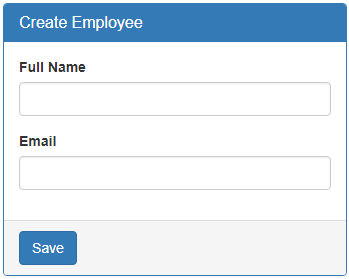
**There are 2 ways to create forms in Angular**

1. Template Driven Forms
2. Reactive Forms (Also called Model Driven Forms)

As the name implies, Template Driven Forms are heavy on the template meaning we create the form completely in HTML. Template driven forms are easy to build and understand. They are great for creating simple forms. However, creating complex forms using template driven approach is not recommended as the HTML can get very complicated and messy. It is not easy to unit test template forms as the logic is in the HTML.  
  
We discussed template driven forms in [Part 5](https://www.youtube.com/watch?v=pwQ3L3UFEjk) of [Angular CRUD Tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5)  
  
**Reactive forms**on the other hand allow us to build the form completely in code. This is more flexible and has many benefits over template forms. For example, it is easy to add form input elements dynamically and adjust validation at run-time based on the decisions made in code. It is also easy to unit test as most of the logic and validation is in the component class. The only downside of reactive forms is that they require more code than template forms.  
  
In this video and in our upcoming videos we will discuss everything we need to know to build complex reactive forms.  
  
With a reactive form, we create the entire form control tree in the component class code. Let us understand this by creating a simple form with just 2 form controls as shown below.  
  
  
  
**Creating a form group model :**Two classes that we commonly use to create a form control tree is **FormGroup**and **FormControl**. As the names imply to create a form with a group of controls, we create an instance of FormGroup class and to create each input element i.e a form control, we create an instance of FormControl class. So in the CreateEmployeeComponent (create-employee.component.ts) class modify the code as shown below.

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

// Import FormGroup and FormControl classes

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

@Component({

  selector: 'app-create-employee',

  templateUrl: './create-employee.component.html',

  styleUrls: ['./create-employee.component.css']

})

export class CreateEmployeeComponent implements OnInit {

  // This FormGroup contains fullName and Email form controls

  employeeForm: FormGroup;

  constructor() { }

  // Initialise the FormGroup with the 2 FormControls we need.

  // ngOnInit ensures the FormGroup and it's form controls are

  // created when the component is initialised

  ngOnInit() {

    this.employeeForm = new FormGroup({

      fullName: new FormControl(),

      email: new FormControl()

    });

  }

}

Right click and go to the definition on **FormGroup**class constructor. Notice it has 3 parameters.

constructor(controls: { [key: string]: AbstractControl;},

validatorOrOpts?: ValidatorFn | ValidatorFn[] | AbstractControlOptions | null,

asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[] | null);

The first parameter (**controls**) is required, but the rest of the 2 parameters are optional. As you can see, the controls parameter is used to pass the collection of child controls. In our case we want 2 child controls in the FormGroup - fullName and email. So we pass an object with key/value pairs. The key is the name for the control and the value is an instance of the FormControl class. But, wait a minute, from the intellisense, I see that the value is AbstractControl and not FormControl.

constructor(controls: { [key: string]: AbstractControl;}

So the obvious question that comes to our mind is, **how are we able to pass a FormControl instance when it is expecting AbstractControl instance.**  
  
Well, FormControl class inherits from AbstractControl class. This allows us to pass FormControl instance as the value. Both FormControl and FormGroup classes inherit from AbstractControl class. This allows us to pass either a FormControl or a FromGroup instance as the value.   
  
If you are wondering, **why do we need to pass a FromGroup instance as the value**.  
  
Well, a FormGroup can have a nested FormGroup. We will discuss nested form groups in our upcoming videos.  
  
