

Log in Get Started Home / Articles / Programming Sameeksha Medewar | 30 Jan, 2025

Download Angular Cheat Sheet PDF for 2025 [Updated] The open-source Angular

framework has a modular architecture and comes with reusable UI components — perfect for building large-scale applications. Angular is based on TypeScript, a programming language heavily influenced by JavaScript. Moreover, Angular leverages new technology called Ivy, which is the code name for Angular's compilation and rendering pipeline. Since Angular is one of the most widely used frameworks, many tech professionals study it to attain stable job opportunities. But you can't remember all Angular methods and commands by heart. That's where this handy Angular cheat sheet comes in! Ready to dive into Angular syntax and breeze through your next Angular Disclosure: Hackr.io is supported by its audience. When you purchase through links on our site, we may earn an affiliate commission. In this article Angular Cheat Sheet Conclusion Frequently Asked Questions project? Use this updated Angular 4 cheat sheet to supplement your web development career. Download the Complete Angular Cheat Sheet PDF Here. Angular Cheat Sheet Before we get started, let's touch briefly upon Angular JS. The background will help you understand the greater purpose behind using the Angular framework in your work. What is Angular JS? Google developed AngularJS, a Javascript framework that helps developers to create fully scaled singlepage web apps. The term "single page" refers to the necessity for web pages to be updated. A single web page consists of multiple moving parts and their organization– navigation bar, footer, sidebar, and more. To inject these components into the same web page, we use AngularJS. Bottom line? AngularJS makes the web page dynamic. Instead of refreshing the page when a user navigates to a URL, AngularJS injects the needed components into the same page. It basically reuses the components that don't change, which cuts down on load time and offers a better browsing experience. Now, let's get into some Angular commands. Angular CLI Cheat Sheet The Angular command-line interface CLI is a set of commands that help you initialize, develop, and maintain highly scalable and speedy Angular apps directly from the command shell. In this Angular CLI commands cheat sheet section, we'll cover various Angular CI commands. 1. Setup The following command installs Angular CLI globally: `npm install -g @angular/cli` 2. New Application The following command sets the prefix to "best:" `ng new best-practises --prefix best` This command checks the available command list. `ng new --help` This command simulates the "ng new" command: `ng new best-practises --dry-run` 3. Lint for Formatting The Lint command fixes code smells and corrects improper formatting. `ng lint my-app --fix` This next command shows warnings: `ng lint my-app` If you want to format the code, you can use the following command. `ng lint my-app --format stylish` Next, this command verifies the accessible list of commands. `ng lint my-app --help` 4. Blueprints Generate spec: `--spec` Check whether the template will be a .ts file or not: `--inline-template (-t)` Check whether the style will be in the .ts file or not: `--inline-style (-s)` Create a directive:

ng g d directive-name Create a pipeline: ng g p init-caps Create customer class in the models folder: ng g cl models/customer Creates a component without the need for the creation of a new folder. ng g c my-component --flat true Assign a prefix: --prefix Create an interface in the models folder: ng g i models/person Create an ENUM gender in the models folder: ng g e models/gender Create a service: ng g s service-name 5. Building Serving Build an app to /dist folder: ng build Optimize and build an app without using unnecessary code: ng build --aot Create a build for production: ng build --prod Specify serve with opening a browser: ng serve -o Reload when changes occur: ng serve --live-reload Serve using SSL ng serve -ssl Recommended Courses 6. Add New Capabilities Add angular material to project: ng add @angular/material Create a material navigation component: ng g @angular/material:material -nav --name nav Components and Templates Components are the most fundamental Angular UI building pieces. An Angular component tree makes up an Angular app. Sample Component ts File import { Component } from '@angular/core'; @Component({ // component attributes selector: 'app-root', templateUrl: './app.component.html', styleUrls: ['./app.component.less'] }) export class AppComponent { title = 'Hello World'; } Component Attributes changeDetection The change-detection strategy to use for this component. viewProviders Defines the set of injectable objects visible to its view DOM children. moduleId The module ID of the module that contains the component. encapsulation An encapsulation policy for the template and CSS styles. interpolation Overrides the default encapsulation start and end delimiters ({{ and }}. entryComponents A set of components that should be compiled along with this component. preserveWhitespaces True to preserve or false to remove potentially superfluous whitespace characters from the compiled template. Component Life Cycles ngOnInit Called once, after the first ngOnChanges() ngOnChanges Called before ngOnInit() and whenever one of the input properties changes. ngOnDestroy Called just before Angular destroys the directive/component. ngDoCheck Called during every change detection run. ngAfterContentChecked Called after the ngAfterContentInit() and every subsequent ngDoCheck() ngAfterViewChecked Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked(). ngAfterContentInit Called once after the first ngDoCheck(). ngAfterViewInit Called once after the first ngAfterContentChecked(). Template Syntax {{user.name}} Interpolation - generates user name. property binding - bind image url for user to src attribute Event - assign function to click event Show button when user.showSth is true *ngFor="let item of items" Iterate through the items list

