

## Unit-5

### **What is Angular:**

AngularJS was developed in 2008-2009 by Misko Hevery and Adam Abrons and is now maintained by Google. AngularJS is a Javascript open-source front-end framework that is mainly used to develop single-page web applications(SPAs). It is a continuously growing and expanding framework which provides better ways for developing web applications.

It changes the static HTML to dynamic HTML. Its features like dynamic binding and dependency injection eliminate the need for code that we have to write otherwise. AngularJS is rapidly growing and because of this reason, we have different versions of AngularJs with the latest stable being 1.7.7. It is also important to note that Angular is different from AngularJs. It is an open-source project which can be freely used and changed by anyone. It extends HTML attributes with Directives, and data is bound with HTML.

### **Key Points:**

- AngularJS is a JavaScript framework that is mainly used for Frontend Development.
- It is used for making Single Page Applications(SPA).
- It is open source and is completely free for everyone.
- It uses the Model, View, Control(MVC) pattern for developing projects.

### **Why use it?**

- **Easy to work with:** All you need to know to work with AngularJs is basics of HTML,CSS and Javascript,not necessary to be an expert in these technologies.
- **Time-saving:** AngularJs allows us to work with components and hence we can use them again which saves time and unnecessary code.
- **Ready to use template:** AngularJs is mainly plain HTML, and it mainly makes use of the plain HTML template and passes it to the DOM and then the AngularJS compiler. It traverses the templates and then they are ready to use.

### **Example:**

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>AngularJS</title>

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">

</script>

</head>

<body style="text-align:center">

<h2 style="color:green">Aditya Engineering College</h2>

<div ng-app="" ng-init="name='Aditya'">

  <p>{{ name }} is the portal for adity.</p>

</div>

</body>

</html>
```

**Output:**

**Aditya Engineering College**

Aditya is the portal for adity.

**Angular Application Setup****Angular CLI | Angular Project Setup**

**ngular** is a front-end framework which is used to create web applications. It uses typescript by default for creating logics and methods for a class but the browser doesn't know typescript. Here webpack comes in picture, webpack is used to compile these typescript files to JavaScript. In addition, there are so many

configuration files you will need to run an angular project on your computer.

**Angular CLI** is a tool that does all these things for you in some simple commands. Angular CLI uses webpack behind to do all this process.

**Note:** Please make sure you have installed node and npm in your system. You can check your node version and npm version by using the following command:

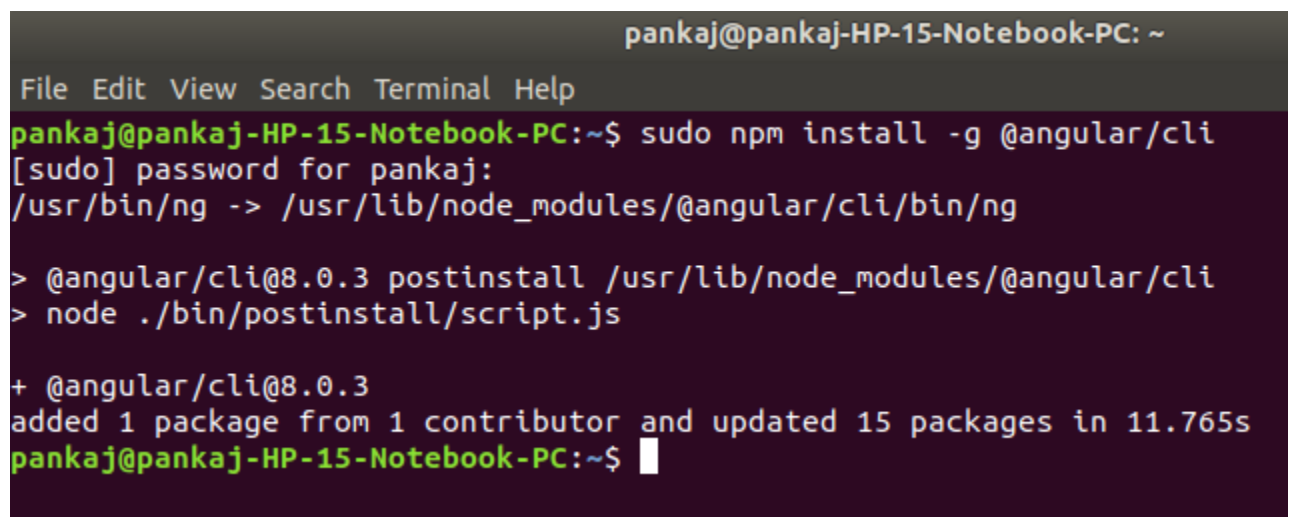
```
node --version
```

```
npm --version
```

### Steps to Create your first application using angular CLI:

- **Step-1: Install angular cli**

```
npm install -g @angular/cli
```

A terminal window titled 'pankaj@pankaj-HP-15-Notebook-PC: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'sudo npm install -g @angular/cli' being executed. It prompts for a password, then shows the installation path. It then shows the postinstall script running. Finally, it shows the installation of @angular/cli@8.0.3, adding 1 package and updating 15 packages in 11.765s.

