

# Angular 2

## Lesson 9—Forms



# Learning Objectives

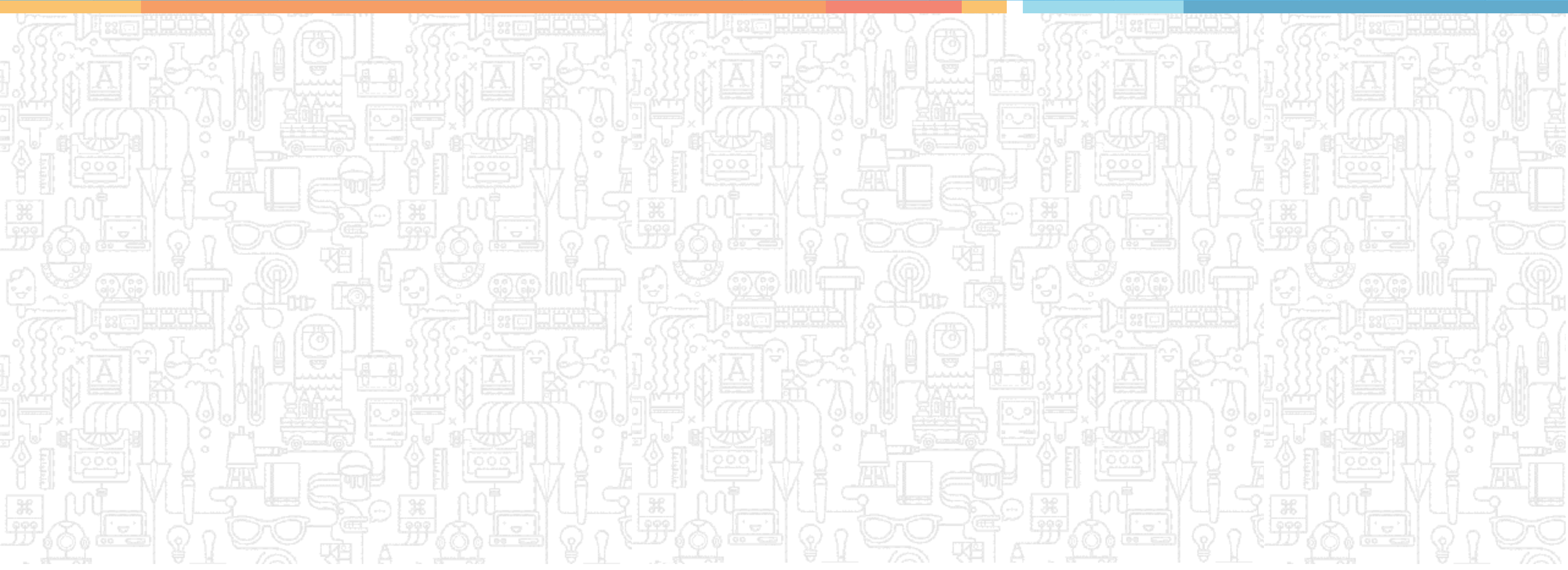
---



- ✓ Angular 2 Form benefits
- ✓ Template-Driven Form Approach
- ✓ Model-Driven Form Approach
- ✓ Angular 2 Form Validation

# Forms

## Topic 1—Angular 2 Form Benefits

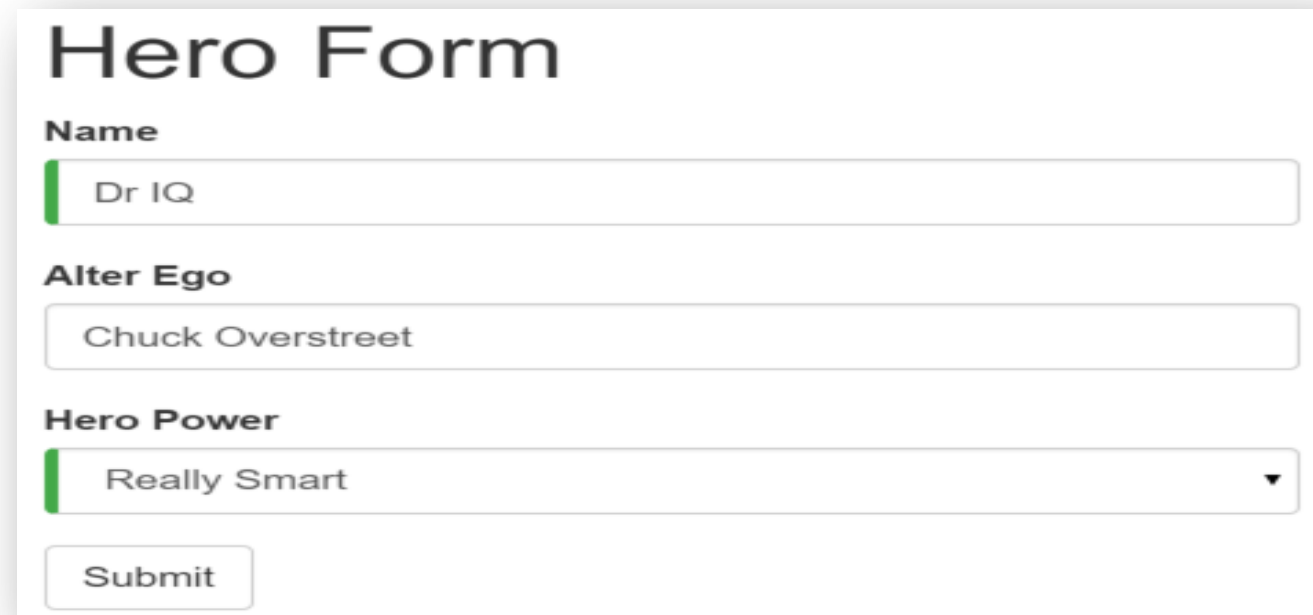


# Angular 2 Form Benefits

---

A form creates a cohesive, effective, and compelling data entry experience.

