**Angular Component Lifecycle Sequence**

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

**Hook :Purpose and Timing**

1. **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.
2. **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().*
3. **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().*
4. **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().*
5. **ngAfterContentChecked() :**Respond after Angular checks the content projected into the directive/component. Called after the *ngAfterContentInit()* and every subsequent *ngDoCheck().*
6. **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()*.
7. **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().*
8. **ngOnDestroy() :**Clean-up 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.
9. **ngOnChanges** − When the value of a data bound property changes, then this method is called.
10. **ngOnInit** − This is called whenever the initialization of the directive/component after Angular first displays the data-bound properties happens.
11. **ngDoCheck** − This is for the detection and to act on changes that Angular can't or won't detect on its own.
12. **ngAfterContentInit** − This is called in response after Angular projects external content into the component's view.
13. **ngAfterContentChecked** − This is called in response after Angular checks the content projected into the component.
14. **ngAfterViewInit** − This is called in response after Angular initializes the component's views and child views.
15. **ngAfterViewChecked** − This is called in response after Angular checks the component's views and child views.
16. **ngOnDestroy** − This is the clean-up phase just before Angular destroys the directive/component.

**Reactive Form or Model Driven Form:** Most of the code is in the component class.

<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>

**Template Drive Form Example:** Most of the code is written in component 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>

**Three ways to specify selector for component:**

**1-** selector : **’app-test’**

Use : **<app-test></app-test>**

**2-** selector : **’.app-test’**

Use: **<div class=’app-test’></div>**

**3-** selector : **’[app-test]’**

Use: **<div app-test></div>**

**Use /\*\* text \*/ to specify multiple line template**

**What is Angular and Why Angular?**

Angular (2/4/5) is not a programming language like Javascript or Typescript. Angular is a front-end or client-side Framework, which needs a [programming](https://www.greycampus.com/programming) language like Typescript (developed by Microsoft). Now, what does it mean when one says, "Angular is a client-side framework"? It means that it runs on the client-side or user's browser and not on a Web Server (where java/python/ruby/vb.net run). Of course, there is the Angular Universal which allows Angular to run on a server; but, primarily, angular is a client-side framework. Angular is a product developed and maintained by the techie giant Google and has adopted the SPA (Single Page Application) principles.

### 1. What’s new in Angular 5?

Certain tools are optimized in the new version of [Angular](https://www.greycampus.com/angularjs-training-instructor-led), 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

### 2. Name the building blocks of Angular.

The Angular application is made using the following:

Modules, Component, Template, Directives, Data Binding, Services, Dependency Injection, Routing

### 3. 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.

### 4. 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.

### 5. Differentiate between Components and Directives in Angular 5.

Components break up the application into smaller parts; whereas, Directives add behaviour to an existing DOM element.

### 6. 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 Parent Component to receive an event from Child to Parent Component.

### 7. What is ng-content Directive?

The HTML elements like p (paragraph) or h1 (heading) have some content between the tags. For example, <p>this is a paragraph</p> 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.

### 8. What does a router.navigate do?

When we want to route to a component we use router.navigate.  Syntax: this.router.navigate([‘/component\_name’]);

**Steps to define Route**

1. Import **RouterModule** and **Routes** into your routing module. **(AppRoutingModule)**

**import { Routes, RouterModule } from '@angular/router';** // CLI imports router

1. Define your routes in your Routes array.

**const routes: Routes = [**

**{ path: 'first-component', component: FirstComponent },**

**{ path: 'second-component', component: SecondComponent },**

**];**

1. Add your routes to your application.

<a [routerLink](https://angular.io/api/router/RouterLink)="/first-component" [routerLinkActive](https://angular.io/api/router/RouterLinkActive)="active">First [Component](https://angular.io/api/core/Component)</a> **(In HTML Page)**

**OR**

this.router.navigate(['color']); **(In Component)**

### 9. What is ViewEncapsulation?

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:

Emulated: styles from other HTML spread to the component.

Native: styles from other HTML do not spread to the component.

None: styles defined in a component are visible to all components.

### 10. What are Services in Angular and what command is used to create a service?

Services help us in not repeating the code. With the creation of services, we can use the same code from different components. Here is the command to create a service in angular, ng g service User (a UserService is created when this command is used).

### 11. What is Dependency Injection in Angular 4?

When a component is dependent on another component the dependency is injected/provided during runtime.

### 12. What is Routing in Angular 5?

Routing helps a user in navigating to different pages using links.

### 13. 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).

### 14. 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.

### 15. Explain the usage of {{}}?

The set of brackets {{}} when used with an HTML tag, represent data from a component. For example, on a 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.

