

# Angular : formulaire



**Source :**

Achref El Mouelhi

# Plan

- 1 Définition
- 2 Évènements
- 3 Template-driven forms
  - Liaison (binding) bidirectionnelle
  - Validation de formulaire
  - Soumission de formulaire
- 4 Reactive forms
  - FormControl
  - FormGroup
  - Soumission de formulaire
  - Validation de formulaire
  - Affichage de messages d'erreur
  - FormGroup imbriqué
  - FormBuilder
  - FormArray

# Angular

## Définition : Un formulaire

- un outil graphique que nous créons avec le langage de description HTML
- il permet à l'utilisateur de saisir des données
- et de les envoyer vers une autre page, vers une base de données...

# Angular

## Alors pourquoi **Angular** ?

Angular nous facilite

- la récupération des données saisies
- la validation et le contrôle des valeurs saisies
- la gestion d'erreurs
- ...

# Angular

## Que propose **Angular** ?

- Template-driven forms : utilisant `FormsModule`, facile et conseillé pour les formulaires simples
- Reactive forms basé sur `ReactiveFormsModule` : robuste, évolutif et conçu pour les applications nécessitant des contrôles particuliers
  - Form Group
  - Form Builder

# Angular

## Un évènement

- action appliquée par l'utilisateur ou simulée par le développeur sur un `.component.html`
- un évènement déclenché  $\Rightarrow$  une méthode, dans le `.component.ts` associé, exécutée

# Angular

## Un évènement

- action appliquée par l'utilisateur ou simulée par le développeur sur un `.component.html`
- un évènement déclenché  $\Rightarrow$  une méthode, dans le `.component.ts` associé, exécutée

## Pour commencer

- créer un composant `formulaire`
- créer un chemin `/formulaire` permettant d'afficher ce composant

# Angular

## Le fichier `formulaire.component.ts`

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

@Component({
  selector: 'app-formulaire',
  templateUrl: './formulaire.component.html',
  styleUrls: ['./formulaire.component.css']
})
export class FormulaireComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }
}
```

## Le fichier

`formulaire.component.html`

```
<p>
  formulaire works!
</p>
```



# Angular

## Le fichier `formulaire.component.ts`

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

@Component({
  selector: 'app-formulaire',
  templateUrl: './formulaire.component.html',
  styleUrls: ['./formulaire.component.css']
})
export class FormulaireComponent implements OnInit {

  constructor() { }

  ngOnInit() { }
  direBonjour() {
    console.log("Bonjour");
  }
}
```

## Le fichier

`formulaire.component.html`

```
<div>
  <button (click)="
    direBonjour()">
    cliquer
  </button>
</div>
```

# Angular

## (eventBinding)

La valeur de l'attribut évènement situé entre () sera le nom d'une méthode de `FormulaireComponent`

# Angular

## (eventBinding)

La valeur de l'attribut évènement situé entre () sera le nom d'une méthode de `FormulaireComponent`

## Explication

- En cliquant sur le bouton `cliquer`, la méthode `direBonjour()` est exécutée.
- `Bonjour` est affiché dans la console

# Angular

## (eventBinding)

La valeur de l'attribut événement situé entre `()` sera le nom d'une méthode de `FormulaireComponent`

## Explication

- En cliquant sur le bouton `cliquer`, la méthode `direBonjour()` est exécutée.
- `Bonjour` est affiché dans la console

Et si on veut récupérer la valeur saisie dans une zone texte et l'afficher dans une autre partie du composant.

Le fichier `formulaire.component.ts`

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

@Component({
  selector: 'app-formulaire',
  templateUrl: './formulaire.component.html',
  styleUrls: ['./formulaire.component.css']
})
export class FormulaireComponent implements OnInit {
  result = "";
  constructor() { }

  ngOnInit() { }
  direBonjour(nom:string) {
    this.result = nom;
  }
}
```

`#nom` est une variable locale contenant les attributs de cet élément HTML. Les variables locales peuvent être utilisées avec tout élément HTML, `<p>`, `<h>`...

## Le fichier

`formulaire.component.html`

