

@angular/forms

1. How to look at forms in Angular

Forms

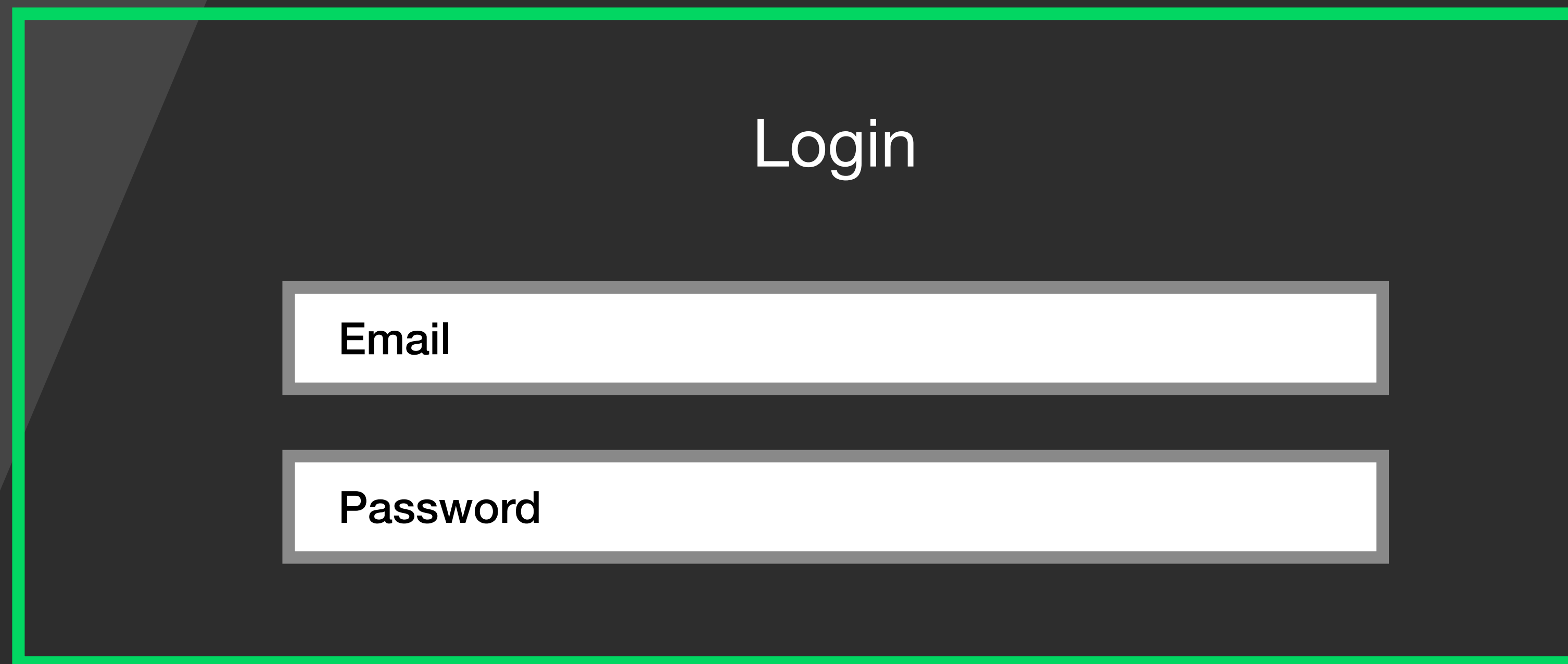
- Forms are the main tool we use to get input from our users
- With forms users can:
 - Enter text in text fields
 - Toggle check boxes
 - Select value from a select box
 - Pick from a list of radio buttons
 - etc.
- In this lesson we will learn how to work with forms with Angular

How Angular wants us to look at forms

- When creating a form, angular wants us to break apart the form to it's building block
- The building blocks of a form are
 - ▶ A single form control
 - ▶ A group of controls
 - ▶ An array of controls
- Let's try and examine a few forms and break them apart to their building blocks

Login form

- We can break this login form to:
 - ▶ Email form control
 - ▶ Password form control



Login

Email

Password

Register form

- We can break this Register form to:
 - ▶ Email form control
 - ▶ Group holding the password and repeat (cause they share validation logic)
 - Password form control
 - Repeat form control

