

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.
(AFFILIATED TO SAURASHTRA UNIVERSITY)



Lt. Shree Chimanbhai Shukla

MSCIT SEM-3 ANGULARJS

**Shree H.N.Shukla college2
vaishali nagar
Near Amrapali Under Bridge,
Raiya road
Rajkot
Ph No:-0281 2440478**

**Shree H.N.Shukla college3
vaishali nagar
Near Amrapali Under Bridge,
Raiya road
Rajkot
Ph No:-0281 2440478**

Unit : 2

DATA BINDING IN ANGULAR

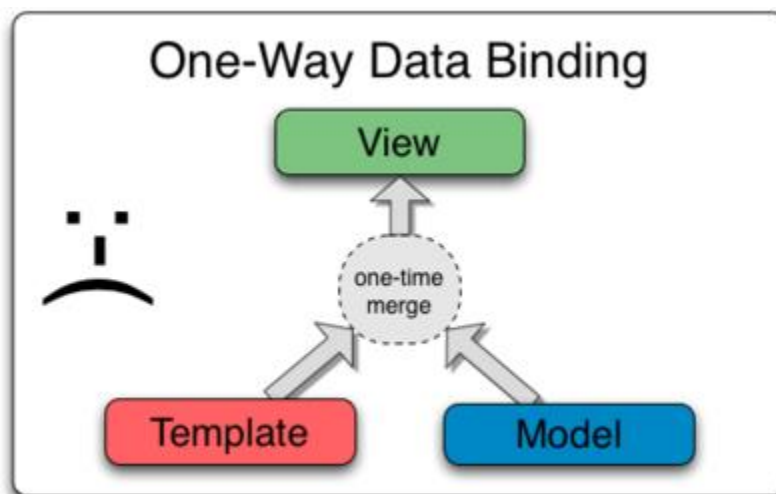
Data Binding?

- ✓ Data-binding in AngularJS apps is the automatic synchronization of data between the model and view components.
- ✓ The way that AngularJS implements data-binding lets you treat the model as the single-source-of-truth in your application.
- ✓ The view is a projection of the model at all times.
- ✓ When the model changes, the view reflects the change, and vice versa.

❖ **AngularJS provides two types of Data Binding:**

- One-way data binding.
- Two-way data binding.

❖ Data Binding in Classical Template Systems



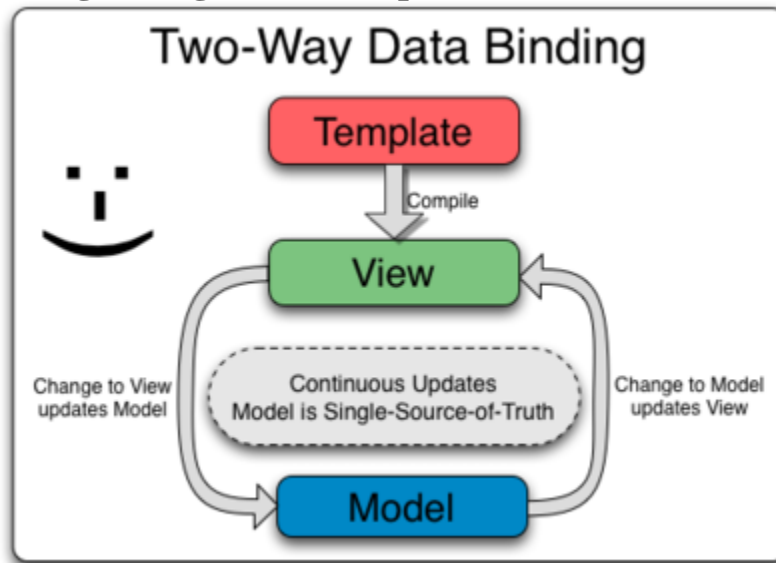
Most templating systems bind data in only one direction: they merge template and model components together into a view. After the merge occurs, changes to the model or related sections of the view are NOT automatically reflected in the view.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

Worse, any changes that the user makes to the view are not reflected in the model. This means that the developer has to write code that constantly syncs the view with the model and the model with the view.

❖ Data Binding in AngularJS Templates



AngularJS templates work differently. First the template (which is the uncompiled HTML along with any additional markup or directives) is compiled on the browser. The compilation step produces a live view. Any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view. The model is the single-source-of-truth for the application state, greatly simplifying the programming model for the developer. You can think of the view as simply an instant projection of your model.

