https://www.guru99.com/angularjs-testing-unit.html
https://www.techgeeknext.com/angular/angular-interview-questions

How is SPA (Single Page Application) technology different from traditional web technology?

In traditional web technology, the client requests for a web page (HTML/JSP/asp) and the server sends the resource (or HTML page), and the client again requests for another page and the server responds with another resource. The problem here is a lot of time is consumed in requesting/responding or due to a lot of reloading. Whereas, in the SPA technology, we maintain only one page (index.HTML) even though the URL keeps on changing.

What is AngularJS? (Required Software: NPM, Ionic, Git/SVN, Visual Studio Code)

AngularJS has been introduced by Google. It is open source, completely free, and used by thousands of developers around the world. AngularJS provides a clean implementation MVC (Model View Controller) way to develop cross browser and cross platform web and mobile application development.

ES6 includes the following new features:

- Arrows Function and String Interpolation

```
const greetings = (name) => {
   return `hello ${name}`;
}
```

- Modules exporting & importing

```
const myModule = { x: 1, y: () => { console.log('This is ES5') }}
export default myModule;
import myModule from './myModule';
```

- Classes & Interface

ES6 introduces language support for classes (class keyword), constructors (constructor keyword), and the extend keyword for inheritance.

- Default parameter values

```
// Basic syntax
function multiply (a, b = 2) {
  return a * b;
}
multiply(5); // 10
```

- enhanced object literals
- template strings
- destructuring
- default + rest + spread
- let + const
- let allows developers to scope variables at the block level. Let doesn't hoist in the same way var does.

```
const NAMES = [];
NAMES.push("Jim");
console.log(NAMES.length === 1);
```

- iterators + for..of

The for...of statement creates a loop iterating over iterable objects.

- map + set + weakmap + weakset
- math + number + string + array + object APIs
- binary and octal literals

What's new in Angular 5?

Certain tools are optimized in the new version of Angular, let us see what the tools are:
Angular 5 supports Typescript version 2.4
Angular 5 supports RxJS 5.5 which has new features like Pipeable Operators
A build tool to make the js bundles (files) lighter
Ahead of Time (AOT) is updated to be on by default
Events like ActivationStart and ActivationEnd are introduced in Router

What is TypeScript?
TypeScript is a superset of JavaScript created by Microsoft that adds optional types, classes, async/await, and many other features, and compiles to plain JavaScript. Angular built entirely in TypeScript and used as a primary language. You can install it globally as **npm install -g typescript**

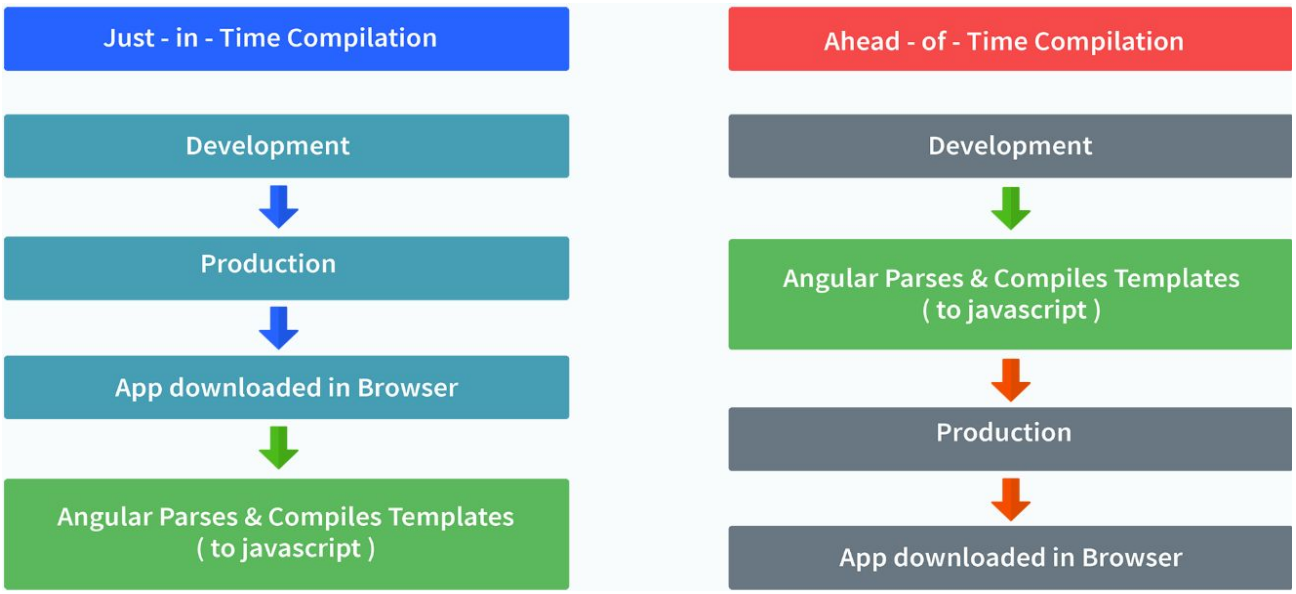
Angular 5 vs 6+
Arrows: These are a function which is described by the '=>' syntax
Objects: Object literals are used to support the prototype assignments.
Classes: ES6 classes can be easily implemented over the prototype based object oriented pattern
Destructing: It allows binding patterns which is based primarily on the pattern matching.
String interpolation, Default, Spread, Module Loaders, **Weak set**, **Map**, New Library, **Promises, Proxies**

What is Component in Angular Terminology?
A web page in Angular has many components involved in it. A Component is basically a block in which the data can be displayed on HTML using some logic usually written in typescript.

What is Transpiling in Angular?
Transpiling is the process of converting the typescript into javascript (using Traceur, a JS compiler). Though typescript is used to write code in the Angular applications, the code is internally transpiled into javascript.

How to handle Events in Angular 5?
Any activity (button click, mouse click, mouse hover, mouse move, etc) of a user on a frontend/web screen is termed as an event. Such events are passed from the view (.HTML) page to a typescript component (.ts).

