Angular 2

Templates

ngIf

The ngIf directive is used when you want to display or hide an element based on a condition. The condition is determined by the result of the expression that you pass in to the directive.

```
<div *ngIf="false"></div> <!-- never displayed -->
<div *ngIf="a > b"></div> <!-- displayed if a is more than b -->
<div *ngIf="str == 'yes'"></div> <!-- displayed if str holds the string "yes" →
```

Angular 2 offers no built-in alternative for **ng-show**. So, if your goal is to just change the CSS visibility of an element, you should look into either the ngStyle or the class directives.

```
<!-- isSpecial is true -->
<div [class.hidden]="!isSpecial">Show with class</div>
<div [class.hidden]="isSpecial">Hide with class</div>

<div [style.display]="isSpecial ? 'block' : 'none'">Show with style</div>
<div [style.display]="isSpecial ? 'none'  : 'block'">Hide with style</div>
```

When NgIf is false, Angular physically removes the element subtree from the DOM.

ngSwitch

Sometimes you need to render different elements depending on a given condition.

```
<div class="container" [ngSwitch]="myVar">
   <div *ngSwitchWhen="'A'">Var is A</div>
   <div *ngSwitchWhen="'B'">Var is B</div>
   <div *ngSwitchDefault>Var is something else</div>
</div>
```

ngSwitchDefault element is optional

ngStyle

With the ngStyle directive, you can set a given DOM element CSS properties from Angular expressions.

The simplest way to use this directive is by doing [style.<cssproperty>]="value":

```
<div [style.background-color]="'yellow'">
   Uses fixed yellow background
</div>
```

Another way to set fixed values is by using the ngStyle attribute:

```
<div [ngStyle]="{color: 'white', 'background-color': 'blue'}">
   Uses fixed white text on blue background
</div>
```

ngStyle: dynamic values

The real power of the `NgStyle` directive comes with using dynamic values.

```
<div class="ui input">
  <input type="text" name="color" value="{{color}}" #colorinput>
</div>
<div class="ui input">
  <input type="text" name="fontSize" value="{{fontSize}}" #fontinput>
</div>
```

We're setting the font size based on the input value:

```
<span [ngStyle]="{color: 'red'}" [style.font-size.px]="fontSize">
  red text
</span>

<span [ngStyle]="{color: colorinput.value}">
  {{ colorinput.value }} text
</span>
```

Otherwise we can use this:

```
<div [style.background-color]="colorinput.value" style="color: white;">
  {{ colorinput.value }} background
</div>
```

ngClass

`ngClass` directive, represented by a `ngClass` attribute in your HTML template, allows you to dynamically set and change the CSS classes for a given DOM element.

```
.bordered {
    border: 1px dashed black; background-color: #eee;
}

<div [ngClass]="{bordered: false}">This is never bordered</div>
<div [ngClass]="{bordered: true}">This is always bordered</div>

<div [ngClass]="{bordered: isBordered}">
    Using object literal. Border {{ isBordered ? "ON" : "OFF" }}
</div>
```

List of classes:

```
<div class="base" [ngClass]="['blue', 'round']">
    This will always have a blue background and round corners
</div>
```

ngFor

The role of this directive is to repeat a given DOM element (or a collection of DOM elements), each time passing it a different value from an array.

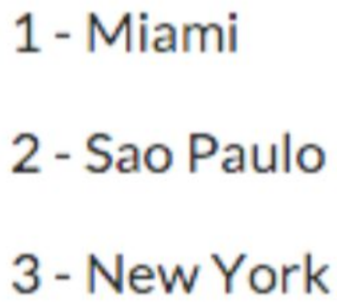
The syntax is *ngFor="let item of items".

```
this.cities = ['Miami', 'Sao Paulo', 'New York'];

<div class="ui list" *ngFor="let c of cities">
    <div class="item">{{ c }}</div>
</div>
```

ngFor with index:

```
<div class="ui list" *ngFor="let c of cities; let num =
index">
    <div class="item">{{ num+1 }} - {{ c }}</div>
</div>
```



```
1 - Miami

2 - Sao Paulo

3 - New York
```

* and <template>

When we reviewed the NgFor, NgIf, and NgSwitch built-in directives, we used asterisk (*) that appears before the directive names.

We can do what Angular does ourselves and expand the * prefix syntax to template syntax:

```
<hero-detail *ngIf="currentHero" [hero]="currentHero">
</hero-detail>
```

Is the same as

```
<template [ngIf]="currentHero">
   <hero-detail [hero]="currentHero"></hero-detail>
</template>
```

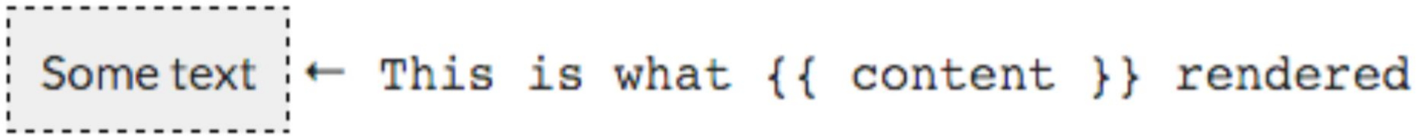
ngNonBindable

We use `ngNonBindable` when we want tell Angular not to compile or bind a particular section of our page.