Angular ngClass attribute

Angular ngStyle attribute Input and Output Input() To pass value into child component

Sample child component implementation: export class SampleComponent { @Input()

value: any/string/object/...; ... } Sample parent component usage: Output() Emitting event

to parent component
Sample child component: `@Output() myEvent: EventEmitter = new EventEmitter(); onRemoved(item: MyModel) { this.myEvent.emit(item); }`
Sample parent component: `onRemoved` in the child component is calling the `someFunction()` method in the parent component, as we can see in the above two child and parent components.
Content Projection
Content projection in Angular is a pattern in which you inject the desired content into a specific component. Here's an example of a parent component template:

(some html here)

Child component template: Let us now inject the following HTML code in the parent component template:

(some html here)

It will look like:

(some html here)

(some html here)

When we combine both the above parent and child template, you get the following result:

(some html here)

(some html here)

ViewChild Decorator Offers access to child component/directive/element:

```
@ViewChild(NumberComponent) private numberComponent: NumberComponent;
increase() { this.numberComponent.increaseBy One(); //method from child component }
decrease() { this.numberComponent.decreaseBy One(); //method from child component }
```

Sample for element: html:

component: `@ViewChild('myElement') myElement: ElementRef` Instead `ElementRef` can be used for specific elements like `FormControl` for forms. Reference to element in html:

Routing
The Angular Router enables navigation from one view to the next as users perform application tasks. Sample routing ts file: `const appRoutes: Routes = [{ path: 'crisis-center', component: CrisisListComponent }, { path: 'prod/:id', component: HeroDetailComponent }, { path: 'products', component: ProductListComponent, data: { title: 'Products List' } }, { path: '', redirectTo: '/products', pathMatch: 'full' }, { path: '**', component: PageNotFoundComponent }]`; Then, this should be added inside

Angular.module imports: `RouterModule.forRoot(appRoutes)` You can also turn on console tracking for your routing by adding `enableTracing`: `imports: [RouterModule.forRoot(routes, {enableTracing: true})]`, Usage `CrisisCenter routerLinkActive="active"` will add active class to element when the link's route becomes active

//Navigate from code
`this.router.navigate(['/heroes']);` // with parameters `this.router.navigate(['/heroes', { id: herold, foo: 'foo' }]);` // Receive parameters without Observable let
`id=this.route.snapshot.paramMap.get('id');` CanActivate and CanDeactivate
In Angular routing, two route guards are CanActivate and CanDeactivate. The former decides whether

the route can be activated by the current user, while the latter decides whether the router can be deactivated by the current user.

```
CanActivate: class UserToken {} class Permissions
{ canActivate(user: UserToken, id: string): boolean { return true; }}
CanDeactivate: class UserToken {} class Permissions { canDeactivate(user: UserToken, id: string): boolean
{ return true; }}
```

Modules Angular apps are modular and Angular has its own modularity system called NgModules. NgModules are containers for a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities.

