VAR IS GLOBAL

VAR X=5;

X=6;

POSSIBLE

LET IS SCOPE,FUNCTIONAL,ETC

LET X=5;

X=6;

NOT POSSIBLE

CONST CAN NEVER BE MODIFIED

BUT IF ARRAY ITS ELEMENT CAN BE MODIFIED

A template **expression** produces a value. Angular executes the expression and assigns it to a property of a binding target; the target might be an HTML element, a component, or a directive.

The interpolation braces in {{1 + 1}} surround the template expression 1 + 1. In the [property binding](https://angular.io/guide/template-syntax#property-binding) section below, a template expression appears in quotes to the right of the = symbol as in [property]="expression".

An expression may also refer to properties of the *template's* context such as a [template input variable](https://angular.io/guide/template-syntax#template-input-variable) (let hero) or a [template reference variable](https://angular.io/guide/template-syntax#ref-vars) (#heroInput).

src/app/app.component.html

<div \*[ngFor](https://angular.io/api/common/NgForOf)="let hero of heroes">{{hero.name}}</div> <input #heroInput> {{heroInput.value}}

If you reference a name that belongs to more than one of these namespaces, the template variable name takes precedence, followed by a name in the directive's context, and, lastly, the component's member names

A template **statement** responds to an **event** raised by a binding target such as an element, component, or directive. You'll see template statements in the [event binding](https://angular.io/guide/template-syntax#event-binding) section, appearing in quotes to the right of the = symbol as in (event)="statement".supports both basic assignment (=) and chaining expressions (with ; or ,).

The statement context may also refer to properties of the template's own context. In the following examples, the template $event object, a [template input variable](https://angular.io/guide/template-syntax#template-input-variable) (let hero), and a [template reference variable](https://angular.io/guide/template-syntax#ref-vars) (#heroForm) are passed to an event handling method of the component.

src/app/app.component.html

content\_copy<button (click)="onSave($event)">Save</button>

<button \*[ngFor](https://angular.io/api/common/NgForOf)="let hero of heroes" (click)="deleteHero(hero)">{{hero.name}}</button>

<form #heroForm (ngSubmit)="onSubmit(heroForm)"> ... </form>

Template context names take precedence over component context names. In deleteHero(hero) above, the hero is the template input variable, not the component's hero property.

**Attributes**initialize**DOM properties and then they are done. Property values can change; attribute values can't.**

The HTML attribute value specifies the initial value; the DOM value property is the current value.

This fact bears repeating: **Template binding works with**properties**and**events**, not**attributes**.**

<button disabled="false">Still Disabled</button>. Disabled as attribute

<button [disabled]="isUnchanged">Save</button> as property

Element properties may be the more common targets, but Angular looks first to see if the name is a property of a known directive.

<app-hero-detail [hero]="currentHero"></app-hero-detail>

If hero expected then supply hero object only…..

The brackets tell Angular to evaluate the template expression. If you omit the brackets, Angular treats the string as a constant and initializes the target property with that string. It does not evaluate the string!

Prefer this:

The following example initializes the prefix property of the HeroDetailComponent to a fixed string, not a template expression. Angular sets it and forgets about it.

<app-hero-detail prefix="You are my" [hero]="currentHero"></app-hero-detail>

**will not** allow HTML with script tags to leak into the browser, neither with interpolation nor property binding.

Consider the [ARIA](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA), [SVG](https://developer.mozilla.org/en-US/docs/Web/SVG), and table span attributes.

Attribute binding syntax resembles property binding. Instead of an element property between brackets, start with the prefix attr, followed by a dot (.) and the name of the attribute. You then set the attribute value, using an expression that resolves to a string.

Bind [attr.colspan] to a calculated value:

src/app/app.component.html

content\_copy<table border=1>

<!-- expression calculates colspan=2 -->

<tr><td [attr.colspan]="1 + 1">One-Two</td></tr>

You can add and remove CSS class names from an element's class attribute with a **class binding**.

Class binding syntax resembles property binding. Instead of an element property between brackets, start with the prefix class, optionally followed by a dot (.) and the name of a CSS class: [class.class-name].

