

ANGULAR CHEAT SHEET

A quick guide to Angular syntax. (Content is provisional and may change.)

Angular for TypeScript Cheat Sheet (v2.1.0)

Bootstrapping	<pre>import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';</pre>
<pre>platformBrowserDynamic().bootstrapModule(AppModule);</pre>	Bootstraps the app, using the root component from the specified <code>NgModule</code> .
NgModules	<pre>import { NgModule } from '@angular/core';</pre>
<pre>@NgModule({ declarations: ..., imports: ..., exports: ..., providers: ..., bootstrap: ... }) class MyModule {}</pre>	Defines a module that contains components, directives, pipes, and providers.

<code>declarations: [MyRedComponent, MyBlueComponent, MyDatePipe]</code>	List of components, directives, and pipes that belong to this module.
<code>imports: [BrowserModule, SomeOtherModule]</code>	List of modules to import into this module. Everything from the imported modules is available to <code>declarations</code> of this module.
<code>exports: [MyRedComponent, MyDatePipe]</code>	List of components, directives, and pipes visible to modules that import this module.
<code>providers: [MyService, { provide: ... }]</code>	List of dependency injection providers visible both to the contents of this module and to importers of this module.
<code>bootstrap: [MyAppComponent]</code>	List of components to bootstrap when this module is bootstrapped.

Template syntax	
<code><input [value]="firstName"></code>	Binds property <code>value</code> to the result of expression <code>firstName</code> .
<code><div [attr.role]="myAriaRole"></code>	Binds attribute <code>role</code> to the result of expression <code>myAriaRole</code> .
<code><div [class.extra-sparkle]="isDelightful"></code>	Binds the presence of the CSS class <code>extra-sparkle</code> on the element to the truthiness of the expression <code>isDelightful</code> .
<code><div [style.width.px]="mySize"></code>	Binds style property <code>width</code> to the result of expression <code>mySize</code> in pixels. Units are optional.
<code><button (click)="readRainbow(\$event)"></code>	Calls method <code>readRainbow</code> when a click event is triggered on this button element (or its children) and passes in the event object.

<code><div title="Hello {{ponyName}}"></code>	Binds a property to an interpolated string, for example, "Hello Seabiscuit". Equivalent to: <code><div [title]='Hello ' + ponyName"></code>
<code><p>Hello {{ponyName}}</p></code>	Binds text content to an interpolated string, for example, "Hello Seabiscuit".
<code><my-cmp [(title)]="name"></code>	Sets up two-way data binding. Equivalent to: <code><my-cmp [title]="name" (titleChange)="name=\$event"></code>
<code><video #movieplayer ...> <button (click)="movieplayer.play()"> </video></code>	Creates a local variable <code>movieplayer</code> that provides access to the <code>video</code> element instance in data-binding and event-binding expressions in the current template.
<code><p *myUnless="myExpression">...</p></code>	The <code>*</code> symbol turns the current element into an embedded template. Equivalent to: <code><template [myUnless]="myExpression"><p>...</p> </template></code>
<code><p>Card No.: {{cardNumber myCardNumberFormatter}}</p></code>	Transforms the current value of expression <code>cardNumber</code> via the pipe called <code>myCardNumberFormatter</code> .
<code><p>Employer: {{employer?.companyName}}</p></code>	The safe navigation operator (<code>?</code>) means that the <code>employer</code> field is optional and if <code>undefined</code> , the rest of the expression should be ignored.
<code><svg:rect x="0" y="0" width="100" height="100"/></code>	An SVG snippet template needs an <code>svg:</code> prefix on its root element to disambiguate the SVG element from an HTML component.
<code><svg> <rect x="0" y="0" width="100" height="100"/> </svg></code>	An <code><svg></code> root element is detected as an SVG element automatically, without the prefix.

Built-in directives	import { CommonModule } from '@angular/common';
<section *ngIf="showSection">	Removes or recreates a portion of the DOM tree based on the showSection expression.
<li *ngFor="let item of list">	Turns the li element and its contents into a template, and uses that to instantiate a view for each item in list.
<pre><div [ngSwitch]="conditionExpression"> <template [ngSwitchCase]="case1Exp">...</template> <template ngSwitchCase="case2LiteralString">... </template> <template ngSwitchDefault>...</template> </div></pre>	Conditionally swaps the contents of the div by selecting one of the embedded templates based on the current value of conditionExpression .
<pre><div [ngClass]=" {active: isActive, disabled: isDisabled}"></pre>	Binds the presence of CSS classes on the element to the truthiness of the associated map values. The right-hand expression should return {class-name: true/false} map.

