

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>
```

Exercice - Refactorisation de l'application de démo

```
<p>What do you want to do next with your app?</p>

<input type="hidden" #selection>

<div class="card-container">
  <button class="card card-small" (click)="selection.value = 'component'" tabindex="0">
    <svg class="material-icons" xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24"><path d=
    <span>New Component</span>
  </button>

  <button class="card card-small" (click)="selection.value = 'material'" tabindex="0">
    <svg class="material-icons" xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24"><path d=
    <span>Angular Material</span>
  </button>

  <button class="card card-small" (click)="selection.value = 'pwa'" tabindex="0">
    <svg class="material-icons" xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24"><path d=
    <span>Add PWA Support</span>
  </button>

  <button class="card card-small" (click)="selection.value = 'dependency'" tabindex="0">
    <svg class="material-icons" xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24"><path d=
```

Exercice - Refactorisation de l'application de démo

- Deux choses à changer dans ce code :
 - Ne pas utiliser le champ hidden pour stocker la selection
 - Ne pas répéter le code pour chaque nouveau bouton

```
<button class="card card-small" (click)="selection.value = 'component'" tabindex="0">  
  <svg class="material-icons" xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24">  
    <path d="M19 13h-6v6h-2v-6H5v-2h6V5h2v6h6v2z" />  
  </svg>  
  <span>New Component</span>  
</button>
```

- Pour le moment, stocker la liste des topics dans le composant
- Assigner pour chaque topic une couleur de texte différent
- Pas d'onglet sélectionné par défaut, avec un message indiquant pas d'onglet sélectionné

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'

Template variables

- Dans le template, il est possible d'utiliser le symbole # pour déclarer une variable de template


```
<input #classe [value]="classeString" />  
<input #style [value]="styleString" />
```

- La variable référence l'élément, et peut être utilisée de partout dans le template

```
<div>La valeur de mon input est {{ classe.value }}</div>  
<button (click)="maj(classe.value, style.value)">Mise à jour</button>
```

- Le type de cette variable dépend d'où elle est déclarée
 - Pour un élément HTML standard, l'interface correspondante (HTMLInputElement pour un input)
 - Pour un composant, le type du composant
- https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API

Priorité et portée des variables

- Les variables de templates sont globales dans le template du composant, sauf celles à l'intérieur d'une directive structurelle
- Plusieurs éléments peuvent avoir la même variable, dans ce cas la première sera prioritaire
- Les variables de template peuvent avoir le même nom que des attributs du composant
- Si plusieurs variables ont le même nom dans le template, la priorité suivante est appliquée
 - Variables définies dans une directive structurelle
 - Variable de template
 - Attribut du composant
-  Utiliser des noms différents pour les variables

Standalone components

```
@Component({
  standalone: true,
  imports: [CommonModule, OtherStandaloneComponent],

  selector: 'standalone-component',
  template: ' ... ',
  styles: [ ... ]
})
export class StandaloneComponent {
```

- Depuis Angular 14, il est possible de créer des composants standalone (ainsi que des directives et pipes), sans passer par des modules
- Dans le décorateur, il suffit d'ajouter le flag standalone à vrai
- Il faut également spécifier les dépendances directement dans le composant
- Si l'on souhaite réutiliser le composant standalone, on peut l'importer directement (dans un module ou directement dans un autre composant standalone)

Standalone components

```
bootstrapApplication(StandaloneComponent).catch(err => console.error(err));
```

- On peut bootstrap directement sur un composant standalone

Exercice

- https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API
- <https://developer.mozilla.org/fr/docs/Web/API/Event>
- 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/CSS/transform>