```
pankaj@pankaj-HP-15-Notebook-PC: ~  
File Edit View Search Terminal Help  
pankaj@pankaj-HP-15-Notebook-PC:~$ sudo npm install -g @angular/cli  
[sudo] password for pankaj:  
/usr/bin/ng -> /usr/lib/node_modules/@angular/cli/bin/ng  
  
> @angular/cli@8.0.3 postinstall /usr/lib/node_modules/@angular/cli  
> node ./bin/postinstall/script.js  
  
+ @angular/cli@8.0.3  
added 1 package from 1 contributor and updated 15 packages in 11.765s  
pankaj@pankaj-HP-15-Notebook-PC:~$
```

- **Step-2: Create new project by this command**

Choose yes for routing option and, CSS or SCSS.

```
ng new myNewApp
```

```

pankaj@pankaj-HP-15-Notebook-PC: ~
File Edit View Search Terminal Help
pankaj@pankaj-HP-15-Notebook-PC:~$ ng new myNewApp
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? SCSS [ http://sass-lang.com/documentation/file
_SASS_REFERENCE.html#syntax ]
CREATE myNewApp/README.md (1025 bytes)
CREATE myNewApp/.editorconfig (246 bytes)
CREATE myNewApp/.gitignore (629 bytes)
CREATE myNewApp/angular.json (3529 bytes)
CREATE myNewApp/package.json (1283 bytes)
CREATE myNewApp/tsconfig.json (470 bytes)
CREATE myNewApp/tslint.json (1985 bytes)
CREATE myNewApp/browserslist (429 bytes)
CREATE myNewApp/karma.conf.js (1020 bytes)
CREATE myNewApp/tsconfig.app.json (210 bytes)
CREATE myNewApp/tsconfig.spec.json (270 bytes)
CREATE myNewApp/src/favicon.ico (5430 bytes)
CREATE myNewApp/src/index.html (295 bytes)
CREATE myNewApp/src/main.ts (372 bytes)
CREATE myNewApp/src/polyfills.ts (2838 bytes)
CREATE myNewApp/src/styles.scss (80 bytes)
CREATE myNewApp/src/test.ts (642 bytes)
CREATE myNewApp/src/assets/.gitkeep (0 bytes)
CREATE myNewApp/src/environments/environment.prod.ts (51 bytes)
CREATE myNewApp/src/environments/environment.ts (662 bytes)
CREATE myNewApp/src/app/app-routing.module.ts (245 bytes)
CREATE myNewApp/src/app/app.module.ts (393 bytes)
CREATE myNewApp/src/app/app.component.scss (0 bytes)
CREATE myNewApp/src/app/app.component.html (1152 bytes)
CREATE myNewApp/src/app/app.component.spec.ts (1101 bytes)
CREATE myNewApp/src/app/app.component.ts (213 bytes)
CREATE myNewApp/e2e/protractor.conf.js (810 bytes)
CREATE myNewApp/e2e/tsconfig.json (214 bytes)
CREATE myNewApp/e2e/src/app.e2e-spec.ts (637 bytes)
CREATE myNewApp/e2e/src/app.po.ts (251 bytes)

CREATE myNewApp/src/styles.scss (80 bytes)
CREATE myNewApp/src/test.ts (642 bytes)
CREATE myNewApp/src/assets/.gitkeep (0 bytes)
CREATE myNewApp/src/environments/environment.prod.ts (51 bytes)
CREATE myNewApp/src/environments/environment.ts (662 bytes)
CREATE myNewApp/src/app/app-routing.module.ts (245 bytes)
CREATE myNewApp/src/app/app.module.ts (393 bytes)
CREATE myNewApp/src/app/app.component.scss (0 bytes)
CREATE myNewApp/src/app/app.component.html (1152 bytes)
CREATE myNewApp/src/app/app.component.spec.ts (1101 bytes)
CREATE myNewApp/src/app/app.component.ts (213 bytes)
CREATE myNewApp/e2e/protractor.conf.js (810 bytes)
CREATE myNewApp/e2e/tsconfig.json (214 bytes)
CREATE myNewApp/e2e/src/app.e2e-spec.ts (637 bytes)
CREATE myNewApp/e2e/src/app.po.ts (251 bytes)

> core-js@2.6.9 postinstall /home/pankaj/myNewApp/node_modules/babel-runtime/node_modules/core-js
> node scripts/postinstall || echo "ignore"