Forms	import { FormsModule } from '@angular/forms';
<input [(ngModel)]="userName">	Provides two-way data-binding, parsing, and validation for form controls.

Class decorators	import { Directive, ... } from '@angular/core';

<pre>@Component({...}) class MyComponent() {}</pre>	Declares that a class is a component and provides metadata about the component.
<pre>@Directive({...}) class MyDirective() {}</pre>	Declares that a class is a directive and provides metadata about the directive.
<pre>@Pipe({...}) class MyPipe() {}</pre>	Declares that a class is a pipe and provides metadata about the pipe.
<pre>@Injectable() class MyService() {}</pre>	Declares that a class has dependencies that should be injected into the constructor when the dependency injector is creating an instance of this class.

Directive configuration	@Directive({ property1: value1, ... })
<pre>selector: '.cool-button:not(a)'</pre>	<p>Specifies a CSS selector that identifies this directive within a template. Supported selectors include <code>element</code>, <code>[attribute]</code>, <code>.class</code>, and <code>:not()</code>.</p> <p>Does not support parent-child relationship selectors.</p>
<pre>providers: [MyService, { provide: ... }]</pre>	List of dependency injection providers for this directive and its children.

Component configuration	@Component extends @Directive, so the @Directive configuration applies to components as well
<pre>moduleId: module.id</pre>	If set, the <code>templateUrl</code> and <code>styleUrl</code> are resolved relative to the component.

<code>viewProviders: [MyService, { provide: ... }]</code>	List of dependency injection providers scoped to this component's view.
<code>template: 'Hello {{name}}'</code> <code>templateUrl: 'my-component.html'</code>	Inline template or external template URL of the component's view.
<code>styles: ['.primary {color: red}']</code> <code>styleUrls: ['my-component.css']</code>	List of inline CSS styles or external stylesheet URLs for styling the component's view.

Class field decorators for directives and components	<code>import { Input, ... } from '@angular/core';</code>
<code>@Input() myProperty;</code>	Declares an input property that you can update via property binding (example: <code><my-cmp [myProperty]="someExpression"></code>).
<code>@Output() myEvent = new EventEmitter();</code>	Declares an output property that fires events that you can subscribe to with an event binding (example: <code><my-cmp (myEvent)="doSomething()"></code>).
<code>@HostBinding('[class.valid]') isValid;</code>	Binds a host element property (here, the CSS class <code>valid</code>) to a directive/component property (<code>isValid</code>).
<code>@HostListener('click', ['\$event']) onClick(e) {...}</code>	Subscribes to a host element event (<code>click</code>) with a directive/component method (<code>onClick</code>), optionally passing an argument (<code>\$event</code>).
<code>@ContentChild(myPredicate) myChildComponent;</code>	Binds the first result of the component content query (<code>myPredicate</code>) to a property (<code>myChildComponent</code>) of the class.
<code>@ContentChildren(myPredicate) myChildComponents;</code>	Binds the results of the component content query (<code>myPredicate</code>) to a property (<code>myChildComponents</code>) of the class.

<code>@ViewChild(myPredicate) myChildComponent;</code>	Binds the first result of the component view query (<code>myPredicate</code>) to a property (<code>myChildComponent</code>) of the class. Not available for directives.
<code>@ViewChildren(myPredicate) myChildComponents;</code>	Binds the results of the component view query (<code>myPredicate</code>) to a property (<code>myChildComponents</code>) of the class. Not available for directives.

Directive and component change detection and lifecycle hooks	(implemented as class methods)
<code>constructor(myService: MyService, ...) { ... }</code>	Called before any other lifecycle hook. Use it to inject dependencies, but avoid any serious work here.
<code>ngOnChanges(changeRecord) { ... }</code>	Called after every change to input properties and before processing content or child views.
<code>ngOnInit() { ... }</code>	Called after the constructor, initializing input properties, and the first call to <code>ngOnChanges</code> .
<code>ngDoCheck() { ... }</code>	Called every time that the input properties of a component or a directive are checked. Use it to extend change detection by performing a custom check.
<code>ngAfterContentInit() { ... }</code>	Called after <code>ngOnInit</code> when the component's or directive's content has been initialized.
<code>ngAfterContentChecked() { ... }</code>	Called after every check of the component's or directive's content.
<code>ngAfterViewInit() { ... }</code>	

	Called after <code>ngAfterContentInit</code> when the component's view has been initialized. Applies to components only.
<code>ngAfterViewChecked() { ... }</code>	Called after every check of the component's view. Applies to components only.
<code>ngOnDestroy() { ... }</code>	Called once, before the instance is destroyed.