Sample Module with Comments

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent], // components, pipes, directives
  imports: [BrowserModule, AppRoutingModule], // other modules
  providers: [], // services
  bootstrap: [AppComponent] // to present component
})
export class AppModule {}
```

Services Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. Instead, they should focus on presenting data and delegate data access to a service.

Sample service with one function:

```
@Injectable()
export class MyService {
  public items: Item[];
  constructor() {}
  getSth() { // some implementation }
}
```

When you create any new instance of the component class, Angular determines the services and other dependencies required by that component by looking at the parameters defined in the constructor as follows:

```
constructor(private dogListService: MyService) {}
```

The above constructor requires the service: MyService

Register MyService in the providers module:

```
providers: [MyService]
```

HttpClient This command handles and consumes http requests.

Add import to module:

```
import { HttpClientModule } from '@angular/common/http';
```

You can use the above statement in the following way:

```
import { HttpClient } from '@angular/common/http';
... // GET
public getData(): Observable { return this.http.get('api/users/2'); }
// POST
public send(val1: any, val2: any): Observable {
  const object = new SendModel(val1, val2);
  const options = { headers: new HttpHeaders({ 'Content-type': 'application/json' }) };
  return this.http.post(environment.apiUrl + 'api/login', object, options);
}
```

Dependency Injection This injects a class into another class:

```
@Injectable({ providedIn: 'root', })
export class SomeService {}
```

It accepts 'root' as a value or any module of your application.

Declare Global Values Class:

```
import { InjectionToken } from '@angular/core';
export const CONTROLS_GLOBAL_CONFIG = new InjectionToken('global-values');
export interface ControlsConfig { firstGlobalValue: string; }
Module: providers: [{ provide: CONTROLS_GLOBAL_CONFIG, useValue: { firstGlobalValue: 'Some value' } },
```

Usage (for example in component):

```
constructor(@Optional() @Inject(CONTROLS_GLOBAL_CONFIG) globalVlues: ControlsConfig) {}
```

Pipes Pipes transform data and values to a specific format.

For example: Show date in shortDate format:

```
{{model.birthsDay | date:'short Date'}}
```

Pipe implementation:

```
@Pipe({ name: 'uselessPipe' })
export class uselessPipe implements
```

PipeTransform { transform(value: string, before: string, after: string): string { let newStr = `\${before}\${value}\${after}`; return newStr; } } usage: {{ user.name | uselessPipe:"Mr.":"the great" }}

Directives An Attribute directive changes A DOM element's appearance or behavior. For example, [ngStyle] is a directive. Custom directive: import { Directive, ElementRef, HostListener, Input } from '@angular/core'; @Directive({ selector: '[appHighlight]' }) export class HighlightDirective { constructor(private el: ElementRef) { } @Input('appHighlight') highlightColor: string; @Input('otherPar') otherPar: any; //it will be taken from other attribute named [otherPar] @HostListener('mouseenter') onMouseEnter() { this.highlight(this.highlightColor || 'red'); } private highlight(color: string) { this.el.nativeElement.style.backgroundColor = color; } } Usage:

Highlight me!

Animations Animations allow you to move from one style state to another before adding BrowserModule and BrowserAnimationsModule to the module. Implementation: animations: [trigger('openClose', [state('open', style({ height: '400px', opacity: 1.5, })), state('closed', style({ height: '100px', opacity: 0.5, })), transition('open => closed', [animate('1s')]), transition('closed => open', [animate('1s')])]] Usage:

Angular Forms In this section of our Angular 4 cheat sheet, we'll discuss different types of Angular forms. Template Driven Forms Form logic (validation, properties) are kept in the template. sample html

First Name

First Name is required

Register

Sample component: @ViewChild("f") form: any; firstName: string = ""; langs: string[] = ["English", "French", "German"]; onSubmit() { if (this.form.valid) { console.log("Form Submitted!"); this.form.reset(); } } Reactive Forms Form logic (validation, properties) are kept in the component. Sample HTML

Email

Email is required

Email must be a valid email address

Register

Sample component: registerForm: FormGroup; submitted = false; constructor(private formBuilder: FormBuilder) { } ngOnInit() { this.registerForm = this.formBuilder.group({ firstName: [{value: 'here default value'}], Validators.required, lastName: ['', Validators.required], email: ['', [Validators.required, Validators.email]], password: ['', [Validators.required, Validators.minLength(6)]] }); // convenience getter for easy access to form fields get f() { return this.registerForm.controls; } onSubmit() { this.submitted =