> core-js@2.6.9 postinstall /home/pankaj/myNewApp/node_modules/karma/node_modules/core-js
> node scripts/postinstall || echo "ignore"

> @angular/cli@8.0.3 postinstall /home/pankaj/myNewApp/node_modules/@angular/cli
> node ./bin/postinstall/script.js

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","ar
ch":"x64"})

added 1015 packages from 1041 contributors and audited 19005 packages in 95.797s
found 0 vulnerabilities

Successfully initialized git.
pankaj@pankaj-HP-15-Notebook-PC:~$

```

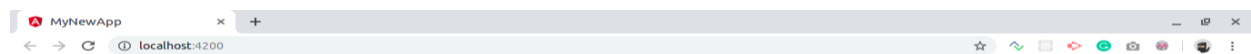
- **Step-3: Go to your project directory**  
cd myNewApp

```
pankaj@pankaj-HP-15-Notebook-PC: ~/myNewApp
File Edit View Search Terminal Help
pankaj@pankaj-HP-15-Notebook-PC:~$ cd myNewApp/
pankaj@pankaj-HP-15-Notebook-PC:~/myNewApp$
```

- **Step-4: Run server and see your application in action**

ng serve -o --poll=2000

```
pankaj@pankaj-HP-15-Notebook-PC: ~/myNewApp
File Edit View Search Terminal Help
pankaj@pankaj-HP-15-Notebook-PC:~/myNewApp$ ng serve -o --poll=2000
93% after chunk asset optimization SourceMapDevToolPlugin polyfills.js generate SourceMa
Date: 2019-06-24T16:58:12.903Z
Hash: 55a034f4077a23b171a3
Time: 24279ms
chunk {main} main.js, main.js.map (main) 11.4 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 248 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.6 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.94 MB [initial] [rendered]
** Angular Live Development Server is listening on localhost:4200, open your browser on h
ttp://localhost:4200/ **
i [wdm]: Compiled successfully.
```



Welcome to myNewApp!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

**Component in angular js:**

In AngularJS, a Component is a special kind of [directive](#) that uses a simpler configuration which is suitable for a component-based application structure.

This makes it easier to write an app in a way that's similar to using Web Components or using the new Angular's style of application architecture.

Advantages of Components:

- simpler configuration than plain directives
  - promote sane defaults and best practices
  - optimized for component-based architecture
  - writing component directives will make it easier to upgrade to Angular
- 
- **app.component.html** Edit this file to make changes to the page. You can edit this file as an HTML file. Work directly with div or any other tag used inside body tags, these are components and do not add **html head body** tags.

```
<h1>
```

```
  Hello world
```

```
</h1>
```

```
<div>
```

```
  <p>
```

```
    This is my First Angular app.
```

```
  </p>
```

```
</div>
```

- **app.component.spec.ts** These are automatically generated files which contain unit tests for source component.
- **app.component.ts** You can do the processing of the HTML structure in the .ts file. The processing will include activities such as connecting to the database, interacting with other components, routing, services, etc.
- **app.component.scss** Here you can add CSS for your component. You can write scss which further compiled to CSS by a transpiler.

## AngularJS Module

In AngularJS, a module defines an application. It is a container for the different parts of your application like controller, services, filters, directives etc.

A module is used as a Main() method. Controller always belongs to a module.

---

### How to create a module

The angular object's module() method is used to create a module. It is also called AngularJS function angular.module

```
<div ng-app="myApp">...</div>
```

```
<script>
```

```
var app = angular.module("myApp", []);
```

```
</script>
```

Here, "myApp" specifies an HTML element in which the application will run.

Now we can add controllers, directives, filters, and more, to AngularJS application.

---

### How to add controller to a module

If you want to add a controller to your application refer to the controller with the ng-controller directive.

See this example:

```
<!DOCTYPE html>
```

1. `<html>`
2. `<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">`  
`</script>`
3. `<body>`
4. `<div ng-app="myApp" ng-controller="myCtrl">`
5. `{{ firstName + " " + lastName }}`
6. `</div>`
7. `<script>`
8. `var app = angular.module("myApp", []);`
9. `app.controller("myCtrl", function($scope) {`
10. `$scope.firstName = "Ajeet";`
11. `$scope.lastName = "Maurya";`
12. `});`
13. `</script>`
14. `</body>`
15. `</html>`

## Structural Directives in Angular

Structural directives are responsible for the Structure and Layout of the DOM Element. It is used to hide or display the things on the DOM. Structural Directives can be easily identified using the ‘\*’. Every Structural Directive is preceded by a ‘\*’ symbol.

Some of the Build in Structural Directives with Examples are as follows:

### 1. \*ngIf: Next Generation

ngIf is used to display or hide the DOM Element based on the expression value assigned to it. The expression value may be either true or false.

**Syntax:**



