

Class / Style binding

Classe

```
<div [class.class1]="boolean"> ... </div>
```

```
<div [class]="classExpression"> ... </div>
```

```
classExpression = 'class1 class2'
```

```
classExpression = { class1: true, class2: false }  
classExpression = ['class1', 'class2']
```

- Ici le div aura la classe "class1" si l'attribut "boolean" du composant est truthy
- classExpression peut être sous forme de string, de tableau ou d'objet

Style


```
<div [style.font-size]="fontSize"> ... </div>  
<div [style.font-size.px]="size"> ... </div>
```

```
<div [style]="styleExpression"> ... </div>
```

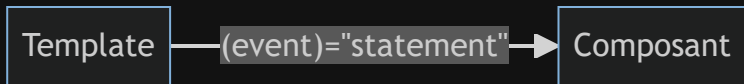
```
styleExpression = 'color: blue; font-size: 50px'  
styleExpression = { color: 'blue', 'font-size': '50px' }
```

- Il est possible de lier un style qui a une unité avec un nombre
- styleExpression peut être sous forme de string ou d'objet
- ⚠ Même problème pour les tableaux et objets que l'interpolation

Template expression

- Une expression de template est très similaire à du JavaScript, avec quelques exceptions
 - Il n'est pas possible de faire des assignements (`=`, `+=`, `-=`, ...) ni d'incrément/décrément (`++`/`--`)
 - Les opérateurs binaires (`|`, `&`) sont interdits
 - Il n'est pas possible de chainer les expressions avec `;` or `,`
 - Pas de `new`, `typeof` ou `instanceof`
 - L'opérateur binaire `|` (OR) de JavaScript est remplacé par le pipe
 - La fonction `$any()` pour cast une expression en `any`
-  Privilégier des expressions simples, avec la logique dans le composant

Event binding



- Lie un évènement avec un template statement

```
<button (click)="i=i+1">Incrémenter</button>
```

- Il est possible d'utiliser les méthode du composant

```
<button (click)="reset()">Réinitialiser</button>
```

- Il est possible de récupérer des information sur l'évènement grace à \$event

```
<input (input)="logInput($event)" />
```

```
logInput(event: Event){  
  console.log(event);  
}
```

- Le type de l'objet \$event dépend du type d'évènement, qui sont basés sur l'interface Event
- <https://developer.mozilla.org/fr/docs/Web/API/Event>

Template statement

- Comme les expressions, les templates statements sont proches du JavaScript, avec les différences suivantes :
 - L'assignement basique (=) est permis
 - Le chainage des expressions avec ; ou , est permis
 - La fonction \$any() est autorisée
 - new, typeof et instanceof sont interdits
 - Les opérateurs (+=, -=, ++ et --) sont interdits
 - Les opérateurs binaires sont interdits
 - L'opérateur pipe est interdit

Directives

- Les directives sont des classes Angular, avec le décorateur @Directive
- Elles sont rattachées à des éléments HTML par un sélecteur, et modifient le comportement de l'élément
- Les “Structural Directive” ajoutent ou suppriment des éléments du DOM

```
<ng-container *ngIf="monTableau.length else empty">
```

- Les “Attribute directive” modifient l'apparence ou le comportement d'un élément du DOM

```
<div [ngStyle]="styleExpression"> ... </div>
```

- Les composants sont également des directives

```
export declare interface Component extends Directive {
```

- Des directives sont fournies par Angular, mais il est possible de créer ses propres directives

Structural Directives (*ngIf, *ngFor, *ngSwitch)

- Les directives structurelles ajoutent ou suppriment des éléments du DOM
- Les balises *ngIf, *ngFor, *ngSwitch sont très similaires aux structures de contrôle (if, for et switch) que l'on peut utiliser dans d'autres langages
- Le * indique une syntaxe raccourcie, par exemple :

```
<div *ngIf="condition">Texte affiché si la condition est vraie</div>
```