```
<div>
  <input type=text #nom>
</div>
<div>
  <button (click)="
    direBonjour(nom.
      value)">
    cliquer
  </button>
</div>
<div>
  Bonjour {{ result }}
</div>
```

# Angular

## Remarque

- Il est possible de simplifier le code précédant en utilisant le `two way binding`

# Angular

## Plusieurs formes de binding

- `{{ interpolation }}` : permet de récupérer la valeur d'un attribut déclarée dans le `.component.ts`
- `[ one way binding ]` : permet aussi de récupérer la valeur d'un attribut déclarée dans le `.component.ts`
- `( event binding )` : permet au `.component.ts` de récupérer des valeurs passées par le `.component.html`

# Angular

## Plusieurs formes de binding

- `{{ interpolation }}` : permet de récupérer la valeur d'un attribut déclarée dans le `.component.ts`
- `[ one way binding ]` : permet aussi de récupérer la valeur d'un attribut déclarée dans le `.component.ts`
- `( event binding )` : permet au `.component.ts` de récupérer des valeurs passées par le `.component.html`

`{{ interpolation }}` **est un raccourci de** `[ one way binding ]`

```
<p [textContent]= "result"></p>
<p> {{ result }} </p>
<!-- Les deux écritures sont équivalentes -->
```



# Angular

Il est possible de combiner `one way binding` et `event binding`

- Résultat : `two way binding`
- Un changement de valeur dans `.component.ts` sera aperçu dans `.component.html` et un changement dans `.component.html` sera reçu dans `.component.ts`

# Angular

Il est possible de combiner `one way binding` et `event binding`

- Résultat : `two way binding`
- Un changement de valeur dans `.component.ts` sera aperçu dans `.component.html` et un changement dans `.component.html` sera reçu dans `.component.ts`

`two way binding`

- Pour la liaison bidirectionnelle, il nous faut la propriété `ngModel`
- Pour pouvoir utiliser la propriété `ngModel`, il faut ajouter le module `FormsModule` dans `app.module.ts`

# Angular

## Nouveau contenu d'app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
//+ les autres imports

@NgModule({
  declarations: [
    AppComponent,
    AdresseComponent,
    PersonneComponent,
    FormulaireComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

## Le fichier formulaire.component.ts

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

@Component({
  selector: 'app-formulaire',
  templateUrl: './formulaire.component.html',
  styleUrls: ['./formulaire.component.css']
})
export class FormulaireComponent implements
  OnInit {
  nom = "";
  result = "";
  constructor() { }

  ngOnInit() { }
  direBonjour() {
    this.result = this.nom;
  }
}
```

## Le fichier

formulaire.component.html

```
<div>
  <input type=text [(
    ngModel)] =nom>
</div>
<div>
  <button (click)="
    direBonjour()">
    cliquer
  </button>
</div>
<div>
  Bonjour {{ result }}
</div>
```

Nous n'avons pas besoin de cliquer pour envoyer la valeur saisie dans le champ texte, elle est mise à jour simultanément dans la classe quand elle est modifiée dans la vue.

# Angular

## Le fichier `formulaire.component.ts`

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

@Component({
  selector: 'app-formulaire',
  templateUrl: './formulaire.component.html',
  styleUrls: ['./formulaire.component.css']
})
export class FormulaireComponent implements
  OnInit {
  nom = "";
  constructor() { }

  ngOnInit() { }
}
```

## Le fichier

`formulaire.component.html`

```
<div>
  <input type=text [(
    ngModel)]=nom>
</div>
<div>
  Bonjour {{ nom }}
</div>
```

# Angular

**Commençons par créer une interface `personne` qui va nous servir de modèle**

```
ng generate interface interfaces/personne
```

# Angular

**Commençons par créer une interface `personne` qui va nous servir de modèle**

```
ng generate interface interfaces/personne
```

**On peut aussi utiliser le raccourci**

```
ng g i interfaces/personne
```

# Angular

Commençons par créer une interface `personne` qui va nous servir de modèle

```
ng generate interface interfaces/personne
```

On peut aussi utiliser le raccourci

```
ng g i interfaces/personne
```

Mettons à jour le contenu de `personne.ts`

```
export interface Personne {  
  id?: number;  
  nom?: string;  
  prenom?: string;  
}
```



# Angular

Considérant le formulaire `formulaire.component.html`

```
<form>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.nom>
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=personne.
              prenom>
  </div>
  <div>
    <button>
      Ajouter
    </button>
  </div>
</form>
```

# Angular

Considérant le formulaire `formulaire.component.html`

```
<form>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.nom>
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=personne.
              prenom>
  </div>
  <div>
    <button>
      Ajouter
    </button>
  </div>
</form>
```

**Pour utiliser `ngModel` dans un formulaire, il faut définir une valeur pour l'attribut `name` de chaque élément du formulaire.**

# Angular

## Le fichier `formulaire.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { Personne } from '../interfaces/personne';

@Component({
  selector: 'app-formulaire',
  templateUrl: './formulaire.component.html',
  styleUrls: ['./formulaire.component.css']
})
export class FormulaireComponent implements OnInit {

  personne: Personne = { };

  constructor() { }

  ngOnInit() { }
}
```