Explain Authentication and Authorization.
Authentication: The user login credentials are passed to an authenticate API (on the server). On the server side validation of the credentials happens and a JSON Web Token (JWT) is returned. JWT is a JSON object that has some information or attributes about the current user. Once the JWT is given to the client, the client or the user will be identified with that JWT.
Authorization: After logging in successfully, the authenticated or genuine user does not have access to everything. The user is not authorized to access someone else's data, he/she is authorized to access some data.
Just in-time (JIT) compilation: This is a standard development approach which compiles our Typescript and html files in the browser at runtime, as the application loads. It is great but has disadvantages. Views take longer to render because of the in-browser compilation step. App size increases as it contains angular compiler and other library code that won't actually need.
Ahead-of-time (AOT) compilation: Every angular application gets compiled internally. The angular compiler takes javascript code, compiles it and produces javascript code again. Ahead-of-Time Compilation does not happen every time or for every user, as is the case with Just-In-Time (JIT) Compilation.
With AOT, the compiler runs at the build time and the browser downloads only the pre compiled version of the application. The browser loads executable code so it can render the application immediately, without waiting to compile the app first. This compilation is better than JIT because of Fast rendering, smaller application size, security and detect template errors earlier.



Key Components/Features of Angular / building blocks of Angular

- Modules − This is used to break up the application into logical pieces of code. Each piece of code or module is designed to perform a single task.
- Components − These are the basic building blocks of angular application to control HTML views. And can be used to bring the modules together.
- Templates − This is used to define the views of an Angular application.
- Service − This is used to create components which can be shared across the entire application. So for example if you had a data component that picked data from a database, you could have it as a shared service that could be used across multiple applications.
- Metadata − This can be used to add more data to an Angular JS class.
- Directives - Directives add behaviour to an existing DOM element or an existing component instance.
- Data Binding - Data Binding happens between the HTML (template) and typescript (component). Data binding can be done in 3 ways:
  - (i) Property Binding: `<input type="email" [value]="user.email"> {{ user.name }}`
  - (ii) Event Binding: `<button (click)="logout()"></button>`
  - (iii) Two-Way Data Binding: `<input type="email" [(ngModel)]="user.email">`
- Routing - Angular Router is a mechanism in which navigation happens from one view to the next as users perform application tasks. It borrows the concepts or model of browser's application navigation.
- Dependency Injection -

How do you describe various dependencies in an angular application?

The dependencies section of package.json with in an angular application can be divided as follow,
Angular packages: Angular core and optional modules; their package names begin @angular/.
Support packages: Third-party libraries that must be present for Angular apps to run.
Polyfill packages: Polyfills plug gaps in a browser's JavaScript implementation.

What is the difference between AngularJS and Angular?

Angular is a completely revived component-based framework in which an application is a tree of individual components.

Angular JS	Angular
It is based on MVC architecture	This is based on Service/Controller
This uses use JavaScript to build the application	Introduced the typescript to write the application
Based on controllers concept	This is a component based UI approach
Not a mobile friendly framework	Developed considering mobile platform

Introduction to modules

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.
The NgModule decorator identifies AppModule as a NgModule class.
The NgModule takes a metadata object that tells Angular how to compile and launch the application.

NgModule metadata

- declarations: The components, directives, and pipes that belong to this NgModule.
- exports: The subset of declarations that should be visible and usable in the component templates of other NgModules.
- imports: Other modules whose exported classes are needed by component templates declared in this NgModule.

- providers: Creators of services that this NgModule contributes to the global collection of services; they become accessible in all parts of the app. (You can also specify providers at the component level, which is often preferred.)
- bootstrap: The main application view, called the root component, which hosts all other app views. Only the root NgModule should set the bootstrap property.

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

There are four types of NgModules –

- Features Module -
- Routing Module -
- Service Module -
- Widget Module -
- Shared Module -

Introduction to Component Components are a logical piece of code for Angular application.

- Template − This is used to render the view for the application. This contains the HTML that needs to be rendered in the application. This part also includes the binding and directives.
- Class − This is like a class defined in any language such as C. This contains properties and methods. This has the code which is used to support the view. It is defined in TypeScript.
- Metadata − This has the extra data defined for the Angular class. It is defined with a decorator.

Template is view/html code there are two type of template inline template and templateURL.
Metadata The @Component decorator identifies the class immediately below it as a component class, and specifies its metadata.

```
@Component({
  selector:    'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers:  [ HeroService ]
})
export class HeroListComponent implements OnInit {
/* . . . */
}
```

@Component configuration options:

- **selector:** A CSS selector that tells Angular to create and insert an instance of this component wherever it finds the corresponding tag in template HTML. For example, if an app's HTML contains <app-hero-list></app-hero-list>, then Angular inserts an instance of the HeroListComponent view between those tags.
- **templateUrl:** The module-relative address of this component's HTML template. Alternatively, you can provide the HTML template inline, as the value of the template property. This template defines the component's host view.
- **Providers:** An array of providers for services that the component requires. In the example, this tells Angular how to provide the HeroService instance that the component's constructor uses to get the list of heroes to display.

Introduction to Template A template is HTML view where you can display data by binding controls to properties of an Angular component. You can store your component's template in one of two places.

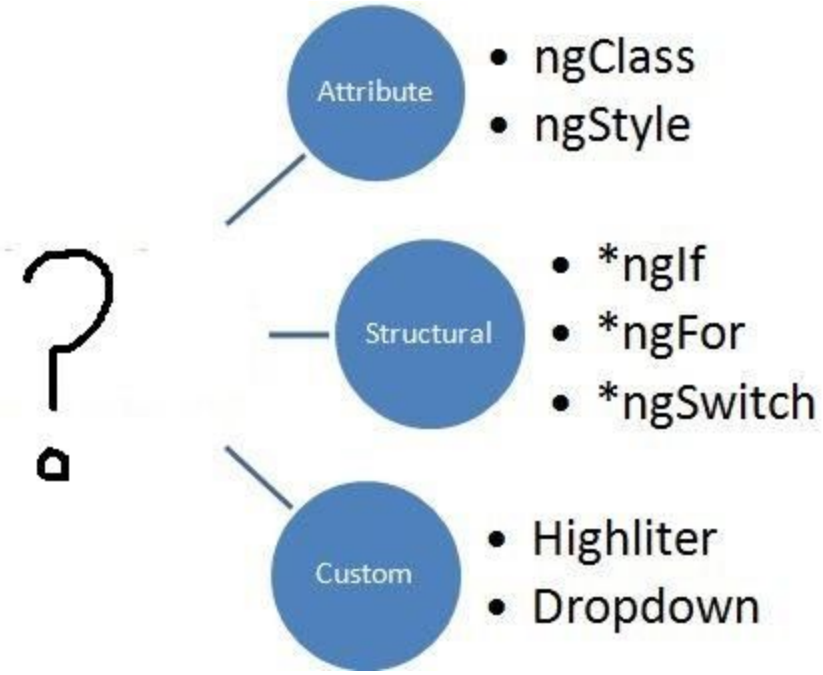
Introduction to Service A service is used when a common functionality needs to be provided to various modules. Services allow for greater separation of concerns for your application and better modularity by allowing you to extract common functionality out of components. Let's create a repoService which can be used across components.