An Angular form coordinates a set of data-bound user controls, tracks changes, validates input, and presents errors.



**Hero Form**

**Name**

**Alter Ego**

**Hero Power**

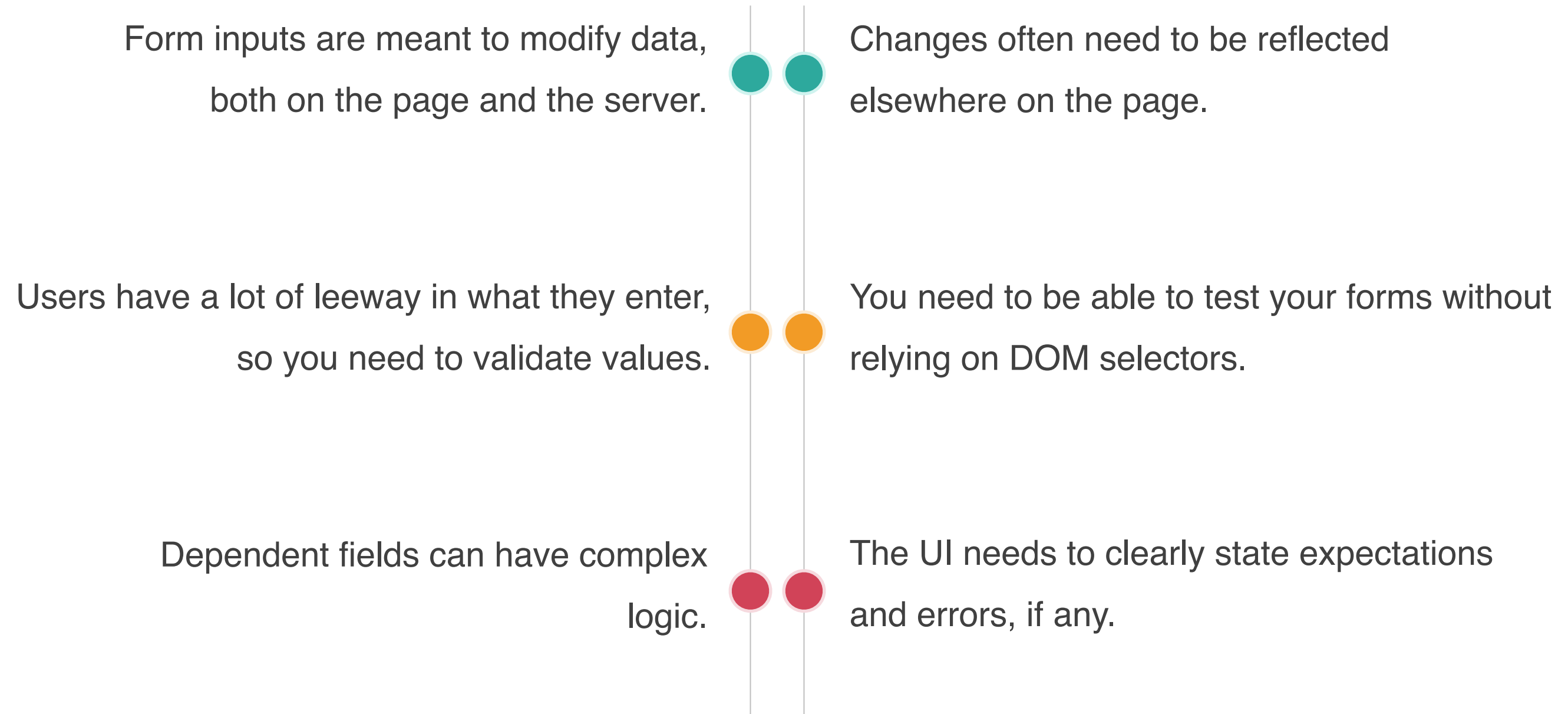
Forms are probably the most crucial aspect of your web application.

You can often get events by clicking on links or moving the mouse, but it's through forms that you get the majority of your rich data input from users.

# Angular 2 Form—Challenges

---

**The form challenges in Angular are listed below:**



# Angular 2 Form—Tools

---

Angular 2 has these tools to help with all the challenges.

## FormControls

- Encapsulate the inputs in your forms and give objects to work with them

## Validators

- Give you the ability to validate inputs, any way you like

## Observers

- Allow you to watch your form for changes and respond accordingly

# Angular 2 Form—FormControl

---

Angular 2 FormControl comes in two forms: **Control** and **ControlGroup**.