# Angular

Pour soumettre le formulaire, il faut qu'il soit valide. Supposant que

- les deux zones textes sont obligatoires
- le bouton doit être initialement désactivé, on l'active que lorsque le formulaire est valide

# Angular

## Commençons par rendre les deux zones textes obligatoires

```
<form>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.
          nom required>
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=
             personne.prenom required>
  </div>
  <div>
    <button>
      Ajouter
    </button>
  </div>
</form>
```

# Angular

## Désactivons le bouton

```
<form>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.
          nom required>
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=
             personne.prenom required>
  </div>
  <div>
    <button disabled>
      Ajouter
    </button>
  </div>
</form>
```

# Angular

## Question : comment réactiver le bouton ?

- Écrire une fonction JavaScript pour tester si les deux champs ne sont pas vides
- Écrire une méthode dans la classe qui vérifiera à chaque saisie si les deux champs ne sont pas vides pour réactiver le bouton

# Angular

## Question : comment réactiver le bouton ?

- Écrire une fonction JavaScript pour tester si les deux champs ne sont pas vides
- Écrire une méthode dans la classe qui vérifiera à chaque saisie si les deux champs ne sont pas vides pour réactiver le bouton

## Avec les directives **Angular**, il y a plus simple

- Utiliser la directive `ngForm` pour créer une variable locale associée au formulaire
- Exploiter la propriété `valid` de `ngForm` pour valider le formulaire



# Angular

## Pour le réactiver

```
<form #monForm=ngForm>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.
          nom required>
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=
             personne.prenom required>
  </div>
  <div>
    <button [disabled]=!monForm.valid>
      Ajouter
    </button>
  </div>
</form>
```

# Angular

Question : et si on veut afficher en rouge les champs obligatoires non-renseignés ?

- Utiliser les variables locales
- Exploiter les propriétés CSS fournies par Angular

# Angular

Utilisons les variables locales et affichons les classes CSS associées attribuées par Angular

```
<form #monForm=ngForm>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.nom
          required #nom>
  </div>
  {{ nom.className}}
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=personne.
              prenom required #prenom>
  </div>
  {{ prenom.className}}
  <div>
    <button [disabled]=!monForm.valid>
      ajouter
    </button>
  </div>
</form>
```

# Angular

## Les classes CSS affichées pour les deux champs

- `ng-untouched` : classe Angular appliquée quand le champ n'est pas touché (son inverse est `ng-touched`)
- `ng-pristine` : classe Angular appliquée quand le champ est vide (son inverse est `ng-dirty`)
- `ng-invalid` : classe Angular appliquée quand le champ n'est pas valid (son inverse est `ng-valid`)

# Angular

## Nettoyons le code précédent

```
<form #monForm=ngForm>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.nom
          required #nom>
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=personne.
             prenom required #prenom>
  </div>
  <div>
    <button [disabled]=!monForm.valid>
      ajouter
    </button>
  </div>
</form>
```

# Angular

## Définissons des propriétés pour quelques classes CSS fournies par Angular

```
.ng-invalid:not(form) {  
    border-left: 5px solid red;  
}  
  
.ng-valid:not(form) {  
    border-left: 5px solid green;  
}
```

# Angular

On peut aussi afficher un message en cas de violation de contrainte

```
<form #monForm=ngForm>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.nom required
          #nom="ngModel">
  </div>
  <div [hidden]="nom.valid">
    Le nom est obligatoire
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=personne.prenom
              required #prenom="ngModel">
  </div>
  <div [hidden]="prenom.valid">
    Le prénom est obligatoire
  </div>
  <div>
    <button [disabled]=!monForm.valid>
      ajouter
    </button>
  </div>
</form>
```

# Angular

## Pour ne pas afficher les messages d'erreur au chargement de la page

```
<form #monForm=ngForm>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.nom required #
      nom="ngModel">
  </div>
  <div [hidden]="nom.valid || nom.pristine">
    Le nom est obligatoire
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=personne.prenom
      required #prenom="ngModel">
  </div>
  <div [hidden]="prenom.valid || prenom.pristine">
    Le prénom est obligatoire
  </div>
  <div>
    <button [disabled]=!monForm.valid>
      ajouter
    </button>
  </div>
</form>
```



