside and outside Angular's zone.

NgZone exposes a set of Observables that allow us to determine the current status, or stability, of Angular's zone.

* onUnstable – Notifies when code has entered and is executing within the Angular zone.
* onMicrotaskEmpty - Notifies when no more microtasks are queued for execution. Angular subscribes to this internally to signal that it should run change detection.
* onStable – Notifies when the last onMicroTaskEmpty has run, implying that all tasks have completed and change detection has occurred.
* onError – Notifies when an error has occurred. Angular subscribes to this internally to send uncaught errors to its own error handler, i.e. the errors you see in your console prefixed with 'EXCEPTION:'.

We can inject the NgZone service in our component/services/etc. and can subscribe to these observables.



Subscribing to these can help you determine if your code is unexpectedly triggering change detection as a result of operations that do not affect application state.

**What is Traceur compiler?**

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

Traceur itself is written in ES6, compiled to ES5.

**What Are Components In Angular?**

**The Concepts of Angular Components  -**

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

Technically components are basically TypeScript classes that interact with the HTML files of the components, which get displayed on the browsers.

The component is the core functionality of Angular applications but you need to know to pass the data into the components to configure them.

**What's New In Angular 6? What Are Improvements In Angular 6?**

The Angular Team are working on lots of bug fixes, new features and added/update/remove/ re-introduce/ and many more things.

**Let’s start to explore all changes of Angular 6 step by step:**

**Added ng update -** This CLI commands will update your angular project dependencies to their latest versions. The ng update is normal package manager tools to identify and update other dependencies.

**What Are The Ngmodule Metadata Properties?**

The NgModule decorator identifies AppModule as aNgModule class.

The NgModule takes a metadata object that tells Angular how to compile and launch the application.

**The NgModule importance metadata properties are as follows –**

* + providers : service classes
  + declarations : components
  + imports: module to run the application like browserModule, client module, router module
  + exports: components, to available for other modules ( means packages)
  + entryComponents
  + bootstrap: app component , the component to start with
  + schemas
  + id

**What Types Of Ngmodules?**

There are four types of NgModules –

* + Features Module
  + Routing Module
  + Service Module
  + Widget Module
  + Shared Module

**What Is A Cookie?**

A cookie is a small piece of data sent from a website and stored on the user's machine by the user's web browsers while the user is browsing.

**What Is Pure Pipe?**

Angular executes a pure pipe only when it detects a pure change to the input value. A pure change can be primitive or non-primitive.

Primitive data are only single values, they have not special capabilities and the non-primitive data types are used to store the group of values.

@Pipe({

**name**: 'currency'

})

**What Is 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.

If you want to make a pipe impure that time you will allow the setting pure flag to false.

@Pipe({

**name:** 'currency',

  pure:false

})

**What Is Parameterizing Pipe?**

A pipe can accept any number of optional parameters to achieve output. The parameter value can be any valid template expressions. To add optional parameters follow the pipe name with a colon (:). Its looks like- currency: 'INR'

**In the following example –**

<h2>The birthday is - {{ birthday | date:"MM/dd/yy" }} </h2>

<!-- Output - The birthday is - 10/03/1984 -->

**What Is Chaining Pipe?**

The chaining Pipe is used to perform the multiple operations within the single expression. This chaining operation will be chained using the pipe (I).

In the following example, to display the birthday in the upper case- will need to use the inbuilt date-pipe and upper-case-pipe.

**In the following example –**

{{ birthday | date | uppercase}}

**Why You Use Browsermodule, Commonmodule, Formsmodule, Routermodule, AndHttpclientmodule?**

**BrowserModule –** The browser module is imported from @angular/platform-browser and it is used when you want to run your application in a browser.

**CommonModule –** The common module is imported from @angular/common and it is used when you want to use directives - NgIf, NgFor and so on.

**FormsModule –** The forms module is imported from @angular/forms and it is used when you build template driven forms.

**RouterModule –** The router module is imported from @angular/router and is used for routing RouterLink, forRoot, and forChild.

**HttpClientModule –**The HttpClientModule is imported from @angular/common/http and it used to initiate HTTP request and responses in angular apps. The HttpClient is more modern and easy to use the alternative of HTTP.