```

true; // stop here if form is invalid if (this.registerForm.invalid) { return; }
alert('SUCCESS!! :-)') } Custom Validator for Reactive Forms Function: validateUrl(control:
AbstractControl) { if (!control.value || control.value.includes('.png') || control.value.includes('.jpg')) { return null; } return { validUrl: true }; } Usage:
this.secondFormGroup = this.formBuilder.group({ imageCtrl: ['', [Validators.required, this.validateUrl]] }); Multi-field validation: validateNameShire(group: FormGroup) { if (group) { if (group.get('isShireCtrl').value && !group.get('nameCtrl').value.toString().toLowerCase().includes('shire')) { return { nameShire : true }; } } return null; }
Multi-field validation usage: * this.firstFormGroup.setValidators(this.validateNameShire);
Error handling:

```

Name is too long

Shire dogs should have "shire" in name

Custom Validator Directive for TemplateDriven Forms Shortly, we'll cover how to register our custom validation directive to the NG_VALIDATORS service. Thanks to multi-parameter we won't override NG_VALIDATORS but just add CustomValidator to NG_VALIDATORS.

Here's what you use: @Directive({ selector: '[CustomValidator]', providers: [{provide: NG_VALIDATORS, useExisting: CustomValidator, multi:true}] }) Example:

```

@Directive({ selector: '[customValidation]', providers: [{provide: NG_VALIDATORS, useExisting: EmailValidationDirective, multi: true}] }) export class CustomValidation implements Validator { constructor() {} validate(control: AbstractControl): ValidationErrors { return (control.value && control.value.length <= 300) ? {myValue : true} : null; } } For

```

```

multiple fields: validate(formGroup: FormGroup): ValidationErrors { const passwordControl = formGroup.controls["password"]; const emailControl = formGroup.controls["login"]; if (!passwordControl || !emailControl || !passwordControl.value || !emailControl.value) { return null; } if (passwordControl.value.length > emailControl.value.length) { passwordControl.setErrors({ too Long: true }); } else { passwordControl.setErrors(null); } return formGroup; } ngModel in Custom Component

```

Add to module: providers: [{ provide: NG_VALUE_ACCESSOR, useExisting: forwardRef(() => TextAreaComponent), multi: true }] Implement ControlValueAccessor interface

```

interface ControlValueAccessor { writeValue(obj: any): void registerOnChange(fn: any): void registerOnTouched(fn: any): void setDisabledState(isDisabled: boolean)?: void }

```

registerOnChange Register a function to tell Angular when the value of the input changes.

registerOnTouched Register a function to tell Angular when the value was touched.

writeValue Tell Angular how to write a value to the input. Sample implementation:

```

@Component({ selector: 'app-text-area', templateUrl: './text-area.component.html', styleUrls: ['./text-area.component.less'], providers: [ { provide: NG_VALUE_ACCESSOR, useExisting: forwardRef(() => TextAreaComponent), multi: true } ] }) export class TextAreaComponent implements ControlValueAccessor, OnInit { @Input() value: string;

```

```
private _onChange = (data: any) => { console.log('changed: ' + data); }; private _onTouched
= (data?: any) => { console.log('touched: ' + data); }; ngOnInit(): void { const self = this; }
constructor() { writeValue(obj: any): void { this.value = obj; } registerOnChange(fn
{ this._onChange = fn; } registerOnTouched(fn: any): void { this._onTouched = fn; } } Tests
```

Every software application under development needs to be tested to verify its correctness, and so do the Angular applications. Testing implies executing various tests or test cases on an application to validate its functionality and correctness. Unit tests Unit testing, in general, is a type of software testing level that checks various components of an application separately. In Angular, the default unit testing framework is Jasmine. It is widely utilized while developing an Angular project using CLI. Service: ??