# Angular

Pour soumettre un formulaire, on utilise la directive `ngSubmit`

```
<form #monForm=ngForm (ngSubmit)=ajouterPersonne()>
  <div>
    Nom : <input type=text name=nom [(ngModel)]=personne.nom required #
          nom="ngModel">
  </div>
  <div [hidden]="nom.valid || nom.pristine">
    Le nom est obligatoire
  </div>
  <div>
    Prénom : <input type=text name=prenom [(ngModel)]=personne.prenom
             required #prenom="ngModel">
  </div>
  <div [hidden]="prenom.valid || prenom.pristine">
    Le prénom est obligatoire
  </div>
  <div>
    <button [disabled]=!monForm.valid>
      ajouter
    </button>
  </div>
</form>
```

# Angular

Le fichier `formulaire.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { Personne } from '../interfaces/personne';

@Component({
  selector: 'app-formulaire',
  templateUrl: './formulaire.component.html',
  styleUrls: ['./formulaire.component.css']
})
export class FormulaireComponent implements OnInit {
  personnes: Array<Personne> = [];
  personne: Personne = { };

  constructor() { }
  ngOnInit() { }

  ajouterPersonne() {
    this.personnes.push({ ...this.personne });
    this.personne.nom = '';
    this.personne.prenom = '';
    console.log(this.personnes);
  }
}
```

# Angular

## Exercice 1

- Modifier le composant `formulaire` pour afficher les personnes ajoutées en-dessous du formulaire
- À chaque ajout, la nouvelle personne ajoutée apparaît comme dernier élément de la liste des personnes (affichée en-dessous du formulaire)

# Angular

## Exercice 2

- Créez un composant `calculette` (n'oubliez pas de lui associer une route `calculette`)
- Dans `calculette.component.html`, définissez deux champs `input` de type `number` et quatre boutons : un pour chaque opération arithmétique
- Le résultat sera affiché en-dessous du formulaire.

# Angular

## Pour commencer

- créer un composant `form`
- créer un chemin `/form` permettant d'afficher ce composant
- ajouter `ReactiveFormsModule` dans la section `imports` de `app.module.ts`

# Angular

## FormControl

myModule

- Une classe **Angular**
- Permettant d'associer un attribut de composant à un champ de formulaire défini dans le template associé afin de faciliter
  - le binding
  - le contrôle et la validation

# Angular

Dans `form.component.ts`, déclarons un attribut nom de type `FormControl`

```
import { Component, OnInit } from '@angular/core';  
import { FormControl } from '@angular/forms';
```

```
@Component({  
  selector: 'app-form',  
  templateUrl: './form.component.html',  
  styleUrls: ['./form.component.css']  
})  
export class FormComponent implements OnInit {  
  nom = new FormControl('');  
  constructor() { }  
  ngOnInit() { }  
}
```


nom

Dans `form.component.html`, on aura un champ de formulaire associé à l'objet `nom`.

# Angular

Dans `form.component.html`, on ajoute un champ associé à l'attribut `nom`

```
<div>  
  Nom :  
  <input type="text" [formControl]="nom">  
</div>
```





# Angular

Dans `form.component.html`, on ajoute un champ associé à l'attribut `nom`

```
<div>
  Nom :
  <input type="text" [formControl]="nom">
</div>
```

On peut ajouter un bouton avec un `event binding`

```
<label>
  Nom :
  <input type="text" [formControl]="nom">
</label>
<button (click)='afficherNom()'>cliquer</button>
```

# Angular

Dans `form.component.html`, on ajoute un champ associé à l'attribut `nom`

```
<div>
  Nom :
  <input type="text" [formControl]="nom">
</div>
```

On peut ajouter un bouton avec un `event binding`

```
<label>
  Nom :
  <input type="text" [formControl]="nom">
</label>
<button (click)='afficherNom()'>cliquer</button>
```

N'oublions pas de définir la méthode `afficherNom()` dans `form.component.ts`.

Dans `form.component.ts`, on ajoute la méthode `afficherNom()`

```
import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {

  nom = new FormControl('');

  constructor() { }
  ngOnInit() { }
  afficherNom() {
    console.log(this.nom.value);
  }
}
```

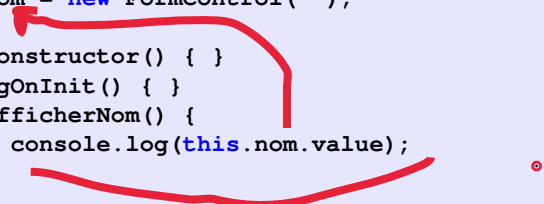
Dans `form.component.ts`, on ajoute la méthode `afficherNom()`