**Binding the FormGroup model and the view :**Copy and paste the following HTML in create-employee.component.html file. This is pure HTML. There is no Angular code in this HTML. We have a <form> element and 2 text input elements (for fullName and email)

<form class="form-horizontal">

  <div class="panel panel-primary">

    <div class="panel-heading">

      <h3 class="panel-title">Create Employee</h3>

    </div>

    <div class="panel-body">

      <div class="form-group">

        <label class="col-sm-2 control-label" for="fullName">Full Name</label>

        <div class="col-sm-8">

          <input id="fullName" type="text" class="form-control">

        </div>

      </div>

      <div class="form-group">

        <label class="col-sm-2 control-label" for="email">Email</label>

        <div class="col-sm-8">

          <input id="email" type="text" class="form-control">

        </div>

      </div>

    </div>

    <div class="panel-footer">

      <button class="btn btn-primary" type="submit">Save</button>

    </div>

  </div>

</form>

Now we need to **bind the view template to the form group model**we created in the component class. For this we make use of the following 2 directives provided by Angular ReactiveFroms module.

* formGroup
* formControlName

Here is the modified HTML. Notice the use of **formGroup**and **formControlName**directives in the <form> element and the 2 input elements.

<form class="form-horizontal" ***[formGroup]="employeeForm"***>

  <div class="panel panel-primary">

    <div class="panel-heading">

      <h3 class="panel-title">Create Employee</h3>

    </div>

    <div class="panel-body">

      <div class="form-group">

        <label class="col-sm-2 control-label" for="fullName">Full Name</label>

        <div class="col-sm-8">

          <input id="fullName" type="text" class="form-control" ***formControlName="fullName"***>

        </div>

      </div>

      <div class="form-group">

        <label class="col-sm-2 control-label" for="email">Email</label>

        <div class="col-sm-8">

          <input id="email" type="text" class="form-control" ***formControlName="email"***>

        </div>

      </div>

    </div>

    <div class="panel-footer">

      <button class="btn btn-primary" type="submit">Save</button>

    </div>

  </div>

</form>

**Please note :**

* To bind the <form> element to the employeeForm group in the component class we use the formGroup directive. Since "employeeForm" is a property we use square brackets around the formGroup directive to indicate that we are binding to a property.
* To bind each input element to the associated FormControl in the FormGroup model, we use formControlName directive. Notice we are not using square brackets with formControlName directive. This is because, in this case we are binding to a form control name which is a string and not a property.

At this point, if you view the page in the browser, you will see the following error in the browser console.  
**Can't bind to 'formGroup' since it isn't a known property of 'form'**

This is because, the 2 directives (formGroup and formControlName) are in ReactiveForms module, but we have not yet imported it in our root module. So in the AppModule (**app.module.ts file**), import ReactiveFormsModule and include it in the **imports**array.

import { ReactiveFormsModule } from '@angular/forms';

**Accessing form data :**To access form data, bind to the ngSubmit event on the <form> element. This ngSubmit event is raised when a button with input type=submit is clicked.

<form class="form-horizontal" [formGroup]="employeeForm"

      (ngSubmit)="onSubmit()">

In the component class (**create-employee.component.ts**), include **onSubmit()** method as shown below.

onSubmit(): void {

  console.log(this.employeeForm.value);