```
<div *ngIf="boolean"> </div>
```

In the above Syntax, boolean stands for either true or false value. Hence, it leads to 2 valid syntaxes as below :

```
<div *ngIf="true"> </div>
```

```
<div *ngIf="false"> </div>
```

### **Example of \*ngIf:**

```
<div *ngIf="false">
```

This text will be hidden

```
<h1 [ngStyle]="{'color': '#FF0000'}">
```

GFG Structural Directive Example

```
</h1>
```

```
</div>
```

```
<div *ngIf="true">
```

This text will be displayed

```
<h1 [ngStyle]="{'color': '#00FF00'}">
```

GFG Structural Directive Example

```
</h1>
```

```
</div>
```

### **Output:**

This text will be displayed

**GFG Structural Directive Example**

## *\*ngIf Example*

### **2. \*ngIf-else:**

ngIf-else works like a simple If-else statement, wherein if the condition is true then 'If' DOM element is rendered, else the other DOM Element is rendered. Angular uses ng-template with element selector in order to display the else section on DOM.

#### **Syntax:**

```
<div *ngIf="boolean; else id_selector"> </div>
```

```
<ng-template #id_selector> </ng-template>
```

In the above Syntax, boolean stands for either true or false value. If the boolean value is true then Element in If is rendered on the DOM, else another element is rendered on the DOM .

#### **Example of \*ngIf- else:**

```
<div *ngIf="false;else id_selector">
```

This text will be hidden

```
<h1 [ngStyle]='{'color': '#FF0000'}'>
```

GFG Structural Directive

If Part

```
</h1>
```

```
</div>
```

```
<ng-template #id_selector>
```

This text will be displayed

```
<h1 [ngStyle]='{'color': '#00FF00'}'>
```

## GFG Structural Directive

### Else Part

```
</h1>
```

```
</ng-template>
```

### Output:

This text will be displayed

**GFG Structural Directive Else Part**

*\*ngIf – else Example*

### 3. \*ngFor:

\*ngFor is used to loop through the dynamic lists in the DOM. Simply, it is used to build data presentation lists and tables in HTML DOM.

#### Syntax:

```
<div *ngFor="let item of item-list"> </div>
```

#### Example of \*ngFor:

Consider that you are having a list as shown below:

```
items = ["GfG 1", "GfG 2", "GfG 3", "GfG 4"];
```

```
<div *ngFor="let item of items">
```

```
<p> {{item}} </p>
```

```
</div>
```

### Output:

GfG 1

GfG 2

GfG 3

GfG 4

*\*ngFor example*

#### 4. \*ngSwitch :

ngSwitch is used to choose between multiple case statements defined by the expressions inside the \*ngSwitchCase and display on the DOM Element according to that. If no expression is matched, the default case DOM Element is displayed.

##### Syntax:

```
<div [ngSwitch]="expression">  
  <div *ngSwitchCase="expression_1"></div>  
  <div *ngSwitchCase="expression_2"></div>  
  <div *ngSwitchDefault></div>  
</div>
```

In the above syntax, the expression is checked with each case and then the case matching with the expression is rendered on DOM else the Default case is rendered on the DOM.

##### Example of \*ngSwitch:

```
<div [ngSwitch]="one">  
  
  <div *ngSwitchCase="one">One is Displayed</div>  
  
  <div *ngSwitchCase="two">Two is Displayed</div>  
  
  <div *ngSwitchDefault>Default Option is Displayed</div>
```

</div>

In the above example, the expression 'one' in ngSwitch is matched to the expression in ngSwitchCase. Hence, the Element displayed on DOM is " One is Displayed ".

**Output:**

---

One is Displayed

**Attribute Directives:**

Directives are classes that add additional behavior to elements in your Angular applications. Use Angular's built-in directives to manage forms, lists, styles, and what users see.

The different types of Angular directives are as follows:

**DIRECTIVE  
TYPES**

**DETAILS**

---

**Components**

Used with a template. This type of directive is the most common directive type.

---

## DIRECTIVE TYPES

## DETAILS

### Attribute directives

Change the appearance or behavior of an element, component, or another directive.

### Structural directives

Change the DOM layout by adding and removing DOM elements.

### Built-in attribute directives

Attribute directives listen to and modify the behavior of other HTML elements, attributes, properties, and components.

Many NgModules such as the [RouterModule](#) and the [FormsModule](#) define their own attribute directives. The most common attribute directives are as follows:

## COMMON DIRECTIVES

## DETAILS

### NgClass

Adds and removes a set of CSS classes.

COMMON  
DIRECTIVES

## DETAILS

[NgStyle](#)

Adds and removes a set of HTML styles.

[NgModel](#)

Adds two-way data binding to an HTML form element.

---

### [Adding and removing classes with \[NgClass\]\(#\)](#)

Add or remove multiple CSS classes simultaneously with [ngClass](#).

To add or remove a *single* class, use [class binding](#) rather than [NgClass](#).

Using [NgClass](#) with an expression

On the element you'd like to style, add [[ngClass](#)] and set it equal to an expression.

In this case, `isSpecial` is a boolean set to true in `app.component.ts`.

Because `isSpecial` is true, [ngClass](#) applies the class of `special` to the `<div>`.