Register

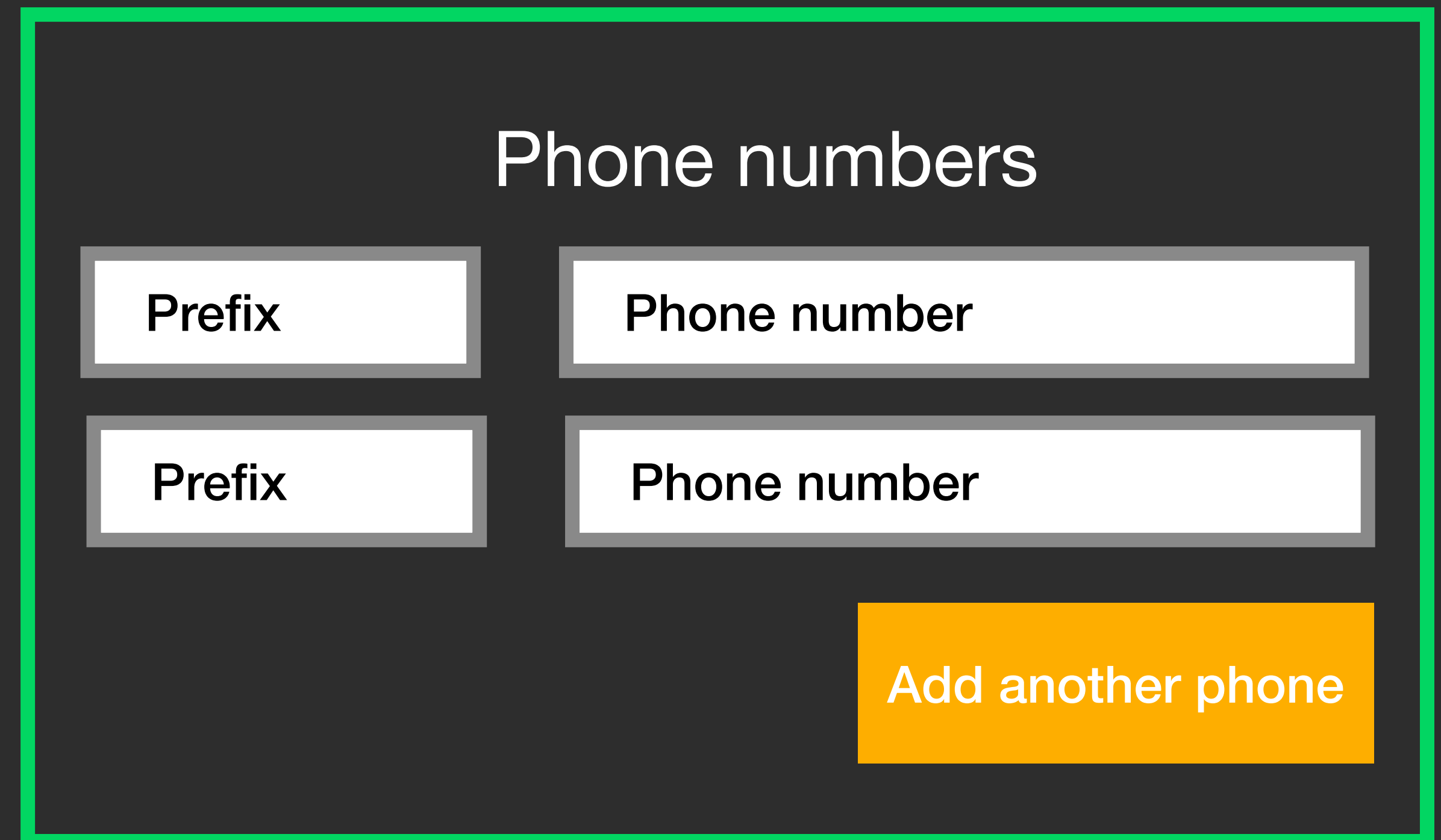
Email

Password

Repeat Password

Phone numbers form

- We can break this phone number form to:
 - ▶ An array of phone numbers
 - Group of prefix and phone
 - * Prefix form control
 - * Phone number form control



The diagram illustrates a form titled "Phone numbers" enclosed in a red border. It contains two rows of input fields. Each row consists of a "Prefix" input field and a "Phone number" input field. Below these fields is a red button labeled "Add another phone".

Phone numbers	
Prefix	Phone number
Prefix	Phone number

Add another phone

Classes for describing your form

- You decided the structure of your form with: form control, form group, form array
- Angular provides you with the following classes to describe the structure of your form:
 - ▶ AbstractControl
 - ▶ FormControl
 - ▶ FormArray
 - ▶ FormGroup

AbstractControl

- The common parent of all the building blocks of the forms: FormControl, FormArray, FormGroup

```
export declare abstract class AbstractControl {  
  readonly value: any;  
  readonly status: string;  
  get valid(): boolean;  
  get invalid(): boolean;  
  readonly errors: ValidationErrors | null;  
  readonly valueChanges: Observable<any>;  
  abstract setValue(value: any, options?: Object): void;  
}
```


- Represents a single element in the form like a text input or a select box

```
export declare class FormControl extends AbstractControl {  
  setValue(value: any, options?: {  
    onlySelf?: boolean;  
    emitEvent?: boolean;  
    emitModelToViewChange?: boolean;  
    emitViewToModelChange?: boolean;  
  }): void;  
}
```

FormArray

- Represents an array in the form, same group of form elements that can repeat multiple times
- Often used in dynamic forms where a group can be added dynamically
- An array can be made from other group, array, controls

```
export declare class FormArray extends AbstractControl {  
  controls: AbstractControl[];  
  push(control: AbstractControl): void;  
  removeAt(index: number): void;  
  setValue(value: any[], options?: {  
    onlySelf?: boolean;  
    emitEvent?: boolean;  
  }): void;  
}
```

FormGroup

- The entire form is a group
- A group can be made from sub groups, array, controls

```
export declare class FormGroup extends AbstractControl {
  controls: {
    [key: string]: AbstractControl;
  };
  addControl(name: string, control: AbstractControl): void;
  removeControl(name: string): void;
  setValue(value: {
    [key: string]: any;
  }, options?: {
    onlySelf?: boolean;
    emitEvent?: boolean;
  }): void;
}
```

Summary

- Angular wants us to break our form into
 - ▶ FormGroup
 - ▶ FormArray
 - ▶ FormControl
- Each of the above has a common parent abstract class of AbstractControl

Thank You

Next Lesson: 2. Reactive VS Template Driven Forms