<div class="special" [class.special]="!isSpecial">This one is not so special</div>

Here special class is applied to div at frst but based on isSpecial is again removed or added

start with the prefix [style](https://angular.io/api/animations/style), followed by a dot (.) and the name of a CSS style property: [style.style-property].

<button [style.background-color]="canSave ? 'cyan': 'grey'" >Save</button>

<button [style.font-size.%]="!isSpecial ? 150 : 50" >Small</button

Note that a style property name can be written in either [dash-case](https://angular.io/guide/glossary#dash-case), as shown above, or [camelCase](https://angular.io/guide/glossary#camelcase), such as fontSize.

Doubt::

<input [value]="currentHero.name" (input)="currentHero.name=$event.target.value" >

<app-hero-detail (deleteRequest)="deleteHero($event)" [hero]="currentHero"></app-hero-detail>

::

Javascript:

const squaredIntegers = arr.filter((x)=>(x)>0&&x%1==0).map(x=>x\*x);

However, the spread operator only works in-place, like in an argument to a function;

const sum = (function() {

"use strict";

return function sum(...args) {

args as array is obtained args=[1,2,3]

return args.reduce((a, b) => a + b, 0);

};

})();

console.log(sum(1, 2, 3)); // 6

or in an array literal:

var parts = ['shoulders', 'knees'];

var lyrics = ['head', ...parts, 'and', 'toes'];

Here's the same assignment statement with ES6 destructuring syntax:

const { x, y, z } = voxel; // x = 3.6, y = 7.4, z = 6.54

If instead you want to store the values of voxel.xinto a, voxel.yinto b, and voxel.zinto c, you have that freedom as well.

const { x : a, y : b, z : c } = voxel // a = 3.6, b = 7.4, c = 6.54

if not declared than…

({x:a,y:}=voxel)

<app-sizer [(size)]="fontSizePx"></app-sizer>

currentClasses: {};

setCurrentClasses() {

// CSS [classes](https://angular.io/api/core/DebugElement#classes): added/removed per current [state](https://angular.io/api/animations/state) of component [properties](https://angular.io/api/core/DebugElement#properties)

this.currentClasses = {

'saveable': this.canSave,

'modified': !this.isUnchanged,

'special': this.isSpecial

};

}

Adding an [ngClass](https://angular.io/api/common/NgClass) property binding to currentClasses sets the element's classes accordingly:

src/app/app.component.html

content\_copy<div [[ngClass](https://angular.io/api/common/NgClass)]="currentClasses">This div is initially saveable, unchanged, and special</div>

currentStyles: {};

setCurrentStyles() {

// CSS styles: set per current [state](https://angular.io/api/animations/state) of component [properties](https://angular.io/api/core/DebugElement#properties)

this.currentStyles = {

'font-style': this.canSave ? 'italic' : 'normal',

'font-weight': !this.isUnchanged ? 'bold' : 'normal',

'font-size': this.isSpecial ? '24px' : '12px'

};

}

Adding an [ngStyle](https://angular.io/api/common/NgStyle) property binding to currentStyles sets the element's styles accordingly:

src/app/app.component.html

content\_copy<div [[ngStyle](https://angular.io/api/common/NgStyle)]="currentStyles">

This div is initially italic, normal weight, and extra large (24px).

</div>

The <input> box is one of those elements. Angular provides value accessors for all of the basic HTML form elements and the [Forms](https://angular.io/guide/forms)guide shows how to bind t

You don't need a value accessor for an Angular component that you write because you can name the value and event properties to suit Angular's basic [two-way binding syntax](https://angular.io/guide/template-syntax#two-way) and skip [NgModel](https://angular.io/api/forms/NgModel) altogether.  o them.