Dependency injection configuration	
<code>{ provide: MyService, useClass: MyMockService }</code>	Sets or overrides the provider for <code>MyService</code> to the <code>MyMockService</code> class.
<code>{ provide: MyService, useFactory: myFactory }</code>	Sets or overrides the provider for <code>MyService</code> to the <code>myFactory</code> factory function.
<code>{ provide: MyValue, useValue: 41 }</code>	Sets or overrides the provider for <code>MyValue</code> to the value <code>41</code> .

Routing and navigation	
	<pre>import { Routes, RouterModule, ... } from '@angular/router';</pre> <p>Configures routes for the application. Supports static, parameterized, redirect, and wildcard routes. Also supports custom route data and resolve.</p>


```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'path/:routeParam', component: MyComponent },
  { path: 'staticPath', component: ... },
  { path: '**', component: ... },
  { path: 'oldPath', redirectTo: '/staticPath' },
  { path: ..., component: ..., data: { message: 'Custom' } }
]);
```

```
const routing = RouterModule.forRoot(routes);
```

```
<router-outlet></router-outlet>
<router-outlet name="aux"></router-outlet>
```

Marks the location to load the component of the active route.

```
<a routerLink="/path">
<a [routerLink]="[ '/path', routeParam ]">
<a [routerLink]="[ '/path', { matrixParam: 'value' } ]">
<a [routerLink]="[ '/path' ]" [queryParams]="{ page: 1 }">
<a [routerLink]="[ '/path' ]" fragment="anchor">
```

Creates a link to a different view based on a route instruction consisting of a route path, required and optional parameters, query parameters, and a fragment. To navigate to a root route, use the / prefix; for a child route, use the ./ prefix; for a sibling or parent, use the ../ prefix.

```
<a [routerLink]="[ '/path' ]" routerLinkActive="active">
```

The provided classes are added to the element when the routerLink becomes the current active route.

An interface for defining a class that the router should call first to determine if it should activate this component. Should return a boolean or an Observable/Promise that resolves to a boolean.

```

class CanActivateGuard implements CanActivate {
    canActivate(
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot
    ): Observable<boolean>|Promise<boolean>|boolean { ... }
}

{ path: ..., canActivate: [CanActivateGuard] }

```

```

class CanDeactivateGuard implements CanDeactivate<T> {
    canDeactivate(
        component: T,
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot
    ): Observable<boolean>|Promise<boolean>|boolean { ... }
}

{ path: ..., canDeactivate: [CanDeactivateGuard] }

```

An interface for defining a class that the router should call first to determine if it should deactivate this component after a navigation. Should return a boolean or an Observable/Promise that resolves to a boolean.

```

class CanActivateChildGuard implements CanActivateChild {
    canActivateChild(
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot
    ): Observable<boolean>|Promise<boolean>|boolean { ... }
}

{ path: ..., canActivateChild: [CanActivateGuard],
  children: ... }

```

An interface for defining a class that the router should call first to determine if it should activate the child route. Should return a boolean or an Observable/Promise that resolves to a boolean.

```
class ResolveGuard implements Resolve<T> {  
    resolve(  
        route: ActivatedRouteSnapshot,  
        state: RouterStateSnapshot  
    ): Observable<any>|Promise<any>|any { ... }  
}
```

```
{ path: ..., resolve: [ResolveGuard] }
```

An interface for defining a class that the router should call first to resolve route data before rendering the route. Should return a value or an Observable/Promise that resolves to a value.

```
class CanLoadGuard implements CanLoad {  
    canLoad(  
        route: Route  
    ): Observable<boolean>|Promise<boolean>|boolean { ... }  
}
```

```
{ path: ..., canLoad: [CanLoadGuard], loadChildren: ... }
```

An interface for defining a class that the router should call first to check if the lazy loaded module should be loaded. Should return a boolean or an Observable/Promise that resolves to a boolean.