```
import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {

  nom = new FormControl('');

  constructor() { }
  ngOnInit() { }
  afficherNom() {
    console.log(this.nom.value);
  }
}
```



En cliquant sur le bouton, la valeur saisie dans le champ texte sera affichée dans la console.

Dans `form.component.ts`, on peut utiliser le constructeur de `FormControl` pour définir une valeur initiale à afficher au chargement du composant

```
import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';
```

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

  nom = new FormControl('wick');

  constructor() { }
  ngOnInit() { }

  afficherNom() {
    console.log(this.nom.value);
  }
}
```

# Angular

## FormGroup

- Une classe **Angular**
- Composée de plusieurs objets de type `FormControl`



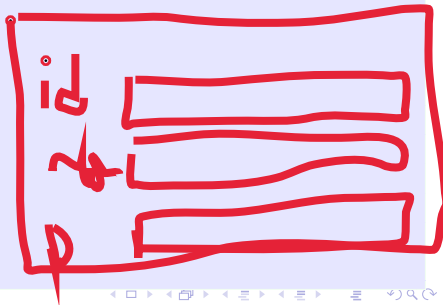
Dans `form.component.ts`, commençons par définir un objet de type `FormGroup`

```
import { Component, OnInit } from '@angular/core';  
import { FormControl, FormGroup } from '@angular/forms';
```

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

```
  personneForm = new FormGroup({  
    id: new FormControl(''),  
    nom: new FormControl(''),  
    prenom: new FormControl('')  
  });
```

```
  constructor() { }  
  ngOnInit() { }  
}
```



# Angular

Dans `form.component.html`, créons maintenant le formulaire prenant les trois éléments déclarés dans le `FormGroup`

```
<form [formGroup]="personneForm">
  <div>
    Identifiant :
    <input type="number" formControlName="id">
  </div>
  <div>
    Nom :
    <input type="text" formControlName="nom">
  </div>
  <div>
    Prénom :
    <input type="text" formControlName="prenom">
  </div>
</form>
```



# Angular

On peut ajouter un bouton avec un event binding

```
<form [formGroup]="personneForm">
  <div>
    Identifiant :
    <input type="number" formControlName="id">
  </div>
  <div>
    Nom :
    <input type="text" formControlName="nom">
  </div>
  <div>
    Prénom :
    <input type="text" formControlName="prenom">
  </div>
  <button (click)='afficherTout()'>cliquer</button>
</form>
```

# Angular

On peut ajouter un bouton avec un event binding

```
<form [formGroup]="personneForm">
  <div>
    Identifiant :
    <input type="number" formControlName="id">
  </div>
  <div>
    Nom :
    <input type="text" formControlName="nom">
  </div>
  <div>
    Prénom :
    <input type="text" formControlName="prenom">
  </div>
  <button (click)='afficherTout()'>cliquer</button>
</form>
```

N'oublions pas de définir la méthode `afficherTout()` dans `form.component.ts`.

# Angular

Dans `form.component.ts`, ajoutons la méthode `afficherTout()`

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {

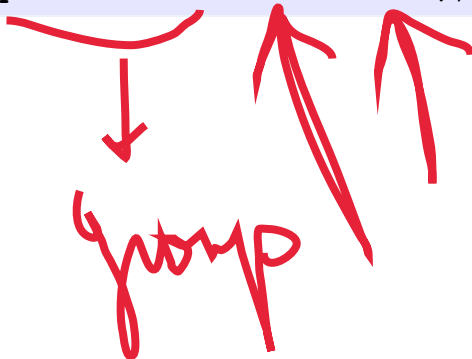
  personneForm = new FormGroup({
    id: new FormControl(''),
    nom: new FormControl(''),
    prenom: new FormControl('')
  });

  constructor() { }
  ngOnInit() { }
  afficherTout() {
    console.log(this.personneForm.value);
  }
}
```

# Angular

Pour récupérer le `FormControl` associé à `nom`

```
console.log(this.personneForm.controls.nom);
```



# Angular

**Pour récupérer le `FormControl` associé à `nom`**

```
console.log(this.personneForm.controls.nom);
```

**Ou aussi**

```
console.log(this.personneForm.get('nom'));
```



# Angular

**Pour récupérer le `FormControl` associé à `nom`**

```
console.log(this.personneForm.controls.nom);
```

**Ou aussi**

```
console.log(this.personneForm.get('nom'));
```

**Pour vider les champs d'un formulaire**

```
this.personneForm.reset();
```

On peut initialiser les champs du formulaire avec la méthode `setValue()`

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {

  personneForm = new FormGroup({
    id: new FormControl(''),
    nom: new FormControl(''),
    prenom: new FormControl('')
  });

  constructor() { }
  ngOnInit() {
    this.personneForm.setValue({nom: 'wick', prenom: 'john', id: 1});
  }
  afficherTout() {
    console.log(this.personneForm.value);
  }
}
```


