Pipes

Pipes

- Un pipe est une classe Angular, qui possède une méthode de transformation
- Un pipe est utilisée dans le template pour transformer une expression en une autre
- Les pipes sont utilisés avec l'opérateur |

```
{{ entrée | pipe }}
```

Pipes

- Un pipe est une classe Angular, qui possède une méthode de transformation
- Un pipe est utilisée dans le template pour transformer une expression en une autre
- Les pipes sont utilisés avec l'opérateur |

```
{{ entrée | pipe | secondPipe }}
```

- Il est possible de faire des enchainements de pipes, où la sortie d'un pipe devient l'entrée de la suivante

Créer un pipe

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

@Pipe({
  name: 'exponential',
  standalone: true
})
export class ExponentialPipe implements PipeTransform {

  transform(value: number, exponent = 1): number {
    return Math.pow(value, exponent);
  }
}
```

Créer un pipe

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

@Pipe({
  name: 'exponential',
  standalone: true
})
export class ExponentialPipe implements PipeTransform {

  transform(value: number, exponent = 1): number {
    return Math.pow(value, exponent);
  }
}
```

Décorateur @Pipe

- Indique à Angular que la classe est un pipe
- Utilisation de métadonnées pour paramétrer le pipe
- Un pipe peut être standalone

Créer un pipe

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

@Pipe({
  name: 'exponential',
  standalone: true
})
export class ExponentialPipe implements PipeTransform {

  transform(value: number, exponent = 1): number {
    return Math.pow(value, exponent);
  }
}
```

Sélecteur CSS du pipe

```
...
{{ 12 | exponential:2 | exponential | exponential:3 }}
...
```

Créer un pipe

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

@Pipe({
  name: 'exponential',
  standalone: true
})
export class ExponentialPipe implements PipeTransform {

  transform(value: number, exponent = 1): number {
    return Math.pow(value, exponent);
  }
}
```

Implémentation :

- Un pipe implémente l'interface PipeTransform
- La méthode transform est une fonction variadique, et a au moins un paramètre, l'expression à transformer
- ⚠ Attention au type d'entrée de sortie des pipes

Exercice

- Créer un pipe qui tronque une chaîne de caractères trop longue, et ajoute "..." à la fin
- La taille maximale de la chaîne en entrée est paramétrable, avec une valeur par défaut de 10

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

@Pipe({
  name: 'tronque',
  standalone: true
})
export class TronquePipe implements PipeTransform {

  ...

}
```

Pipes Angular

- Un certain nombre de pipes sont disponibles dans le package CommonModule :

- UpperCasePipe (uppercase) et LowerCasePipe (lowercase) : Convertit la valeur respectivement en majuscules ou minuscules
- DatePipe (date) : Formatte une date
- DecimalPipe (number), PercentPipe (percent), CurrencyPipe (currency) : Formatte un nombre
- JsonPipe (json) : Convertit un objet sous forme JSON (utile pour débug)
- AsyncPipe (async) : Souscrit à un Observable ou une Promise et retourne la dernière valeur émise

- https://angular.io/api?type=pipe

Pourquoi les pipes ?

Performance

- Angular execute le pipe uniquement lorsque l'expression d'entrée change

```
{{ toUpperCase(texte) }}                                    {{ texte | toUpperCase }}
```

Lisibilité

- En particulier lors de chaînage de pipes, l'expression finale est bien plus lisible

