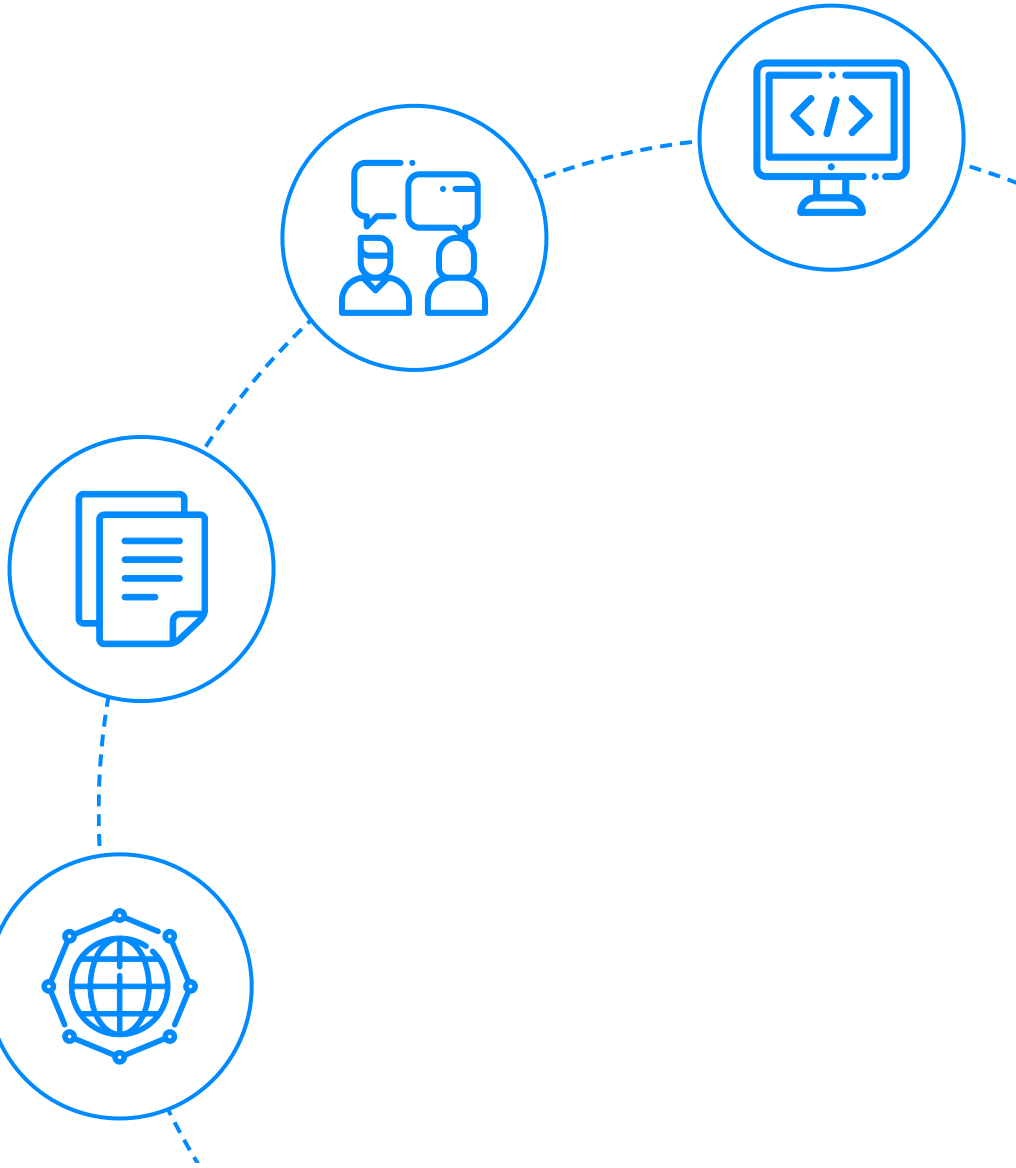
InterviewBit

# Angular Cheat Sheet



To view the live version of the page, click here.

# Contents

## Angular Tutorial: Basics to Advanced

# Let's get Started

Angular is a JavaScript front-end framework, which is used to develop web and mobile apps. As the name suggests, it is a declarative approach to developing an application. It is, therefore, different from other **JavaScript frameworks** like React, which is an application of higher-order JavaScript. The development process of an Angular application is not different from other traditional web apps. The structure of the code is similar to the following: directives, services, models, pipes, and Rural routes. The only difference is that the source code of the application is written in angular format.

You may hear of too many versions of Angular as a beginner, and as a result, it is likely that you will get confused with so many different versions out there for the same framework. There are versions like AngularJS, Angular 2, Angular 4, Angular 5, Angular 6, Angular 7, Angular 8, and now Angular 9. There are actually two different frameworks - AngularJS and Angular.

In 2010, AngularJS was the initial release and was known as AngularJS. It was a JavaScript-based web development framework that was developed and maintained by Google. The type-setting language JavaScript is super-set to the language of Java. In September 2016, Angular 2 was created, which was a complete rewrite of the framework using TypeScript, a superset of JavaScript.

