Instructor Notes:

Add instructor notes here.

Instructor Notes:

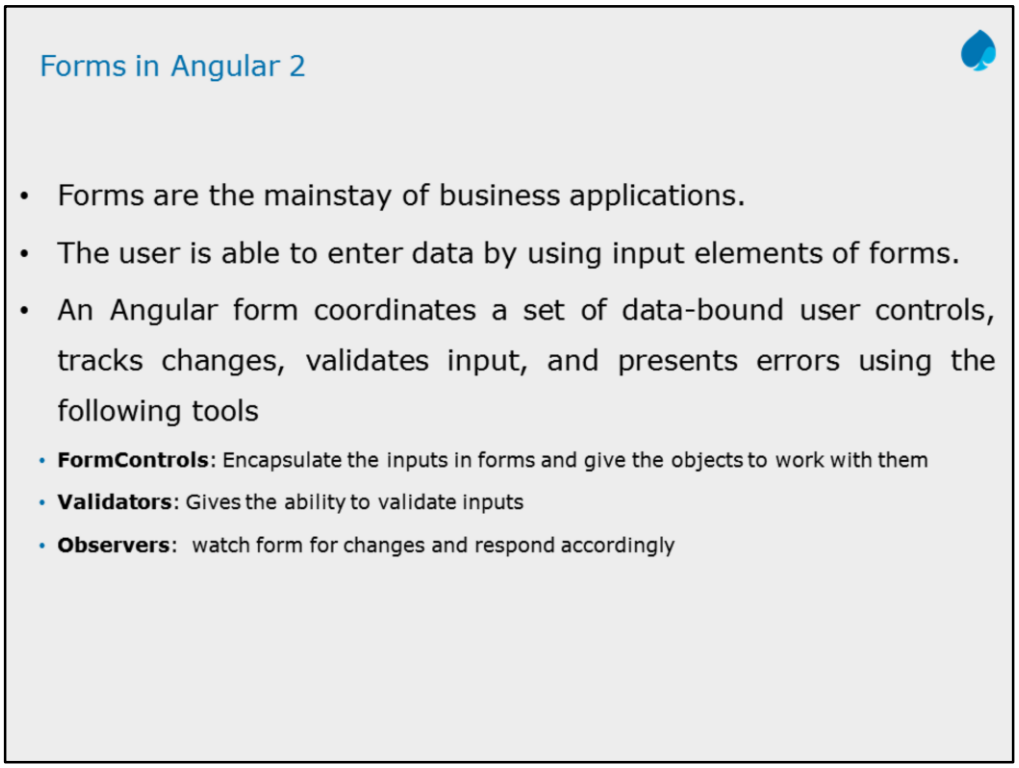
Add instructor notes here.

Lesson Objectives

- Forms in Angular 2
- Template & Model Driven Forms
- A Basic Angular Form
- Binding Input Fields
- Displaying Form Validation State & Field Validation State
- Displaying Validation State Using Classes
- Disabling Submit when Form is Invalid

Instructor Notes:

Add instructor notes here.