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';

@Injectable({ // The Injectable decorator is required for dependency injection to work
  // providedIn option registers the service with a specific NgModule
  providedIn: 'root',  // This declares the service with the root app (AppModule)
})
export class RepoService{
  constructor(private http: Http){
  }

  fetchAll(){
    return this.http.get('https://api.github.com/repositories');
  }
}
```

Introduction to Module Modules are logical boundaries in your application and the application is divided into separate modules to separate the functionality of your application. Lets take an example of app.module.ts root module declared with @NgModule decorator as below

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }  from './app.component';
@NgModule ({
   imports:      [ BrowserModule ],
   declarations: [ AppComponent ],
   bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Introduction to Directives directive changes the appearance or behavior of a DOM element. There are three kinds of directives in Angular:



- **Components**—directives with a template.
- **Structural directives**—change the DOM layout by adding and removing DOM elements. Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements. `*ngIf`, `*ngFor`, `*ngSwitchCase`, `ngModel`, `*appUnless` Creating a directive is similar to creating a component.
    - Import the Directive decorator (instead of the Component decorator).

- Import the Input, TemplateRef, and ViewContainerRef symbols; you'll need them for any structural directive.
- Apply the decorator to the directive class.
- Set the CSS attribute selector that identifies the directive when applied to an element in a template.

```
<p *appUnless="condition">Show this sentence unless the condition is true.</p>
import { Directive, Input, TemplateRef, ViewContainerRef } from '@angular/core';
@Directive({
      selector: '[appUnless]'
})
export class UnlessDirective {
      constructor() { }
}
```

```
<ul>
  <li *ngFor="let name of names">{{name}}</li>
</ul>
```

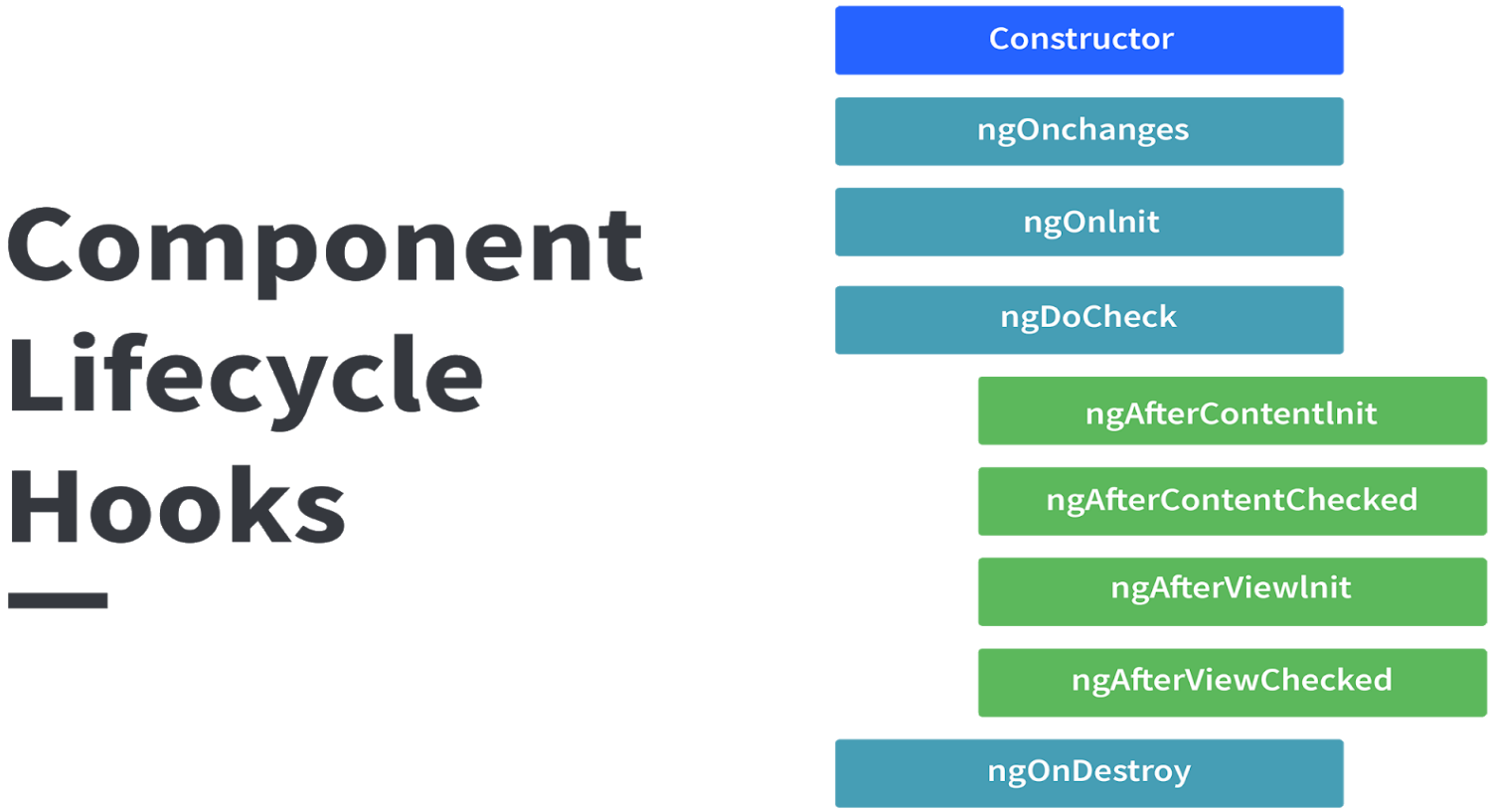
- **Attribute directives**—change the appearance or behavior of an element, component, or another directive.
  For example, *ngStyle and *ngClass