**valueChanges** permet de surveiller le changement de valeur d'un champ du formulaire

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {

  personneForm = new FormGroup({
    id: new FormControl(''),
    nom: new FormControl(''),
    prenom: new FormControl('')
  });

  constructor() { }
  ngOnInit() {
    this.personneForm.controls.nom.valueChanges.subscribe(change => {
      console.log(change);
    });
  }
  afficherTout() { console.log(this.personneForm.value); }
}
```





# Angular

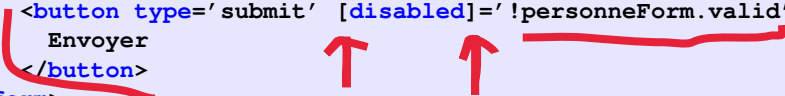
Pour la soumission de formulaire, on ajoute un event binding (`ngSubmit`) et on ajoute un bouton de type `submit`

```
<form [formGroup]="personneForm" (ngSubmit)='afficherTout()'>
  <div>
    Identifiant :
    <input type="number" formControlName="id">
  </div>
  <div>
    Nom :
    <input type="text" formControlName="nom">
  </div>
  <div>
    Prénom :
    <input type="text" formControlName="prenom">
  </div>
  <button type='submit'>Envoyer</button>
</form>
```

# Angular

Pour la validation de formulaire, on commence par désactiver le bouton tant que le formulaire n'est pas valide

```
<form [formGroup]="personneForm" (ngSubmit)='afficherTout()'>
  <div>
    Identifiant :
    <input type="number" formControlName="id">
  </div>
  <div>
    Nom :
    <input type="text" formControlName="nom">
  </div>
  <div>
    Prénom :
    <input type="text" formControlName="prenom">
  </div>
  <button type='submit' [disabled]='!personneForm.valid'>
    Envoyer
  </button>
</form>
```



The diagram illustrates the logic for disabling the submit button. A red bracket on the left groups the entire form content. Two red arrows point upwards to the `[disabled]` attribute in the button tag. A red line connects the `!personneForm.valid` expression to the `[disabled]` attribute, indicating that the button is disabled when the form is not valid.

# Angular

## Étape suivante : définir les règles de validation

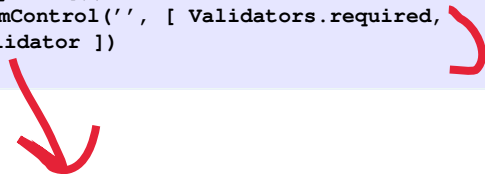
- La classe `FormControl` peut prendre deux paramètres : la valeur initiale à afficher dans le formulaire et la deuxième une règle de validation
- Pour définir une règle de validation, on peut utiliser la classe **Angular** `Validators` contenant plusieurs règles de validation

# Angular

Dans `form.component.ts`, définissons quelques règles de validation

```
import { FormControl, FormGroup, Validators } from '@angular/forms';

personneForm = new FormGroup({
  id: new FormControl('', Validators.required),
  nom: new FormControl('', [Validators.pattern(/^[A-Z][a-z]{2,10}/),
    Validators.required]),
  prenom : new FormControl('', [ Validators.required,
    checkPrenomValidator ])
});
```



# Angular

Dans `form.component.ts`, définissons quelques règles de validation

```
import { FormControl, FormGroup, Validators } from '@angular/forms';

personneForm = new FormGroup({
  id: new FormControl('', Validators.required),
  nom: new FormControl('', [Validators.pattern(/^[A-Z][a-z]{2,10}/),
    Validators.required]),
  prenom : new FormControl('', [ Validators.required,
    checkPrenomValidator ])
});
```

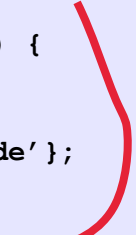
## Explication

- Le champ `id` est obligatoire
- Le champ `nom` est obligatoire et doit respecter une expression régulière qui exige que la première soit en majuscule et que le nombre de caractère soit entre 3 et 11
- Le champ `prenom` est aussi obligatoire et doit respecter une fonction (qu'on a préparée pour ça) appelée `checkPrenomValidator`

# Angular

**La fonction** `checkPrenomValidator()`