```
<!-- toggle the "special" class on/off with a property -->
```

```
<div [ngClass]="isSpecial ? 'special' : ''>This div is special</div>
```

Using [NgClass](#) with a method

1. To use [NgClass](#) with a method, add the method to the component class. In the following example, `setCurrentClasses()` sets the property `currentClasses` with an object that adds or removes three classes based on the true or false state of three other component properties. Each key of the object is a CSS class name. If a key is true, [ngClass](#) adds the class. If a key is false, [ngClass](#) removes the class.  
`src/app/app.component.ts`

```
content_copycurrentClasses: Record<string, boolean> = {};
```

```
/* ... */

setCurrentClasses() {

  // CSS classes: added/removed per current state of component
  properties

  this.currentClasses = {

    saveable: this.canSave,

    modified: !this.isUnchanged,

    special: this.isSpecial

  };

}
```

2. In the template, add the [ngClass](#) property binding to currentClasses to set the element's classes:

src/app/app.component.html

```
<div [ngClass]="currentClasses">This div is initially saveable,
unchanged, and special.</div>
```

---

## Displaying and updating properties with [ngModel](#)

Use the [NgModel](#) directive to display a data property and update that property when the user makes changes.

1. Import [FormsModule](#) and add it to the NgModule's imports list.

```
import { FormsModule } from '@angular/forms'; // <--- JavaScript
import from Angular

/* ... */

@NgModule({

  /* ... */

  imports: [
```



```

    BrowserModule,

    FormsModule // <--- import into the NgModule

  ],
  /* ... */
})

export class AppModule { }

```

Add an [(ngModel)] binding on an HTML <form> element and set it equal to the property, here name.

src/app/app.component.html (NgModel example)

```

<label for="example-ngModel">[(ngModel)]:</label>

<input [(ngModel)]="currentItem.name" id="example-ngModel">

```

### Property binding in angular

Property Binding is a **one-way data-binding** technique. In property binding, we bind a property of a DOM element to a field which is a defined property in our component TypeScript code. Actually, Angular internally converts string interpolation into property binding.

In this, we bind the property of a defined element to an HTML DOM element.

#### Syntax:

```
<element [property]= 'typescript_property'>
```

- Approach: Define a property element in the app.component.ts file.
- In the app.component.html file, set the property of the HTML element by assigning the property value to the app.component.ts file's element.

**Example 1:** setting value of an input element using property binding.

**app.component.html**

```

<input style = "color:green;

margin-top: 40px;

```

```
margin-left: 100px;"
```

```
[value]='title'>
```

### **app.component.ts**

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

