Cours Angular : Les Pipes

1. Introduction

Les **pipes** en Angular permettent de **transformer les données** dans le template de manière déclarative, sans modifier le modèle.

Objectifs : - Comprendre le rôle des pipes - Utiliser les pipes intégrés - Créer des pipes personnalisés - Appliquer les bonnes pratiques professionnelles

---

2. Pipes intégrés

Angular fournit de nombreux pipes pour les types de données courants :

Pipe	Description
`date`	Formate une date
`uppercase` / `lowercase`	Met en majuscule / minuscule
`currency`	Formate une valeur monétaire
`percent`	Formate un pourcentage
`decimal`	Formate un nombre
`json`	Convertit un objet en JSON

2.1 Exemple dans un template

```
<p>Date actuelle : {{ today | date:'fullDate' }}</p>
<p>Nom : {{ name | uppercase }}</p>
<p>Prix : {{ price | currency:'USD' }}</p>
```

---

3. Pipes avec paramètres

```
<p>{{ birthday | date:'shortDate' }}</p>
<p>{{ amount | currency:'EUR':'symbol':'1.2-2' }}</p>
```

- `shortDate`, `longDate`, `1.2-2` = format nombre décimal

---

4. Pipes personnalisés

4.1 Création d'un pipe

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'truncate' })
export class TruncatePipe implements PipeTransform {
  transform(value: string, limit: number = 20): string {
    return value.length > limit ? value.substring(0, limit) + '...' : value;
  }
}
```

4.2 Utilisation dans le template

```
<p>{{ text | truncate:30 }}</p>
```

---

5. Pipes purs et impurs

- **Pipes purs** : Angular ne recalculera le pipe que si la valeur de l'input change
- **Pipes impurs** : Angular recalculera le pipe à chaque détection de changement

```
@Pipe({ name: 'impurePipe', pure: false })
```

⚠️Les pipes impurs sont coûteux pour la performance, à utiliser avec parcimonie.

---

6. Composition de pipes

```
<p>{{ text | truncate:20 | uppercase }}</p>
```

- On peut **enchaîner plusieurs pipes** pour transformer les données.

---

7. Pipes asynchrones

- Le **pipe** `async` permet de gérer les **Observables** et **Promises** directement dans le template

```
users$ = this.userService.getUsers();
```

```
<ul>
  <li *ngFor="let user of users$ | async">{{ user.name }}</li>
</ul>
```

- Angular souscrit automatiquement à l'Observable et se désinscrit pour éviter les fuites mémoire.

---

8. Bonnes pratiques

- Préférer les pipes **purs** pour la performance
- Utiliser les pipes intégrés pour les transformations simples
- Créer des pipes personnalisés pour les besoins spécifiques à l'application
- Éviter les calculs lourds dans les pipes impurs
- Utiliser `async` pipe pour les Observables et Promises

---

9. Cas pratique complet

Objectif

Afficher une liste d'articles avec : - Date formatée - Prix formaté en USD - Description tronquée - Nom en majuscule

```
<ul>
  <li *ngFor="let item of items">
    <h3>{{ item.name | uppercase }}</h3>
    <p>{{ item.description | truncate:50 }}</p>
    <p>{{ item.price | currency:'USD' }}</p>
    <p>Publié le {{ item.date | date:'shortDate' }}</p>
  </li>
</ul>
```

---

10. Résumé

- Les pipes transforment les données directement dans le template
- Angular propose des **pipes intégrés**, mais les **pipes personnalisés** sont très utiles
- Pipes **purs** = performance, **impurs** = recalcul constant
- Le **pipe** `async` est incontournable pour les données asynchrones

---

Prochaine étape recommandée : **création de pipes complexes + optimisation de performance**