**Cheat Sheet**

|  |  |
| --- | --- |
| **Bootstrapping** | **import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';** |
| **platformBrowserDynamic().bootstrapModule**(AppModule); | Bootstraps the app, using the root component from the specified NgModule. |

|  |  |
| --- | --- |
| **NgModules** | **import { NgModule } from '@angular/core';** |
| @**NgModule**({ declarations: ..., imports: ...,exports: ..., providers: ..., bootstrap: ...})class MyModule {} | Defines a module that contains components, directives, pipes, and providers. |
| **declarations:** [MyRedComponent, MyBlueComponent, MyDatePipe] | List of components, directives, and pipes that belong to this module. |
| **imports:** [BrowserModule, SomeOtherModule] | List of modules to import into this module. Everything from the imported modules is available to declarations of this module. |
| **exports:** [MyRedComponent, MyDatePipe] | List of components, directives, and pipes visible to modules that import this module. |
| **providers:** [MyService, { provide: ... }] | List of dependency injection providers visible both to the contents of this module and to importers of this module. |
| **entryComponents:** [SomeComponent, OtherComponent] | List of components not referenced in any reachable template, for example dynamically created from code. |
| **bootstrap:** [MyAppComponent] | List of components to bootstrap when this module is bootstrapped. |

|  |  |
| --- | --- |
| **Template syntax** |  |
| <input **[value]**="firstName"> | Binds property value to the result of expression firstName. |
| <div **[attr.role]**="myAriaRole"> | Binds attribute role to the result of expression myAriaRole. |
| <div **[class.extra-sparkle]**="isDelightful"> | Binds the presence of the CSS class extra-sparkle on the element to the truthiness of the expression isDelightful. |
| <div **[style.width.px]**="mySize"> | Binds style property width to the result of expression mySize in pixels. Units are optional. |
| <button**(click)**="readRainbow($event)"> | Calls method readRainbow when a click event is triggered on this button element (or its children) and passes in the event object. |
| <div title="Hello **{{ponyName}}**"> | Binds a property to an interpolated string, for example, "Hello Seabiscuit". Equivalent to: <div [title]="'Hello ' + ponyName"> |
| <p>Hello **{{ponyName}}**</p> | Binds text content to an interpolated string, for example, "Hello Seabiscuit". |
| <my-cmp **[(title)]**="name"> | Sets up two-way data binding. Equivalent to: <my-cmp [title]="name" (titleChange)="name=$event"> |
| <video **#movieplayer** ...> <button**(click)**="movieplayer.play()"></video> | Creates a local variable movieplayer that provides access to the videoelement instance in data-binding and event-binding expressions in the current template. |
| <p **\*myUnless**="myExpression">...</p> | The \* symbol turns the current element into an embedded template. Equivalent to: <ng-template [myUnless]="myExpression"><p>...</p></ng-template> |
| <p>Card No.: **{{cardNumber | myCardNumberFormatter}}**</p> | Transforms the current value of expression cardNumber via the pipe called myCardNumberFormatter. |
| <p>Employer:**{{employer?.companyName}}**</p> | The safe navigation operator (?) means that the employer field is optional and if undefined, the rest of the expression should be ignored. |
| <**svg:**rect x="0" y="0" width="100" height="100"/> | An SVG snippet template needs an svg: prefix on its root element to disambiguate the SVG element from an HTML component. |
| <**svg**> <rect x="0" y="0" width="100" height="100"/></**svg**> | An <svg> 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. |
| <div **[ngSwitch]**="conditionExpression"> <ng-template**[ngSwitchCase]**="case1Exp">...</ng-template> <ng-template**ngSwitchCase**="case2LiteralString">...</ng-template> <ng-template **ngSwitchDefault**>...</ng-template></div> | Conditionally swaps the contents of the div by selecting one of the embedded templates based on the current value of conditionExpression. |
| <div **[ngClass]**="{'active': isActive, 'disabled': isDisabled}"> | 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. |
| <div **[ngStyle]**="{'property': 'value'}"> <div **[ngStyle]**="dynamicStyles()"> | Allows you to assign styles to an HTML element using CSS. You can use CSS directly, as in the first example, or you can call a method from the component. |

|  |  |
| --- | --- |
| **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';** |
| **@Component({...})**class MyComponent() {} | Declares that a class is a component and provides metadata about the component. |
| **@Directive({...})**class MyDirective() {} | Declares that a class is a directive and provides metadata about the directive. |
| **@Pipe({...})**class MyPipe() {} | Declares that a class is a pipe and provides metadata about the pipe. |
| **@Injectable()**class MyService() {} | 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, ... })** |
| **selector:** '.cool-button:not(a)' | Specifies a CSS selector that identifies this directive within a template. Supported selectors include element, [attribute], .class, and :not().  Does not support parent-child relationship selectors. |
| **providers:** [MyService, { provide: ... }] | 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** |
| **moduleId:** module.id | If set, the templateUrl and styleUrl are resolved relative to the component. |
| **viewProviders:** [MyService, { provide: ... }] | List of dependency injection providers scoped to this component's view. |
| **template:** 'Hello {{name}}'**templateUrl:** 'my-component.html' | Inline template or external template URL of the component's view. |
| **styles:** ['.primary {color: red}']**styleUrls:**['my-component.css'] | List of inline CSS styles or external stylesheet URLs for styling the component’s view. |