```
describe('MyService', () => { let service: MyService; beforeEach(() => service = new
MyService(); it('#fetch should update data', () => { service.fetchData();
expect(service.data.length).toBe(4); expect(service.data[0].id).toBe(1); }); }); For async
functions: it('#fetch should update data', (done // some code done(); // we need 'done' to avoid
// some code }); example async test: it('http client works', (done: DoneFn) =>
```

```
{ service.getUser().subscribe((data) => { expect(data).toBe('test'); done(); }); }); Spy and
stub When you make calls during the testing process, a stub provides canned answers to
all those calls. It does not respond to anything outside the program under test. A spy is a
stub that records the information based on the calls you make during the test. Spy: //
create spied object by copy getDataAsync from HttpService const valueServiceSpy =
jasmine.createSpyObj('HttpService', ['getDataAsync']); Stub: const stubValue =
of('StubValue'); valueServiceSpy.getDataAsync.and.returnValue(stubValue); TestBed
Mock whole module/environment for unit tests: beforeEach(() => { let httpClientMock =
TestBed.configureTestingModule({ providers: [{ provide: MyService, useValue: new
MyService(httpClientMock)} ] }); }); Then use tested object (for example service) like this:
service = TestBed.get(MyService); We can add schemas: NO_ERRORS_SCHEMA. This
```

means that we don't have to mock child component dependencies of this component because Angular won't yell if we don't include them! Miscellaneous Http Interceptor An HTTP interceptor can handle any given HTTP request. Class: @Injectable() export class MyInterceptor implements HttpInterceptor { constructor() { } intercept(request: HttpRequest, next: HttpHandler): Observable > { // do sth (like check and throw error) return next.handle(request); // if want continue } } Parameters: req: HttpRequest - It is the object that handles outgoing requests. next: HttpHandler - It indicates the next interceptor in the line or the backend in case there are no interceptors. Returns: An HTTP interceptor returns the observable of the event stream. Observable > Conclusion Here we have reached the end of the Angular cheat sheet. We discussed various domains like forms, pipes, directives, routes, etc. along with their usage and examples. Web development work is lengthy most of the time, so a quick reference for Angular commands and syntax goes a

long way. Interested in finding more references to speed up your work? Try another cheat sheet! Explore the JavaScript Cheat Sheet Frequently Asked Questions

1. What is the Angular Cheat Sheet? The Angular cheat sheet is a set of short notes that covers all Angular topics, from pipes to directives, and comes in handy for a quick reference.
2. What Are Components in Angular? Angular components are the most fundamental building blocks of any Angular application.
3. What Is a Question Mark in Angular? An Angular question mark refers to a “Safe Navigation operator.” It makes sure that there are no null or undefined values in an Angular application when we try to access an object’s properties.
4. What Is Ng in Angular? In Angular, ‘ng’ is a prefix for all the builtin directives that ship with Angular.

Explore More By Sameeksha Medewar Sameeksha is a freelance content writer for more than half and a year. She has a hunger to explore and learn new things. She possesses a bachelor's degree in Computer Science. View all post by the author

[Learn More](#) [7 Best Angular Courses Online You Should Signup in 2025](#) [JavaScript Courses](#) [Web Development Frameworks](#) [Angular JavaScript Frameworks](#) [JavaScript Libraries](#) [Cool, Fun & Easy Angular Projects for Beginners \(with Source Code\)](#) [JavaScript Web Development Frameworks](#) [How to Learn Angular? - A Comprehensive Guide](#) [JavaScript Web Development Frameworks](#) [JavaScript Frameworks](#) [JavaScript Libraries](#) [Angular search...](#)

[SHOW ALL CATALOG](#) [Courses](#) [Projects](#) [Blog](#) [User Resources](#) [RESOURCES](#) [Projects](#) [Blog](#) [Cheat Sheets](#) [User Tutorials](#) [Python Editor](#) [HTML Editor](#) [JavaScript Editor](#) [Mentor](#) [AI Interviewer](#) [PRICING](#) [Plans For Students](#) [ACCOUNT](#) [Dashboard](#) [Premium For Students](#) [COMPANY](#) [About Us](#) [Contact Us](#) [Advertise / Partner](#) [SUPPORT](#) [Help Center](#) [Refund Policy](#) [Privacy Policy](#) [Cookie Policy](#) [Terms & Conditions](#) [Disclosure](#) [Disclaimer](#) © 2025 Hackr.io, All rights reserved.