Because the view is just a projection of the model, the controller is completely separated from the view and unaware of it. This makes testing a snap because it is easy to test your controller in isolation without the view and the related DOM/browser dependency.

reflected in the view. Worse, any changes that the user makes to the view are not reflected in the model.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

Property binding

Property binding in Angular helps you set values for properties of HTML elements or directives. Use property binding to do things such as toggle button features, set paths programmatically, and share values between components.

❖ Understanding the flow of data

Property binding moves a value in one direction, from a component's property into a target element property.

For more information on listening for events, see Event binding.

To read a target element property or call one of its methods, see the API reference for ViewChild and ContentChild.

Binding to a property

To bind to an element's property, enclose it in square brackets, [], which identifies the property as a target property.

A target property is the DOM property to which you want to assign a value.

- ✓ To assign a value to a target property for the image element's src property, type the following code:

```
src/app/app.component.html  
content_copy<img alt="item" [src]="itemImageUrl">
```

In most cases, the target name is the name of a property, even when it appears to be the name of an attribute.

In this example, src is the name of the `` element property.

The brackets, [], cause Angular to evaluate the right-hand side of the assignment as a dynamic expression.

Without the brackets, Angular treats the right-hand side as a string literal and sets the property to that static value.

- ✓ To assign a string to a property, type the following code:

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

src/app.component.html

```
content_copy<app-item-detail childItem="parentItem"></app-item-detail>
```

Omitting the brackets renders the string parentItem, not the value of parentItem.

Setting an element property to a component property value

To bind the src property of an element to a component's property, place src in square brackets followed by an equal sign and then the property.

- ✓ Using the property itemImageUrl, type the following code:

✓

src/app/app.component.html

```
content_copy<img alt="item" [src]="itemImageUrl">
```

- ✓ Declare the itemImageUrl property in the class, in this case AppComponent.

src/app/app.component.ts

```
content_copyitemImageUrl = '../assets/phone.svg';
```

colspan and colSpan

A common point of confusion is between the attribute, colspan, and the property, colSpan. Notice that these two names differ by only a single letter.

- ✓ To use property binding using colSpan, type the following:

src/app/app.component.html

```
<tr><td [colSpan]="1 + 1">Three-Four</td></tr>
```

- ✓ To disable a button while the component's isUnchanged property is true, type the following:

src/app/app.component.html

```
<button type="button" [disabled]="isUnchanged">Disabled Button</button>
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

- ✓ To set a property of a directive, type the following:

src/app/app.component.html

content_copy<p [ngClass]="classes">[ngClass] binding to the classes property making this blue</p>

- ✓ To set the model property of a custom component for parent and child components to communicate with each other, type the following:

src/app/app.component.html

content_copy<app-item-detail [childItem]="parentItem"></app-item-detail>

Toggle button features

- ✓ To use a Boolean value to disable a button's features, bind the disabled DOM attribute to a Boolean property in the class.

src/app/app.component.html

content_copy<!-- Bind button disabled state to `isUnchanged` property -->

<button type="button" [disabled]="isUnchanged">Disabled Button</button>

Because the value of the property isUnchanged is true in the AppComponent, Angular disables the button.

src/app/app.component.ts

content_copyisUnchanged = true;

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.
(AFFILIATED TO SAURASHTRA UNIVERSITY)

 **Difference between String interpolation and Property Binding**

	String Interpolation	Property Binding
Syntax	<code>`\${variable}`</code>	<code>`[]`</code>
Usage	Used to insert variables into strings, such as in template literals or in JSX in React.	Used to bind variables or properties to HTML attributes or DOM properties.
Variable/Property Evaluation	is Evaluated at runtime, meaning any changes to the variables will be reflected in the interpolated string.	Evaluated when the property is bound and does not update automatically if the value changes later.
Applicable to	Strings and templates.	HTML elements and DOM properties.
Dynamic Updating	Automatically update the string when the variable changes.	Require manual re-binding if the variable changes.
Data Types	Work with strings, numbers, and boolean values.	Can bind to any data type, including arrays, objects, and functions.
Compatibility	Works with most modern browsers.	Works with Angular, React, and other front-end frameworks.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

Event binding

- ✓ Event binding lets you listen for and respond to user actions such as keystrokes, mouse movements, clicks, and touches.
- ✓ Binding to events
- ✓ To bind to an event you use the Angular event binding syntax. This syntax consists of a target event name within parentheses to the left of an equal sign, and a quoted template statement to the right.
- ✓ Create the following example; the target event name is `click` and the template statement is `onSave()`.
- ✓ Event binding syntax
- ✓