```
<p appHighlight>Highlight me!</p>
import { Directive } from '@angular/core';
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor() { }
}
```

Component lifecycle hooks overview

After creating a component/directive by calling its constructor, Angular calls the lifecycle hook methods in the following sequence at specific moments:

OnChange() - OnInit() - DoCheck() - AfterContentInit() - AfterContentChecked() - AfterViewInit() - AfterViewChecked() - OnDestroy().



Hook	Purpose and Timing
ngOnChanges()	Respond when Angular (re)sets data-bound input properties. The method receives a SimpleChanges object of current and previous property values. Called before ngOnInit() and whenever one or more data-bound input properties change.
ngOnInit()	Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties. Called once, after the first ngOnChanges().
ngDoCheck()	Detect and act upon changes that Angular can't or won't detect on its own. Called during every change detection run, immediately after ngOnChanges()and ngOnInit().
ngAfterContentInit()	Respond after Angular projects external content into the component's view / the view that a directive is in. Called once after the first ngDoCheck().
ngAfterContentChecked()	Respond after Angular checks the content projected into the directive/component. Called after the ngAfterContentInit() and every subsequent ngDoCheck().
ngAfterViewInit()	Respond after Angular initializes the component's views and child views / the view that a directive is in. Called once after the first ngAfterContentChecked().
ngAfterViewChecked()	Respond after Angular checks the component's views and child views / the view that a directive is in. Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().
ngOnDestroy()	Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks. Called just before Angular destroys the directive/component.

Which of the Angular life cycle component execution happens when a data-bound input value updates?

ngOnChanges is the life cycle hook that gets executed whenever a change happens to the data that was bound to an input.

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

Annotations in angular are “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 that are used for separating decoration or modification of a class without actually altering the original source code.

What are the basic rules of Metadata/Decorators?

If you come from a Java background they might look to you just like java annotations. They can be used for all the purposes runtime annotations are used but they are more powerful Annotations are merely a mechanism to store metadata on a type. An annotation on its own does not add new behaviour to its target – to make any effect it needs a processor that can act based on the stored metadata. Decorators on the other hand are functions that take their target as the argument. With decorators we can run arbitrary code around the target execution or even entirely replace the target with a new definition.

1. Class decorators

A class decorator is a function that accepts a constructor function and returns a constructor function. Returning undefined is equivalent to returning the constructor function passed in as argument.

```
import { NgModule, Component } from '@angular/core';

@Component({
  selector: 'my-component',
  template: '<div>Class decorator</div>',
})
export class MyComponent {
  constructor() {
    console.log('Hey I am a component!');
  }
}

@NgModule({
  imports: [],
  declarations: [],
})
export class MyModule {
  constructor() {
    console.log('Hey I am a module!');
  }
}
```

2. Method decorators

A method decorator is a function that accepts 3 arguments: the object on which the method is defined, the key for the property (a string name or symbol) and a property descriptor. The function returns a property descriptor; returning undefined is equivalent to returning the descriptor passed in as argument.

```
import { Component, HostListener } from '@angular/core';

@Component({
    selector: 'my-component',
    template: '<div>Method decorator</div>'
})
export class MyComponent {
    @HostListener('click', ['$event'])
    onHostClick(event: Event) {
        // clicked, `event` available
    }
}
```

3. Property decorators

Property decorators are similar to method decorators. The only difference is they do not accept property descriptors as arguments and do not return anything.

```
import { Component, Input } from '@angular/core';

@Component({
    selector: 'my-component',
    template: '<div>Property decorator</div>'
})

export class MyComponent {
    @Input()
    title: string;
}
```

4. Parameter decorators

A parameter decorator is a function that accepts 3 arguments: the object on which the method is defined or the construction function if the decorator is on a constructor argument, the key for the method (a string name or symbol) or undefined in case of constructor argument and the index of the parameter in the argument list. A property decorator does not return anything.

```
import { Component, Inject } from '@angular/core';
import { MyService } from './my-service';

@Component({
    selector: 'my-component',
    template: '<div>Parameter decorator</div>'
})
export class MyComponent {
    constructor(@Inject(MyService) myService) {
        console.log(myService); // MyService
    }
}
```

Dependency Injection

DI is wired into the Angular framework and used everywhere to provide new components with the services or other things they need. Components consume services; that is, you can inject a service into a component, giving the component access to that service class.

To define a class as a service in Angular, use the **@Injectable()** decorator to provide the metadata that allows Angular to inject it into a component as a dependency.

- The injector is the main mechanism. Angular creates an application-wide injector for you during the bootstrap process, and additional injectors as needed. You don't have to create injectors.
- An injector creates dependencies, and maintains a container of dependency instances that it reuses if possible.
- A provider is an object that tells an injector how to obtain or create a dependency.

Services encapsulate 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 the difference between constructor and ngOnInit?

TypeScript classes have a default method called constructor which is normally used for initialization purposes. Whereas ngOnInit method is specific to Angular, especially used to define Angular bindings. Even though the constructor gets called first, it is preferred to move all of your Angular bindings to ngOnInit method. In order to use ngOnInit, you need to implement the OnInit interface as below.

```
export class App implements OnInit{
  constructor(){
     //called first time before the ngOnInit()
  }

  ngOnInit(){
     //called after the constructor and called  after the first ngOnChanges()
  }
}
```

ngOnInit and ionViewDidLoad:

ngOnInit is a life cycle hook called by Angular to indicate that Angular is done creating the component.

ionViewDidLoad is related to the Ionic's NavController lifeCycle events. It runs when the page has loaded. This event only happens once per page being created.

Basically both are good places for initializing the component's data.

What is Interpolation or String interpolation?

Interpolation is a special syntax that Angular converts into property binding. It's a convenient alternative to property binding. It is represented by double curly braces({{}}). 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. Let's take an example.

```
<h3>
  {{title}}
  <img src="{{url}}" style="height:30px">
</h3>
```

What are template expressions?

A template expression produces a value similar to any Javascript expressions. Angular executes the expression and assigns it to a property of a binding target; the target might be an HTML element, a component, or a directive. In the property binding, a template expression appears in quotes to the right of the = symbol as in [property]="expression". In interpolation syntax, the template expression is surrounded by double curly braces. For example, in the below interpolation, the template expression is {{username}},

```
<h3>{{username}}, welcome to Angular</h3>
```

The below javascript expressions are prohibited in template expression

i. assignments (=, +=, -=, ...)
ii. new
iii. chaining expressions with ; or ,
iv. increment and decrement operators (++ and --)

What are the Pipes?

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

This feature is used to change the output on the template; something like changing the string into uppercase and displaying it on the template. It can also change the Date format accordingly.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-birthday',
  template: `<p>Birthday is {{ birthday | date }}</p>`
})
export class BirthdayComponent {
  birthday = new Date(1987, 6, 18); // June 18, 1987
}
```

What is a parameterized pipe?