## Control

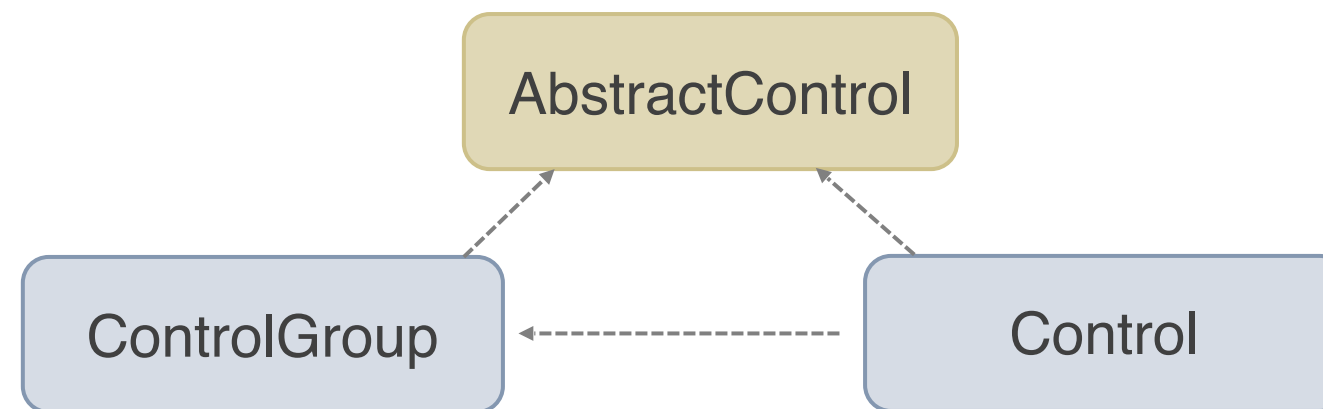
A Control is one of the fundamental building blocks used to define forms in Angular 2; however, it cannot be divided into other Controls.

## Control Group

A ControlGroup can define all or part of a form; it aggregates many Controls. A ControlArray resembles a ControlGroup and can also aggregate Controls.

## Angular 2 Form—AbstractControl

ControlGroup and Control extend from the same **abstract class**, which means that these are in **AbstractControl** and refer to each other using this same abstract class.



**AbstractControl** gives you enough flexibility to match any HTML form structure. These building blocks play an important role in the structure.



# Angular 2 Form—Control Class

---

Control class represents a single input field in the form.



- By default, a Control is created for every `<input>` or other form component.
- You can bind an existing Control to a DOM element using directives such as `NgFormControl` and `NgControlName`

## Angular 2 Form—Control Class (Contd.)

---

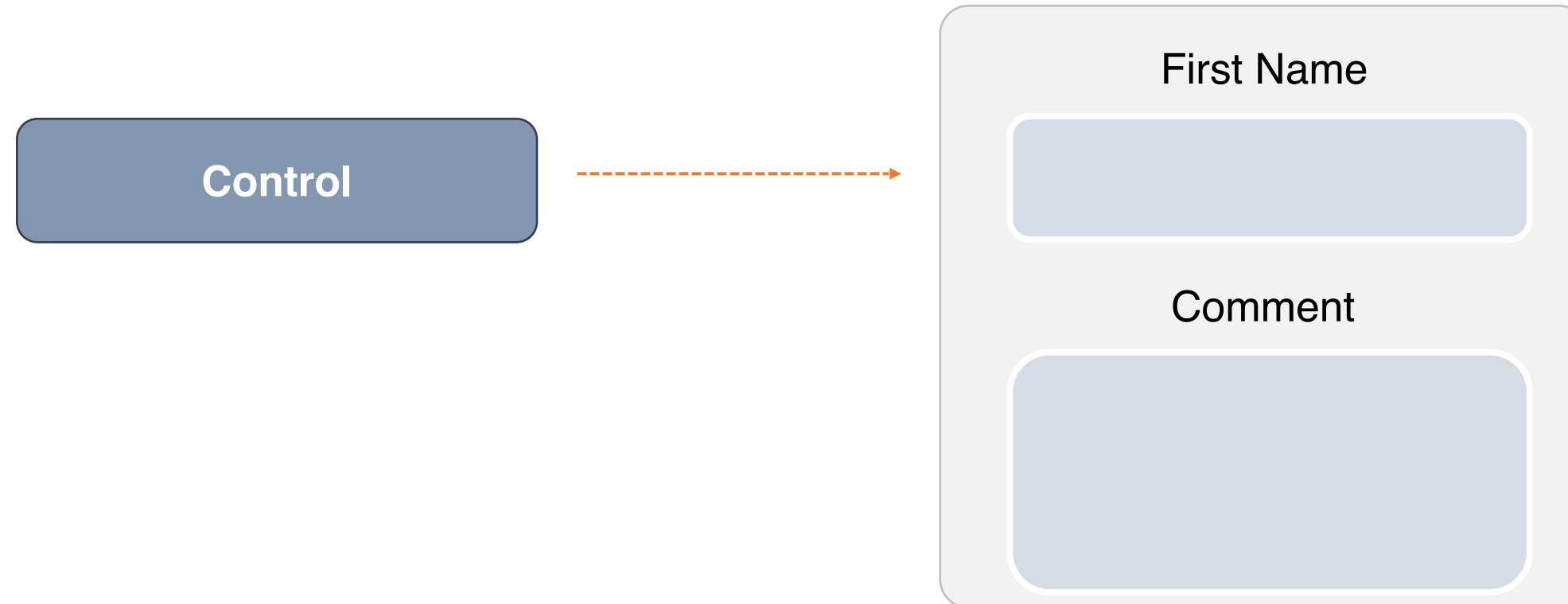
Using Control class, you can access other properties too.