<input [[ngModel](https://angular.io/api/forms/NgModel)]="currentHero.name" (ngModelChange)="currentHero.name=$event">

If two way data binding is used:

<input [([ngModel](https://angular.io/api/forms/NgModel))]="currentHero.name">

If additional reform is needed while giving from html to .ts

<input [[ngModel](https://angular.io/api/forms/NgModel)]="currentHero.name" (ngModelChange)="setUppercaseName($event)">

Completely add/remove component:

<app-hero-detail \*[ngIf](https://angular.io/api/common/NgIf)="[isActive](https://angular.io/api/router/RouterLinkActive#isActive)"></app-hero-detail>

Is isactive null never displayed…

Hide/show:

<app-hero-detail [class.hidden]="isSpecial"></app-hero-detail>

Ngfor:

<div \*[ngFor](https://angular.io/api/common/NgForOf)="let hero of heroes">{{hero.name}}</div>

<app-hero-detail \*[ngFor](https://angular.io/api/common/NgForOf)="let hero of heroes" [hero]="hero"></app-hero-detail>

**Trackby:it will help to update dom only when hero id changes….**

trackByHeroes(index: number, hero: Hero): number { return hero.id; }

<div \*[ngFor](https://angular.io/api/common/NgForOf)="let hero of heroes; trackBy: trackByHeroes"> ({{hero.id}}) {{hero.name}} </div>

Ngswitch:

<div [[ngSwitch](https://angular.io/api/common/NgSwitch)]="currentHero.emotion"> <app-happy-hero \*[ngSwitchCase](https://angular.io/api/common/NgSwitchCase)="'happy'" [hero]="currentHero"></app-happy-hero> <app-sad-hero \*[ngSwitchCase](https://angular.io/api/common/NgSwitchCase)="'sad'" [hero]="currentHero"></app-sad-hero> <app-confused-hero \*[ngSwitchCase](https://angular.io/api/common/NgSwitchCase)="'confused'" [hero]="currentHero"></app-confused-hero> <app-unknown-hero \*[ngSwitchDefault](https://angular.io/api/common/NgSwitchDefault) [hero]="currentHero"></app-unknown-hero> </div>

For elements:

<div \*[ngSwitchCase](https://angular.io/api/common/NgSwitchCase)="'confused'">Are you as confused as {{currentHero.name}}?</div>

template reference variable:

<input #phone placeholder="phone number"> <!-- lots of other elements --> <!-- phone refers to the input element; pass its `value` to an event handler --> <button (click)="callPhone(phone.value)">Call</button>

In context of form:

<form (ngSubmit)="onSubmit(heroForm)" #heroForm="[ngForm](https://angular.io/api/forms/NgForm)"> <div class="form-group"> <label for="name">Name <input class="form-control" name="name" required [([ngModel](https://angular.io/api/forms/NgModel))]="hero.name"> </label> </div> <button type="submit" [disabled]="!heroForm.form.valid">Submit</button> </form> <div [hidden]="!heroForm.form.valid"> {{submitMessage}} </div>

 The #heroForm is actually a reference to an Angular [NgForm](https://angular.io/api/forms/NgForm) directive with the ability to track the value and validity of every control in the form. The scope of a reference variable is the entire template. Do not define the same variable name more than once in the same template

Input and output:

In app..component.ts:

<app-hero-detail [hero]="currentHero" (deleteRequest)="deleteHero($event)"> </app-hero-detail>

Hero is inpu bcoz currentherro is supplied to hero-detail from app.component.ts

deleteRequest is output coz event is fired from herodetail to app.component.ts

In hero-detail:

@[Input](https://angular.io/api/core/Input)() hero: Hero; @[Output](https://angular.io/api/core/Output)() deleteRequest = new [EventEmitter](https://angular.io/api/core/EventEmitter)<Hero>();

If want to give alisas name that can be used in component:

@[Output](https://angular.io/api/core/Output)('myClick') clicks = new [EventEmitter](https://angular.io/api/core/EventEmitter)<string>();

Note:

<div>Birthdate: {{currentHero?.birthdate | date:'longDate'}}</div>

?. if not present and currenthero is null then component will not render,and if present and is null then this div is not siaplayed nut app runs.. It works perfectly with long property paths such as [a](https://angular.io/api/router/RouterLinkWithHref)?.b?.c?.d.

|:pipe

: for supplying parameters to pipe…

The Angular **non-null assertion operator (**!**)** serves the same purpose in an Angular template.