|  |  |
| --- | --- |
| **Class field decorators for directives and components** | **import { Input, ... } from '@angular/core';** |
| **@Input()** myProperty; | Declares an input property that you can update via property binding (example:<my-cmp [myProperty]="someExpression">). |
| **@Output()** myEvent = new EventEmitter(); | Declares an output property that fires events that you can subscribe to with an event binding (example: <my-cmp (myEvent)="doSomething()">). |
| **@HostBinding('class.valid')**isValid; | Binds a host element property (here, the CSS class valid) to a directive/component property (isValid). |
| **@HostListener('click', ['$event'])** onClick(e) {...} | Subscribes to a host element event (click) with a directive/component method (onClick), optionally passing an argument ($event). |
| **@ContentChild(myPredicate)**myChildComponent; | Binds the first result of the component content query (myPredicate) to a property (myChildComponent) of the class. |
| **@ContentChildren(myPredicate)**myChildComponents; | Binds the results of the component content query (myPredicate) to a property (myChildComponents) of the class. |
| **@ViewChild(myPredicate)**myChildComponent; | Binds the first result of the component view query (myPredicate) to a property (myChildComponent) of the class. Not available for directives. |
| **@ViewChildren(myPredicate)**myChildComponents; | Binds the results of the component view query (myPredicate) to a property (myChildComponents) of the class. Not available for directives. |

|  |  |
| --- | --- |
| **Directive and component change detection and lifecycle hooks** | **(implemented as class methods)** |
| **constructor(myService: MyService, ...)** { ... } | Called before any other lifecycle hook. Use it to inject dependencies, but avoid any serious work here. |
| **ngOnChanges(changeRecord)** { ... } | Called after every change to input properties and before processing content or child views. |
| **ngOnInit()** { ... } | Called after the constructor, initializing input properties, and the first call to ngOnChanges. |
| **ngDoCheck()** { ... } | 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. |
| **ngAfterContentInit()** { ... } | Called after ngOnInit when the component's or directive's content has been initialized. |
| **ngAfterContentChecked()** { ... } | Called after every check of the component's or directive's content. |
| **ngAfterViewInit()** { ... } | Called after ngAfterContentInit when the component's views and child views / the view that a directive is in has been initialized. |
| **ngAfterViewChecked()** { ... } | Called after every check of the component's views and child views / the view that a directive is in. |
| **ngOnDestroy()** { ... } | Called once, before the instance is destroyed. |

|  |  |
| --- | --- |
| **Dependency injection configuration** |  |
| { **provide**: MyService, **useClass**: MyMockService } | Sets or overrides the provider for MyService to the MyMockServiceclass. |
| { **provide**: MyService, **useFactory**: myFactory } | Sets or overrides the provider for MyService to the myFactory factory function. |
| { **provide**: MyValue, **useValue**: 41 } | Sets or overrides the provider for MyValue to the value 41. |

|  |  |
| --- | --- |
| **Routing and navigation** | **import { Routes, RouterModule, ... } from '@angular/router';** |
| 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); | Configures routes for the application. Supports static, parameterized, redirect, and wildcard routes. Also supports custom route data and resolve. |
| <**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. |
| class **CanActivate**Guard implements **CanActivate** { canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot ): Observable<boolean>|Promise<boolean>|boolean { ... }}{ path: ..., canActivate: [**CanActivate**Guard] } | 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 **CanDeactivate**Guard implements **CanDeactivate**<T> {canDeactivate( component: T, route: ActivatedRouteSnapshot,state: RouterStateSnapshot ): Observable<boolean>|Promise<boolean>|boolean { ... }}{ path: ..., canDeactivate: [**CanDeactivate**Guard] } | 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 **CanActivateChild**Guard 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 **Resolve**Guard implements **Resolve**<T> { resolve( route: ActivatedRouteSnapshot, state: RouterStateSnapshot ): Observable<any>|Promise<any>|any { ... }}{ path: ..., resolve: [**Resolve**Guard] } | 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 **CanLoad**Guard implements **CanLoad** { canLoad( route: Route): Observable<boolean>|Promise<boolean>|boolean { ... }}{ path: ..., canLoad: [**CanLoad**Guard], 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. |