# Angular 2 Form—ControlGroup

Most forms have more than one field, so you need a way to manage multiple Controls.

If you want to check the validity of your form, it's cumbersome to iterate over an array of Controls and check each Control for validity.

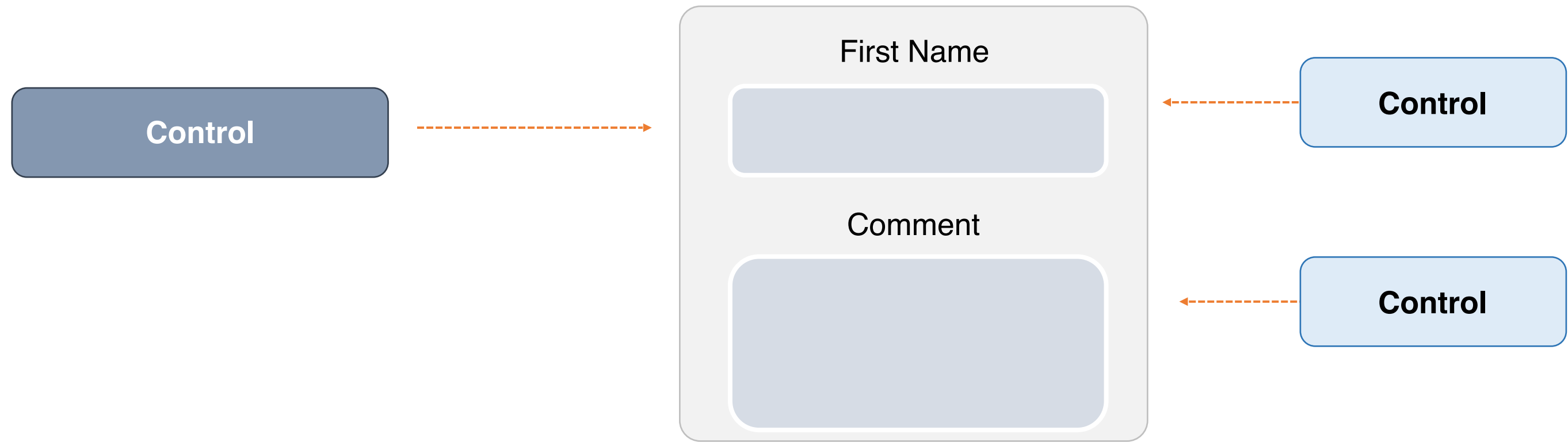


ControlGroups solve this issue by providing a wrapper interface around a collection of Controls.

## Angular 2 Form—ControlGroup (Contd.)

---

ControlGroup can have multiple controls.



# Angular 2 Form—Validator

---

Validators are provided by the Validators module. The simplest validator is “required,” which simply says that the designated field is required or the Control will be considered invalid.

**To use validators, you need to do two things:**

**01**

Assign a validator to the Control object

**02**

Check the status of the validator in the view and take action accordingly

# Angular 2 Form—Observer

---

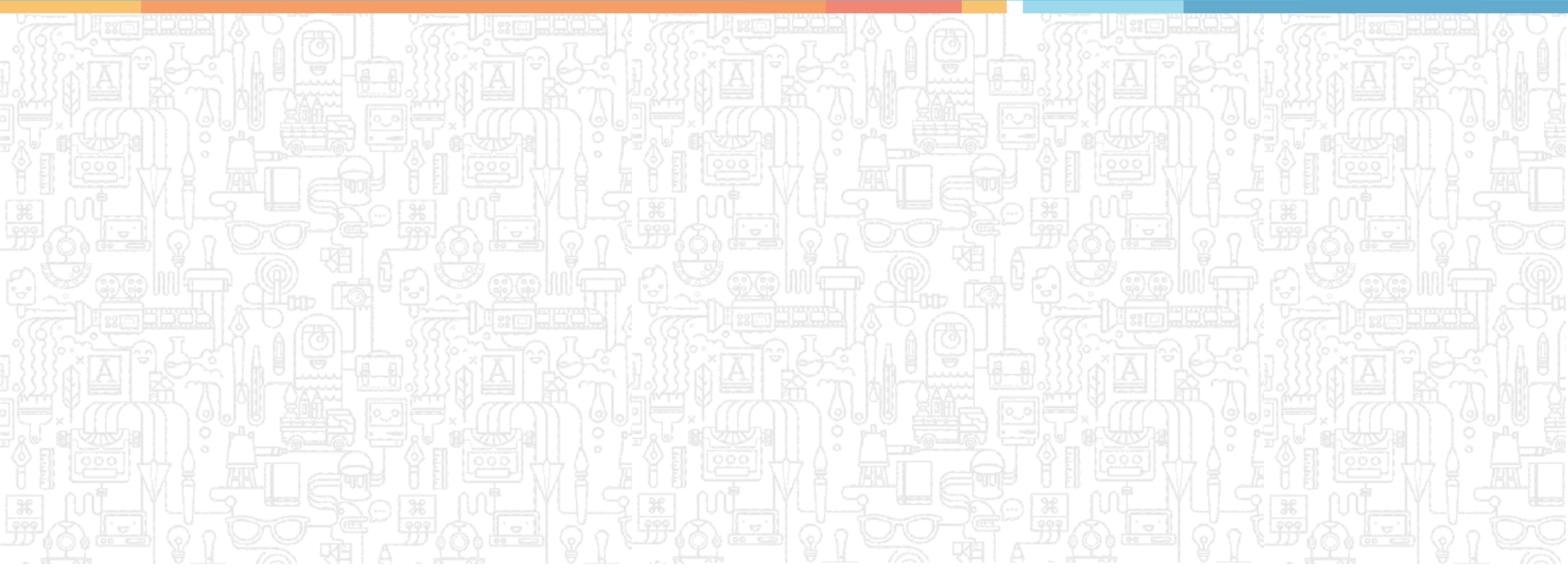
Observers allow you to watch your form for changes and respond accordingly. In an imperative way, a variable is only changed when its state is mutated by assigning a new or updated value.

