

Angular : concepts de base

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`



- 1 Introduction
- 2 Liaison template/classe (composant)
 - Interpolation `{{ ... }}`
 - Property binding `[...]`
 - Event binding `(...)`
- 3 Directives Angular
 - `*ngFor`
 - `*ngIf`
 - `ngSwitch`
 - `ngPlural`
 - `ngStyle`
 - `ngClass`

4 Control Flow Syntax

- @for
- @empty
- @if
- @else
- @switch

5 Valeurs réactives

- signal
- computed
- effect

4 modes de liaison (binding) avec **Angular**

- One way binding
 - Property binding [...]
 - Interpolation {{ ... }}
 - Event binding (...)
- Two way binding [(...)]

Angular

Le fichier `app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
}
```

Le fichier `app.component.html`

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  ...
</div>
```

© Achref EL BOUELA

Angular

Le fichier `app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
}
```

Le fichier `app.component.html`

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  ...
</div>
```

```
{{ Interpolation }}
```

Welcome to {{ title }}!:
title sera remplacé par sa valeur dans `app.component.ts`

Angular

Objectif : afficher les attributs et les valeurs sous forme d'une liste **HTML**

- Créer un répertoire `classes` dans `app` où toutes les classes seront stockées.
- Créer une classe `Stagiaire` avec les attributs `num`, `nom` et `prénom`.
- Créer un objet de cette classe dans `app.component.ts`
- Afficher les valeurs de cet objet sous forme de liste dans `app.component.html`

Angular

On peut aussi utiliser la commande

```
ng generate class classes/stagiaire
```

© Achref EL MOUELHI ©

Angular

On peut aussi utiliser la commande

```
ng generate class classes/stagiaire
```

Pour générer une classe sans le fichier de test (.spec.ts)

```
ng generate class classes/stagiaire --skip-tests=true
```

Angular

On peut aussi utiliser la commande

```
ng generate class classes/stagiaire
```

Pour générer une classe sans le fichier de test (.spec.ts)

```
ng generate class classes/stagiaire --skip-tests=true
```

Remarque

L'option `--skip-tests=true` peut être utilisée avec la commande `generate` quel que soit l'élément généré.

Contenu de stagiaire.ts

```
export class Stagiaire {
  constructor(private _num?: number, private _nom?: string, private
    _prenom?: string) { }

  get num(): number | undefined {
    return this._num;
  }
  set num(_num: number | undefined) {
    this._num = _num;
  }
  get nom(): string | undefined {
    return this._nom;
  }
  set nom(_nom: string | undefined) {
    this._nom = _nom;
  }
  get prenom(): string | undefined {
    return this._prenom;
  }
  set prenom(_prenom: string | undefined) {
    this._prenom = _prenom;
  }
}
```

Angular

Créons un objet de type `Stagiaire` dans `app.component.ts`

```
import { Component } from '@angular/core';
import { Stagiaire } from './classes/stagiaire';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
  stagiaire: Stagiaire = new Stagiaire(100, 'Wick', 'John');
  constructor() {
  }
}
```

Angular

Affichons l'objet stagiaire **dans** app.component.html

```
<ul>
  <li> Nom : {{ stagiaire.nom }} </li>
  <li> Prénom : {{ stagiaire.prenom }} </li>
</ul>
```

Angular

L'écriture suivante

```
{{ stagiaire }}
```

© Achref EL MOUELHI ©

Angular

L'écriture suivante

```
{{ stagiaire }}
```

affiche

```
[object Object]
```

© Achref EL M... HI ©

Angular

L'écriture suivante

```
{{ stagiaire }}
```

affiche

```
[object Object]
```

On peut utiliser un pipe pour affiche un objet au format JSON

```
{{ stagiaire | json }}
```


Angular

L'écriture suivante

```
{{ stagiaire }}
```

affiche

```
[object Object]
```

On peut utiliser un pipe pour affiche un objet au format JSON

```
{{ stagiaire | json }}
```

Le résultat est

```
{ "_num": 100, "_nom": "Wick", "_prenom": "John" }
```

Angular

Modifions l'objet `stagiaire` **dans** `app.component.ts`

```
import { Component } from '@angular/core';
import { Stagiaire } from './classes/stagiaire';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
  stagiaire: Stagiaire = new Stagiaire(100, 'Wick');
  constructor() {
  }
}
```

Angular

Dans `app.component.html`, et depuis Angular 12, on peut utiliser La coalescence nulle

```
<ul>
  <li> Nom : {{ stagiaire.nom }} </li>
  <li> Prénom : {{ stagiaire.prenom ?? "prénom indéfini" }} </li>
</ul>
```

© Achille

Angular

Créer un tableau d'entiers `tab` dans `app.component.ts`

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  title = 'cours-angular';
  stagiaire: Stagiaire = new Stagiaire(100, 'Wick', 'John');
  tab: number[] = [2, 3, 5, 8];

  constructor() {
  }
}
```

Afficher le tableau `tab` dans `app.component.html`

```
<ul>
  <li> {{ tab[0] }} </li>
  <li> {{ tab['1'] }} </li>
  <li> {{ tab["2"] }} </li>
  <li> {{ tab[3] }} </li>
</ul>
```

© Achref EL MOUELHI ©

Afficher le tableau `tab` dans `app.component.html`

```
<ul>
  <li> {{ tab[0] }} </li>
  <li> {{ tab['1'] }} </li>
  <li> {{ tab["2"] }} </li>
  <li> {{ tab[3] }} </li>
</ul>
```

Remarques

- Ce code est trop répétitif
- Et si on ne connaissait pas le nombre d'éléments
- Ou si on ne voulait pas afficher tous les éléments

Afficher le tableau `tab` dans `app.component.html`

```
<ul>
  <li> {{ tab[0] }} </li>
  <li> {{ tab['1'] }} </li>
  <li> {{ tab["2"] }} </li>
  <li> {{ tab[3] }} </li>
</ul>
```

Remarques

- Ce code est trop répétitif
- Et si on ne connaissait pas le nombre d'éléments
- Ou si on ne voulait pas afficher tous les éléments

Solution

utiliser les directives (section suivante)

Angular

Ajouter une méthode `direBonjour()` dans `app.component.ts`

```
import { Component } from '@angular/core';
import { Stagiaire } from '../classes/stagiaire';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
  stagiaire: Stagiaire = new Stagiaire(100, 'Wick', 'John');
  tab: number[] = [2, 3, 5, 8];
  constructor() {
  }
  direBonjour(): string {
    return 'bonjour Angular';
  }
}
```


Angular

Appeler la méthode `direBonjour()` **dans** `app.component.html`

```
<p> {{ direBonjour() }} </p>
```

Angular

Pour afficher le contenu de l'attribut `title` dans le template, on peut utiliser l'interpolation

```
<p> {{ title }} </p>
```

© Achref EL MOUELHI ©

Angular

Pour afficher le contenu de l'attribut `title` dans le template, on peut utiliser l'interpolation