```
@Component({
```

```
  selector: 'app-root',
```


```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  title = 'GeeksforGeeks';
```

The logo for GeeksforGeeks, featuring the text "GeeksforGeeks" in a green, sans-serif font, enclosed within a thin black rectangular border.

## Angular Lifecycle

Angular is a dominant and broadly classified **client-side** platform that has impressed millions of developers. Ever since the inception of the Angular platform, making applications has turned way easier than ever. Angular is extensively used in data visualization and building applications for both **mobile** and **desktop**. The seamless potential to leverage data makes angular unique in its class. With the new

features being added every year, angular has captured the marketing trends, and each year the lifecycle of Angular keeps evolving at a great pace.

Back in 2009, when Google introduced angular, it has limited lifecycle dependencies and features and was initially an initiative to be served on HTML and JavaScript. However, in the latest version, **TypeScript** has replaced the former JavaScript and other popular scripting languages. In another perspective, Angular has given the freedom to the developers to build applications that run on the **web, mobile, and desktop** with fewer footprints. Let's now discuss some more aspects of the Angular lifecycle.

### What is Angular lifecycle?

In Angular, each component is channeled through 8 different phases in its lifecycle. To be precise, it is first initialized, and then the root is created and is later presented to its components. It is always checked whenever the components get loaded during the development of the application and are gradually updated. When the component remains unused, the death phase is approached by decimating and expelling it from the **DOM**.

It is interesting to note that Angular can oversee all the lifecycle of its components and directives and can smartly understand the result with the previous data, making the application's integration smooth. Collectively, components make the primary building blocks of the version, and Directives channelize these versions with the build.

Another interesting thing about the Angular lifecycle is that each component has its lifecycle, and at every stage, the lifecycle moves from initialization to destruction. Each component goes through 8 stages. The steps can be explained in the following points.

1. On initializing any Angular lifecycle component, it creates and gets presented to its root components designed to be produced to heirs.
2. The next phase involves the components that get loaded during application development, and data binding techniques are changed and updated gradually.
3. The later section involves unused components that are not utilized and are approached to the death phase and moved out of DOM.

## Overview of Lifecycle with Hooks

All the events that occur in the component's life are termed "**lifecycle hooks**" in Angular. Hooks are nothing but simple functions that a developer can call anytime in the Angular application. You can think of hooks as callback methods that raise positive events happening in the lifecycle of a component. As discussed earlier, eight stages greatly determine the lifecycle of Angular. Let's learn about them.

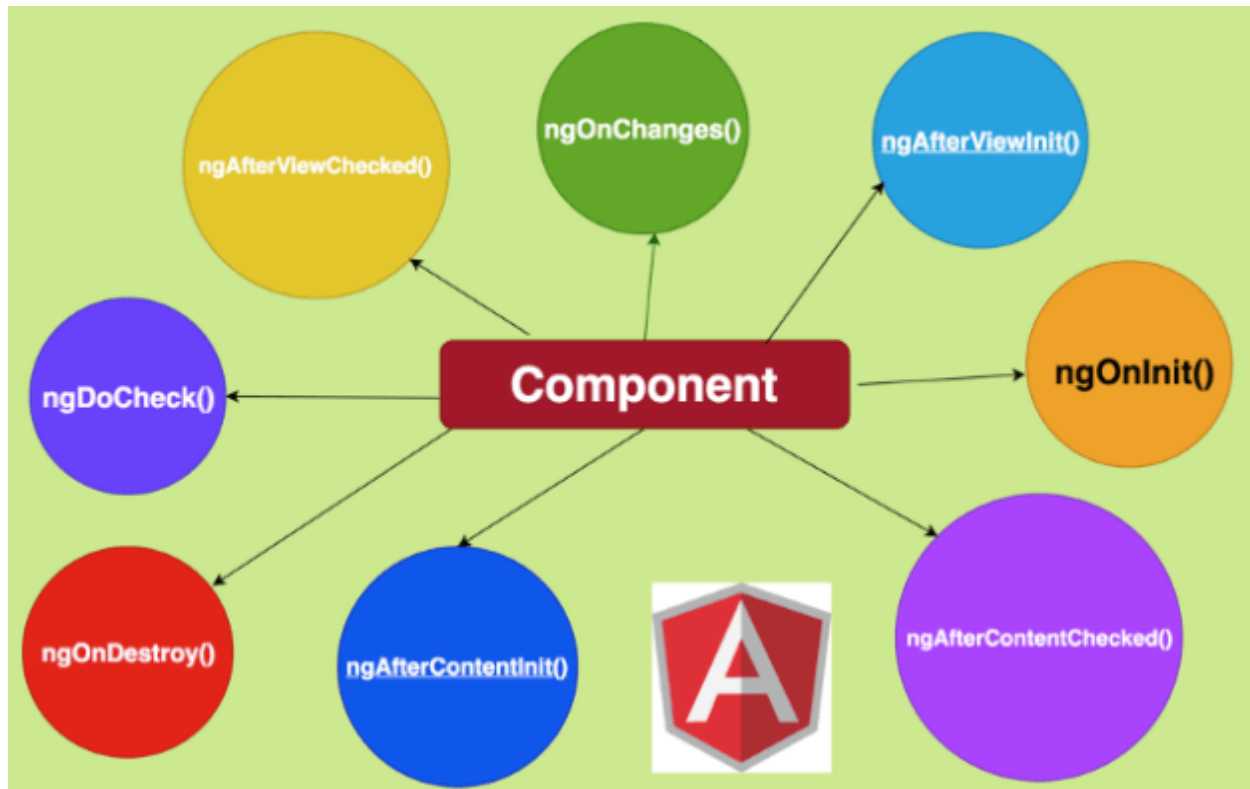
### Online vs. Constructor

Being a programmer, it is important to generate your component and introduce it. You always need to keep two options, i.e., either utilize the constructor or use **OnInit** lifecycle method. Whenever the component is utilized, the **OnInit** lifecycle method is automatically triggered.

Although both Constructor and **OnInit** methods can be used for the same cause yet, the developer prefers using **ngOnInit** for declaration or initialization and avoiding using constructors most of the time.

### Interface

The interface is an important asset of hook in the lifecycle methods because the component class of the application must be implemented in the virtual interface. As per recommendation, the method that triggers the implementation, i.e., **AfterViewInit**, should be suitable for **ngAfterViewInit**.



## ngOnChanges

This is a callback function used to bound data property variations that occur for a component. In simple terms, you can think of it as an event that gets executed as and when the input controls get renewed inside the component. The changed data is mapped when Angular receives it, and the current and the previous data are encapsulated in a simple change. Therefore, by using these lifecycle hooks, a Parent component can easily communicate to its child components provided the decorator is exposed to **@InputDecorator** of the child. Even if the Parent component gets off track from the input property, this hook gets automatically summoned to the child component. This helps the developers to worry less about the implementation details. Thus, some of its properties include utilizing all components that have input, invoking when the input is changed, and getting an initial call raised before **ngOnInit**.

Properties of this phase include practical utilization of all the input components, which gets invoked whenever the value gets changed, and raising initial calls before **ngOnInit**.