## Observers design patterns:

- 01 Subject tracks the Observer
- 02 Subject notifies observers of state changes

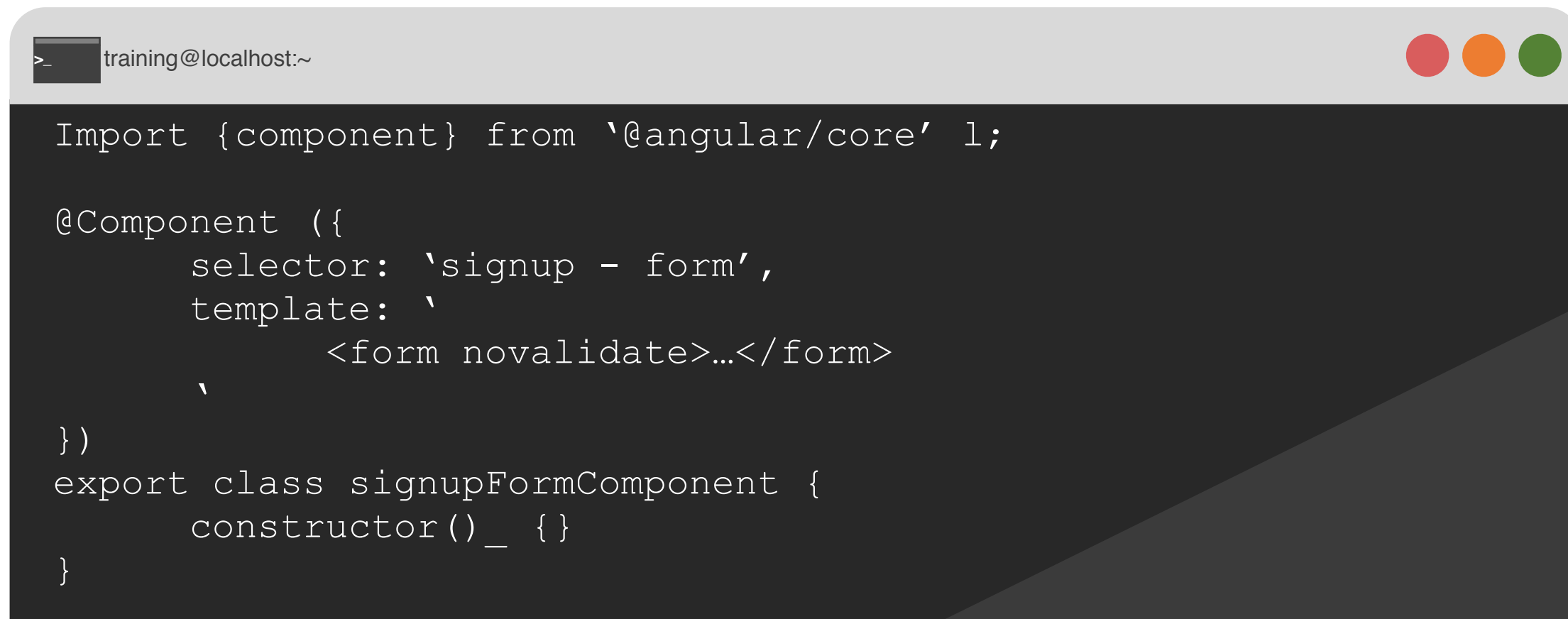
# Forms

## Topic 2—Template-Driven Approach



# Template-Driven Approach

With template-driven forms, you can essentially leave a component class empty until you need to read/write values (such as submitting and setting initial or future data).

A terminal window with a light gray title bar. The title bar contains a terminal icon, the text 'training@localhost:~', and three window control buttons (red, orange, green). The terminal area has a dark background with white text. The code is an Angular component definition for a signup form.

```
>_ training@localhost:~  
  
Import {component} from '@angular/core' 1;  
  
@Component ({  
  selector: 'signup - form',  
  template: `  
    <form novalidate>...</form>  
  `,  
})  
export class signupFormComponent {  
  constructor()_ {}  
}
```