```
<p> {{ title }} </p>
```

On peut aussi utiliser le Property binding sur la propriété JavaScript `textContent`

```
<p [textContent]=title></p>
```

Angular

Pour afficher le contenu de l'attribut `title` dans le template, on peut utiliser l'interpolation

```
<p> {{ title }} </p>
```

On peut aussi utiliser le Property binding sur la propriété JavaScript `textContent`

```
<p [textContent]=title></p>
```

```
[ property binding ]
```

`[property] = 'value'` : permet de remplacer `value` par sa valeur dans la classe `app.component.ts`

Angular

Évènement (Event)

- Action appliquée par l'utilisateur ou simulée par le développeur sur un `.component.html`.
- Évènement déclenché \Rightarrow Méthode, dans le `.component.ts` associé, exécutée.

© Achref EL M...

Angular

Évènement (Event)

- Action appliquée par l'utilisateur ou simulée par le développeur sur un `.component.html`.
- Évènement déclenché \Rightarrow Méthode, dans le `.component.ts` associé, exécutée.

Event binding (...)

- Objectif : associer un évènement à une méthode de la classe associée au template.
- Pas besoin de préfixer l'évènement par 'on' comme en **JavaScript**.

Angular

Ajoutons la méthode `alertBonjour()` dans `app.component.ts`

```
import { Component } from '@angular/core';
import { Stagiaire } from '../classes/stagiaire';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  title = 'cours-angular';
  stagiaire: Stagiaire = new Stagiaire(100, 'Wick', 'John');
  tab: number[] = [2, 3, 5, 8];

  constructor() {
  }

  alertBonjour(): void {
    console.log('bonjour Angular');
  }
}
```

Angular

Pour exécuter la méthode `direBonjour()` lorsqu'on clique sur un bouton

```
<button (click)="alertBonjour()">Dire Bonjour</button>
```

© Achref EL ...

Angular

Exercice

- Créer deux boutons **HTML** dont l'un est initialement désactivé (`disabled`).
- À chaque clic sur le bouton activé, celui-ci se désactive tandis que l'autre s'active.

Angular

Solution

```
<button [disabled]="disabled" (click)="switchDisabled()">
  bouton 1
</button>

<button [disabled]="!disabled" (click)="switchDisabled()">
  bouton 2
</button>
```

© Achrel

Angular

Solution

```
<button [disabled]="disabled" (click)="switchDisabled()">  
  bouton 1  
</button>
```

```
<button [disabled]="!disabled" (click)="switchDisabled()">  
  bouton 2  
</button>
```

Sans oublier de déclarer `disabled` dans `data`

```
disabled = true
```

Angular

Considérons la zone de saisie suivante dans laquelle nous ne voudrions accepter que des lettres en minuscule

```
<div>  
  <label for="texte"> Merci de bien saisir un texte :  
  </label>  
  <input type="text" id="texte" (input)="showValue($event)">  
</div>
```

© Achref EL MOU

Angular

Considérons la zone de saisie suivante dans laquelle nous ne voudrions accepter que des lettres en minuscule

```
<div>
  <label for="texte"> Merci de bien saisir un texte :
</label>
  <input type="text" id="texte" (input)="showValue($event)">
</div>
```

Pour connaître le caractère saisi, utilisons l'objet `event` dans `showValue()`

```
showValue(event: Event) {
  console.log((event.target as HTMLInputElement).value);
  // affiche tous les caractères saisis

  console.log((event as InputEvent).data);
  // affiche le dernier caractère saisi
}
```

Angular

Plusieurs directives possibles

- *ngFor
- *ngIf
- *ngSwitch
- ngStyle
- ngClass

© Achre

Angular

Plusieurs directives possibles

- *ngFor
- *ngIf
- *ngSwitch
- ngStyle
- ngClass

Ces directives s'utilisent conjointement avec les composants web suivant

- ng-container
- ng-template

Angular

*ngFor

- permet de répéter un traitement (affichage d'un élément **HTML**)
- s'utilise comme un attribut de balise et sa valeur est une instruction itérative **TypeScript**

Angular

Afficher les éléments du tableau `tab` en utilisant `*ngFor`

```
<ul>
  <li *ngFor="let elt of tab">
    {{ elt }}
  </li>
</ul>
```

© Achref EL MOUËLHAJ

Angular

Afficher les éléments du tableau `tab` en utilisant `*ngFor`

```
<ul>
  <li *ngFor="let elt of tab">
    {{ elt }}
  </li>
</ul>
```

Le code précédent est une forme simplifiée du code suivant

```
<ul>
  <ng-template ngFor let-elt [ngForOf]="tab">
    <li>{{ elt }} </li>
  </ng-template>
</ul>
```

Angular

Afficher les éléments du tableau `tab` en utilisant `*ngFor`

```
<ul>
  <li *ngFor="let elt of tab">
    {{ elt }}
  </li>
</ul>
```

Le code précédent est une forme simplifiée du code suivant

```
<ul>
  <ng-template ngFor let-elt [ngForOf]="tab">
    <li>{{ elt }} </li>
  </ng-template>
</ul>
```

`ng-template` n'apparaîtra pas dans le **DOM**.

Et pour avoir l'indice de l'itération courante

```
<ul>  
  <li *ngFor="let elt of tab; let i = index">  
    {{ i }} : {{ elt }}  
  </li>  
</ul>
```

© Achref EL MOUELHI ©

Et pour avoir l'indice de l'itération courante

```
<ul>
  <li *ngFor="let elt of tab; let i = index">
    {{ i }} : {{ elt }}
  </li>
</ul>
```

ou

```
<ul>
  <li *ngFor="let elt of tab; index as i">
    {{ i }} : {{ elt }}
  </li>
</ul>
```

Et pour avoir l'indice de l'itération courante

```
<ul>
  <li *ngFor="let elt of tab; let i = index">
    {{ i }} : {{ elt }}
  </li>
</ul>
```

ou

```
<ul>
  <li *ngFor="let elt of tab; index as i">
    {{ i }} : {{ elt }}
  </li>
</ul>
```

Ou la version la plus longue

```
<ul>
  <ng-template ngFor let-i="index" let-elt [ngForOf]="tab">
    <li> {{ i }} : {{ elt }} </li>
  </ng-template>
</ul>
```

Angular

On peut aussi utiliser `first` pour savoir si l'élément courant est le premier de la liste

```
<ul>
  <li *ngFor="let elt of tab; let i = index; let isFirst = first">
    {{ i }} : {{ elt }} : {{ isFirst }}
  </li>
</ul>
```

© Achref EL MOULI

Angular

On peut aussi utiliser `first` pour savoir si l'élément courant est le premier de la liste