In this article, we will discuss some of the Angular features. You can follow this angular cheat sheet to build your application. We've tried to cover Angular CLI, Angular Lifecycle Hooks, Angular Routing, and a lot more in this post.

## Angular Tutorial: Basics to Advanced

### 1. Angular CLI

The Angular CLI or the command line interface is a very powerful and sophisticated tool that enables you to perform a lot of tasks in an Angular project by utilizing simple commands. Everything is handled by the CLI. In order to scaffold a brand-new Angular project, for example, the CLI generates the application, compiles the application, and ships it to you for testing. The development server monitors the source code files for changes and when you change any of them, it automatically compiles the source code files and refreshes the app in the browser.

| Command | Meaning |
| --- | --- |
| `npm install -g @angular/cli` | To install the Angular CLI into our local machine using npm, run this command. |
| `ng version` | Displays the information about the currently installed CLI. |
| `ng new <application name>` | Using the ng new command, a new Angular application will be created. |
| `ng new <application name> --prefix best` | New project is created, and the project prefix is set to new. |
| `ng new --help` | All available Angular commands are returned by this command. |
| `ng lint my-app` | Linting warnings are checked against this command in our entire application. |
| `ng lint my-app --fix` | This command will correct any form of linting errors. |
| `ng lint my-app --format stylish` | Our entire codebase is formatted using this command. |
| `ng lint my-app --help` | The list of linting commands is returned by this command. |
| `ng add <package` | To use this command, you must first enable your package manager. Then, this command will use your package manager to |

## 2. Angular Lifecycle Hooks

Angular apps are made up of pieces. There are pieces in an Angular app that are tree-structured, and pieces that consist of more pieces. An Angular app is made up of components, which is a tree of components. A component is a template, a typescript class, and a stylesheet file. Angular components have a lifecycle that is administered by Angular.

The Angular lifecycle hooks provide fine-grained control of Angular by capturing different phases of birth to death. You can see how Angular phases change in certain portions of its lifecycle. Here's how you can control the phases of Angular using Angular lifecycle hooks.

| Hook | Significance |
| --- | --- |
| `ngOnChanges` | The content is processed or child views are loaded before this hook is executed. It is also executed when the input properties of the component change. |
| `ngOnInit` | Data can be initialized in a component by calling this hook after input values are set. It is performed only once after input values are set. |
| `ngOnDestroy` | You can use this hook to clean up memory and release resources and subscriptions after a component is destroyed. |
| `ngDoCheck` | Any changes detected are handled using this hook. |
| `ngAfterContentInit` | After performing content projection into the component's view, Angular invokes this hook before evaluating the expression. |
| `ngAfterContentChecked` | Angular's change detection mechanism checks the content of all components once every time they are rendered, so this hook is called each time change is detected. |
| `ngAfterViewInit` | When the component's view has been fully initialized, this hook is called |

| Template Syntax | Details |
| --- | --- |
| `<input [val]="name">` | Binds the “name” expression result to the property “val” |
| `<div [attr.role]="myAriaRole">` | An expression that binds an attribute role to a result of expression “myAriaRole”. |
| `<div [class.extra]="isADelight">` | The truthiness of the expression isADelight binds to the CSS class extra |
| `<div [style.height.px]="myHeight">` | The result of the expression myHeight binds to the style property height |

## 3. DECORATORS

Classes and fields can be decorated with Angular's dozens of decorators. These are some of the most commonly used decorators.

| Class Decorators | Details |
| --- | --- |
| `import { Directive, ... } from '@angular/core';` | This imports the Directive from @angular/core |
| `@Component({...}) class MyComponent() {}` | Metadata about a component is declared as part of the class definition. |
| `@Directive({...}) class MyDirective() {}` | Declares the class as a directive and provides metadata about the directive |
| `@Pipe({...}) class MyPipe() {}` | Declares the class as a pipe and provides metadata about the pipe. |
| `@Injectable() class MyService() {}` | This declares that class can be injected and provided. Without this decorator, the compiler does not generate enough metadata to allow the class to be created properly when it is injected somewhere. |

| CLASS FIELD DECORATORS | Details |
| --- | --- |
| `import { Inp } from '@angular/core';` | Import Inp from @angular/core. |
| `@Input() myProperty;` | You can declare input properties that you can bind to using property binding |
| `@Output() myEvent = new EventEmitter();` | An output property is declared that can fire subscribable events. |
| `@HostBinding('class.valid') isValid;` | Host element property is binded to a component property |
| `@HostListener('click', ['$event']) onClick(e) {...}` | Host element event is subscribed with a directive method |
| `@ContentChild(myPredicate) myChildComponent;` | First result of the query in the component content is binded to a property of the class |
| `@ContentChildren(myPredicate) myChildComponents;` | Results of the query in the component content is binded to a property of the class |
| `@ViewChild(myPredicate) myChildComponent;` | First result of the query in the component view is binded to a property of the class |
| | Results of the query in the |