# Template-Driven Approach—Advantages and Disadvantages

---

## Advantage

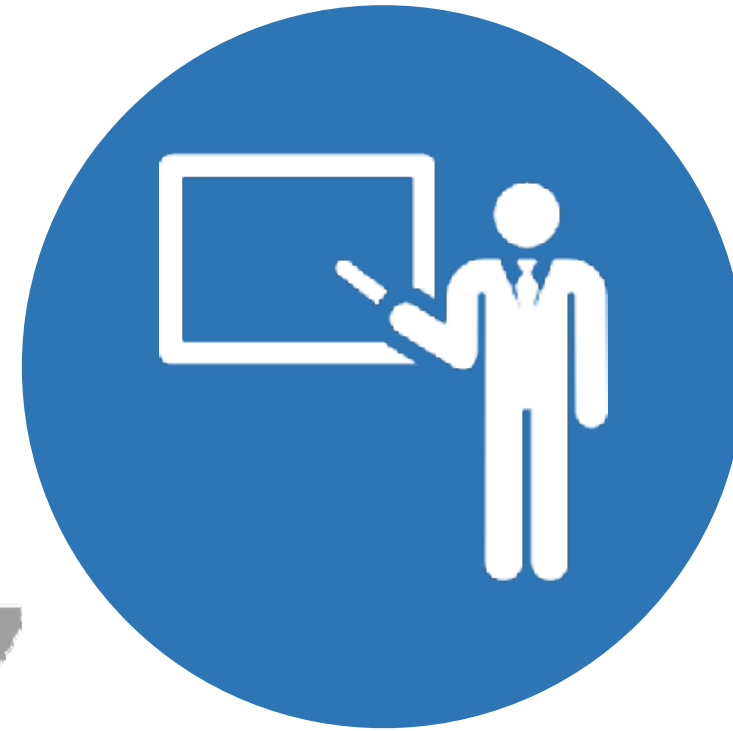
The upside of this way of handling forms is its simplicity, and it is probably more than enough to build a large range of forms.

## Disadvantage

On the downside, the form validation logic cannot be unit tested. The only way to test this logic is to run an end-to-end test with a browser, for example, using a headless browser like PhantomJS.

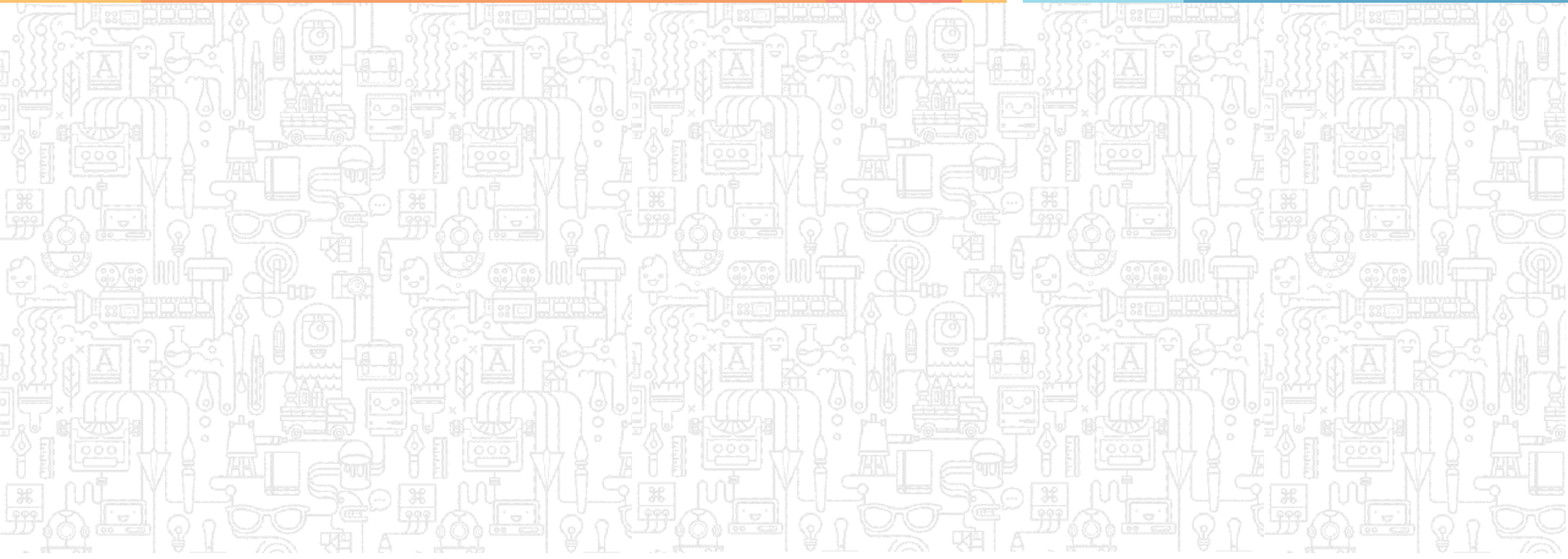
# Demo—Template-Driven Approach

---



# Forms

## Topic 3—Angular 2 Validation



# Angular 2 Validation

---

**Angular 2 comes with 2 types of validation:**

**Built-in form validation**

**Custom form validation**

# Angular 2 Validation (Contd.)

## Built-in form validation

Angular 2 provides three out-of-the-box validators that can either be applied using the “Control” Class, or using HTML properties.