```
<ul>
  <li *ngFor="let elt of tab; let i = index; let isFirst = first">
    {{ i }} : {{ elt }} : {{ isFirst }}
  </li>
</ul>
```

Autres paramètres possible

- `last` : retourne `true` si l'élément courant est le dernier de la liste, `false` sinon.
- `even` : retourne `true` si l'indice de l'élément courant est pair, `false` sinon.
- `odd` : retourne `true` si l'indice de l'élément courant est impair, `false` sinon.

Angular

Considérons la liste suivante (à déclarer dans `app.component.ts`)

```
stagiaires: Array<Stagiaire> = [  
  new Stagiaire(100, 'Wick', 'John'),  
  new Stagiaire(101, 'Abruzzi', 'John'),  
  new Stagiaire(102, 'Marley', 'Bob'),  
  new Stagiaire(103, 'Segal', 'Steven')  
];
```

© Achref EL

Angular

Considérons la liste suivante (à déclarer dans `app.component.ts`)

```
stagiaires: Array<Stagiaire> = [  
  new Stagiaire(100, 'Wick', 'John'),  
  new Stagiaire(101, 'Abruzzi', 'John'),  
  new Stagiaire(102, 'Marley', 'Bob'),  
  new Stagiaire(103, 'Segal', 'Steven')  
];
```

Exercice

Écrire un script **Angular** qui permet d'afficher dans une liste **HTML** les nom et prénom de chaque stagiaire de la liste `stagiaires`.

Angular

Solution

```
<ul>
  <li *ngFor="let st of stagiaires">
    {{ st.prenom }} : {{ st.nom }}
  </li>
</ul>
```

Angular

Considérons la méthode suivante (à déclarer dans `app.component.ts`)

```
ajouterStagiaires() {  
  this.stagiaires = [  
    new Stagiaire(100, 'Wick', 'John'),  
    new Stagiaire(101, 'Abruzzi', 'John'),  
    new Stagiaire(102, 'Marley', 'Bob'),  
    new Stagiaire(103, 'Segal', 'Steven'),  
    new Stagiaire(104, 'Dalton', 'Jack'),  
    new Stagiaire(105, 'Maggio', 'Carol')  
  ];  
}
```

© Achrel

Angular

Considérons la méthode suivante (à déclarer dans `app.component.ts`)

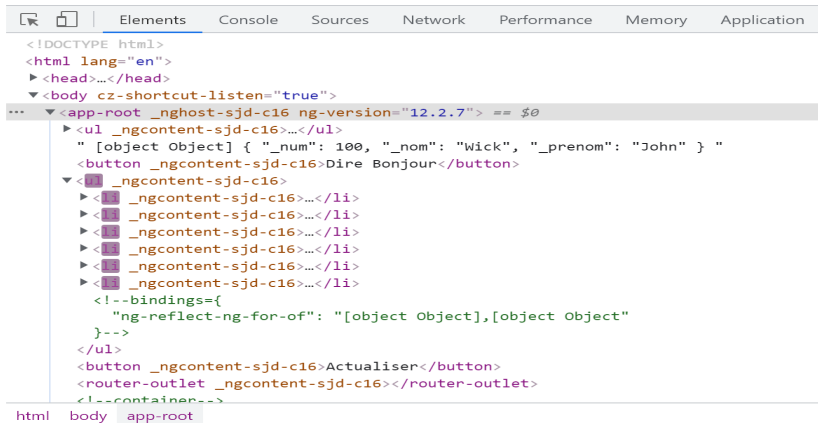
```
ajouterStagiaires() {  
  this.stagiaires = [  
    new Stagiaire(100, 'Wick', 'John'),  
    new Stagiaire(101, 'Abruzzi', 'John'),  
    new Stagiaire(102, 'Marley', 'Bob'),  
    new Stagiaire(103, 'Segal', 'Steven'),  
    new Stagiaire(104, 'Dalton', 'Jack'),  
    new Stagiaire(105, 'Maggio', 'Carol')  
  ];  
}
```

Et le bouton suivant permettant de mettre à jour la liste de stagiaires

```
<ul>  
  <li *ngFor="let st of stagiaires">  
    {{ st.prenom }} : {{ st.nom }}  
  </li>  
</ul>  
<button (click)="ajouterStagiaires()">Actualiser</button>
```

Angular

En cliquant sur le bouton, la liste (et la partie du DOM associée) sera entièrement reconstruite



```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body cz-shortcut-listen="true">
    <app-root _ngghost-sjd-c16 ng-version="12.2.7"> == $0
      <ul _ngcontent-sjd-c16>...</ul>
      " [object Object] { "_num": 100, "_nom": "Wick", "_prenom": "John" } "
      <button _ngcontent-sjd-c16>Dire Bonjour</button>
      <li _ngcontent-sjd-c16>
        <li _ngcontent-sjd-c16>...</li>
        <li _ngcontent-sjd-c16>...</li>
        <li _ngcontent-sjd-c16>...</li>
        <li _ngcontent-sjd-c16>...</li>
        <li _ngcontent-sjd-c16>...</li>
        <li _ngcontent-sjd-c16>...</li>
        <!--bindings={
          "ng-reflect-ng-for-of": "[object Object],[object Object]"
        }-->
      </ul>
      <button _ngcontent-sjd-c16>Actualiser</button>
      <router-outlet _ngcontent-sjd-c16></router-outlet>
      <!--container-->
    </app-root>
  </body>
</html>
```

Angular

Question

Comment faire pour ajouter seulement les éléments manquants sans reconstruire toute la liste ?

© Achref EL MOUADIB

Angular

Question

Comment faire pour ajouter seulement les éléments manquants sans reconstruire toute la liste ?

Réponse

Utiliser `trackBy` dans `*ngFor` qui permet de suivre les éléments lorsqu'ils sont ajoutés ou supprimés du tableau pour des meilleures performances.

Angular

Nous devons définir une méthode `trackBy` qui identifie chaque élément de manière unique

```
trackByStagiaireNum(index: number, stagiaire: Stagiaire) {  
  return stagiaire.num;  
}
```

© Achref EL MOUËL

Angular

Nous devons définir une méthode `trackBy` qui identifie chaque élément de manière unique

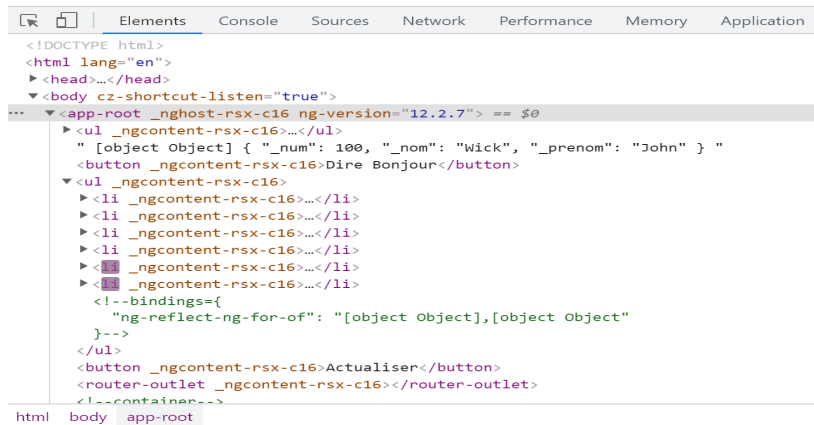
```
trackByStagiaireNum(index: number, stagiaire: Stagiaire) {  
  return stagiaire.num;  
}
```

Associons cette méthode à la directive `*ngFor`