A pipe can accept any number of optional parameters to fine-tune its output. The parameterized pipe can be created by declaring the pipe name with a colon (:) and then the parameter value. If the pipe accepts multiple parameters, separate the values with colons. Let's take a birthday example with a particular format(dd/mm/yyyy):

```
import { Component } from '@angular/core';

    @Component({
      selector: 'app-birthday',
      template: `<p>Birthday is {{ birthday | date:'dd/mm/yyyy'}}</p>` // 18/06/1987
    })
    export class BirthdayComponent {
      birthday = new Date(1987, 6, 18);
    }
```

What Is Chaining Pipe?

The chaining Pipe is used to perform 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}}

What is custom Pipe/Filter?

Apart from built-in pipes, you can write your own custom pipe, You can create custom reusable pipes for the transformation of existing value.

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'customFileSizePipe'})
export class FileSizePipe implements PipeTransform {
  transform(size: number, extension: string = 'MB'): string {
    return (size / (1024 * 1024)).toFixed(2) + extension;
  }
}
```

```
import { Injectable, Pipe, PipeTransform } from '@angular/core';
@Pipe({
        name: 'ateTagModeratorFilter'
})
@Injectable()
export class ATETagModeratorFilterPipe implements PipeTransform {
        transform(items: any[], field: string, searchKey: string): any[] {
        if(!items) return [];
        if(!searchKey) return items;
        searchKey = searchKey.toLowerCase();
        return items.filter( it => {
        if(it.ContactPerson!=null && it.Name!=null)
        return (it.ContactPerson.toLowerCase().includes(searchKey) ||
it.Name.toLowerCase().includes(searchKey));
        });
        }
}
```

```
<ion-item [style.display]="(tagMemberList | ateTagModeratorFilter : 'Name' : searchKey).length == 0 ?
'block' : 'none'">
 No result found.
</ion-item>
```

What is an AsyncPipe in Angular?

When an observable or promise returns something, we use a temporary property to hold the content. Later, we bind the same content to the template. With the usage of AsyncPipe, the promise or observable can be directly used in a template and a temporary property is not required.

What are Pure and Impure Pipes?

Pure pipes are stateless that flow input date without remembering anything or causing detectable side-effects. Pipes are pure by default, hence most pipes are pure. We can make a pipe impure by setting its pure flag to false. Angular executes a pure pipe only when it detects a pure change to the input value. A pure change is either a change to a primitive input value or a changed object reference.

Impure pipes are those which can manage the state of the data they transform. A pipe that creates an HTTP request, stores the response and displays the output, is an impure or stateful pipe. Stateful Pipes should be used cautiously. Angular provides AsyncPipe, which is stateful. In the following code, the pipe only calls the server when the request URL changes and it caches the server response. The code uses the Angular http client to retrieve data:

```
@Pipe({
  name: 'fetch',
  pure: false
})
export class FetchJsonPipe  implements PipeTransform {
  private cachedData: any = null;
  private cachedUrl = '';

  constructor(private http: Http) { }

  transform(url: string): any {
    if (url !== this.cachedUrl) {
      this.cachedData = null;
      this.cachedUrl = url;
      this.http.get(url)
        .map( result => result.json() )
       .subscribe( result => this.cachedData = result );
     }

    return this.cachedData;
  }
}
```

What is Observables?

Observables provide support for passing messages between publishers and subscribers in your application. Observables offer significant benefits over other techniques for event handling, asynchronous programming, and handling multiple values.

Observables are declarative—that is, you define a function for publishing values, but it is not executed until a consumer subscribes to it. The subscribed consumer then receives notifications until the function completes, or until they unsubscribe.

An observable can deliver multiple values of any type—literals, messages, or events, depending on the context. The API for receiving values is the same whether the values are delivered synchronously or asynchronously.

As a publisher, you create an Observable instance that defines a subscriber function. This is the function that is executed when a consumer calls the **subscribe()** method. The subscriber function defines how to obtain or generate values or messages to be published.

Notification	Description
next	Required. A handler for each delivered value. Called zero or more times after execution starts.
error	Optional. A handler for an error notification. An error halts execution of the observable instance.
complete	Optional. A handler for the execution-complete notification. Delayed values can continue to be delivered to the next handler after execution is complete.

```
const source = range(1, 5);
// Create observer object
```

```
const myObserver = {
  next: x => console.log('Observer got a next value: ' + x),
  error: err => console.error('Observer got an error: ' + err),
  complete: () => console.log('Observer got a complete notification'),
};
// Execute with the observer object and Prints out each item
myObservable.subscribe(myObserver);     OR
myObservable.subscribe({
  next(num) { console.log('Next num: ' + num)},
  error(err) { console.log('Received an errror: ' + err)}
});
```

RxJS https://www.techgeeknext.com/angular/angular-interview-questions

RxJS is a library for composing asynchronous and callback-based code in a functional, reactive style using Observables. Many APIs such as HttpClient produce and consume RxJS Observables and also use operators for processing observables.

What are the operators in RxJS ?

- tap()
- catchError()
- switchAll()
- finalize()
- throwError()

What are the utility functions provided by RxJS?

The RxJS library also provides below utility functions for creating and working with observables.

i. Converting existing code for async operations into observables
ii. Iterating through the values in a stream
iii. Mapping values to different types
iv. Filtering streams
v. Composing multiple streams

What is multicasting?

Multi-casting is the practice of broadcasting to a list of multiple subscribers in a single execution. Let's demonstrate the multi-casting feature.

```
var source = Rx.Observable.from([1, 2, 3]);
var subject = new Rx.Subject();
var multicasted = source.multicast(subject);

// These are, under the hood, `subject.subscribe({...})`:
multicasted.subscribe({
  next: (v) => console.log('observerA: ' + v)
});
multicasted.subscribe({
  next: (v) => console.log('observerB: ' + v)
});
```

What is an rxjs subject in Angular?

An RxJS Subject is a special type of Observable that allows values to be multicasted to many Observers. While plain Observables are unicast (each subscribed Observer owns an independent execution of the Observable), Subjects are multicast.

A Subject is like an Observable, but can multicast to many Observers. Subjects are like EventEmitters: they maintain a registry of many listeners.