For example: `<input required>` will automatically apply the required validator

- Required
- minLength
- maxLength

Apply them to the Control using the second parameter.

```
this.name = new Control('', Validators.minLength(4));
```

# Angular 2 Validation (Contd.)

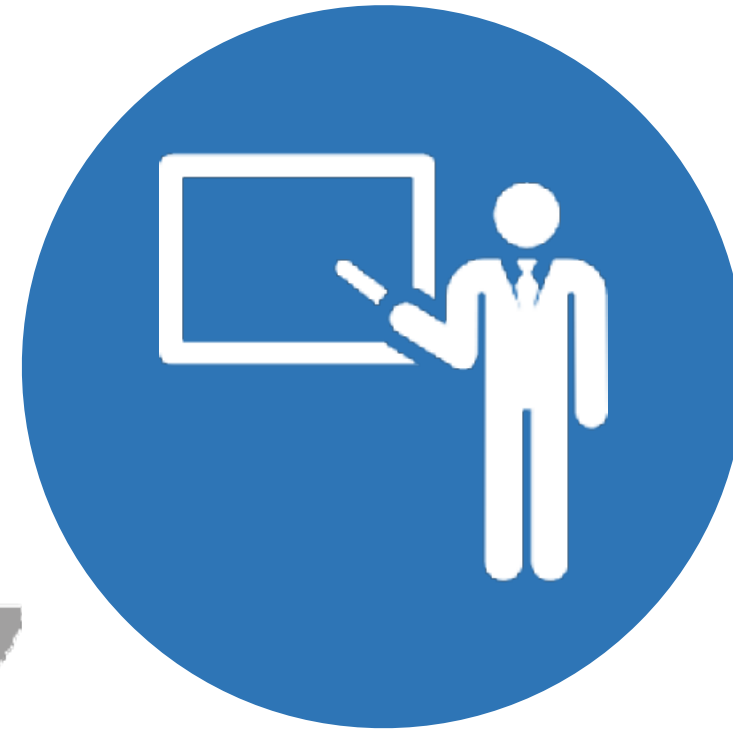
## Custom form validation

You can also write your own custom validators. Here is an example of a validator that checks if the first character is not a number:

```
Interface ValidationResult {  
  [key:string] :boolean;  
}  
  
class UsernameValidator {  
  static startWithNumber (control: Control) :ValidationResult {  
    if ( control.value != "" && !isNaN  
(control.value.charAt(0)) ) {  
    }  
    return null;  
  
  }  
}
```

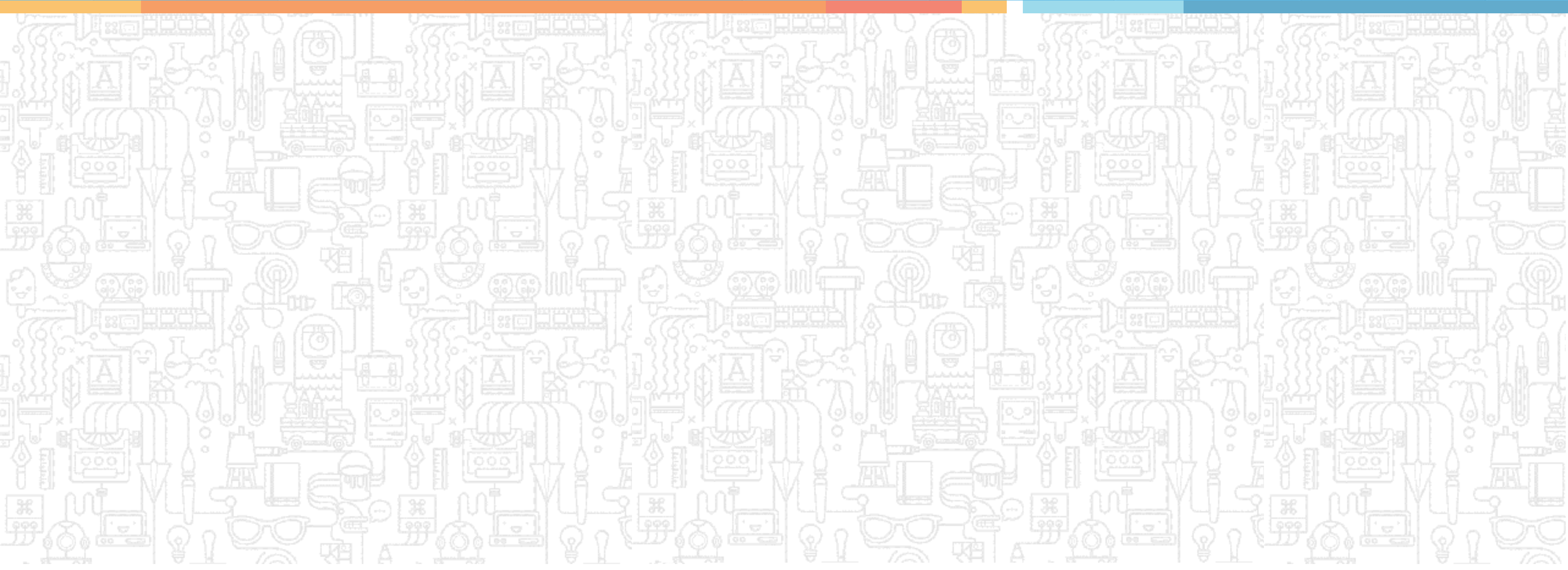
# Demo—Validation

---



# Forms

## Topic 4—Model-Driven Approach





# Model-Driven Approach


---

A lot of features and concerns were improved in Angular 2, such as communication between components, code-reuse, separation of concerns, unit tests, and others.

To promote some of those benefits in the context of forms, Angular 2 uses a “model-driven” or reactive technique of development.

# Model-Driven Approach—Benefits

---



Model-Driven approach (Reactive Forms) refers to a strategy of building your own Model object (FormGroup and related items) and binding it to Component HTML.

It's implied in a more concise HTML Template and provides more control in your forms (but with more code on it).

The same FormGroup object that is automatically created and associated with the form can be created manually using ReactiveForms.