```
<ul>  
  <li *ngFor="let st of stagiaires; trackBy:trackByStagiaireNum">  
    {{ st.prenom }} : {{ st.nom }}  
  </li>  
</ul>  
<button (click)="ajouterStagiaires()">Actualiser</button>
```

Angular

En cliquant sur le bouton, les deux derniers éléments ont été ajoutés au DOM sans reconstruire intégralement la liste



```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body cz-shortcut-listen="true">
    <app-root _ngghost-rsx-c16 ng-version="12.2.7"> == $0
      <ul _ngcontent-rsx-c16>...</ul>
        [object Object] { "_num": 100, "_nom": "Wick", "_prenom": "John" } "
        <button _ngcontent-rsx-c16>Dire Bonjour</button>
        <ul _ngcontent-rsx-c16>
          <li _ngcontent-rsx-c16>...</li>
          <li _ngcontent-rsx-c16>...</li>
          <li _ngcontent-rsx-c16>...</li>
          <li _ngcontent-rsx-c16>...</li>
          <li _ngcontent-rsx-c16>...</li>
          <li _ngcontent-rsx-c16>...</li>
          <!--bindings={
            "ng-reflect-ng-for-of": "[object Object],[object Object]"
          }-->
        </ul>
        <button _ngcontent-rsx-c16>Actualiser</button>
        <router-outlet _ngcontent-rsx-c16></router-outlet>
        <!--container-->
  </app-root>
</body>
</html>
```

html > body > app-root

Angular

Ou la version la plus longue

```
<ul>
  <ng-template ngFor let-st [ngForOf]="stagiaires" [ngForTrackBy]="
    trackByStagiaireNum">
    <li>
      {{ st.prenom }} : {{ st.nom }}
    </li>
  </ng-template>
</ul>
<button (click)="ajouterStagiaires()">Actualiser</button>
```

Angular

Remarques

- *ngFor nous permet d'itérer sur les tableaux, mais pas sur les objets.
- Il est possible d'itérer sur les objets en utilisant le pipe `keyvalue`.

Angular

Pour tester puis afficher si le deuxième élément du tableau est impair

```
<ul>
  <li *ngIf="tab[1] % 2 != 0">
    {{ tab[1] }} est impair
  </li>
</ul>
```

© Achref EL MOUL

Angular

Pour tester puis afficher si le deuxième élément du tableau est impair

```
<ul>
  <li *ngIf="tab[1] % 2 != 0">
    {{ tab[1] }} est impair
  </li>
</ul>
```

Ou en plus long

```
<ul>
  <ng-template [ngIf]="tab[1] % 2 != 0">
    <li>
      {{ tab[1] }} est impair
    </li>
  </ng-template>
</ul>
```

Angular

Exercice

Utiliser les directives **Angular** pour afficher le premier élément du tableau (`tab`) ainsi que sa parité (pair ou impair).

Une première solution avec *ngIf et else

```
<ul>
  <li *ngIf="tab[0] % 2 != 0; else pair">
    {{ tab[0] }} est impair
  </li>
  <ng-template #pair>
    <li>
      {{ tab[0] }} est pair
    </li>
  </ng-template>
</ul>
```

© Achref EL MOUËL

Une première solution avec *ngIf et else

```
<ul>
  <li *ngIf="tab[0] % 2 != 0; else pair">
    {{ tab[0] }} est impair
  </li>
  <ng-template #pair>
    <li>
      {{ tab[0] }} est pair
    </li>
  </ng-template>
</ul>
```

Ou en plus long

```
<ul>
  <ng-template [ngIf]="tab[0] % 2 != 0" [ngIfElse]=pair>
    <li>
      {{ tab[0] }} est impair
    </li>
  </ng-template>
  <ng-template #pair>
    <li>
      {{ tab[0] }} est pair
    </li>
  </ng-template>
</ul>
```

Une deuxième solution avec *ngIf, then et else

```
<ul>
  <li *ngIf="tab[0] % 2 != 0; then impair else pair">
    Ce code ne sera jamais pris en compte
  </li>
  <ng-template #impair>
    <li>
      {{ tab[0] }} est impair
    </li>
  </ng-template>
  <ng-template #pair>
    <li>
      {{ tab[0] }} est pair
    </li>
  </ng-template>
</ul>
```

© Achref EL

Une deuxième solution avec *ngIf, then et else

```
<ul>
  <li *ngIf="tab[0] % 2 != 0; then impair else pair">
    Ce code ne sera jamais pris en compte
  </li>
  <ng-template #impair>
    <li>
      {{ tab[0] }} est impair
    </li>
  </ng-template>
  <ng-template #pair>
    <li>
      {{ tab[0] }} est pair
    </li>
  </ng-template>
</ul>
```

Ou en plus long

```
<ul>
  <ng-template [ngIf]="tab[0] % 2 != 0" [ngIfThen]=impair [ngIfElse]=pair></ng-template>
  <ng-template #impair>
    <li>
      {{ tab[0] }} est impair
    </li>
  </ng-template>
  <ng-template #pair>
    <li>
      {{ tab[0] }} est pair
    </li>
  </ng-template>
</ul>
```

Angular

Remarque

*ngIf et *ngFor ne doivent pas être utilisées dans la même balise.

Angular

Exercice

Utiliser les directives **Angular** pour afficher sous forme d'une liste **HTML** tous les éléments du tableau précédent (`tab`) ainsi que leur parité.

Angular

Solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li *ngIf="elt % 2 != 0; else sinon">
      {{ elt }} est impair
    </li>
    <ng-template #sinon>
      <li>
        {{ elt }} est pair
      </li>
    </ng-template>
  </ng-container>
</ul>
```

Angular

Solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li *ngIf="elt % 2 != 0; else sinon">
      {{ elt }} est impair
    </li>
    <ng-template #sinon>
      <li>
        {{ elt }} est pair
      </li>
    </ng-template>
  </ng-container>
</ul>
```

ng-container n'apparaît pas dans le **DOM**.

Angular

Exercice

Utiliser les directives **Angular** pour afficher sous forme d'une liste **HTML** les éléments du tableau `moyennes`: `number[] = [18, 5, 11, 15]` avec le message suivant :

- Si $0 \leq \text{valeur} < 10$ alors échec,
- Si $10 \leq \text{valeur} < 13$ alors moyen,
- Si $13 \leq \text{valeur} < 16$ alors bien,
- Sinon très bien.

Angular

Une solution possible