### 16. In how many ways the Data Binding can be done?

Data binding happens between the HTML (template) and typescript (component). Data binding can be done in 3 ways:

1. **Property binding :** with the help of square brackets []
2. **Event Binding** : with the help of small brackets ()
3. **Two-Way Data Binding**. With the help of **[( )]** (model and view will be in sync)

### 17. What is the sequence of Angular Lifecycle Hooks?

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

### 18. 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.

### 19. How is SPA (Single Page Application) technology different from the 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 the 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.

### 20. What is Component in Angular Terminology?

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

### 21. 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 [()].

### 22. 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.

### 23. 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.

### 24. What is an AsyncPipe (| async) 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 (| async), the promise or observable can be directly used in a template and a temporary property is not required.

### 25. 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.

### 26. What is 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.

### 27. What is Redux?

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.

### 28. What are Pipes?

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 Date format accordingly.

### 29. 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.

### 30. 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 an Object Oriented Programming Language. Typescript was thus developed by Microsoft in a way that it can work as Javascript and also offer what javascript cannot; i.e., OOPS.

//Convert Object to Array

const mapped = Object.keys(this.optionsLists).map(key=> ({ colorid:key, sizeid:this.optionsLists[key] }));

for (varxinmapped) {

//Convert Inner Object to Array

C

3onst dd = Object.keys(mapped[x]['sizeid']).map(key=> ({ size:key, value:mapped[x]['sizeid'][key] }));

mapped[x]['sizeid'] = dd;

}

this.optionsListsArray = mapped;

**31. @Directive v/s @Component in Angular**

**Directives** add behaviour to an existing DOM element or an existing component instance.

import { Directive, ElementRef, AfterViewInit } from '@angular/core';

@Directive({

selector: '[CpColor]'

})

export class CpColorDirective implements AfterViewInit {

constructor(private elRef: ElementRef) {

}

ngAfterViewInit() {

this.elRef.nativeElement.style.color = 'red';

}

change(changedColor: String) {

this.elRef.nativeElement.style.color = changedColor;

}

}

**Component**, rather than adding/modifying behaviour, actually creates its own view (hierarchy of DOM elements) with attached behaviour.

import { Component, OnInit, ViewChild } from '@angular/core';

import { CpColorDirective } from '../cp-color.directive';

@Component({

selector: 'app-cpcolor-parent',

templateUrl: './cpcolor-parent.component.html',

styleUrls: ['./cpcolor-parent.component.css']

})

export class CpcolorParentComponent implements OnInit {

constructor() { }

ngOnInit() {

}

@ViewChild(CpColorDirective)

private cpColorDirective: CpColorDirective;

changeColor(color: string) {

this.cpColorDirective.change(color);

}

}

<h3>@ViewChild using Directive Color</h3>

Color Example:

<p CpColor>Change my Color </p>

<div>

Change Color:

<input type="radio" name="rad" (click)="changeColor('green')"> Green

<input type="radio" name="rad" (click)="changeColor('cyan')"> Cyan

<input type="radio" name="rad" (click)="changeColor('blue')"> Blue

</div>

**32. What is the use of systemjs? How is webpack better to use in Angular 2/4?**

1- Systemjs is a client side module bundler in angular as it loads modules (components and other files) on demand instead of loading an entire application at startup.

2- This largely reduces load times while starting up the app.

3- The up side of Webpack over Systemjs is that it bundles and creates a single file called bundle.js, which contains HTML, CSS and JS etc.

4- While the initial load time might take a few seconds once the app is cached it becomes lightning fast and will lead to a large boost in performance.

**33. How to enable lazy loading in Angular 2/4?**

Steps to enable lazy loading in Angular 2/4 are as follows:

Add loadChildren in place of components in the routes in the exported class in the Main Routing module of your application.

Now change forRoot to forChild in the module or feature level router config module and you are done.

**34. Is it good to use JQuery in Angular 2/4?**

Angular 2/4 and JQuery at its core are very different concept all together.

JQuery deals with manipulating DOM directly while Angular is used primarily for binding data.

In some scenarios, using JQuery libraries can be used as quick solutions but doing this we might face problems in the future in terms of pre-rendering.

**35. What Are Event Emitters in Angular 2/4?**

If the change in one of the child components needs to be reflected to any of its parent component, we can emit the event by using Event Emitter API in Angular.

EventEmitter is class present in @angular/core module, which is used by components and directives to emit custom events.

**36. What are Directives in Angular 2/4?**

Directives in Angular are used to add behaviour to our existing DOM elements.

We use @Directive meta-data annotation to register directives.