### DEPENDENCY INJECTION CONFIGURATION

A dependency is a piece of information needed by a class to carry out its task. A service, on the other hand, is an object that a class creates and uses to carry out its tasks.A dependency injection container such as Angular's Dependency Injection(DI) framework is used to create the dependencies. Most of the time, a class depends on other classes, rather than on itself, to create the required dependencies. Dependencies are created by external sources, such as services and other classes. Following are dependency injection configuration as part of Angular's DI framework:

| DEPENDENCY INJECTION CONFIGURATION | DETAILS |
|---|---|
| `{ provide: InterviewBitService, useClass: InterviewBitMockService }` | InterviewBitService's provider is set or overridden to InterviewBitMockService class |
| `{ provide: InterviewBitService, useFactory: InterviewBitFactory }` | InterviewBitService's provider is set or overridden to InterviewBitFactory factory function |
| `{ provide: InterviewBitValue, useValue: 56 }` | InterviewBitValue's provider is set or overridden to the value 56 |

## 4. Angular Directives

An element or component can be assigned an attribute directive or a structural directive to modify its behaviour. An attribute directive is an attribute that is associated with an element or component. A structural directive is a directive that modifies the structure of an element or component.

**1. Attribute Directives:** An element, component, or other directive can be decorated with an attribute directive. Angular exports the following attribute directives:

| Directive | Details | Example |
| --- | --- | --- |
| NgClass | A CSS class can be added or removed via NgClass. | `<div [ngClass]="isInterviewBitSpecial ? 'Yes' : ''">This company is special</div>` |
| NgStyle | HTML styles can be added or removed via NgStyle.. | `<div [ngStyle]="{ 'font-height': 3 +3 === 6 ? 'light' : 'normal', }"> This div is light. </div>` |
| NgModel | Two-way data binding to an HTML form element can be added via NgModel.. | `<input [(ngModel)]="interviewBit">` |

**2. Structural Directives:** Elements that are added or removed from the DOM in Angular's structure are referred to as structural directives. Here are the most prevalent structural directives in Angular:

| Directive | Details |
|---|---|
| `NgIf` | The Angular conditional NgIf directive conditionalizes the value of NgIf. If the NgIf directive's value is false, Angular removes the element. |
| `NgFor` | The Angular NgFor directive loops through an array or list. It comes in two types: The ng-for directive, which loops through a ul> or ol> element; and the ng-for-each directive, which iterates through a collection. |
| `NgSwitch` | NgSwitch is a structural directive, meaning that it should be assigned a particular value depending on the context in which it is used. |
| `NgSwitchCase` | A NgSwitchCase structure stores a matched value for NgSwitch, and it can also be used to refer to a matched value. |
| `NgSwitchDefault` | When the expression does not match any of the specified values, NgSwitchDefault performs the function. |

## 5. Pipes

Templates typically use pipes to change content but it does not directly affect data. For example:

| Pipe | Details | Example |
|---|---|---|
| `DatePipe` | Locale-specific date formatting is performed. | `{{ value_expr \| date: 'long'}}` |
| `UpperCasePipe` | Given text is transformed into upper case text. | `{{ 'InterviewBit' \| uppercase }}` |
| `LowerCasePipe` | Given text is transformed into lower case text. | `{{ 'InterviewBit' \| lowercase }}` |
| `CurrencyPipe` | Given number is transformed into a currency string. | `{{ 4.4324 \| currency:'USD' }}` |
| `DecimalPipe` | Given number is transformed into a decimal point string. | `{{ 1.1334354654 \| number }}` |
| `PercentPipe` | Given number is transformed into a percentage point string. | `{{ 0.245 \| percent }}` |

## Conclusion

In this document, we've covered the basics of Angular, its features and some of the important cheat sheets. Now, it's time for you to head out and try what we've covered here and more. More than memorizing syntax, do pay attention to practising them and solving problems.

## Additional Resources

- https://www.interviewbit.com/javascript-interview-questions/
- https://www.interviewbit.com/angular-interview-questions/
- https://www.interviewbit.com/angularjs-interview-questions/
- https://www.interviewbit.com/blog/angular-architecture/
- https://www.interviewbit.com/angular-8-interview-questions/
- https://www.interviewbit.com/angular-mcq/

# Links to More Interview Questions

| | | |
|---|---|---|
| C Interview Questions | Php Interview Questions | C Sharp Interview Questions |
| Web Api Interview Questions | Hibernate Interview Questions | Node Js Interview Questions |
| Cpp Interview Questions | Oops Interview Questions | Devops Interview Questions |
| Machine Learning Interview Questions | Docker Interview Questions | Mysql Interview Questions |
| Css Interview Questions | Laravel Interview Questions | Asp Net Interview Questions |
| Django Interview Questions | Dot Net Interview Questions | Kubernetes Interview Questions |
| Operating System Interview Questions | React Native Interview Questions | Aws Interview Questions |
| Git Interview Questions | Java 8 Interview Questions | Mongodb Interview Questions |
| Dbms Interview Questions | Spring Boot Interview Questions | Power Bi Interview Questions |
| Pl Sql Interview Questions | Tableau Interview Questions | Linux Interview Questions |
| Ansible Interview Questions | Java Interview Questions | Jenkins Interview Questions |