```
<ul>
  <ng-container *ngFor="let elt of moyennes">
    <li *ngIf="elt >= 0 && elt < 10; else moyen">
      {{ elt }} : échec
    </li>
    <ng-template #moyen>
      <li *ngIf="elt < 13; else bien">
        {{ elt }} : moyen
      </li>
    </ng-template>
    <ng-template #bien>
      <li *ngIf="elt < 16; else tbien">
        {{ elt }} : bien
      </li>
    </ng-template>
    <ng-template #tbien>
      <li *ngIf="elt >= 16; else autre">
        {{ elt }} : très bien
      </li>
    </ng-template>
    <ng-template #autre>
      <li>
        autre
      </li>
    </ng-template>
  </ng-container>
</ul>
```

Angular

*ngIf VS hidden

- *ngIf commentera le contenu si la condition est fausse. L'élément ne sera pas attaché au **DOM**.
- [hidden] attachera l'élément au **DOM** et le marquera avec l'attribut `hidden`.

Angular

ngSwitch

- Équivalent à `switch` en **JavaScript**.
- Acceptant des valeurs de type `number`, `string` et `boolean`.

Exemple avec ngSwitch

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <ng-container [ngSwitch]="elt">
      <li *ngSwitchCase="1">
        {{ elt }} = un
      </li>
      <li *ngSwitchCase="2">
        {{ elt }} = deux
      </li>
      <li *ngSwitchCase="3">
        {{ elt }} = trois
      </li>
      <li *ngSwitchDefault>
        {{ elt }} : autre
      </li>
    </ng-container>
  </ng-container>
</ul>
```

Exemple avec ngSwitch

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <ng-container [ngSwitch]="elt">
      <li *ngSwitchCase="1">
        {{ elt }} = un
      </li>
      <li *ngSwitchCase="2">
        {{ elt }} = deux
      </li>
      <li *ngSwitchCase="3">
        {{ elt }} = trois
      </li>
      <li *ngSwitchDefault>
        {{ elt }} : autre
      </li>
    </ng-container>
  </ng-container>
</ul>
```

Pour comparer avec une chaîne de caractère, il faut ajouter les simples quotes (par exemple : `*ngSwitchCase="'bonjour' "`).

Angular

Refaire cet exercice avec `ngSwitch`

Utiliser les directives **Angular** pour afficher sous forme d'une liste les éléments du tableau `moyennes = [18, 5, 11, 15]` avec le message suivant :

- Si $0 \leq \text{valeur} < 10$ alors échec
- Si $10 \leq \text{valeur} < 13$ alors moyen
- Si $13 \leq \text{valeur} < 16$ alors bien
- Sinon très bien

Angular

Solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <ng-container [ngSwitch]="true">
      <li *ngSwitchCase="elt >= 0 && elt < 10">
        {{ elt }} : échec
      </li>
      <li *ngSwitchCase="elt >= 10 && elt < 13">
        {{ elt }} : moyen
      </li>
      <li *ngSwitchCase="elt >= 13 && elt < 16">
        {{ elt }} : bien
      </li>
      <li *ngSwitchCase="elt >= 16">
        {{ elt }} : très bien
      </li>
      <li *ngSwitchDefault>
        autre
      </li>
    </ng-container>
  </ng-container>
</ul>
```


Contrairement au `*ngIf`, plusieurs `*ngSwitchCase` peuvent être exécutée pour une même valeur. Par conséquent, le code suivant ne réalise pas le travail demandé

```
<ul>
  <ng-container *ngFor="let elt of moyennes">
    <ng-container [ngSwitch]="true">
      <li *ngSwitchCase="elt >= 0 && elt < 10">
        {{ elt }} : échec
      </li>
      <li *ngSwitchCase="elt < 13">
        {{ elt }} : moyen
      </li>
      <li *ngSwitchCase="elt < 16">
        {{ elt }} : bien
      </li>
      <li *ngSwitchCase="elt >= 16">
        {{ elt }} : très bien
      </li>
      <li *ngSwitchDefault>
        autre
      </li>
    </ng-container>
  </ng-container>
</ul>
```

Angular

Remarques

- L'objectif de l'exercice précédent était une excellente occasion de se familiariser avec les directives **Angular** telles que `*ngSwitchCase` et `*ngIf`.
- Ceci est essentiel pour comprendre et maîtriser la création d'interfaces utilisateur dynamiques dans **Angular**.
- Quand c'est possible, on préfère déplacer la logique de calcul dans les fichiers **TypeScript** (`*.component.ts`).
- Cela réduit considérablement la complexité du **HTML** et rend le code plus lisible et plus facile à maintenir.

Angular

ngPlural

- Introduit dans **Angular 10**.
- Similaire à `ngSwitch` mais n'acceptant que des valeurs numériques.
- Pas de mot-clé pour le cas par défaut, mais on peut utiliser la constante `other`.

Angular

Exemple avec ngPlural

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li>
      <ng-container [ngPlural]="elt">
        <ng-template ngPluralCase="1">
          {{ elt }} : un
        </ng-template>
        <ng-template ngPluralCase="2">
          {{ elt }} : deux
        </ng-template>
        <ng-template ngPluralCase="3">
          {{ elt }} : trois
        </ng-template>
        <ng-template ngPluralCase="other">
          {{ elt }} : autre
        </ng-template>
      </ng-container>
    </li>
  </ng-container>
</ul>
```

Angular

ngStyle

- permet de modifier dynamiquement le style d'un élément **HTML**.
- permet de récupérer des valeurs définies dans la partie `script` ou `style`.

© Achref EL MOUADJIB

Angular

ngStyle

- permet de modifier dynamiquement le style d'un élément **HTML**.
- permet de récupérer des valeurs définies dans la partie `script` ou `style`.

Remarque

N'utilisez jamais `ngStyle` dans une balise `<ng-template>` ou `<ng-container>` car ces dernières ne sont pas directement présentes dans le **DOM**.

Angular

Ajoutons les deux attributs suivants dans `app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  // le contenu précédent

  couleur = 'white';
  couleurBg = 'red';
}
```

Angular

Pour attribuer une valeur à la propriété CSS d'une balise

```
<p [style.background-color]='couleurBg'>
  {{ title }}
</p>
```

© Achref EL MOUELHI ©

Angular

Pour attribuer une valeur à la propriété CSS d'une balise

```
<p [style.background-color]='couleurBg'>
  {{ title }}
</p>
```

Pour deux propriétés

```
<p [style.background-color]='couleurBg' [style.color]='couleur'>
  {{ title }}
</p>
```

Angular

Pour attribuer une valeur à la propriété CSS d'une balise

```
<p [style.background-color]='couleurBg'>
  {{ title }}
</p>
```

Pour deux propriétés

```
<p [style.background-color]='couleurBg' [style.color]='couleur'>
  {{ title }}
</p>
```

Remarque

L'écriture précédente peut-être simplifiée en utilisant la directive `ngStyle`.