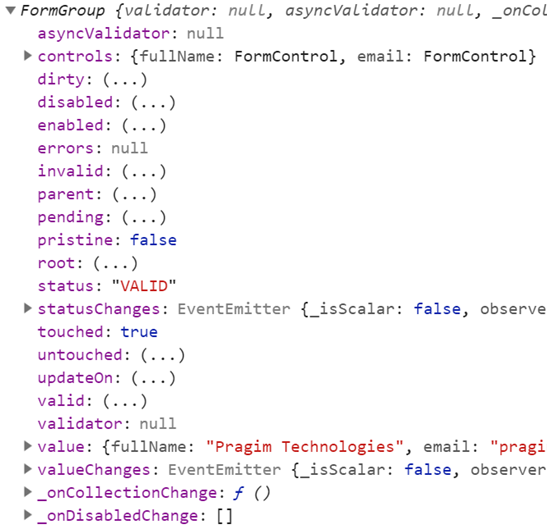
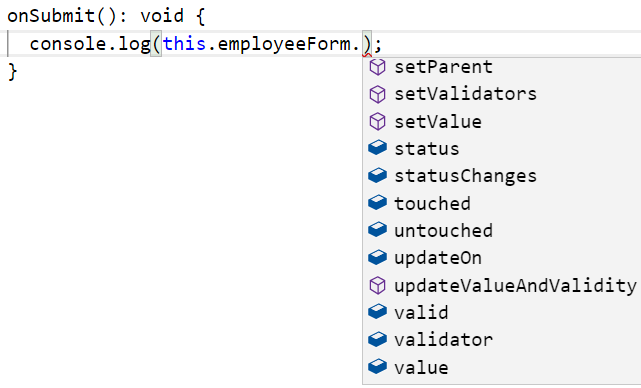
}

At this point, fill out the form and click **Save**button. Notice, the **Formgroup**value property is logged to the console. The value property of the **FormGroup**contains each form control name and it's associated value.  
  
At the moment our reactive form is a very simple form with just 2 text box input controls. As we progress through this course will discuss working with checkboxes, radio buttons, dropdownlists etc. We will also discuss form validation, nested form groups, dynamically creating form controls etc.

* When working with reactive forms we create instances of **FormControl**and **FormGroup**classes to create a form model.
* To bind an HTML <form> tag in the template to the **FromGroup**instance in the component class, we use **formGroup**directive
* To bind an  HTML <input> element in the template to the **FormControl**instance in the component class, we use **formControlName**directive
* **formGroup**and **formControlName**directives are provided by the **ReactiveFormsModule**
* Both **FormControl**and **FormGroup**classes inherit from **AbstractControl**base class
* The **AbstractControl**class has properties that help us track both **FormControl**and **FormGroup**value and state

**The following are some of the useful properties provided by the AbstractControl class**

* value
* errors
* valid
* invalid
* dirty
* pristine
* touched
* untouched

**FormControl**instance tracks the value and state of the individual html element it is associated with  
**FormGroup**instance tracks the value and state of all the form controls in it's group  
  
To see the form model we created using **FormGroup**and **FormControl**classes, log the employeeForm to the console.  
  
onSubmit(): void {  
  console.log(this.employeeForm);  
}  
  
On Save button click, you should see the following form model in the browser console.  
  
  
  
To access the FormGroup properties use, **employeeForm**property in the component class. When you press DOT on the employeeForm property you can see all the available properties and methods.  
  
  
  
To access a **FormControl**in a **FormGroup**, we can use one of the following 2 ways.

employeeForm.controls.fullName.value  
employeeForm.get('fullName').value

**Note:**This same code works, both in the template and component class.  
  
Please include the following HTML, just after the <form> tag, in the template, and you can see the property values change as you interact with the form controls on the form.

<table border="1">

  <tr>

    <th style="padding: 10px">FormGroup</th>

    <th style="padding: 10px">FormControl (fullName)</th>

  </tr>

  <tr>

    <td style="padding: 10px">

      touched : {{ employeeForm.touched }}

      <br/> dirty : {{ employeeForm.dirty }}

      <br/> valid : {{ employeeForm.valid }}

      <br/> Form Values : {{employeeForm.value | json}}

    </td>

    <td style="padding: 10px">

      touched : {{ employeeForm.get('fullName').touched }}

      <br/> dirty : {{ employeeForm.get('fullName').dirty }}

      <br/> valid : {{ employeeForm.get('fullName').valid }}

      <br/> FullName Value : {{employeeForm.get('fullName').value}}

    </td>

  </tr>

</table>

In addition to these properties, **AbstractControl**also provides the following methods. In our upcoming videos we will use these properties and methods for form validation and working with data.

* setValidators()
* clearValidators()
* updateValueAndValidity()
* setValue()
* patchValue()
* Reset()