```
{{ toUpperCase(lowercase(format(transformation(texte)))) }}

{{ texte | transformation | format | lowercase | toUpperCase }}
```

Pipe pur et impur

- La fonction de transformation du pipe doit être pure
  - La valeur de sortie doit dépendre uniquement des valeurs d'entrée
  - La fonction ne modifie pas de valeurs en dehors de ses variables locales

- ⚠ Utilisation du pipe avec les tableaux et les objets

- Un pipe est pur par défaut, mais il est possible de le rendre 'impur'

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

- ⚠ Angular ne 'devine' pas si le pipe est pure ou impur, c'est à vous de lui indiquer selon l'usage

- Un pipe impur n'a plus l'optimisation du pipe pur, et sera executé beaucoup plus souvent

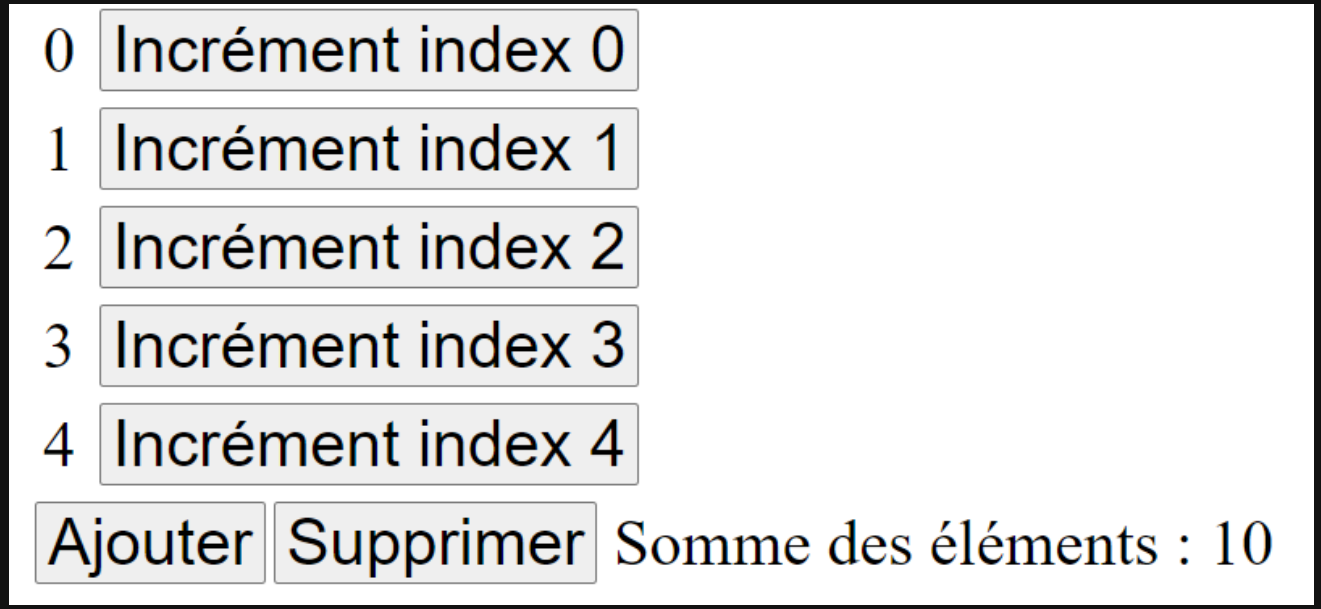
Exercice - Pipes et objets

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

@Pipe({
  standalone: true,
  name: 'sumPipe',
})
export class SumPipe implements PipeTransform {
  transform(array: number[]): number {
    return array.reduce((a, b) => a + b, 0)
  }
}

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

  selector: 'app-root',
  template: `
  <div>
    <div *ngFor="let nombre of tableau; index as index" > {{ nombre }}
        <button (click)="increment(index)">{{'Incrément index ' + index}} </button>
```



- Ce composant affiche les éléments d'un tableau
- Il est possible d'incrémenter les éléments, et d'ajouter ou supprimer un élément
- La somme totale des éléments est affichée, en utilisant un pipe
- Problème : La somme n'est pas mise à jour si l'utilisateur modifie les éléments

Solution 1

- Utiliser un pipe impure

```
@Pipe({
  standalone: true,
  name: 'sumPipe',
  pure: false
})
```

✓ Solution simple

✕ Peut impacter les performances si l'opération est complexe (tri, ...)

- Exercice : trouver une solution sans passer par un pipe impure