Angular

ngStyle

- permet de modifier le style d'un élément **HTML**.
- s'utilise conjointement avec le **property binding** pour récupérer des valeurs définies dans la classe.

Angular

Simplification avec `ngStyle`

```
<p [ngStyle]="{ color: couleur, backgroundColor: couleurBg }">
  {{ title }}
</p>
```

Angular

ngStyle **peut aussi prendre comme valeur un objet**

```
<p [ngStyle]="monStyle">  
  {{ title }}  
</p>
```

© Achref EL M...

Angular

ngStyle **peut aussi prendre comme valeur un objet**

```
<p [ngStyle]="monStyle">
  {{ title }}
</p>
```

Contenu de monStyle dans .component.ts

```
monStyle = { color: 'white', backgroundColor: 'skyblue' }
```

Angular

Il est possible de définir des méthodes dans `app.component.ts` qui gèrent le style d'un élément HTML

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  // le contenu précédent

  getColor(): string {
    return 'white';
  }
  getBgColor(): string {
    return 'red';
  }
}
```

Angular

Pour afficher le contenu de l'attribut `nom` dans le template avec une couleur de fond rouge

```
<p [ngStyle]="{ color: getColor(), bgColor: getBgColor() }">
  {{ title }}
</p>
```


Angular

ngClass

- permet d'attribuer de nouvelles classes d'un élément **HTML**.
- s'utilise conjointement avec le **property binding** pour récupérer des valeurs définies dans la classe ou dans la feuille de style.

© Achref EL M...

Angular

ngClass

- permet d'attribuer de nouvelles classes d'un élément **HTML**.
- s'utilise conjointement avec le **property binding** pour récupérer des valeurs définies dans la classe ou dans la feuille de style.

Remarque

N'utilisez jamais `ngClass` dans une balise `<ng-template>` ou `<ng-container>` car ces dernières ne sont pas directement présentes dans le **DOM**.

Considérons le contenu suivant de `app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  // le contenu précédent
}
```

© Achref EL M...

Considérons le contenu suivant de `app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  // le contenu précédent
}
```

On définit deux classes rouge et bleu dans `app.component.css`

```
.rouge {
  color: red;
}
.bleu {
  color: blue;
}
.gras {
  font-weight: bold;
}
```

Angular

Pour associer la classe rouge à la balise <p>

```
<p [ngClass]="{ 'rouge': true }">
  {{ title }}
</p>
```

© Achref EL MOUELHI ©

Angular

Pour associer la classe `rouge` à la balise `<p>`

```
<p [ngClass]="{ 'rouge': true }">
  {{ title }}
</p>
```

On peut aussi appeler une méthode dans la directive `ngClass`

```
<p [ngClass]="{ 'rouge': afficherEnRouge () }">
  {{ title }}
</p>
```

Angular

Pour associer la classe `rouge` à la balise `<p>`

```
<p [ngClass]="{ 'rouge': true }">
  {{ title }}
</p>
```

On peut aussi appeler une méthode dans la directive `ngClass`

```
<p [ngClass]="{ 'rouge': afficherEnRouge () }">
  {{ title }}
</p>
```

Définissons la méthode `afficherEnRouge` dans `app.component.ts`

```
afficherEnRouge(): boolean {
  return this.couleur === 'blue' ? true : false;
}
```

Angular

Ainsi, on peut faire aussi un affichage conditionnel

```
<p [ngClass]=" {'rouge': nom == 'wick' }">
  {{ title }}
</p>
```

© Achref EL MOUELHI ©

Angular

Ainsi, on peut faire aussi un affichage conditionnel

```
<p [ngClass]=" { 'rouge': nom == 'wick' }">
  {{ title }}
</p>
```

Ou encore (le paragraphe sera affiché en rouge si `nom` contient `wick`, en bleu sinon)

```
<p [ngClass]="{ 'rouge': title == 'ng', 'bleu': title != 'ng' }">
  {{ title }}
</p>
```

Angular

Ainsi, on peut faire aussi un affichage conditionnel

```
<p [ngClass]=" { 'rouge': nom == 'wick' }">
  {{ title }}
</p>
```

Ou encore (le paragraphe sera affiché en rouge si `nom` contient `wick`, en bleu sinon)

```
<p [ngClass]="{ 'rouge': title == 'ng', 'bleu': title != 'ng' }">
  {{ title }}
</p>
```

Plusieurs classes peuvent être associées à un même élément

```
<p [ngClass]="{ 'rouge': title == 'ng', 'bleu': title != 'ng', 'gras':
  title.length!= 2 }">
  {{ title }}
</p>
```

Angular

On peut aussi utiliser les expressions ternaires

```
<p [ngClass]="nom == 'wick' ? 'rouge' : 'bleu'">
  {{ title }}
</p>
```

Angular

Exercice

Utiliser `ngClass` pour afficher en bleu les éléments pairs du tableau précédent (`tab`) et en rouge les éléments impairs.

Angular

Première solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li [ngClass]="{'rouge': elt % 2 !== 0, 'bleu': elt % 2 == 0}">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

© Achref EL MOU

Angular

Première solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li [ngClass]="{'rouge': elt % 2 !== 0, 'bleu': elt % 2 == 0}">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

Deuxième solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li [ngClass]="elt % 2 == 0 ? 'bleu' : 'rouge'">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

Angular

Troisième solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li [ngClass]="{'rouge': !estPair(elt), 'bleu':
      estPair(elt)}">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

© Achille

Angular

Troisième solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li [ngClass]="{'rouge': !estPair(elt), 'bleu':
      estPair(elt)}">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

Dans `app.component.ts`, on définit la méthode `estPair`

```
estPair(elt: number): boolean {
  return elt % 2 === 0;
}
```


Angular

Considérons la liste stagiaires précédente

```
stagiaires: Array<Stagiaire> = [  
  new Stagiaire(100, 'Wick', 'John'),  
  new Stagiaire(101, 'Abruzzi', 'John'),  
  new Stagiaire(102, 'Marley', 'Bob'),  
  new Stagiaire(103, 'Segal', 'Steven')  
];
```

© Achref EL

Angular

Considérons la liste `stagiaires` précédente

```
stagiaires: Array<Stagiaire> = [  
  new Stagiaire(100, 'Wick', 'John'),  
  new Stagiaire(101, 'Abruzzi', 'John'),  
  new Stagiaire(102, 'Marley', 'Bob'),  
  new Stagiaire(103, 'Segal', 'Steven')  
];
```

Exercice

Écrire un script **Angular** qui permet d'afficher dans une liste **HTML** les éléments du tableau `stagiaires` (afficher uniquement nom et prénom). Les éléments d'indice pair seront affichés en rouge, les impairs en bleu.

Angular

Première solution