## ngOnInit

After Angular has completed creating and introducing the callbacks, it gets initialized and displays some data-bound properties. This event takes place only after **ngOnChanges** event and constructor calling. With this hook, the component's logic is initialized, and all the properties of **ngOnInit** can be used. Any child belonging to these properties cannot be used before the hook gets triggered.

## ngDoCheck

After the hooks are sorted in this phase, there remains a vitality to review the inputs and properties associated with them respective to components and directives. You can easily check your logic here. To be precise, you can put custom checks to view test cases and later implement them into the component. This hook comes in handy after **ngOnInit**, and the main function of this hook is to execute even if there is no change in the component's property. This hook comes into view if, for any instance, Angular miscarries to detect any changes in the input properties. Some of the properties of this hook include change detection and automatic function calls.

## ngAfterContentInit

This phase is raised when the **ngDoCheck** is marked complete. Every component in **ngAfterContentInit** gets introduced and check for the first time. Usually, this method is implemented as soon as Angular does some content project and the component view. This method is called when the properties get demarked as **ContentChild** or **ContentChildren**, ensuring that all of these have been initialized. On the other hand, external child components can be included using the method by placing **<ng-content>** and **</ng-content>** tags. Thus, for all the lifecycle components, this hook is called only once. Some properties of this phase may include not initializing the content since it takes place after **ngDoCheck**.

## ngAfterContentChecked

This hook method is designed to investigate changes in the contents using the Angular change detection apparatus. It performs its tasks even if no modifications are made, ensuring that everything is check and is on point. This hook gets its call just after the **ngAfterContentInit** and is automatically executed after every execution of **ngDoCheck**. It plays a vital role in child component initialization.

Some of the main properties of this method include waiting for **ngContentInit** to get executed and ensure everything is executed only after **ngDoCheck**.

### ngAfterViewInit

This lifecycle method is called right after **ngAfterContentChecked** and makes use of only the components. This method is similar to the **ngAfterContentInit** and invokes only after the component and the child view. This method is called only after initialization of view and for only one time.

### ngAfterViewChecked

This method is triggered when there is a subsequent check-in, the component check, and the child view. This method is called right after **ngAfterViewInit** and then for check method using **ngAfterContentChecked**. Like some other lifecycle methods discussed above, this method is used only for components, and it comes in handy when the child component is waiting for execution.

Some of the properties of this method are to get called once the initialization and checking are done, and right after **ngAfterContentCheck** method, it starts executing.

### ngOnDestroy

This method is called when Angular destroys all the directives or components. This is the right place to clean up all the constructed logic and unsubscribe them from the observable. This detaches all the event handlers and is solely done to eliminate memory leakage or buffer-related issues.

One of the main functions or properties of this hook is to get the call after the components are removed from the DOM.

## AngularJS Dependency Injection

AngularJS comes with a built-in dependency injection mechanism. It facilitates you to divide your application into multiple different types of components which can be injected into each other as dependencies.

Dependency Injection is a software design pattern that specifies how components get holds of their dependencies. In this pattern, components are given their dependencies instead of coding them within the component.

Modularizing your application makes it easier to reuse, configure and test the components in your application. Following are the core types of objects and components:

- value
- factory
- service
- provider
- constant

These objects and components can be injected into each other using AngularJS Dependency Injection.

### Value

In AngularJS, value is a simple object. It can be a number, string or JavaScript object. It is used to pass values in factories, services or controllers during run and config phase.

1. `//define a module`
2. `var myModule = angular.module("myModule", []);`
3. `//create a value object and pass it a data.`
4. `myModule.value("numberValue", 100);`
5. `myModule.value("stringValue", "abc");`
6. `myModule.value("objectValue", { val1 : 123, val2 : "abc" } );`

Here, values are defined using the `value()` function on the module. The first parameter specifies the name of the value, and the second parameter is the value itself. Factories, services and controllers can now reference these values by their name.

### Injecting a value

To inject a value into AngularJS controller function, add a parameter with the same when the value is defined.

1. `var myModule = angular.module("myModule", []);`
2. `myModule.value("numberValue", 100);`