For example, after you use [\*ngIf](https://angular.io/guide/template-syntax#ngIf) to check that hero is defined, you can assert that hero properties are also defined.

src/app/app.component.html

content\_copy<!--No hero, no text -->

<div \*[ngIf](https://angular.io/api/common/NgIf)="hero">

The hero's name is {{hero!.name}}

</div>

ometimes a binding expression will be reported as a type error and it is not possible or difficult to fully specify the type. To silence the error, you can use the $any cast function to cast the expression to [the any type](http://www.typescriptlang.org/docs/handbook/basic-types.html#any).

src/app/app.component.html

content\_copy<!-- Accessing an undeclared member -->

<div>

The hero's marker is {{$any(hero).marker}}

</div>

The $any cast function can be used in conjunction with this to allow access to undeclared members of the component.

src/app/app.component.html

content\_copy<!-- Accessing an undeclared member -->

<div>

Undeclared members is {{$any(this).member}}

</div>

You must **add a**[<base href> element](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/base) to the app's index.html for pushState routing to work. The browser uses the <base [href](https://angular.io/api/router/RouterLinkWithHref#href)> value to prefix relative URLs when referencing CSS files, scripts, and images.

You teach the router how to navigate by configuring it with routes.

* When the browser's location URL changes to match the path segment /crisis-center, then the router activates an instance of the CrisisListComponent and displays its view.
* When the application requests navigation to the path /crisis-center, the router activates an instance of CrisisListComponent, displays its view, and updates the browser's address location and history with the URL for that path.

The [RouterOutlet](https://angular.io/api/router/RouterOutlet) is a directive from the router library that marks the spot in the template where the router should display the views for that outlet.

. Query string parameters are provided through the [queryParams] binding which takes an object (e.g. { name: 'value' }), while the URL fragment takes a single value bound to the [fragment] input binding.

[RouterLinkActive](https://angular.io/api/router/RouterLinkActive) directive to a string of classes such as [[routerLinkActive](https://angular.io/api/router/RouterLinkActive)]="'active fluffy'"

The [RouterLinkActive](https://angular.io/api/router/RouterLinkActive) directive toggles css classes for active [RouterLink](https://angular.io/api/router/RouterLink)s based on the current [RouterState](https://angular.io/api/router/RouterState). This cascades down through each level of the route tree, so parent and child router links can be active at the same time. To override this behavior, you can bind to the [[routerLinkActiveOptions](https://angular.io/api/router/RouterLinkActive#routerLinkActiveOptions)] input binding with the { exact: true }expression. By using { exact: true }, a given [RouterLink](https://angular.io/api/router/RouterLink) will only be active if its URL is an exact match to the current URL.

The router selects the route with a [first match wins](https://angular.io/guide/router#example-config) strategy. Wildcard routes are the least specific routes in the route configuration. Be sure it is the last route in the configuration.

{ path: '\*\*', component: PageNotFoundComponent }

Now when the user visits /sidekicks, or any other invalid URL, the browser displays "Page not found". The browser address bar continues to point to the invalid URL.

Order:

const appRoutes: [Routes](https://angular.io/api/router/Routes) = [ { path: 'crisis-center', component: CrisisListComponent }, { path: 'heroes', component: HeroListComponent }, { path: '', [redirectTo](https://angular.io/api/router/Route#redirectTo): '/heroes', [pathMatch](https://angular.io/api/router/Route#pathMatch): '[full](https://angular.io/api/core/Version#full)' }, { path: '\*\*', component: PageNotFoundComponent } ];

static then default then wildcard…

In this app, the router should select the route to the HeroListComponent only when the entire URL matches '', so set the [pathMatch](https://angular.io/api/router/Route#pathMatch) value to '[full](https://angular.io/api/core/Version#full)'.

*Routing module*

 update the app.module.ts file, first importing the newly created AppRoutingModulefrom app-routing.module.ts, then replacing RouterModule.forRoot in the [imports](https://angular.io/api/core/NgModule#imports) array with the AppRoutingModule.

Only call RouterModule.forRoot in the root AppRoutingModule (or the AppModule if that's where you register top level application routes). In any other module, you must call the RouterModule.forChild method to register additional routes.

Import the HeroesModule and add it to the [imports](https://angular.io/api/core/NgModule#imports) array in the @[NgModule](https://angular.io/api/core/NgModule) metadata of the AppModule.