Let's say we want to render the literal text `{{ content }}` in our template. Normally that text will be bound to the value of the `content` variable because we're using the `{{ }}` template syntax.

```
<div>
    <span class="bordered">{{ content }}</span>

    <span class="pre" ngNonBindable>
    &larr; This is what {{ content }} rendered
    </span>
</div>
```



Some text ← This is what {{ content }} rendered

Property binding

We write a template property binding when we want to set a **property of a view element** to the value of a template expression.

binding the src property of an image element to a component's heroImageUrl property:

```
  <img [src]="heroImageUrl">
```

disabling a button when the component says that it isUnchanged:

```
  <button [disabled]="isUnchanged">Cancel is disabled</button>
```

setting a property of a directive:

```
  <div [ngClass]="classes">[ngClass] binding to the classes
property</div>
```

setting the model property of a custom component:

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

Property binding as one-way data binding because it flows a value in one direction, from a component's data property into a target element property.

Attribute binding

We must use attribute binding when there is no element property to bind.

If we try this:

```
<tr><td colspan="{{1 + 1}}">Three-Four</td></tr>
```

We'll get the error:

> *Template parse errors:*
> *Can't bind to 'colspan' since it isn't a known native property*

<td> element does not have a colspan property. It has the "colspan" attribute, but interpolation and property binding can set only properties, not attributes.
We need attribute bindings to create and bind to such attributes.

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

Class binding

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

Replacement binding:

```
<!-- reset/override all class names with a binding  -->
<div class="bad curly special"
    [class]="badCurly">Bad curly</div>
```

Angular adds the class when the template expression evaluates to truthy. It removes the class when the expression is falsey.

```
<!-- toggle the "special" class on/off with a property -->
<div [class.special]="isSpecial">The class binding is special</div>

<!-- binding to `class.special` trumps the class attribute -->
<div class="special"
    [class.special]="!isSpecial">This one is not so special</div>
```

For managing multiple class names it's preferred to use ngClass

Style binding

We can set inline styles with a style binding.

```
<button [style.color] = "isSpecial ? 'red': 'green'”>Red</button>
<button [style.background-color]="canSave ? 'cyan': 'grey'" >
      Save
</button>
```

Some style binding styles have unit extension. Here we conditionally set the font size in “em” and “%” units:

```
<button [style.font-size.em]="isSpecial ? 3 : 1" >Big</button>
<button [style.font-size.%]="!isSpecial ? 150 : 50" >Small</button>
```

When setting several inline styles at the same time ngStyle directive is preferrable

Event binding

User actions may result in a flow of data in the opposite direction: from an element to a component. They are described with event bindings:

```
<button (click)="onSave()">Save</button>
```

The binding conveys information about the event, including data values, through an event object named **$event**.

Event object is determined by the target event. If the target event is a native DOM element event, then **$event** is a **DOM event object**, with properties such as target and target.value:

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

Two-way binding with ngModel

We often want to both display a data property and update that property when the user makes changes.

[( )] = BANANA IN A BOX

Two-way data binding with the NgModel directive makes that easy. Here's an example:

```
<input [(ngModel)]="currentHero.firstName">
```

Note: to make [(ngModel)] available we have to import FormsModule in NgModule.

For <input> it’s the same as

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

That ngModel directive hides these onerous details behind its own ngModel input and ngModelChange output properties:

```
<input  [ngModel]="currentHero.firstName"
    (ngModelChange)="currentHero.firstName=$event">
```

Template reference variables

A template reference variable is a reference to a DOM element or directive within a template.

Note: Do not define the same variable name more than once in the same template. The runtime value will be unpredictable.

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

Binding in templates

Data Direction	Syntax	Binding Type
One way from data source to view target	{{expression}} [target] = "expression"	Interpolation Property Attribute Class Style
One way from view target to data source	(target) = "expression"	Event
Two way	[(target)] = "expr”	Two-way

Binding targets

Binding Type	Target	Examples
Property	Element Property	`<img [src] = "heroImageUrl">`
	Component Property	`<hero-detail [hero]="currentHero"></hero-detail>`
	Directive property	`<div [ngClass] = "{selected: isSelected}"></div>`
Event	Element Event	`<button (click) = "onSave()"> Save </button>`
	Component Event	`<hero-detail (deleted)="onHeroDeleted()"></hero-detail>`
	Directive Event	`<div myClick (myClick)="clicked=$event">click me</div>`
Two-way	Directive Event Property	`<input [(ngModel)]="heroName">`

Binding targets

Binding Type	Target	Examples

Attribute	Attribute (the exception)	`<button [attr.aria-label]="help">`
		`  help`
		`</button>`

Class	class Property	`<div [class.special]="isSpecial">`
		`  Special`
		`</div>`

Style	style Property	`<button [style.color] =`
		`  "isSpecial ? 'red' : 'green'">`

Thank you
and have a great Angular 2 experience!

LXFT LISTED NYSE

LUXOFT