```
3. myModule.controller("MyController", function($scope, numberValue) {  
4.   console.log(numberValue);  
5. });
```

---

## Factory

Factory is a function that is used to return value. When a service or controller needs a value injected from the factory, it creates the value on demand. It normally uses a factory function to calculate and return the value.

Let's take an example that defines a factory on a module, and a controller which gets the factory created value injected:

```
1. var myModule = angular.module("myModule", []);  
2. myModule.factory("myFactory", function() {  
3.   return "a value";  
4. });  
5. myModule.controller("MyController", function($scope, myFactory) {  
6.   console.log(myFactory);  
7. });
```

## Injecting values into factory

To inject a value into AngularJS controller function, add a parameter with the same when the value is defined.

```
1. var myModule = angular.module("myModule", []);  
2. myModule.value("numberValue", 100);  
3. myModule.controller("MyController", function($scope, numberValue) {  
4.   console.log(numberValue);  
5. });
```

*Note: It is not the factory function that is injected, but the value produced by the factory function.*

---

## Service

In AngularJS, service is a JavaScript object which contains a set of functions to perform certain tasks. Services are created by using service() function on a module and then injected into controllers.

1. **//define a module**
2. `var mainApp = angular.module("mainApp", []);`
3. ...
4. **//create a service which defines a method square to return square of a number.**
5. `mainApp.service('CalcService', function(MathService){`
6.     `this.square = function(a) {`
7.         `return MathService.multiply(a,a);`
8.     `}`
9. `});`
10. **//inject the service "CalcService" into the controller**
11. `mainApp.controller('CalcController', function($scope, CalcService, defaultInput) {`
12.     `$scope.number = defaultInput;`
13.     `$scope.result = CalcService.square($scope.number);`
14.     `$scope.square = function() {`
15.         `$scope.result = CalcService.square($scope.number);`
16.     `}`
17. `});`

---

## Provider

In AngularJS, provider is used internally to create services, factory etc. during config phase (phase during which AngularJS bootstraps itself). It is the most flexible form of factory you can create. Provider is a special factory method with a get() function which is used to return the value/service/factory.

1. **//define a module**
2. `var mainApp = angular.module("mainApp", []);`
3. ...

4. `//create a service using provider which defines a method square to return square of a number.`
  5. `mainApp.config(function($provide) {`
  6. `$provide.provider('MathService', function() {`
  7. `this.$get = function() {`
  8. `var factory = {};`
  9. `factory.multiply = function(a, b) {`
  10. `return a * b;`
  11. `}`
  12. `return factory;`
  13. `};`
  14. `});`
  15. `});`
- 

## Constants

You cannot inject values into the `module.config()` function. Instead constants are used to pass values at config phase.

1. `mainApp.constant("configParam", "constant value");`

Let's take an example to deploy all above mentioned directives.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<title>AngularJS Dependency Injection</title>`
5. `</head>`
6. `<body>`
7. `<h2>AngularJS Sample Application</h2>`
8.
9. `<div ng-app = "mainApp" ng-controller = "CalcController">`
10. `<p>Enter a number: <input type = "number" ng-model = "number" /></p>`
11. `<button ng-click = "square()">X<sup>2</sup></button>`
12. `<p>Result: {{result}}</p>`

```
13. </div>
14.
15. <script src = "http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.mi
    n.js"></script>
16.
17. <script>
18.     var mainApp = angular.module("mainApp", []);
19.
20.     mainApp.config(function($provide) {
21.         $provide.provider('MathService', function() {
22.             this.$get = function() {
23.                 var factory = {};
24.
25.                 factory.multiply = function(a, b) {
26.                     return a * b;
27.                 }
28.                 return factory;
29.             };
30.         });
31.     });
32.
33.     mainApp.value("defaultInput", 10);
34.
35.     mainApp.factory('MathService', function() {
36.         var factory = {};
37.
38.         factory.multiply = function(a, b) {
39.             return a * b;
40.         }
41.         return factory;
42.     });
43.
44.     mainApp.service('CalcService', function(MathService){
45.         this.square = function(a) {
```

```
46.         return MathService.multiply(a,a);
47.     }
48. });
49.
50.     mainApp.controller('CalcController', function($scope, CalcService, defaultIn
        put) {
51.         $scope.number = defaultInput;
52.         $scope.result = CalcService.square($scope.number);
53.
54.         $scope.square = function() {
55.             $scope.result = CalcService.square($scope.number);
56.         }
57.     });
58.     </script>
59. </body>
60.</html>
```