```
import { Subject } from 'rxjs';

   const subject = new Subject<number>();

   subject.subscribe({
```

```
    next: (v) => console.log(`observerA: ${v}`)
  });
  subject.subscribe({
    next: (v) => console.log(`observerB: ${v}`)
  });

  subject.next(1);
  subject.next(2);
```

What is the difference between BehaviorSubject and Observable ?

Observable is stateless	BehaviourSubject is stateful
Observable is unidirectional	BehaviourSubject is bidirectional
Observable creates copy of data	BehaviourSubject shares data

What is the difference between Hot and Cold Observables?

COLD is when your observable creates the producer	HOT is when your observable closes over the producer
Emits at controller rate when Observer is ready	Emits immediately whether Observer is ready or not
Created with OnSubscribe	Created with Timer Events

Why Do You Use Browsermodule, Commonmodule, Formsmodule, Routermodule, And Httpclient Module?

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 requests and responses in angular apps. The HttpClient is more modern and easy to use the alternative of HTTP.

Angular Module to API Call:

HttpClientModule is a module provided by the @angular/common/http package that has the classes and services that we can use to implement a network interface in our Ionic 4 and Angular Apps.

Source Code of API Call

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';

@Injectable({ // The Injectable decorator is required for dependency injection to work
  // providedIn option registers the service with a specific NgModule
  providedIn: 'root',  // This declares the service with the root app (AppModule)
})
export class RepoService{
  constructor(private http: Http){
  }

  fetchAll(){
    return this.http.get('https://api.github.com/repositories');
  }
}
```

i. Import HttpClient into root module:

```
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule,
  ],})
 export class AppModule {}
```

ii. Inject the HttpClient into the application: Let's create a userProfileService(userprofile.service.ts) as an example. It also defines get method of HttpClient

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
const userProfileUrl: string = 'assets/data/profile.json';
@Injectable()
export class UserProfileService {
  constructor(private http: HttpClient) { }

  getUserProfile() {
    return this.http.get(this.userProfileUrl);
  }
}
```

iii. Create a component for subscribing service: Let's create a component called UserProfileComponent(userprofile.component.ts) which inject UserProfileService and invokes the service method,

```
fetchUserProfile() {
  this.userProfileService.getUserProfile()
    .subscribe((data: User) => this.user = {
        id: data['userId'],
        name: data['firstName'],
        city:  data['city']
    });
}
```

What is ViewEncapsulation or Shadow DOM?
ViewEncapsulation decides whether the styles defined in a component can affect the entire application or not. There are three ways to do this in Angular:
ViewEncapsulation.None: styles defined in a component are visible to all components.
Angular doesn't use Shadow DOM at all. Styles applied to our component are written to the document head.
ViewEncapsulation.Emulated: styles from other HTML spread to the component.
This view encapsulation is used by default. ViewEncapsulation.Emulated emulates style encapsulation, even if no Shadow DOM is available. This is a very powerful feature in case you want to use a third-party component that comes with styles that might affect your application.

ViewEncapsulation.Native: styles from other HTML do not spread to the component.
Last but not least, we have the native Shadow DOM view encapsulation. This one is super simple to understand since it basically just makes Angular using native Shadow DOM.

What is the use of @Input and @Output?

When it comes to the communication of Angular Components, which are in Parent-Child Relationship; we use @Input in Child Component when we are passing data from Parent to Child Component and @Output is used in Child Component to receive an event from Child to Parent Component.

In Angular, you can pass data from parent component to child component using: @Input()
In Angular, you can pass data from child component to parent component using: @Output()

@input & @output:

Parent TO Child @Input

Parent

```
messageToSendP: string = '';
passBall(id: number) {
        if (id == 1) {
        this.child1.futbols.pop();
        this.child2.futbols.push(true);
        } else if (id == 2) {
        this.child2.futbols.pop();
        this.child1.futbols.push(true);
        }
 }
<app-child [child]="child1" (emitPass)="passBall($event)"></app-child>
```

Child

```
@Input() receivedParentMessage: string;
```

Child TO Parent @Output

Parent

```
receivedChildMessage: string;
getMessage(message: string) {
        this.receivedChildMessage = message;
 }
<app-child (messageToEmit)="getMessage($event)"></app-child>
```

Child

```
@Output() messageToEmit = new EventEmitter<string>();
<button (click)="sendMessageToParent(messageToSendC)">Send to Parent</button>
sendMessageToParent(message: string) {
        this.messageToEmit.emit(message)
}
```

Child to parent using @ViewChild decorator

Child:

```
@Component({
      selector: 'app-child',
      template:`
        <p>{{data}}</p>
```

```
    `,
    styleUrls: ['./child.component.css']
  })
  export class ChildComponent {
    data:string = "Message from child to parent";
    constructor() { }
  }
```

Parent:

```
@Component({
    selector: 'app-parent',
    template: `
      <p>{{dataFromChild}}</p>
    `,
    styleUrls: ['./parent.component.css']
  })
  export class ParentComponent implements AfterViewInit {
    dataFromChild: string;
    @ViewChild(ChildComponent,{static:false}) child;

    ngAfterViewInit(){
      this.dataFromChild = this.child.data;
    }
    constructor() { }
  }
```

If you do not know the number of arguments to be passed to function in advance, you should use _____ parameter type.