```
<ul>
  <ng-container *ngFor="let elt of stagiaires; let isEven =
    even; let isOdd = odd">
    <li [ngClass]="{'rouge': isEven, 'bleu': isOdd}">
      {{ elt.nom }} {{ elt.prenom }}
    </li>
  </ng-container>
</ul>
```

© Achref EL

Angular

Première solution

```
<ul>
  <ng-container *ngFor="let elt of stagiaires; let isEven =
    even; let isOdd = odd">
    <li [ngClass]="{'rouge': isEven, 'bleu': isOdd}">
      {{ elt.nom }} {{ elt.prenom }}
    </li>
  </ng-container>
</ul>
```

Deuxième solution

```
<ul>
  <ng-container *ngFor="let elt of stagiaires; let i = index;">
    <li [ngClass]="i % 2 != 0 ? 'rouge' : 'bleu'">
      {{ elt.nom + " " + elt.prenom }}
    </li>
  </ng-container>
</ul>
```

Angular

Troisième solution

```
<ul>
  <ng-container *ngFor="let stagiaire of stagiaires">
    <li [ngStyle]="{color: getNextColor()}">
      {{ stagiaire.prenom }} {{ stagiaire.nom }}
    </li>
  </ng-container>
</ul>
```

© Achref EL

Angular

Troisième solution

```
<ul>
  <ng-container *ngFor="let stagiaire of stagiaires">
    <li [ngStyle]="{color: getNextColor()}">
      {{ stagiaire.prenom }} {{ stagiaire.nom }}
    </li>
  </ng-container>
</ul>
```

Dans `app.component.ts`, on définit la méthode `getNextColor`

```
couleur = 'blue';
getNextColor(): string {
  this.couleur = this.couleur === 'red' ? 'blue' : 'red';
  return this.couleur;
}
```

Control Flow Syntax

- Introduit dans **Angular 17**
- Ayant comme objectif de remplacer les directives structurales
 - `@for` pour remplacer `*ngFor`
 - `@if` pour remplacer `*ngIf`
 - `@switch` pour remplacer `*ngSwitch`

Pour essayer Control Flow Syntax dans un projet existant, il faut utiliser la migration suivante

```
ng generate @angular/core:control-flow
```


Angular

Afficher les éléments du tableau `tab` en utilisant `*ngFor`

```
<ul>
  @for (elt of tab; track elt) {
    <li> {{ elt }} </li>
  }
</ul>
```

© Achref EL MOU

Angular

Afficher les éléments du tableau `tab` en utilisant `*ngFor`

```
<ul>
  @for (elt of tab; track elt) {
    <li> {{ elt }} </li>
  }
</ul>
```

`track elt`

- permet de spécifier la clé utilisée pour associer les éléments du tableau aux vues dans le **DOM**.
- permet à **Angular** d'exécuter un ensemble minimal d'opérations sur le **DOM** lorsque des éléments sont ajoutés, supprimés ou déplacés dans une collection.

Angular

Et pour avoir l'indice de l'itération courante, on peut utiliser la variable contextuelle `$index`

```
<ul>
  @for (elt of tab; track elt) {
    <li>{{ $index }} {{ elt }} </li>
  }
</ul>
```

© Achref EL

Angular

Et pour avoir l'indice de l'itération courante, on peut utiliser la variable contextuelle `$index`

```
<ul>
  @for (elt of tab; track elt) {
    <li>{{ $index }} {{ elt }} </li>
  }
</ul>
```

ou

```
<ul>
  @for (elt of tab; let i = $index; track i) {
    <li>{{ i }} {{ elt }} </li>
  }
</ul>
```

Angular

Autres variables contextuelles

- `$first` : retourne `true` si l'élément courant est le premier de la liste, `false` sinon.
- `$last` : retourne `true` si l'élément courant est le dernier de la liste, `false` sinon.
- `$even` : retourne `true` si l'indice de l'élément courant est pair, `false` sinon.
- `$odd` : retourne `true` si l'indice de l'élément courant est impair, `false` sinon.
- `$count` : retourne le nombre total d'éléments dans la collection.

Angular

Exemple avec \$first

```
<ul>
  @for (elt of tab; let i = $index; let premier = $first ;
    track i) {
    <li>{{ first }} {{ i }} {{ elt }} </li>
  }
</ul>
```

Angular

Le bloc optionnel `@empty` est exécuté lorsque la liste est vide

```
<ul>
  @for (elt of tab; track i) {
    <li>{{ elt }} </li>
  } @empty {
    La liste ne contient aucun élément.
  }
</ul>
```

Angular

Considérons la liste suivante (à déclarer dans `app.component.ts`)

```
stagiaires: Array<Stagiaire> = [  
  new Stagiaire(100, 'Wick', 'John'),  
  new Stagiaire(101, 'Abruzzi', 'John'),  
  new Stagiaire(102, 'Marley', 'Bob'),  
  new Stagiaire(103, 'Segal', 'Steven')  
];
```

© Achref EL

Angular

Considérons la liste suivante (à déclarer dans `app.component.ts`)

```
stagiaires: Array<Stagiaire> = [  
  new Stagiaire(100, 'Wick', 'John'),  
  new Stagiaire(101, 'Abruzzi', 'John'),  
  new Stagiaire(102, 'Marley', 'Bob'),  
  new Stagiaire(103, 'Segal', 'Steven')  
];
```

Exercice

Écrire un script **Angular** qui permet d'afficher dans une liste **HTML** les nom et prénom de chaque stagiaire de la liste `stagiaires`.

Angular

Solution

```
<ul>
  @for (st of stagiaires; track st.num) {
    <li> {{ st.prenom }} {{ st.nom }} </li>
  }
</ul>
```

Angular

Remarques

- `@for` nous permet d'itérer sur les tableaux, mais pas sur les objets.
- Il est possible d'itérer sur les objets en utilisant le pipe `keyvalue`.

Angular

Exemple d'utilisation de `keyvalue` dans des boucles imbriquées

```
<ul>
  @for (st of stagiaires; track st.num) {
    @for (elt of st | keyvalue ; track elt) {
      <li> {{ elt.key }} {{ elt.value }}</li>
    }
  }
</ul>
```

Angular

Exemple avec @if

```
<ul>
  <li>
    @if(tab[1] % 2 != 0) {
      {{ tab[1] }} est impair
    }
  </li>
</ul>
```

Angular

Exemple avec @else

```
<ul>
  <li>
    @if(tab[1] % 2 != 0) {
      {{ tab[1] }} est impair
    } @else {
      {{ tab[1] }} est pair
    }
  </li>
</ul>
```

Angular

Exercice

Utiliser les Control Flow Syntax d'**Angular** pour afficher sous forme d'une liste **HTML** les éléments du tableau `moyennes: number[] = [18, 5, 11, 15]` avec le message suivant :

- Si $0 \leq \text{valeur} < 10$ alors échec,
- Si $10 \leq \text{valeur} < 13$ alors moyen,
- Si $13 \leq \text{valeur} < 16$ alors bien,
- Sinon très bien.

Angular

Exemple avec @switch