```
content_copy<button (click)="onSave()">Save</button>
```
- ✓ The event binding listens for the button's click events and calls the component's `onSave()` method whenever a click occurs.

`<button (click)="onSave()">Save</button>`

target event name

template statement

- ✓ Determining an event target
- ✓ To determine an event target, Angular checks if the name of the target event matches an event property of a known directive.
- ✓ Create the following example: (Angular checks to see if `myClick` is an event on the custom `ClickDirective`)
- ✓ `src/app/app.component.html`
- ✓

```
content_copy<h4>myClick is an event on the custom ClickDirective:</h4>
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

- ✓ `<button type="button" (myClick)="clickMessage=$event" clickable>click`
with `myClick</button>`
 - ✓ `{{clickMessage}}`
 - ✓ If the target event name, `myClick` fails to match an output property of `ClickDirective`, Angular will instead bind to the `myClick` event on the underlying DOM element.
-
- ✓ Binding to passive events
 - ✓ This is an advanced technique that is not necessary for most applications. You may find this useful if you want to optimize frequently occurring events that are causing performance problems.
 - ✓ Angular also supports passive event listeners. For example, use the following steps to make a scroll event passive.
 - ✓ Create a file `zone-flags.ts` under `src` directory.
 - ✓ Add the following line into this file.
 - `content_copy(window as any)['__zone_symbol__PASSIVE_EVENTS'] = ['scroll'];`
 - ✓ In the `src/polyfills.ts` file, before importing `zone.js`, import the newly created `zone-flags`.
 - ✓ `content_copy`
 - ✓ `import './zone-flags';`
 - `import 'zone.js'; // Included with Angular CLI.`
 - ✓ After those steps, if you add event listeners for the scroll event, the listeners will be passive.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

-
- ✓ Binding to keyboard events
 - ✓ You can bind to keyboard events using Angular's binding syntax. You can specify the key or code that you would like to bind to keyboard events. The key and code fields are a native part of the browser keyboard event object. By default, event binding assumes you want to use the key field on the keyboard event. You can also use the code field.
 - ✓ Combinations of keys can be separated by a . (period). For example, `keydown.enter` will allow you to bind events to the enter key. You can also use modifier keys, such as shift, alt, control, and the command keys from Mac. The following example shows how to bind a keyboard event to `keydown.shift.t`.
 - ✓ `content_copy<input (keydown.shift.t)="onKeydown($event)" />`
 - ✓ Depending on the operating system, some key combinations might create special characters instead of the key combination that you expect. MacOS, for example, creates special characters when you use the option and shift keys together. If you bind to `keydown.shift.alt.t`, on macOS, that combination produces a `~` character instead of a `t`, which doesn't match the binding and won't trigger your event handler. To bind to `keydown.shift.alt.t` on macOS, use the code keyboard event field to get the correct behavior, such as `keydown.code.shiftright.altleft.key` shown in this example.
 - ✓ `content_copy<input (keydown.code.shiftright.altleft.key)="onKeydown($event)" />`
 - ✓ The code field is more specific than the key field. The key field always reports shift, whereas the code field will specify leftshift or rightshift. When using the code field, you might need to add separate bindings to catch all the behaviors you want. Using the code field avoids the need to handle OS specific behaviors such as the shift + option behavior on macOS.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

Angular 7 Components

Components are the key features of Angular. The whole application is built by using different components.

The core idea behind Angular is to build components. They make your complex application into reusable parts which you can reuse very easily.

How to create a new component?

Open WebStorm>> Go to your project source folder>> Expand the app directory and create a new directory named "server".

Now, create the component within server directory. Right click on the server directory and create a new file named as "server.component.ts". It is the newly created component.

Components are used to build webpages in Angular but they require modules to bundle them together. Now, you have to register our new components in module.

Creating component with CLI

Syntax

```
ng generate component component_name
```

Or

```
ng g c component_name
```

Let's see how to create a new component by using command line.

Open Command prompt and stop **ng serve** command if it is running on the browser.