- Est en réalité un raccourci pour :

```
<ng-template [ngIf]="condition">  
  <div>Texte affiché si la condition est vraie</div>  
</ng-template>
```

- La balise <ng-template> est une balise utilisée par Angular pour porter des directives structurelles, sans ajouter d'élément au DOM
- <ng-template> n'est pas affichée par défaut

*ngIf

- Affiche l'élément uniquement si la condition est vraie

```
<div *ngIf="condition">Texte affiché si la condition est vraie</div>
```

- La balise <ng-container> permet de contenir un *ngIf sans créer un élément

```
<ng-container *ngIf="condition">Texte affiché si la condition est vraie</ng-container>
```

- Il est possible de faire un "else" en utilisant un <ng-template>

```
<ng-container *ngIf="loaded; else loading">
  ...
</ng-container>
<ng-template #loading>
  L'application est en cours de chargement
</ng-template>
```

*ngFor

- L'élément à l'intérieur du *ngFor va être dupliqué pour chaque item du tableau monTableau

```
<div *ngFor="let item of monTableau">{{item.name}}</div>
```

- Il est possible d'utiliser les propriétés suivantes à l'intérieur d'un *ngFor
 - index (number) : index de l'objet en cours
 - count (number) : taille totale de l'iterable
 - first (boolean) : vrai si l'objet est le premier de l'iterable
 - last (boolean) : vrai si l'objet est le dernier de l'iterable
 - even (boolean) : vrai si l'index de l'objet est pair
 - odd (boolean) : vrai si l'index de l'objet est impair

```
<div [class.red]="odd" *ngFor="let item of monTableau; index as i; let tailleTotale=count; odd as odd">  
  {{item}} ({{ i }}) / {{ tailleTotale }}  
</div>
```


Built-in control flow

- Depuis la version 17, il est possible de créer des blocs structuraux directement dans le template
- Par exemple, avec le @if :

```
<ng-container *ngIf="loaded; else loading">
  ...
</ng-container>
<ng-template #loading>
  L'application est en cours de chargement
</ng-template>
```

```
@if(loaded) {
  ...
} else {
  L'application est en cours de chargement
}
```

- Il est possible de faire un else facilement sans passer par un ng-template
- Il est également possible de faire un else if

- Avec un @for :

```
<div *ngFor="let item of monTableau">{{item.name}}</div>
```

```
@for(item of monTableau; track item.id) {  
  <div>{{item.name}}</div>  
} @empty {  
  <div>Le tableau est vide</div>  
}
```

- Le bloc @for à des meilleures performances que le *ngFor
- Le track, qui remplace le trackBy, est obligatoire

Attributes Directives (ngModel, ngStyle, ngClass)

- Les directives d'attributs modifient l'aspect et le comportement des éléments du DOM

```
<div maDirective> ... </div>
```

- [ngStyle] et [ngClass] ont le même comportement que [style] et [class], mais détectent le changement du contenu d'un objet ou tableau

```
<div [ngClass]="classExpression"> ... </div>  
<div [ngStyle]="styleExpression"> ... </div>
```

- [(ngModel)] lie une propriété du composant à un input et le modifie lorsque la valeur de l'input change

```
<input [(ngModel)]="data" />
```

- ⚠ Ne pas oublier l'import de FormsModule pour utiliser ngModel
- Notez l'écriture de [(ngModel)], () à l'intérieur de [], aussi connu sous le nom de 'banana-in-a-box syntax'

Exercice

- Le but de l'exercice est de créer un petit lecteur vidéo :
 - Un clic sur la vidéo lance ou pause la vidéo
 - On peut faire défiler la vidéo avec la molette
 - On peut déplacer la vidéo sur la page
- https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API
- <https://developer.mozilla.org/fr/docs/Web/API/Event>
- <https://developer.mozilla.org/en-US/docs/Web/CSS/transform>