Directives in angular are used to design reusable components.

Directives do not have their own views.

**37. What are pipes in Angular 2/4?**

Pipes are used in Angular to edit/transform and format our data in the template itself.

**38. What is component decorators in Angular?**

The main objectives of decorators is to add some metadata to the class that will tell Angular how to process a class. Angular has many decorators that attach metadata to classes so that it knows what those classes mean and how they should work.

If we consider Component in Angular, we will have following options to configure.

**selector**: — define the name of the HTML element in which our component will live.

**template or templateUrl**: — It can be inline string or link an external html file. It allows us to tie logic from our component directly to a view.

**Style or styleUrls**: — the styles array for our specific component. We can also link external CSS by styleUrls.

**providers**: — This is the place we are passing the services that we need insider our components.

**To create package.json from command**

* npm init --yes

**To install dependencies**

* npm install --save express body-parser cors

**express** is a web server

**body-parser** is a middleware to handle form data

**cors** is a package to make request across different ports

**39. Types of compiler in angular and difference between them**

There are two types of compiler JIT and AOT

**40. JIT and AOT Comparison**

The main differences between JIT and AOT in Angular are:

Just-in-Time (JIT), which compiles your app in the browser at runtime.

Ahead-of-Time (AOT), which compiles your app and libraries at build time.

JIT compilation is the default when you run the ng build (build only) or ng serve (build and serve locally) CLI commands. This is for development.

AOT compilation, include the --aot option with the ng build or ng serve command. Another ways is using --prod which by default production mode is configured in Angular.json with AOT is set to true.

**ng build -- aot**

**nb build -- prod**

**Summary**

* JIT and AOT are two ways to compile code in an Angular project.
* We use **JIT in development mode** while **AOT is for production mode**.
* We can easily implement features and debug in JIT mode since we have map file while AOT does not. However, the big **benefit when we use AOT for production is reducing bundle size for faster rendering.**

**41. Ways to communicate between components.**

1- Binding (@Input & @Output)

Data binding is the most basic way to communicate between components.

2- Reference (@ViewChild & @ContentChild)

Directly grab the instance reference of the specific component and manipulate it programmatically

3- Provider (Service)

Instead of establishing a direct linkage between components, a standalone injectable (service) is used as a middleman between them.

**42. What is CommonModule?**

**Exports all the basic Angular directives and pipes, such as NgIf, NgForOf, DecimalPipe, and so on**. Re-exported by BrowserModule, which is included automatically in the root AppModule when you create a new app with the CLI new command.

**43. What is Subjects in Angular?**

1. Subjects are observables as well as observers.
2. First super power of a subject: Data emission.

**let** subject = **new** Subject<**string**>();*// We subscribe to the subject*subject.subscribe((data) => {  
 ***console***.log(**"Subscriber got data >>>>> "**+ data);  
});  
  
*// We use the subject to emit data*subject.next(**"Eureka"**);  
  
*// Console result:* ***Subscriber got data >>>>> Eureka***

1. The second super power of subjects is that they support multiple subscriptions.

**let** subject = **new** Subject<**string**>();  
subject.subscribe((data) => {  
 ***console***.log(**"Subscriber 1 got data >>>>> "**+ data);  
});subject.subscribe((data) => {  
 ***console***.log(**"Subscriber 2 got data >>>>> "**+ data);  
});  
subject.next(**"Eureka"**);  
  
*// Console result:  
// Subscriber 1 got data >>>>> Eureka  
// Subscriber 2 got data >>>>> Eureka*

**44. What is Behavior Subjects in Angular?**

Behavior Subjects are another cool thing about subjects. When you subscribe to a behavior subject, it **will give you the last emitted value right away.** Behavior subjects need a first value.

**let** subject = **new** BehaviorSubject<**string**>(**"First value"**);  
  
subject.asObservable().subscribe((data) => {  
**console**.log(**"First subscriber got data >>>>> "**+ data);  
});  
subject.next(**"Second value"**)  
// Console result:// First subscriber got data >>>>> First value// First subscriber got data >>>>> Second value

**45. What is Replay Subjects in Angular?**

Replay Subjects **keep a given number of historical values** so that those values can be replayed to new subscribers.

// We tell the ReplaySubject how many values should be kept in history

**let** subject = **new** ReplaySubject<**string**>(2);  
  
subject.next(**"First value"**);  
subject.next(**"Second value"**);  
subject.next(**"Third value"**);  
  
subject.asObservable().subscribe((data) => {  
 ***console***.log(**"First subscriber got data >>>>> "**+ data);  
});  
  
subject.next(**"Fourth value"**);  
  
*//Console result:  
// First subscriber got data >>>>> Second value  
// First subscriber got data >>>>> Third value  
// First subscriber got data >>>>> Fourth value*

Angular has built-in pipes for dates, currency, percentage and character cases.