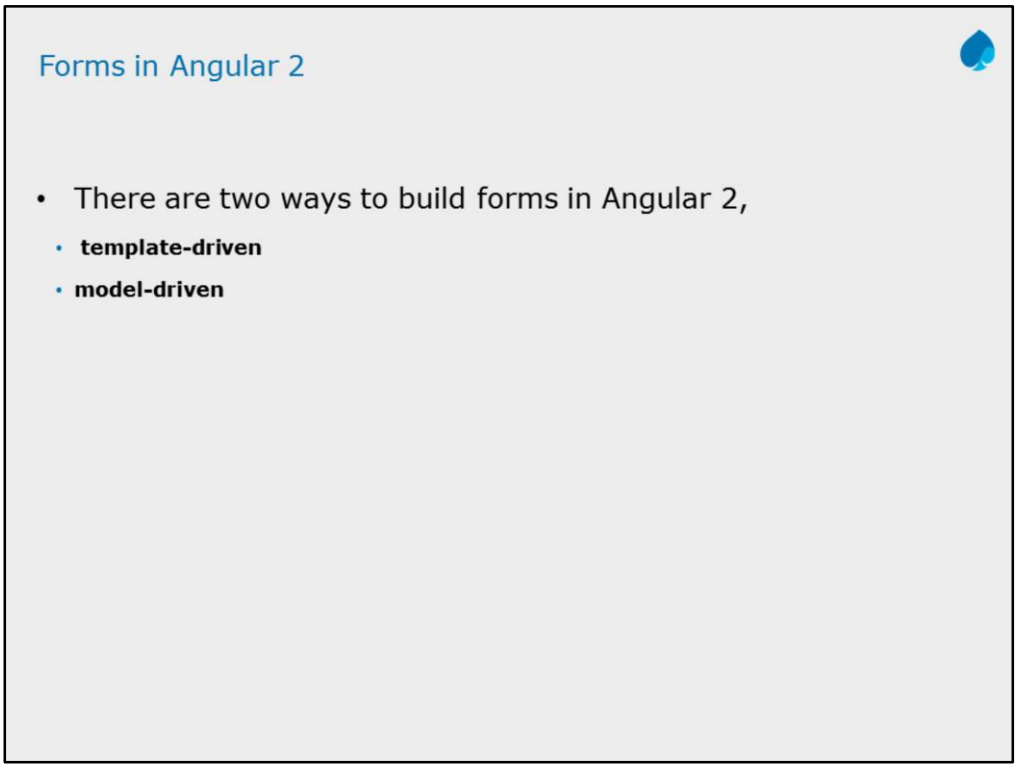


Forms are the cornerstone of any real app. In Angular 2, forms have changed quite a bit from their v1 counterpart.
Where we used to use ngModel and map to our internal data model, in Angular 2 we more explicitly build forms and form controls.

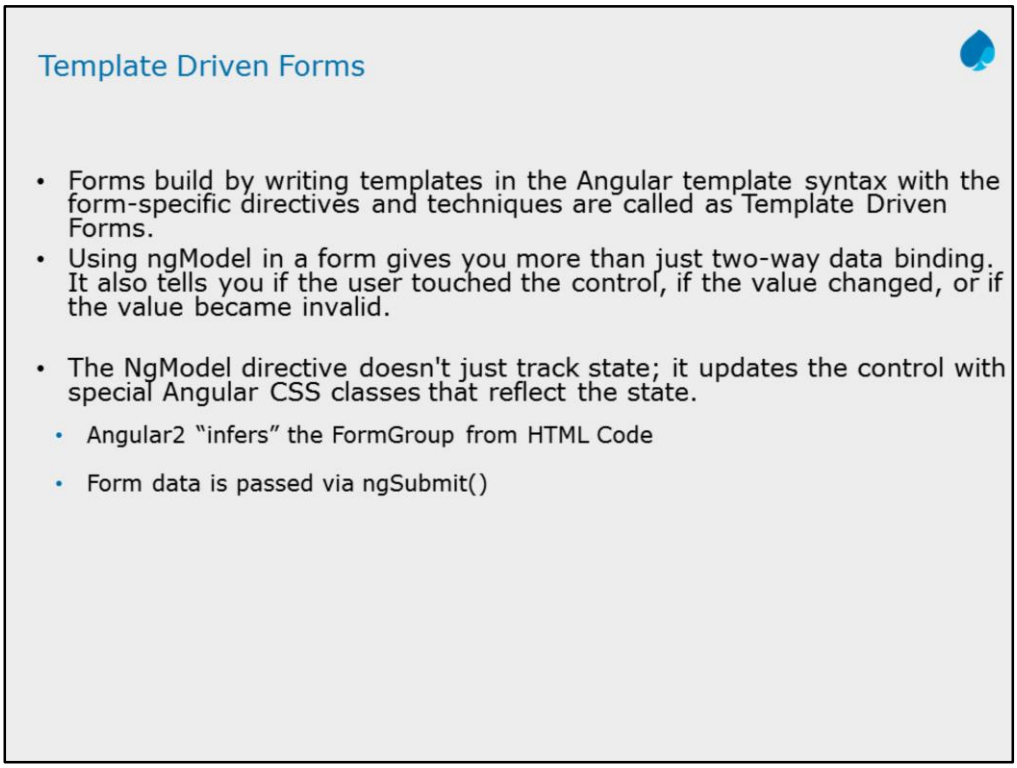
Import FormsModule

To be able to use Angular 2 forms the FormsModule needs to be imported in our application. To make it available application-wide we're importing in *app.module.ts*:
import { FormsModule } from '@angular/forms';