### **Module import order matters**[**link**](https://angular.io/guide/router#module-import-order-matters)

[imports](https://angular.io/api/core/NgModule#imports): [ [BrowserModule](https://angular.io/api/platform-browser/BrowserModule), [FormsModule](https://angular.io/api/forms/FormsModule), HeroesModule, AppRoutingModule ],

if in approuting module default and wildcard routes are present via app.routing.module.ts…

import { switchMap } from 'rxjs/operators';

import { Component, OnInit, HostBinding } from '@angular/core';

import { Router, ActivatedRoute, ParamMap } from '@angular/router';

import { Observable } from 'rxjs';

constructor(

private route: ActivatedRoute,

private router: Router,

private service: HeroService

) {}

ngOnInit() {

this.hero$ = this.route.paramMap.pipe(

switchMap((params: ParamMap) =>

this.service.getHero(params.get('id')))

);

}

gotoHeroes(hero: Hero) {

let heroId = hero ? hero.id : null;

// Pass along the hero id if available

// so that the HeroList component can select that hero.

// Include a junk 'foo' property for fun.

this.router.navigate(['/heroes', { id: heroId, foo: 'foo' }]);

}

}

You might think to use the RxJS map operator. But the HeroService returns an Observable<Hero>. So you flatten the Observable with the switchMap operator instead.

The switchMap operator also cancels previous in-flight requests. If the user re-navigates to this route with a new idwhile the HeroService is still retrieving the old id, switchMap discards that old request and returns the hero for the new id.

#### **ParamMap API**

The [ParamMap](https://angular.io/api/router/ParamMap) API is inspired by the [URLSearchParams interface](https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams). It provides methods to handle parameter access for both route parameters (paramMap) and query parameters (queryParamMap).

|  |  |
| --- | --- |
| **Member** | **Description** |
| has(name) | Returns true if the parameter name is in the map of parameters. |
| get(name) | Returns the parameter name value (a string) if present, or null if the parameter name is not in the map. Returns the first element if the parameter value is actually an array of values. |
| getAll(name) | Returns a string array of the parameter name value if found, or an empty array if the parameter name value is not in the map. Use getAll when a single parameter could have multiple values. |
| [keys](https://angular.io/api/router/ParamMap#keys) | Returns a string array of all parameter names in the map. |

The [ActivatedRoute](https://angular.io/api/router/ActivatedRoute) and its observables are insulated from the [Router](https://angular.io/api/router/Router) itself. The [Router](https://angular.io/api/router/Router) destroys a routed component when it is no longer needed and the injected [ActivatedRoute](https://angular.io/api/router/ActivatedRoute) dies with it.

The route.snapshot provides the initial value of the route parameter map. You can access the parameters directly without subscribing or adding observable operators. It's much simpler to write and read:

src/app/heroes/hero-detail.component.ts (ngOnInit snapshot)

content\_copyngOnInit() {

let id = this.route.snapshot.paramMap.get('id');

this.hero$ = this.service.getHero(id);

}

Remember: you only get the *initial* value of the parameter map with this technique. Stick with the observable paramMap approach if there's even a chance that the router could re-use the component. This sample stays with the observable paramMap strategy just in case

The router supports navigation with optional parameters as well as required route parameters. Define optionalparameters in a separate object after you define the required route parameters.

gotoHeroes(hero: Hero) { let heroId = hero ? hero.id : null; // Pass along the hero id if [available](https://angular.io/api/service-worker/SwUpdate#available) // so that the HeroList component can select that hero. // Include [a](https://angular.io/api/router/RouterLinkWithHref) junk 'foo' property for fun. this.router.navigate(['/heroes', { id: heroId, foo: 'foo' }]); }

localhost:3000/heroes;id=15;foo=foo

The id value appears in the URL as (;id=15;foo=foo), not in the URL path. The path for the "Heroes" route doesn't have an :id token.

The optional route parameters are not separated by "?" and "&" as they would be in the URL query string. They are separated by semicolons ";" This is *matrix URL* notation

Rest use just paramap for retrieving value..even of optional parameter..