```
function checkPrenomValidator(control: FormControl)
{
  const str: string = control.value;
  if (str[0] >= 'A' && str[0] <= 'Z') {
    return null;
  } else {
    return {erreur: 'Prénom non valide'};
  }
}
```



# Angular

## Pour afficher les messages d'erreurs

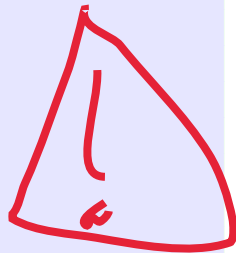
```
<form [formGroup]="personneForm" (ngSubmit)='afficherTout()'>
  <div>
    Identifiant :
    <input type="number" formControlName="id">
  </div>
  <div *ngIf="id.invalid && (id.dirty || id.touched)">
    <div *ngIf="id.errors?.required">L'identifiant est obligatoire</div>
  </div>
  <div>
    Nom :
    <input type="text" formControlName="nom">
  </div>
  <div *ngIf="nom.invalid && (nom.dirty || nom.touched)">
    <div *ngIf="nom.errors?.required">Le nom est obligatoire</div>
    <div *ngIf="nom.errors?.pattern">Première lettre en majuscule et min 3 lettres max 11</div>
  </div>
  <div>
    Prénom :
    <input type="text" formControlName="prenom">
  </div>
  <div *ngIf="prenom.invalid && (prenom.dirty || prenom.touched)">
    <div *ngIf="prenom.errors?.required">Le prénom est obligatoire</div>
    <div *ngIf="prenom.errors?.erreuer">Première lettre en majuscule</div>
  </div>
  <button type='submit' [disabled]='!personneForm.valid'>Envoyer</button>
</form>
```

# Angular

**On ne peut accéder à la propriété `invalid` ni `errors` si on ne définit pas de getter pour chaque `FormControl` dans**

`form.component.ts`

```
get nom() {  
    return this.personneForm.get('nom');  
}  
  
get id() {  
    return this.personneForm.get('id');  
}  
  
get prenom() {  
    return this.personneForm.get('prenom');  
}
```





# Angular

## Remarque

- Il est possible d'imbriquer les `FormGroup`
- Par exemple : un `FormGroup` `adresse` défini dans le `FormGroup` `personne`

Dans `form.component.ts`, définissons les `FormGroup` imbriqués

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {
  personneForm = new FormGroup({
    id: new FormControl(''),
    nom: new FormControl(''),
    prenom: new FormControl(''),
    adresse: new FormGroup({
      rue: new FormControl(''),
      ville: new FormControl(''),
      codePostal: new FormControl('')
    })
  });
  constructor() { }
  ngOnInit() { }
  afficherTout() { console.log(this.personneForm.value); }
}
```

Dans `form.component.html`, créons le formulaire associé aux `FormGroup` imbriqués

```
<form [formGroup]="personneForm" (ngSubmit)='afficherTout()'>
  <div>
    Identifiant : <input type="number" formControlName="id">
  </div>
  <div>
    Nom : <input type="text" formControlName="nom">
  </div>
  <div>
    Prénom : <input type="text" formControlName="prenom">
  </div>
  <div formGroupName="adresse">
    <h3>Adresse</h3>
    <div>
      Rue : <input type="text" formControlName="rue">
    </div>
    <div>
      Ville : <input type="text" formControlName="ville">
    </div>
    <div>
      Code postal : <input type="text" formControlName="codePostal">
    </div>
  </div>
  <button type='submit'>Envoyer</button>
</form>
```

# Angular

## Remarque

- La méthode `setValue()` permet d'initialiser, ou modifier les valeurs de formulaire : il faut préciser une valeur pour chaque `FormControl` du `FormGroup`
- La méthode `patchValue()` permet d'initialiser, ou modifier quelques (ou tous les) `FormControl` du `FormGroup`

# Angular

## Exemple (dans `ngOnInit()`)

```
this.personneForm.patchValue({  
  prenom: 'abruzzo',  
  adresse: {  
    codePostal: '13000'  
  }  
});
```

# Angular

## Exemple (dans `ngOnInit()`)

```
this.personneForm.patchValue({  
  prenom: 'abruzzzi',  
  adresse: {  
    codePostal: '13000'  
  }  
});
```

Les champs `Prénom` et `Code postal` sont initialisés avec les valeurs `abruzzzi` et `13000`

# Angular

## FormBuilder

- Une classe service défini par **Angular**
- Donc, pour l'utiliser, il faut l'injecter dans le constructeur
- Il permet de simplifier la construction d'un formulaire en évitant les répétitions de `FormControl`