Instructor Notes:



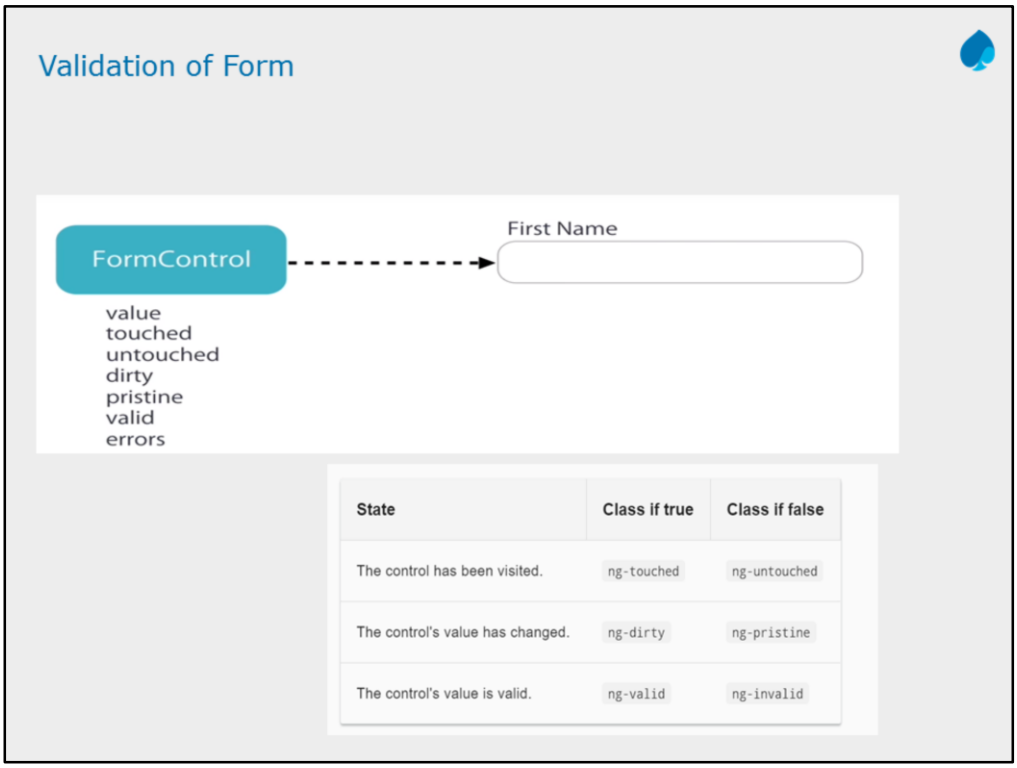
Instructor Notes:



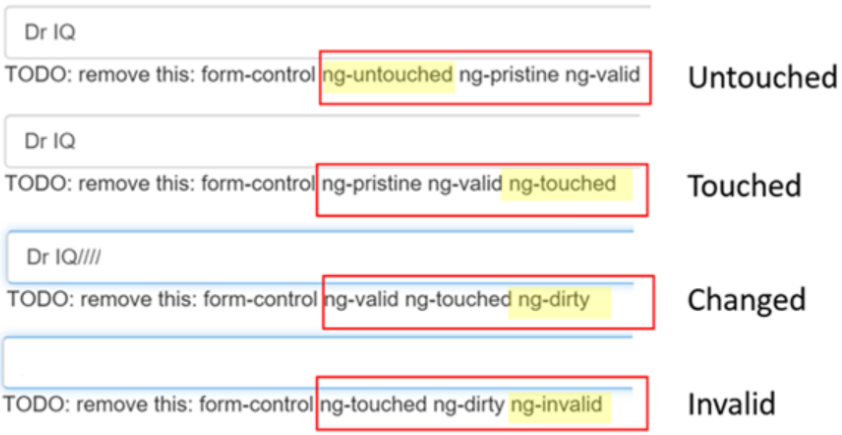
The user should be able to submit this form after filling it in. The *Submit* button at the bottom of the form does nothing on its own, but it will trigger a form submit because of its type (type="submit").
A "form submit" is useless at the moment. To make it useful, bind the form's ngSubmit event property to the hero form component's onSubmit() method: src/app/hero-form/hero-form.component.html (ngSubmit)content_copy<form (ngSubmit)="onSubmit()" #heroForm="ngForm">

The <input> element carries the HTML validation attributes: required and minlength. It also carries a custom validator directive, forbiddenName. For more information, see Custom validatorssection.
#name="ngModel" exports NgModel into a local variable called name. NgModel mirrors many of the properties of its underlying FormControl instance, so you can use this in the template to check for control states such as valid and dirty. For a full list of control properties, see the AbstractControl API reference.
The *ngIf on the element reveals a set of nested message divs but only if the nameis invalid and the control is either dirty or touched.
Each nested can present a custom message for one of the possible validation errors. There are messages for required, minlength, and forbiddenName.

Instructor Notes:



The ng-valid/ng-invalid pair is the most interesting, because you want to send a strong visual signal when the values are invalid.
hide the message when the control is valid or pristine; "pristine" means the user hasn't changed the value since it was displayed in this form.
This user experience is the developer's choice. Some developers want the message to display at all times. If you ignore the pristine state, you would hide the message only when the value is valid. ISome developers want the message to display only when the user makes an invalid change. Hiding the message while the control is "pristine" achieves that goal. You'll see the significance of this choice when you add a new hero to the form.
Required - The form field is mandatroy
Maxlength - Maximum number of characters in the field.
Minlength - Minimum number of characters in the field.
Pattern - Regular expression to match the input values and validate.
Custom Validator - Writing custom validation function to match password and confirm password fields.



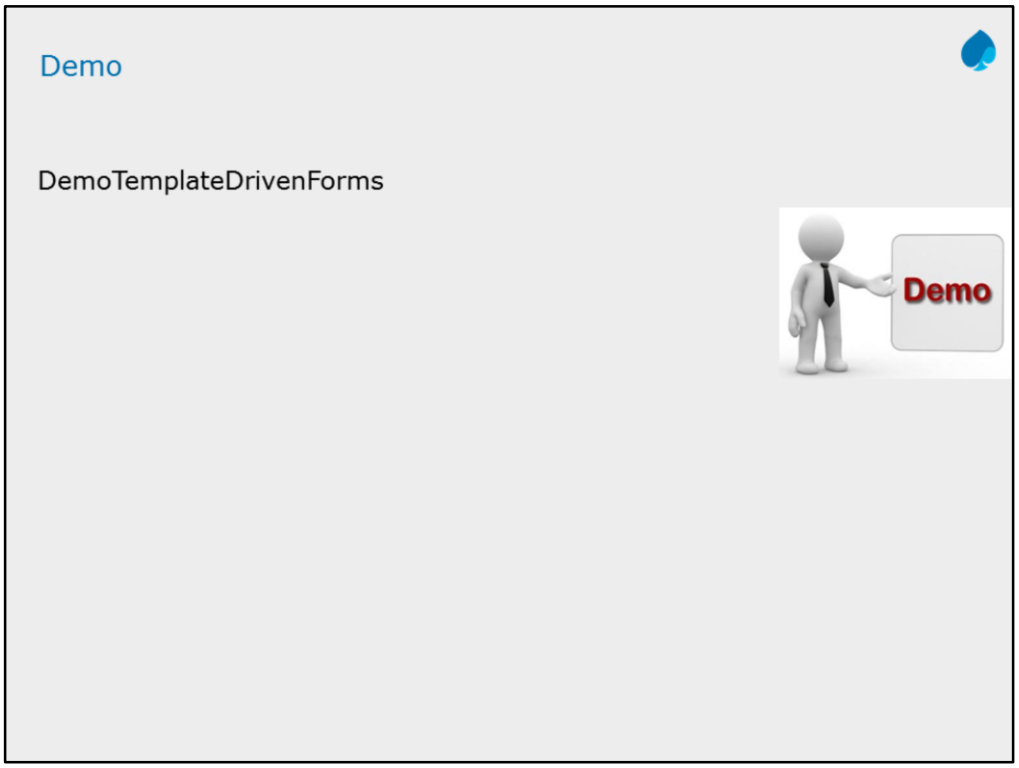
Instructor Notes:

Validation

```
<form #empForm=ngForm (ngSubmit)="getData(empForm)">
<table>
<tr>
<td>Product ID</td>
<td><input required id="eid"  name ="id" [(ngModel)]="emp.eId"
type="text" #idcontrol="ngModel"/>
<span  *ngIf="idcontrol.invalid && idcontrol.touched" > ID is
required</span>
</td>
</tr>
<tr>
<td>Product Name</td>
<td><input required id="ename" name ="empname"
[(ngModel)]="emp.eName" type="text"
#namecontrol="ngModel"/></td>
<span  *ngIf="namecontrol.invalid && namecontrol.touched" >
Name is required</span>
</tr></table>
```

Instructor Notes:

Add instructor notes here.



Add the notes here.

Instructor Notes:

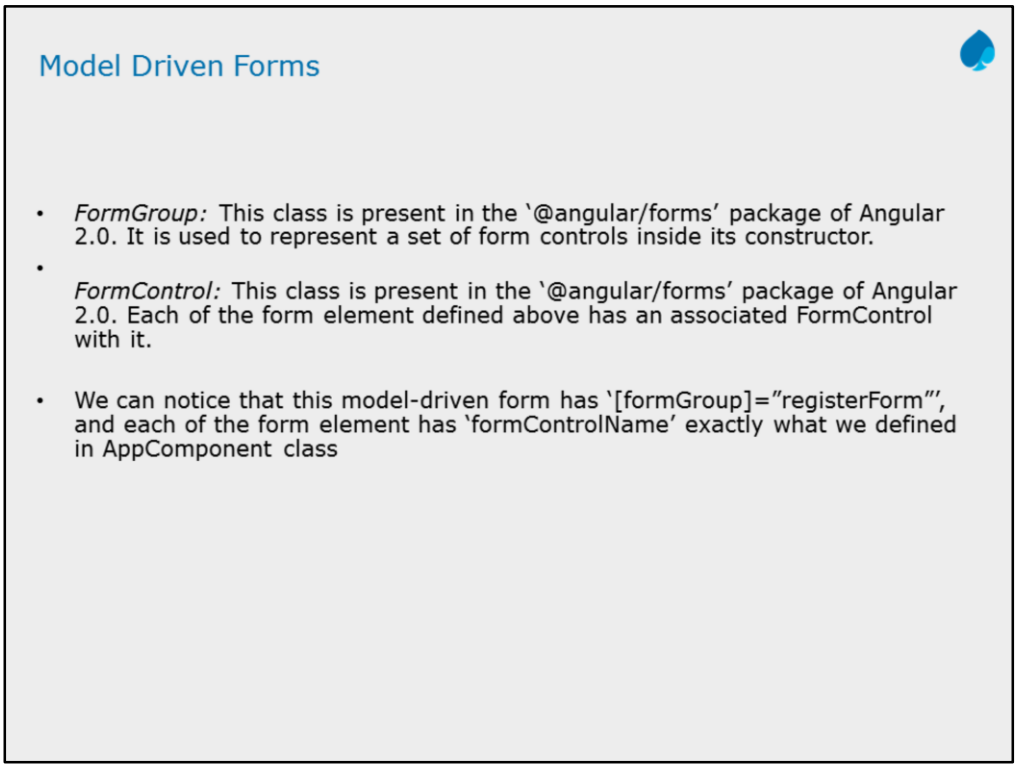
Add instructor notes here.