Type **ng generate component server2** to create a new component named server2.

You can also use a shortcut **ng g c server2** to do the same task.

In the above image, you can see that a new component named "server2" is created. It contains the same other components which you have seen when we create a first app..

1. server2.component.css
2. server2.component.html
3. server2.component.spec.ts
4. server2.component.ts

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

Here, **server2.component.spec.ts** component is used for testing purpose. You can delete it by right click on that.

What is async in Angular?

In Angular, the async pipe is a pipe that essentially does these three tasks: It subscribes to an observable or a promise and returns the last emitted value. Whenever a new value is emitted, it marks the component to be checked. That means Angular will run Change Detector for that component in the next cycle.

Async functions with AngularJS

April 5, 2015 Vijay Thirugnanam

AngularJS allows to define async functions. An example is the **\$http** service. HTTP service calls an API and returns a response as a promise.

```
$http.get(url)
  .then(function(res){
    // handle the response
  }, function(res){
    // handle the error
  });
```

Retrieve the result from the Promise object using then function. The function has two arguments: onSuccess callback and onFailure callback.

The **\$q** service in AngularJS creates a promise using the defer function.

```
$scope.callDb = function () {
  var deferred = $q.defer();
  promise.then(function(res){
    var someObj = {};
    deferred.resolve(someObj);
  }, function(err){
    var errObj = {};
    deferred.reject(errObj);
  });
  return deferred.promise;
};
```

Pass the result using the resolve function. If there is an error, pass the error using reject function.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

I want to give a concrete example. Cordova has a SQLite plugin ([ngCordova](#)). Create a DbService that returns a list of employees.

```
app.service('DbService',
function ($q, $cordovaSQLite) {
  this.get = function() {
    var q = $q.defer();
    var query = "SELECT idemployee, name from employee";
    var db = $cordovaSQLite.openDB("employee.sqlite", 0);
    $cordovaSQLite.execute(db, query, []).then(function (res) {
      var employees = [];
      for (index = 0; index < res.rows.length; index++) {
        employees.push(res.rows.item(index));
      }
      q.resolve(employees);
    }, function (err) {
      q.reject(null);
    });
    return q.promise;
  };
});
```

In the above example, DbService is an Angular service. We have a SQL that retrieves a list of employees. SQLite plugin has an execute function which queries the database using the SQL. The plugin returns a promise. Retrieve a list of employees from the promise. And return a new promise using Angular's \$q service.

Call the DbService to get the employee list. The get method returns a promise. From the then function of the promise, we set the employees to the scope object.

```
function get() {
  DbService.get()
  .then(function(employees){
    $scope.employees = employees;
  });
}
```

Bind the Angular's scope object to a table using [ng-repeat](#) directive.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

Template Interpolation in Angular

In AngularJS, Interpolation is a way to transfer the data from a TypeScript code to an HTML template (view), i.e. it is a method by which we can put an expression in between some text and get the value of that expression. Interpolation basically binds the text with the expression value.

❖ Interpolation and template expressions

Interpolation allows you to incorporate calculated strings into the text between HTML element tags and within attribute assignments. Template expressions are what you use to calculate those strings.

✓ syntax and code snippets in this guide.

Interpolation `{{...}}`

Interpolation refers to embedding expressions into marked up text. By default, interpolation uses as its delimiter the double curly braces, `{{` and `}}`.

❖ In the following snippet, `{{ currentCustomer }}` is an example of interpolation.

src/app/app.component.html

content_copy<h3>Current customer: `{{ currentCustomer }}`</h3>

The text between the braces is often the name of a component property. Angular replaces that name with the string value of the corresponding component property.

src/app/app.component.html

content_copy<p>`{{title}}`</p>

<div></div>

In the example above, Angular evaluates the title and imageUrl properties and fills in the blanks, first displaying some title text and then an image.

More generally, the text between the braces is a template expression that Angular first evaluates and then converts to a string. The following interpolation illustrates the point by adding two numbers:

src/app/app.component.html

content_copy<!-- "The sum of 1 + 1 is 2" -->

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

`<p>The sum of 1 + 1 is {{1 + 1}}.</p>`

The expression can invoke methods of the host component such as `getVal()` in the following example:

`src/app/app.component.html`

`content_copy<!-- "The sum of 1 + 1 is not 4" -->`

`<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}.</p>`

Angular evaluates all expressions in double curly braces, converts the expression results to strings, and links them with neighboring literal strings. Finally, it assigns this composite interpolated result to an element or directive property.