# Model-Driven Approach—Front-end

>\_ training@localhost:~

```
<form [ngFormModel]='form' (ngSubmit)="onSubmit()">
  <div class="form-group" [ngclass]="{'has-error': !name.valid}">
    <label for="name">Name</label>
    <input type="text" class="form-control" id="name"
      ngcontrol='name'
      #name="ngForm">
    <p *ngIf="!name.valid" class="help-block">
      Name is required
    </p>
  </div>
  <button type="submit" class="btn btn-primary"
    [disabled]="!form.valid">Add user</button>
</form>
```

# Model-Driven Approach—Component

>\_ training@localhost:~

```
import {Component} from 'angular2/core';
import {Control, ControlGroup, Validators} from 'angular2/common';

@Component({
  selector: 'form',
  templateUrl: './dev/shared/form.component.html'
})
Export class FormComponent {
  form = new ControlGroup({
    name: new Control('', Validators.required);
  });

  onSubmit() {
    console.log(this.form.value);
  }
}
```

# Model-Driven Approach—FormBuilder

---

- You can create an `AbstractControl` from a user-specified configuration.
- It is essentially syntactic sugar that shortens the `new FormGroup()`, `new FormControl()`, and `new FormArray()` boilerplate that can build up in larger forms.
- Angular has a new helper Class called `FormBuilder`.
- `FormBuilder` allows you to explicitly declare forms in your components. This also allows you to explicitly list each form control's validators.

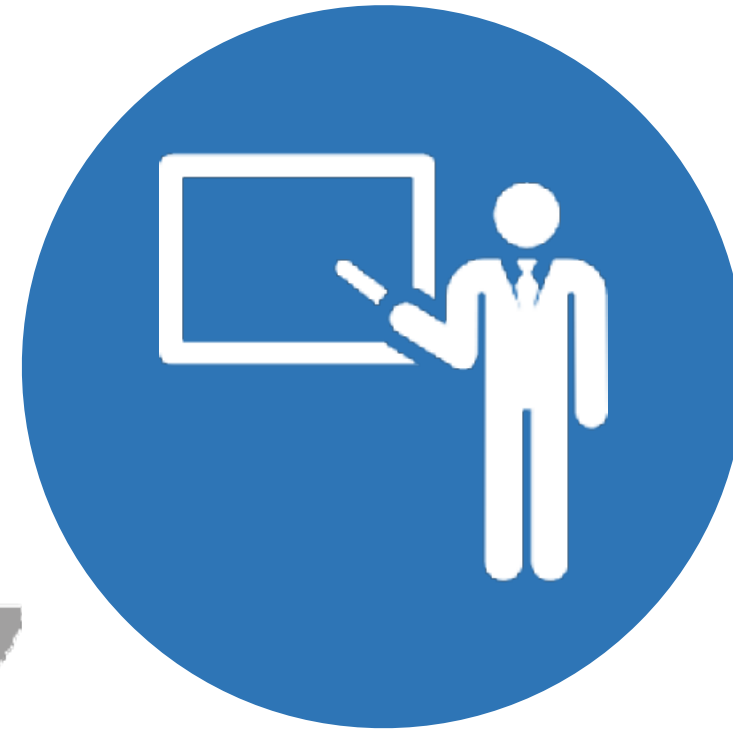
# Advantages of Model-Driven Forms

---

- You can now unit test the form validation logic.
- This can be done by just instantiating the class, setting some values in the form controls, and performing assertions against the form global valid state and the validity state of each control.
- The FormGroup and FormControl classes provide an API that allows you to build UIs using a completely different programming style known as Functional Reactive Programming.

# Demo—Model Driven Approach

---



# Key Takeaways



- ✓ Angular 2 aims to make the creation and validation of forms simple, intuitive, and manageable.
- ✓ Template-Driven approach was the typical way to build forms in the past, and it is still useful for building very simple forms in Angular 2.
- ✓ Angular 2 comes with a brand new approach to forms that makes it easier to construct and apply validation to them.
- ✓ Model-driven approach (reactive forms), while similar to template driven approach, adds an additional layer of complexity and functionality as you need to declare the model of the form in your component class.





## QUIZ

1

There are two ways to build forms in Angular 2, namely \_\_\_\_\_ and \_\_\_\_\_.

- a. Interface and Modular
- b. Model and Template
- c. Interface and Model
- d. Modular and Template



## QUIZ

1

There are two ways to build forms in Angular 2, namely \_\_\_\_\_ and \_\_\_\_\_.

- a. Interface and Modular
- b. Model and Template
- c. Interface and Model
- d. Modular and Template



The correct answer is **b.**

There are two ways to build forms in Angular 2, namely Model and Template.

## QUIZ

2

Which of the following is NOT a built-in validator in Angular 2?

- a. required
- b. minlength
- c. maxlength
- d. notfound



## QUIZ

2

Which of the following is NOT a built-in validator in Angular 2?

- a. required
- b. minlength
- c. maxlength
- d. notfound



The correct answer is **d.**

**Notfound is not a built-in validator in Angular 2.**

## QUIZ

3

**Angular-2 forms Control follows AbstractControl pattern.**

- a. True
- b. False



## QUIZ

3

Angular-2 forms Control follows AbstractControl pattern.

- a. True
- b. False



The correct answer is **a.**

Angular-2 forms Control follows AbstractControl pattern.



# Thank You