Model Driven Forms

Difference Between template driven & model driven form

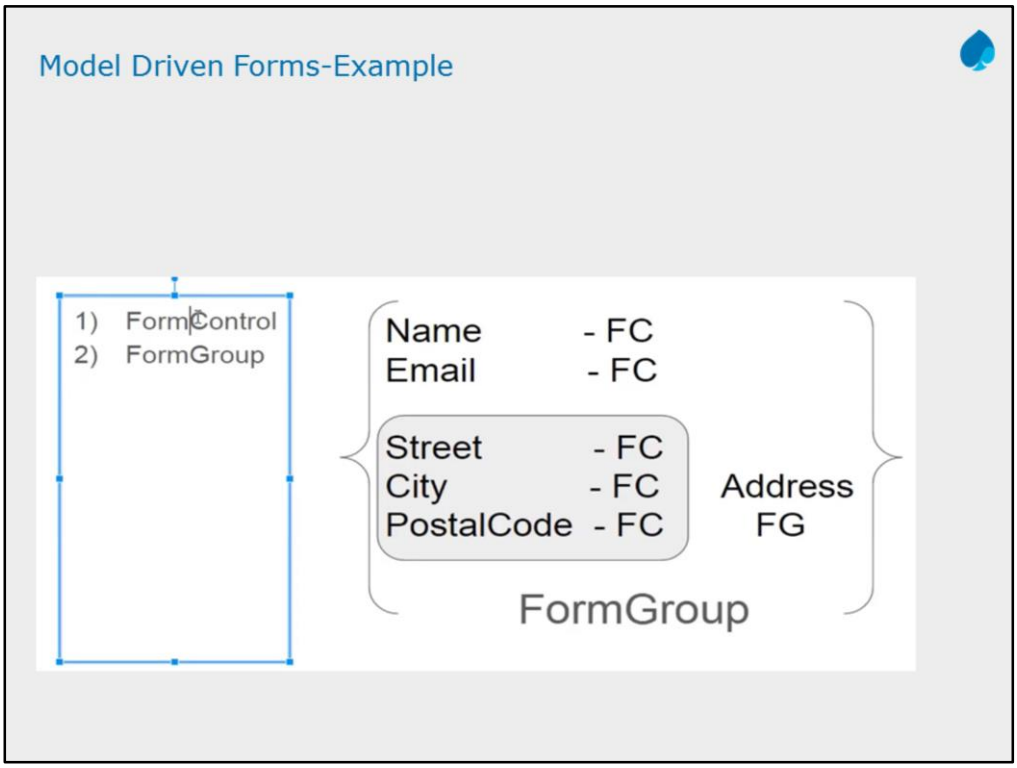
- Both template-driven form and model-driven form differs from each other as the former allows us to write as little
- JavaScript code as possible to prepare the sophisticated forms and the latter makes the testing of a form easy as it doesn't require end-to-end testing. It prepares forms imperatively out of the properties on the available components.
- Model-driven forms are also known as reactive forms and can be thought of as the addition to the template-driven forms such as validators on DOM elements etc.

Instructor Notes:



myForm will be our model driven form. It implements FormGroup interface. FormBuilder is not a mandatory to building model driven form, but it simplify the syntax, we'll cover this later.
A form is a type of **FormGroup**. A **FormGroup** can contain one **FormGroup** or **FormControl**

Instructor Notes:



It has the selector 'my-modal-form-app' which is used on the index.html to load the complete app on the browser. It has the template URL as 'resource/app-component.html' and styleUrls as 'assets/styles.css' as shown below. In the controller class 'AppComponent', we are implementing OnInit interface for the method 'ngOnInit ()', where we are using the FormGroup and FormControl classes to create a model-driven form in Angular 2.0. Both FormGroup and FormControl are low level APIs where the FormGroup always represents a set of FormControls. They collectively form the model-driven form in Angular 2.0.
FormGroup: This class is present in the '@angular/forms' package of Angular 2.0. It is used to represent a set of form controls inside its constructor as shown above.
FormControl: This class is present in the '@angular/forms' package of Angular 2.0. Each of the form element defined above has an associated FormControl with it.
t is the HTML code template for 'app-component-ts'. Here, we can notice that this model-driven form has '[formGroup]="registerForm"', and each of the form element has 'formControlName' exactly what we defined in AppComponent class. Also, for the nested FormGroup 'address', we have used the 'fieldset' tag with the 'formGroupName' directive

Instructor Notes:

Add instructor notes here.

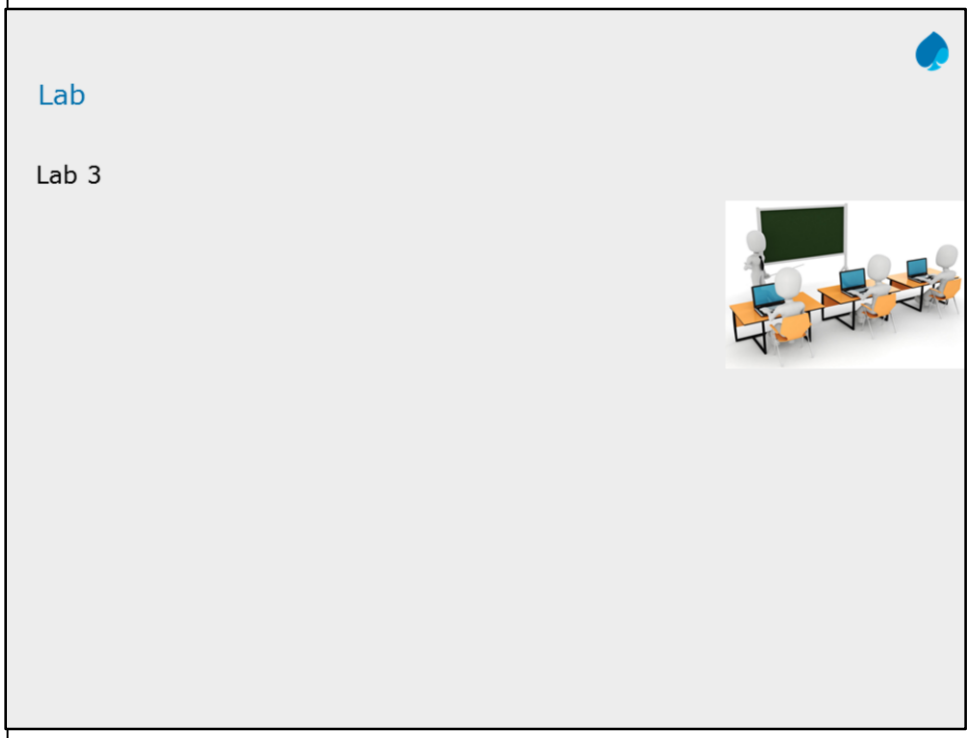
Demo

DemoModelDrivenForms

Add the notes here.

Instructor Notes:

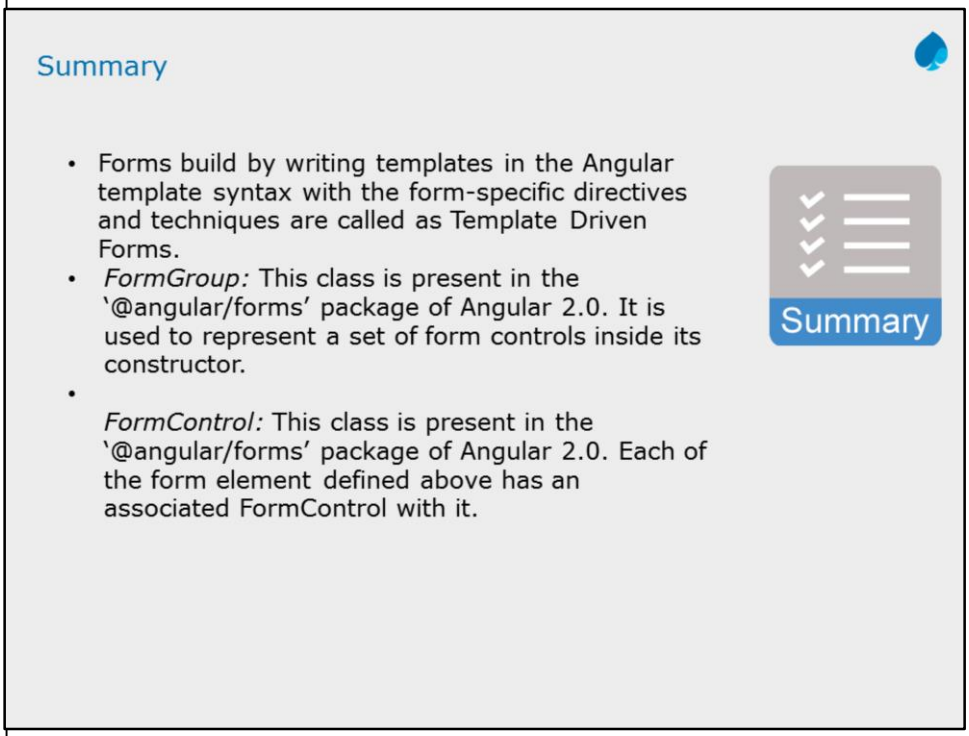
Add instructor notes here.



Add the notes here.

Instructor Notes:

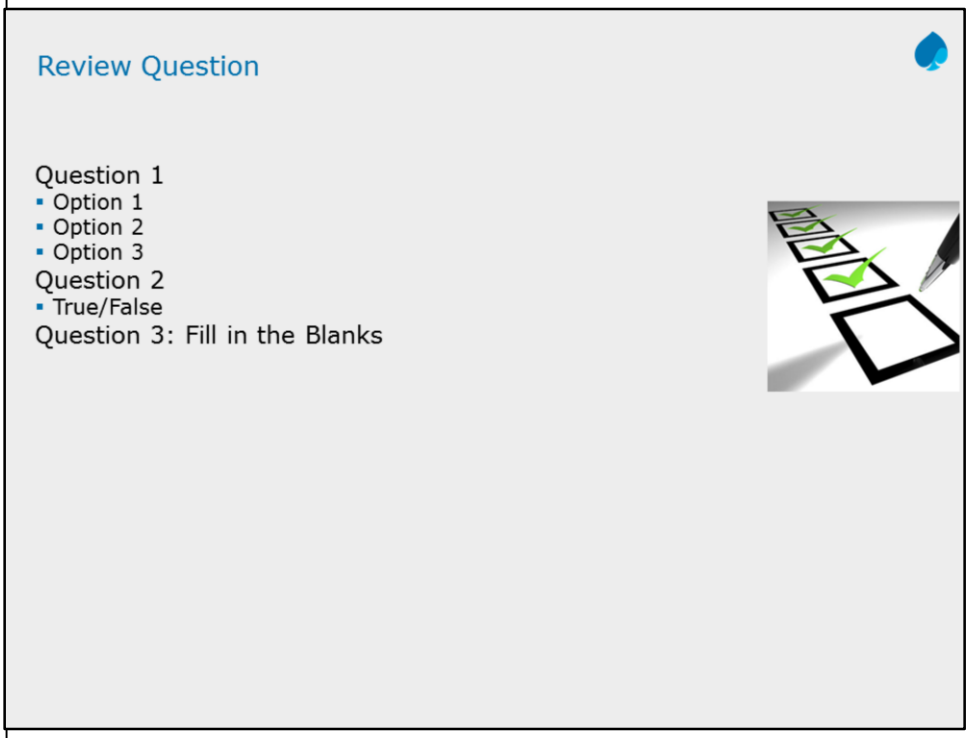
Add instructor notes here.



Add the notes here.

Instructor Notes:

Add instructor notes here.



Add the notes here.