You appear to be inserting the result between element tags and assigning it to attributes. However, interpolation is a special syntax that Angular converts into a *property binding*.

If you'd like to use something other than `{{ and }}`, you can configure the interpolation delimiter via the [interpolation](#) option in the [Component](#) metadata.

Template expressions

A template expression produces a value and appears within the double curly braces, `{{ }}`. Angular executes the expression and assigns it to a property of a binding target; the target could be an HTML element, a component, or a directive. The interpolation braces in `{{1 + 1}}` surround the template expression `1 + 1`. In the property binding, a template expression appears in quotes to the right of the `=` symbol as in `[property]="expression"`.

In terms of syntax, template expressions are similar to JavaScript. Many JavaScript expressions are legal template expressions, with a few exceptions.

You can't use JavaScript expressions that have or promote side effects, including:

- Assignments (`=`, `+=`, `-=`, ...)
- Operators such as `new`, `typeof`, `instanceof`, etc.
- Chaining expressions with `;` or `,`
- The increment and decrement operators `++` and `--`
- Some of the ES2015+ operators

Other notable differences from JavaScript syntax include:

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

- No support for the bitwise operators such as | and &
- New [template expression operators](#), such as |, ?. and !

Expression context

The *expression context* is typically the *component* instance. In the following snippets, the recommended within double curly braces and the `itemImageUrl2` in quotes refer to properties of the `AppComponent`.

src/app/app.component.html

```
content_copy<h4>{{recommended}}</h4>
```

```
<img [src]="itemImageUrl2">
```

An expression may also refer to properties of the *template's* context such as a template input variable,

let customer, or a template reference variable, #customerInput.

src/app/app.component.html (template input variable)

```
content_copy<ul>
  <li *ngFor="let customer of customers">{{customer.name}}</li>
</ul>
```

src/app/app.component.html (template reference variable)

```
content_copy<label>Type something:
  <input #customerInput>{{customerInput.value}}
</label>
```

The context for terms in an expression is a blend of the *template variables*, the directive's *context* object (if it has one), and the component's *members*. If you reference a name that belongs to more than one of these namespaces, the template variable name takes precedence, followed by a name in the directive's *context*, and, lastly, the component's member names.

The previous example presents such a name collision. The component has a customer property and the `*ngFor` defines a customer template variable.

The customer in `{{customer.name}}` refers to the template input variable, not the component's property.

Template expressions cannot refer to anything in the global namespace, except undefined. They can't refer to window or document. Additionally, they

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

can't call `console.log()` or `Math.max()` and they are restricted to referencing members of the expression context.

Expression guidelines

When using template expressions follow these guidelines:

- [Simplicity](#)
- [Quick execution](#)
- [No visible side effects](#)

❖ **Simplicity**

Although it's possible to write complex template expressions, it's a better practice to avoid them.

A property name or method call should be the norm, but an occasional Boolean negation, `!`, is OK. Otherwise, confine application and business logic to the component, where it is easier to develop and test.

❖ **Quick execution**

Angular executes template expressions after every change detection cycle. Change detection cycles are triggered by many asynchronous activities such as promise resolutions, HTTP results, timer events, key presses and mouse moves.

Expressions should finish quickly or the user experience may drag, especially on slower devices. Consider caching values when their computation is expensive.

❖ **No visible side effects**

A template expression should not change any application state other than the value of the target property.

This rule is essential to Angular's "unidirectional data flow" policy. You should never worry that reading a component value might change some other displayed value. The view should be stable throughout a single rendering pass.

An [idempotent](#) expression is ideal because it is free of side effects and improves Angular's change detection performance. In Angular terms, an idempotent expression always returns *exactly the same thing* until one of its dependent values changes.

Dependent values should not change during a single turn of the event loop. If an idempotent expression returns a string or a number, it returns the same string or number when called twice in a row. If the expression returns an object, including an array, it returns the same object *reference* when called twice in a row.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

There is one exception to this behavior that applies to `*ngFor`. `*ngFor` has `trackBy` functionality that can deal with referential inequality of objects when iterating over them. See [*ngFor with trackBy](#) for details.

Two-way Data Binding in AngularJS

- In this article, we will see the Data Binding, along with understanding how the flow of code is from a Typescript file to an HTML file & vice-versa through their implementation.