```
var print_names = function(...names) { for (let i=0; i<names.length; i++) console.log(names[i]); }
```

keyword is used to access class's member variables and functions inside class member function

Let var, const

EventEmitter

Use in directives and components to emit custom events synchronously or asynchronously, and register handlers for those events by subscribing to an instance.

What is ng-content Directive?

The HTML elements like p (paragraph) or h1 (heading) have some content between the tags. For example, this is a paragraph and <h1>this is a heading</h1>. Now, similar to this, what if we want to have some custom text or content between the angular tags like <app-tax>some tax-related content</app-tax> This will not work the way it worked for HTML elements. Now, in such cases, the <ng-content> tag directive is used.

What does a router.navigate do?

When we want to route to a component we use router.navigate. Syntax:
this.router.navigate(['/component_name']);

What is a RouterOutlet?

RouterOutlet is a substitution for templates rendering the components. In other words, it represents or renders the components on a template at a particular location.

Explain the usage of {{}}?

The set of brackets {{}} when used with an HTML tag, represent data from a component. For example, on an HTML page which has <h1>{{variableName}}</h1>, here the 'variableName' is actually typescript

(component) data representing its value on the template; i.e., HTML. This entire concept is called String Interpolation.

What is the purpose of using package.json in the angular project?

With the existence of package.json, it will be easy to manage the dependencies of the project. If we are using typescript in the angular project then we can mention the typescript package and version of typescript in package.json.

What are ngModel and how do we represent it?

ngModel is a directive which can be applied on a text field. This a two-way data binding. ngModel is represented by [()]

What does a Subscribe method do in Angular 4?

It is a method which is subscribed to an observable. Whenever the subscribe method is called, an independent execution of the observable happens.

Differentiate between Observables and Promises.

Observables are lazy, which means nothing happens until a subscription is made. Whereas Promises are eager; which means as soon as a promise is created, the execution takes place. Observable is a stream in which passing of zero or more events is possible and the callback is called for each event. Whereas, promise handles a single event.

Promise	Observable
Emits a single value	Emits multiple values over a period of time
Not Lazy	Lazy. An observable is not called until we subscribe to the observable
Cannot be cancelled	Can be cancelled by using the unsubscribe() method
	Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc.

Differentiate between ng-Class and ng-Style.

In ng-Class, loading of CSS class is possible; whereas, in ng-Style we can set the CSS style.

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.

```
<input #name placeholder="Your name">
{{ name.value }}
```

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 input element 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 is a 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.

What is a bootstrapping module?
Every application has at least one Angular module, the root module that you bootstrap to launch the application is called a bootstrapping module. It is commonly known as AppModule. The default structure of AppModule generated by AngularCLI would be as follows.

```
/* JavaScript imports */
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';

/* the AppModule class with the @NgModule decorator */
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Explain the difference between 'ionic prepare' and 'ionic build.'
The 'ionic prepare' function copies all the files from the WWW folder to the target platform's WWW folder, whereas the 'ionic build' function performs the same while building the app's source code so that the app can be run on an emulator/simulator or a device.

Angular Form
Every form-intensive application has to provide answers for the following problems:
how to keep track of the global form state
know which parts of the form are valid and which are still invalid
properly displaying error messages to the user so that the user knows what to do to make the form valid.

Template Driven Forms:
Forms built with this directive can only be tested in an end to end test because this requires the presence of a DOM, but still, this mechanism is very useful and simple to understand.
Angular now provides an identical mechanism named also ngModel, that allow us to build what is now called Template-Driven forms.
Template Driven from a functional programming point of view and Need to import FormsModule template.

```
<section class="sample-app-content">
    <h1>Template-driven Form Example:</h1>
    <form #f="ngForm" (ngSubmit)="onSubmitTemplateBased()">
        <p>
            <label>First Name:</label>
            <input type="text"
                [(ngModel)]="user.firstName" required>
        </p>
        <p>
```

```
            <label>Password:</label>
            <input type="password"
                [(ngModel)]="user.password" required>
        </p>
        <p>
            <button type="submit" [disabled]="!f.valid">Submit</button>
        </p>
    </form>
</section>
```

Reactive Forms (or Model Driven)
We imported ReactiveFormsModule forms directeve. and its Functional Reactive Programming.
Notice also that the required validator attribute is not applied to the form controls. This means the validation logic must be somewhere in the controller, where it can be unit tested.

```
<section class="sample-app-content">
    <h1>Model-based Form Example:</h1>
    <form [formGroup]="form" (ngSubmit)="onSubmit()">
        <p>
            <label>First Name:</label>
            <input type="text" formControlName="firstName">
        </p>
        <p>
            <label>Password:</label>
            <input type="password" formControlName="password">
        </p>
        <p>
            <button type="submit" [disabled]="!form.valid">Submit</button>
        </p>
    </form>
</section>

import { FormGroup, FormControl, Validators, FormBuilder }
    from '@angular/forms';
@Component({
    selector: "model-driven-form",
    templateUrl: 'model-driven-form.html'
})
export class ModelDrivenForm {
    form: FormGroup;
    firstName = new FormControl("", Validators.required);
    constructor(fb: FormBuilder) {
        this.form = fb.group({
            "firstName": this.firstName,
            "password":["", Validators.required]
```

```
        });
    }
    onSubmitModelBased() {
        console.log("model-based form submitted");
        console.log(this.form);
    }
}
```

Template Driven Vs Reactive Form
Template-driven forms are asynchronous in nature, whereas Reactive forms are mostly synchronous.

Reactive forms are also known as model-driven forms. This means that the HTML content changes depending on the code in the component.

Template-driven forms are driven by derivatives in the template. This means that you will see derivatives such as ngModel in the template as opposed to the code.

Template-driven forms use the FormsModule, while reactive forms use the ReactiveFormsModule.

Template-driven forms are asynchronous, while reactive forms are synchronous.
In template-driven forms, most of the interaction occurs in the template, while in reactive-driven forms, most of the interaction occurs in the component.

Advantages and Disadvantages of Template-Driven Forms
Although template forms are easier to create, they become a challenge when you want to do unit testing, because testing requires the presence of a DOM.

Advantages and Disadvantages of Reactive Forms
It's easier to write unit tests in reactive forms since all the form code and functionality is contained in the component. However, reactive forms require more coding implementation in the component.

Explain the working of timers in JavaScript?
Timers are used to execute a piece of code at a set time or also to repeat the code in a given interval of time. This is done by using the functions **setTimeout**, **setInterval** and **clearInterval**.
The **setTimeout(function, delay)** function is used to start a timer that calls a particular function after the mentioned delay. The **setInterval(function, delay)** function is used to repeatedly execute the given function in the mentioned delay and only halts when cancelled. The **clearInterval(id)** function instructs the timer to stop.
Timers are operated within a single thread, and thus events might queue up, waiting to be executed.

slice() is a method of Array Object and String Object. It is non-destructive since it return a new copy and it doesn't change the content of input array.
split() is a method of String Object.
Splice() is used to add or remove an element in a array, and it would modify the origin array.

forkJoin
'forkJoin' is the easiest way, when you need to wait for multiple HTTP requests to be resolved
'forkJoin' waits for each HTTP request to complete and group's all the observables returned by each HTTP call into a single observable array and finally return that observable array.