Dans `form.component.ts`, on injecte `FormBuilder` dans le constructeur puis on l'utilise pour construire le formulaire

```
import { Component, OnInit } from '@angular/core';
import { Validators, FormBuilder } from '@angular/forms';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {
  personneForm = this.fb.group({
    id: [null],
    nom: ['doe'],
    prenom: ['', [Validators.required, Validators.minLength(2)]],
    adresse: this.fb.group({
      rue: ['', ],
      ville: ['', ],
      codePostal: ['', ]
    })
  });
  constructor(private fb: FormBuilder) { }
  ngOnInit() { }
  afficherTout() { console.log(this.personneForm.value); }
}
```



## Dans form.component.html, rien à changer

```
<form [formGroup]="personneForm" (ngSubmit)='afficherTout()'>
  <div>
    Identifiant : <input type="number" formControlName="id">
  </div>
  <div>
    Nom : <input type="text" formControlName="nom">
  </div>
  <div>
    Prénom : <input type="text" formControlName="prenom">
  </div>
  <div formGroupName="adresse">
    <h3>Adresse</h3>
    <div>
      Rue : <input type="text" formControlName="rue">
    </div>
    <div>
      Ville : <input type="text" formControlName="ville">
    </div>
    <div>
      Code postal : <input type="text" formControlName="codePostal">
    </div>
  </div>
  <button [disabled]='!personneForm.valid'>Envoyer</button>
</form>
```

# Angular



On peut aussi surveiller l'évolution de notre formulaire grâce à l'attribut `status` (à placer dans le formulaire)

```
<div>  
  état : {{ personneForm.status }}  
</div>
```

# Angular

## Exercice

- Modifier le composant `builder` pour afficher les personnes ajoutées en-dessous du formulaire
- À chaque ajout, la nouvelle personne ajoutée apparaît comme dernier élément de la liste des personnes (affichée en-dessous du formulaire)

# Angular

## Exercice

- Ajoutez un bouton supprimer pour chaque personne affichée.
- En cliquant sur le bouton, la personne associée sera supprimée.

# Angular

## FormArray

- Défini dans `FormBuilder`
- Il permet de définir un tableau de taille indéterminée de `FormControl`
- Une personne peut pratiquer plusieurs sports (le nombre peut varier d'une personne à une autre)  $\Rightarrow$  on peut utiliser `FormArray`

# Angular

Dans `form.component.ts`, définissons notre **FormArray**

```
personneForm = this.fb.group({  
  id: [null],  
  nom: ['doe'],  
  prenom: ['', [Validators.required, Validators.  
    minLength(2)]],  
  adresse: this.fb.group({  
    rue: ['',],  
    ville: ['',],  
    codePostal: ['',]  
  }),  
  sports: this.fb.array([  
    this.fb.control('')  
  ])  
});
```

# Angular

Pour afficher instantanément les sports ajoutés par l'utilisateur, on doit retourner notre `FormArray`

```
get sports() {  
    return this.personneForm.get('sports') as  
        FormArray;  
}
```

# Angular

Dans `form.component.html`, ajoutons notre `FormArray` à notre formulaire précédent

```
<div formArrayName="sports">
  <h3>Sports </h3>
  <button type="button" (click)="ajouterSport()">
    Ajouter sport
  </button>
  <div *ngFor="let sport of sports.controls; let i=
    index">
    <div>
      Sport :
      <input type="text" [formControlName]="i">
    </div>
  </div>
</div>
```



# Angular

Définissons maintenant la méthode `ajouterSport()`

```
ajouterSport() {  
    this.sports.push(this.fb.control(''));  
}
```

# Angular

Définissons maintenant la méthode `ajouterSport()`

```
ajouterSport() {  
    this.sports.push(this.fb.control(''));  
}
```

## Remarque

- On ajoute à notre tableau un nouvel élément vide pour que l'utilisateur puisse saisir un nouveau sport.
- Le sport ajouté par l'utilisateur est lié directement à notre `FormArray`

# Angular

## Exercice

- Ajoutez un bouton supprimer pour chaque sport.
- En cliquant sur le bouton, le sport associé sera supprimé.

# Angular

## Remarque

`FormArray` est une classe qui peut être aussi utilisée dans un `FormGroup`

# Angular

## Exercice

- Dans un nouveau composant, créer un formulaire qui permet à une personne de saisir son nom, son prénom ainsi qu'un tableau de commentaire de taille variable.
- Chaque commentaire est composé d'un titre, un contenu et une catégorie.
- En cliquant sur **Ajouter**, les données saisies sont affichées en bas du formulaire et le formulaire est vidé.
- Aucun champ ne doit être vide à l'ajout, les nom et prénom doivent commencer par une lettre majuscule.