Two-way data binding enables the communication between the form and the class. It allows for changes in the fields of the model to automatically appear in the form controller, and for the data entered by the user to immediately update the model.

In AngularJS, **Data Binding** refers to the synchronization between the model and view. In **Two-way data binding**, the flow of data is bidirectional i.e. when the data in the model changes, the changes are reflected in the view and when the data in the view changes it is reflected in the model. Two-way data binding is achieved by using the [ng-model directive](#). The `ng-model` directive transfers data from the view to the model and from the model to the view.

Approach: The following approach will be implemented to achieve the Two-way Binding:

- Create a module

```
var app=angular.module('myApp',[])
```

- Add a controller to the module. Here you can write the logic for updating the model as the view changes.

```
app.controller('myCtrl',function($scope){})
```

- Specify the `ng-model` directive in the element
`<input ng-model="name"/>`

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

Angular form, NgForm and two-way data binding

Published August 15, 2018

One of the most important components of any website is the HTML form because it allows users to interact with the app. In this tutorial, we will explore the Angular way to produce smart and highly interactive forms.

The example in the tutorial is of a form into which users enter the details of a car model, such as the name and price of the model, and whether it is a convertible.

The two-way binding syntax is:

```
[(ngModel)]="fieldName"
```

where `fieldName` stands for the name of the field in the model.

The field name must be unique because it identifies the field.

Add the two-way data binding to the HTML form by attaching a `name` and `ngModel` to each controller. For example:

```
<input type="text" name="name" [(ngModel)]="model.name">
```

app.component.html

```
<form>
```

```
  <label>Name</label>
```

```
  <input name="name" [(ngModel)]="carModel.name" required>
```

```
  <br />
```

```
  <select name="motor" [(ngModel)]="carModel.motor" required>
```

```
    <option *ngFor="let motor of motors" [value]="motor">{{motor}}</option>
```

```
</select>
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
<br />
<label><input type="radio" name="hasSunroof"
[(ngModel)]="carModel.hasSunroof" [value]="true">Has sunroof</label>
<label><input type="radio" name="hasSunroof"
[(ngModel)]="carModel.hasSunroof" [value]="false">No sunroof</label>

<br />
<input type="submit" value="Save">
</form>
```

[Attribute directives](#)

Change the appearance or behavior of an element, component, or another directive.

[Structural directives](#)

Change the DOM layout by adding and removing DOM elements.

This guide covers built-in [attribute directives](#) and [structural directives](#).

Built-in attribute directives

Attribute directives listen to and modify the behavior of other HTML elements, attributes, properties, and components.

Many NgModules such as the [RouterModule](#) and the [FormsModule](#) define their own attribute directives. The most common attribute directives are as follows:

COMMON DIRECTIVES

DETAILS

[NgClass](#)

Adds and removes a set of CSS classes.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

COMMON DIRECTIVES

DETAILS

NgStyle

Adds and removes a set of HTML styles.

NgModel

Adds two-way data binding to an HTML form element.

Built-in directives use only public APIs. They do not have special access to any private APIs that other directives can't access.

Adding and removing classes with [NgClass](#)

Add or remove multiple CSS classes simultaneously with [ngClass](#).

To add or remove a *single* class, use [class binding](#) rather than [NgClass](#).

Using [NgClass](#) with an expression

On the element you'd like to style, add [[ngClass](#)] and set it equal to an expression. In this case, `isSpecial` is a boolean set to true in `app.component.ts`. Because `isSpecial` is true, [ngClass](#) applies the class of `special` to the `<div>`.

`src/app/app.component.html`

```
content_copy<!-- toggle the "special" class on/off with a property -->
```

```
<div [ngClass]="isSpecial ? 'special' : ''>This div is special</div>
```

Using [NgClass](#) with a method

1. To use [NgClass](#) with a method, add the method to the component class. In the following example, `setCurrentClasses()` sets the property `currentClasses` with an object that adds or removes three classes based on the true or false state of three other component properties. Each key of the object is a CSS class name. If a key is true, [ngClass](#) adds the class. If a key is false, [ngClass](#) removes the class.
`src/app/app.component.ts`

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
content_copycurrentClasses: Record<string, boolean> = {};  
  