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import { forkJoin } from 'rxjs';  // RxJS 6 syntax
@Injectable()
export class DataService {
  constructor(private http: HttpClient) { }
  public requestDataFromMultipleSources(): Observable<any[]> {
    let response1 = this.http.get(requestUrl1);
    let response2 = this.http.get(requestUrl2);
    let response3 = this.http.get(requestUrl3);
    // Observable.forkJoin (RxJS 5) changes to just forkJoin() in RxJS 6
    return forkJoin([response1, response2, response3]);
  }
}

import { Component, OnInit } from '@angular/core';
import { DataService } from "../data.service";
@Component({
    selector: 'app-page',
    templateUrl: './page.component.html',
    styleUrls: ['./page.component.css']
})
export class DemoComponent implements OnInit {
    public responseData1: any;
    public responseData2: any;
    public responseData3: any;
    constructor(private dataService: DataService) {}
    ngOnInit() {
        this.dataService.requestDataFromMultipleSources().subscribe(responseList => {
            this.responseData1 = responseList[0];
            this.responseData2 = responseList[1];
            this.responseData3 = responseList[1];
        });
    }
```

How to prevent security threats 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 redirection 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 unnecessary 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.

```
import { Injectable, NgZone } from '@angular/core';

@Injectable()
export class OurZoneWatchingService() {
  constructor(private ngZone: NgZone) {
    this.ngZone.onStable.subscribe(this.onZoneStable);
  }

  onZoneStable() {
    console.log('We are stable');
}

}
```

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 Directive in Angular 4? How does it differ 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 decorator (@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 a reusable set of DOM elements with UI behaviors. And we should use a Directive whenever we want reusable behavior to supplement the DOM.

PWA:
Progressive Web Apps are a new way to offer incredible mobile app experiences that are highly optimized, reliable, and accessible completely on the web. A Progressive Web App (PWA) is a web app that uses modern web capabilities to deliver an app-like experience to users. These apps meet certain requirements, are deployed to servers, accessible through URLs, and indexed by search engines.
To be considered a Progressive Web App, your app must be:
Progressive - Work for every user, regardless of browser choice, because they are built with progressive enhancement as a core tenet.
Responsive - Fit any form factor, desktop, mobile, tablet, or whatever is next.
Connectivity independent - Enhanced with service workers to work offline or on low quality networks.
App-like - Use the app-shell model to provide app-style navigation and interactions.
Fresh - Always up-to-date thanks to the service worker update process.
Safe - Served via HTTPS to prevent snooping and ensure content has not been tampered with.
Discoverable - Are identifiable as “applications” thanks to W3C manifests and service worker registration scope allowing search engines to find them.
Re-engageable - Make re-engagement easy through features like push notifications.
Installable - Allow users to “keep” apps they find most useful on their home screen without the hassle of an app store.
Linkable - Easily share via URL and not require complex installation.
This can work in conjunction with Cordova to provide a multiple deploy targets for all your users. You can deploy your app as a PWA as well as Native app and take advantage of both channels.
The two main requirements of a PWA are a Service Worker and a Web Manifest. While it's possible to add both of these to an app manually, the Angular team has an @angular/pwa package that can be used to automate this. The @angular/pwa package will automatically add a service worker and a app manifest to the app. To add this package to the app run:
ng add @angular/pwa

What is Redux? and @ngRx?
It is a library which helps us maintain the state of the application. Redux is not required in applications that are simple with the simple data flow, it is used in Single Page Applications that have complex data flow.
Redux 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 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.
Why Typescript with Angular?
Typescript is a superset of Javascript. Earlier, Javascript was the only client-side language supported by all browsers. But, the problem with Javascript is, it is not a pure Object Oriented Programming Language. The code written in JS without following patterns like Prototype Pattern becomes messy and finally leading to difficulties in maintainability and reusability. Instead of learning concepts (like patterns) to maintain code, programmers prefer to maintain the code in an OOP approach and is made available with a programming language like Typescript was thus developed by Microsoft in a way that it can work as Javascript and also offer what javascript cannot ie;

pure OOPS as Typescript offers concepts like Generics, Interfaces and Types (a Static Typed Language) which makes it easier to catch incorrect data types passing to variables.
TS provides flexibility to programmers experienced in java, .net as it offers encapsulation through classes and interfaces.

JS version ES5 offers features like Constructor Function, Dynamic Types, Prototypes. The next version of Javascript ie ES6 introduced a new feature like Class keyword but not supported by many browsers.
TS offers Arrow Functions (=>) which is an ES6 feature not supported by many browsers directly but when used in TS, gets compiled into JS ES5 and runs in any browser.
TS is not the only alternative to JS, we have CoffeeScript, Dart(Google).
Finally, it is like, TS makes life easier when compared to JS.
https://www.sparkbit.pl/typescript-decorators/

HTML, CSS & CSS3, JS, Angular

https://www.kirupa.com/apps/webview.htm

https://www.javatpoint.com/html-interview-questions

https://career.guru99.com/top-50-csscascading-style-sheet-interview-questions/
http://a4academics.com/interview-questions/79-web/817-css-css3
https://www.fullstack.cafe/blog/13-tricky-css-interview-questions-and-answers-in-2018
https://www.softwaretestinghelp.com/css-interview-questions
https://www.javatpoint.com/css-media-query

https://hackr.io/blog/bootstrap-interview-questions

https://www.guru99.com/javascript-interview-questions-answers.html

https://www.techgeeknext.com/angular/angular-interview-questions

https://www.techgeeknext.com/angular/angular-interview-questions

How does CSS work (under the hood of browsers)?
When a browser displays a document, it must combine the document's content with its style information. It processes the document in two stages:
The browser converts HTML and CSS into the DOM (Document Object Model). The DOM represents the document in the computer's memory. It combines the document's content with its style.
The browser displays the contents of the DOM.

Grid and flexbox. The basic difference between CSS Grid Layout and CSS Flexbox Layout is that flexbox was designed for layout in one dimension - either a row or a column. Grid was designed for two-dimensional layout - rows, and columns at the same time.
Grid is Container-Based, Flexbox is Content-Based
Flexbox is One Dimensional, Grid is Two Dimensional,
Grid Has a “Gap” Property Flexbox Doesn’t.

slice() is a method of Array Object and String Object. It is non-destructive since it return a new copy and it doesn't change the content of input array.
split() is a method of String Object.
Splice() is used to add or remove an element in a array, and it would modify the origin array.