In Addition to this we can create custom pipes ourselves and use them in angular.

Therefore, we can say that pipes take in data as input and transforms it to output we desire. 1. The **splice()**method returns the removed item(s) in an array and **slice()**method returns the selected element(s) in an array, as a new array object.

2. The **splice()** method changes the original array and **slice()** method doesn’t change the original array.

3. The **splice()** method can take n number of arguments:

**Argument 1:**Index, Required. An integer that specifies at what position to add /remove items, Use negative values to specify the position from the end of the array.

**Argument 2:**Optional. The number of items to be removed. If set to 0(zero), no items will be removed. And if not passed, all item(s) from provided index will be removed.

**Argument 3…n:**Optional. The new item(s) to be added to the array.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | var array=[1,2,3,4,5];  console.log(array.splice(2));  // shows [3, 4, 5], returned removed item(s) as a new array object.    console.log(array);  // shows [1, 2], original array altered.    var array2=[6,7,8,9,0];  console.log(array2.splice(2,1));  // shows [8]    console.log(array2.splice(2,0));  //shows [] , as no item(s) removed.    console.log(array2);  // shows [6,7,9,0]    var array3=[11,12,13,14,15];  console.log(array3.splice(2,1,"Hello","World"));  // shows [13]    console.log(array3);  // shows [11, 12, "Hello", "World", 14, 15]               -5 -4 -3 -2 -1              |  |  |  |  |  var array4=[16,17,18,19,20];               |  |  |  |  |               0  1  2  3  4    console.log(array4.splice(-2,1,"me"));  // shows  [19]    console.log(array4);  // shows [16, 17, 18, "me", 20] |

**If Argument(1) is NaN, it is treated as if it were 0.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var array5=[21,22,23,24,25];  console.log(array5.splice(NaN,4,"NaN is Treated as 0"));  // shows [21,22,23,24]    console.log(array5);  // shows ["NaN is Treated as 0",25] |

**If Argument(2) is less than 0 or equal to NaN, it is treated as if it were 0.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var array6=[26,27,28,29,30];  console.log(array6.splice(2,-5,"Hello"));  // shows []    console.log(array6);  // shows [26,27,"Hello",28,29,30]    console.log(array6.splice(3,NaN,"World"));  // shows []    console.log(array6);  // shows [26,27,"Hello","World",28,29,30] |

**If Argument(1) or Argument(2) is greater than Array’s length, either argument will use the Array’s length.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var array7=[31,32,33,34,35];  console.log(array7.splice(23,3,"Add Me"));  // shows []    console.log(array7);  // shows [31,32,33,34,35,"Add Me"]    console.log(array7.splice(2,34,"Add Me Too"));  // shows [33,34,35,"Add Me"]    console.log(array7);  // shows [31,32,"Add Me Too"] |

4. The **slice()** method can take 2 arguments:

**Argument 1:**Required. An integer that specifies where to start the selection (The first element has an index of 0). Use negative numbers to select from the end of an array.

**Argument 2:**Optional. An integer that specifies where to end the selection. If omitted, all elements from the start position and to the end of the array will be selected. Use negative numbers to select from the end of an array.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | var array=[1,2,3,4,5]  console.log(array.slice(2));  // shows [3, 4, 5], returned selected element(s).    console.log(array.slice(-2));  // shows [4, 5], returned selected element(s).  console.log(array);  // shows [1, 2, 3, 4, 5], original array remains intact.    var array2=[6,7,8,9,0];  console.log(array2.slice(2,4));  // shows [8, 9]    console.log(array2.slice(-2,4));  // shows [9]    console.log(array2.slice(-3,-1));  // shows [8, 9]    console.log(array2);  // shows [6, 7, 8, 9, 0] |

**If either argument is NaN, it is treated as if it were 0.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | var array3=[11,12,13,14,15];  console.log(array3.slice(NaN,NaN));  // shows []    console.log(array3.slice(NaN,4));  // shows [11,12,13,14]    console.log(array3);  // shows [11,12,13,14,15] |

**If either argument is greater than the Array’s length, either argument will use the Array’s length**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var array4=[16,17,18,19,20];  console.log(array4.slice(23,24));  // shows []    console.log(array4.slice(23,2));  // shows []    console.log(array4.slice(2,23));  // shows [18,19,20]    console.log(array4);  // shows [16,17,18,19,20] |