```
<ul>
  @for(elt of tab; track $index) {
    <li>
      @switch(elt) {
        @case(1) {
          {{ elt }} = un
        }
        @case(2) {
          {{ elt }} = deux
        }
        @case(3) {
          {{ elt }} = trois
        }
        @default {
          {{ elt }} = autre
        }
      }
    </li>
  }
</ul>
```


Angular

Refaire cet exercice avec @switch

Utiliser les directives **Angular** pour afficher sous forme d'une liste les éléments du tableau `moyennes = [18, 5, 11, 15]` avec le message suivant :

- Si $0 \leq \text{valeur} < 10$ alors échec
- Si $10 \leq \text{valeur} < 13$ alors moyen
- Si $13 \leq \text{valeur} < 16$ alors bien
- Sinon très bien

Angular

Créons le composant `reactive.component.ts` avec le code suivant

```
export class ReactiveComponent {
  value1 = 2
  value2 = 5
  resultat = 0
  constructor() {
    setInterval(
      () => this.value1 += 10,
      5000
    )
    this.resultat = this.value1 + this.value2
  }
}
```

Angular

Créons le composant `reactive.component.ts` avec le code suivant

```
export class ReactiveComponent {
  value1 = 2
  value2 = 5
  resultat = 0
  constructor() {
    setInterval(
      () => this.value1 += 10,
      5000
    )
    this.resultat = this.value1 + this.value2
  }
}
```

Le contenu de `reactive.component.html` est le suivant

```
<p>
  {{ value1 }} + {{ value2 }} = {{ resultat }}
</p>
```

Angular

Question

Que sera le résultat de ce code ?

© Achref EL MOUELHANI

Angular

Question

Que sera le résultat de ce code ?

Résultat attendu

- Au chargement de la page $2 + 3 = 5$
- 5 secondes plus tard, 3 sera remplacé par 13
- Contenu attendu : $2 + 13 = 15$

Résultat obtenu

- Au chargement de la page $2 + 3 = 5$
- 5 secondes plus tard, 3 est remplacé par 13
- Contenu affiché : $2 + 13 = 5$

Angular

Conclusion

Quand valeur1 ou valeur2 change, resultat ne se met pas à jour.

© Achref EL MOUELHI ©

Angular

Conclusion

Quand valeur1 ou valeur2 change, resultat ne se met pas à jour.

Solution

Utiliser les valeurs réactives (signaux).

Angular

Conclusion

Quand valeur1 ou valeur2 change, resultat ne se met pas à jour.

Solution

Utiliser les valeurs réactives (signaux).

Signal

- une enveloppe autour d'une valeur qui peut informer les consommateurs intéressés lorsque cette valeur change.
- peut contenir n'importe quel type de valeur : primitive ou complexe.

Angular propose deux types de signaux

- **Writable Signals** : valeur modifiable via les méthodes
 - `set(newValue)` ou
 - `update(callback)`
- **Computed Signals** (en lecture seule) : obtient une valeur à partir d'un ou plusieurs autres signaux.

Angular

Commençons par initialiser `value1` et `value2` avec la fonction `signal`

```
value1 = signal(2)
```

```
value2 = signal(5)
```

© Achref EL MOUELHI ©

Angular

Commençons par initialiser `value1` et `value2` avec la fonction `signal`

```
value1 = signal(2)
value2 = signal(5)
```

Remarque

`value1` et `value2` sont de type `WritableSignal`.

Angular

Commençons par initialiser `value1` et `value2` avec la fonction `signal`

```
value1 = signal(2)
value2 = signal(5)
```

Remarque

`value1` et `value2` sont de type `WritableSignal`.

On peut donc typer ainsi

```
value1: WritableSignal<number> = signal(2)
value2: WritableSignal<number> = signal(5)
```

Angular

Préparons l'attribut `resultat` et initialisons le à 0

```
resultat: Signal<number> = signal(0)
```

© Achref EL MOUËL

Angular

Préparons l'attribut `resultat` et initialisons le à 0

```
resultat: Signal<number> = signal(0)
```

Remarque

`resultat` est de type `Signal<number>` donc il est en lecture-seule.

Angular

Dans le constructeur, utilisons `update` pour modifier `value1` et `computed` pour recalculer la valeur de `resultat` après chaque changement de valeur de `value1` ou `value2`

```
constructor() {  
  setInterval(  
    () => this.value1.update(val => val += 10),  
    2000  
  )  
  this.resultat = computed(() => this.value1() + this.value2())  
}
```

© Achre

Angular

Dans le constructeur, utilisons `update` pour modifier `value1` et `computed` pour recalculer la valeur de `resultat` après chaque changement de valeur de `value1` ou `value2`

```
constructor() {  
  setInterval(  
    () => this.value1.update(val => val += 10),  
    2000  
  )  
  this.resultat = computed(() => this.value1() + this.value2())  
}
```

Les imports nécessaires

```
import { Component, Signal, WritableSignal, computed, signal } from '  
@angular/core';
```


Angular

Pour accéder aux valeurs d'un signal, on utilise le getter (on rajoute donc les parenthèses)

```
<p>  
  {{ value1() }} + {{ value2() }} = {{ resultat() }}  
</p>
```

Angular

Pour suivre l'état d'un signal, on utilise `effect`

```
export class ReactiveComponent {
  value1: WritableSignal<number> = signal(2)
  value2: WritableSignal<number> = signal(5)
  resultat: Signal<number> = signal(0)
  constructor() {
    setInterval(
      () => this.value1.update(val => val += 10),
      2000
    )
    this.resultat = computed(() => this.value1() + this.value2())
  }
  r = effect(() => console.log('effet', this.value1(), this.resultat()))
}
```

© Acti

Angular

Pour suivre l'état d'un signal, on utilise `effect`

```
export class ReactiveComponent {
  value1: WritableSignal<number> = signal(2)
  value2: WritableSignal<number> = signal(5)
  resultat: Signal<number> = signal(0)
  constructor() {
    setInterval(
      () => this.value1.update(val => val += 10),
      2000
    )
    this.resultat = computed(() => this.value1() + this.value2())
  }
  r = effect(() => console.log('effet', this.value1(), this.resultat()))
}
```

Remarque

`effect` observe `resultat` et `value1`. Le changement de valeur de tout autre signal ne permettra pas de le déclencher.

Angular

`effect` peut-être utilisé dans les cas suivants

- enregistrer la valeur d'un signal à l'aide d'un framework de journalisation.
- exporter la valeur d'un signal vers le **Web Storage** ou un **Cookie**.
- enregistrer la valeur d'un signal dans la base de données.

Angular

Pour appliquer un traitement particulier à la destruction de l'effect

```
r = effect((onCleanup) => {  
  console.log('effet', this.value1(), this.resultat())  
  onCleanup(() => {  
    console.log("Traitement appliqué à la destruction de l'effect");  
  });  
})
```