/* ... */  
  
setCurrentClasses() {  
  
    // CSS classes: added/removed per current state of component  
    properties  
  
    this.currentClasses = {  
  
        saveable: this.canSave,  
  
        modified: !this.isUnchanged,  
  
        special: this.isSpecial  
  
    };  
  
}
```

2. In the template, add the [ngClass](#) property binding to currentClasses to set the element's classes:

src/app/app.component.html

```
content_copy<div [ngClass]="currentClasses">This div is initially  
saveable, unchanged, and special.</div>
```

For this use case, Angular applies the classes on initialization and in case of changes. The full example calls `setCurrentClasses()` initially with `ngOnInit()` and when the dependent properties change through a button click. These steps are not necessary to implement [ngClass](#).

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

Setting inline styles with [NgStyle](#)

Use [NgStyle](#) to set multiple inline styles simultaneously, based on the state of the component.

1. To use [NgStyle](#), add a method to the component class.

In the following example, `setCurrentStyles()` sets the property `currentStyles` with an object that defines three styles, based on the state of three other component properties.

`src/app/app.component.ts`

```
content_copycurrentStyles: Record<string, string> = {};  
  
/* ... */  
  
setCurrentStyles() {  
  
    // CSS styles: set per current state of component properties  
  
    this.currentStyles = {  
  
        'font-style': this.canSave    ? 'italic' : 'normal',  
  
        'font-weight': !this.isUnchanged ? 'bold' : 'normal',  
  
        'font-size': this.isSpecial    ? '24px'  : '12px'  
  
    };  
  
}
```

2. To set the element's styles, add an [ngStyle](#) property binding to `currentStyles`.
`src/app/app.component.html`

```
content_copy<div [ngStyle]="currentStyles">
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

This div is initially italic, normal weight, and extra large (24px).

</div>

For this use case, Angular applies the styles upon initialization and in case of changes. To do this, the full example calls `setCurrentStyles()` initially with `ngOnInit()` and when the dependent properties change through a button click. However, these steps are not necessary to implement [ngStyle](#) on its own.

Displaying and updating properties with [ngModel](#)

Use the [NgModel](#) directive to display a data property and update that property when the user makes changes.

1. Import [FormsModule](#) and add it to the NgModule's imports list.
src/app/app.module.ts (FormsModule import)

```
content_copyimport { FormsModule } from '@angular/forms'; // <--  
- JavaScript import from Angular
```

```
/* ... */
```

```
@NgModule({
```

```
/* ... */
```

```
imports: [
```

```
  BrowserModule,
```

```
  FormsModule // <--- import into the NgModule
```

```
],
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
/* ... */  
  
})  
  
export class AppModule { }
```

2. Add an `[(ngModel)]` binding on an HTML `<form>` element and set it equal to the property, here name.
src/app/app.component.html (NgModel example)

```
content_copy<label for="example-ngModel">[(ngModel)]:</label>  
  
<input [(ngModel)]="currentItem.name" id="example-ngModel">
```

This `[(ngModel)]` syntax can only set a data-bound property.

To customize your configuration, write the expanded form, which separates the property and event binding. Use [property binding](#) to set the property and [event binding](#) to respond to changes. The following example changes the `<input>` value to uppercase:

src/app/app.component.html

```
content_copy<input[ngModel]="currentItem.name"  
  
(ngModelChange)="setUppercaseName($event)"id="exampleuppercase">
```

Here are all variations in action, including the uppercase version:

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

NgModel examples

Current item name: Teapot

without NgModel:

[(ngModel)]:

bindon-ngModel:

(ngModelChange)="...name=\$event":

(ngModelChange)="setUppercaseName(\$event)"

[NgModel](#) and value accessors

The [NgModel](#) directive works for an element supported by a [ControlValueAccessor](#). Angular provides *value accessors* for all of the basic HTML form elements. For more information, see [Forms](#).

To apply [([ngModel](#))] to a non-form built-in element or a third-party custom component, you have to write a value accessor. For more information, see the API documentation on [DefaultValueAccessor](#).

When you write an Angular component, you don't need a value accessor or [NgModel](#) if you name the value and event properties according to Angular's [two-way binding syntax](#).

Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, and manipulating the host elements to which they are attached.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

This section introduces the most common built-in structural directives:

COMMON
STRUCTURAL DIRECTIVES

BUILT-IN

DETAILS

[NgIf](#)

Conditionally creates or disposes of subviews from the template.

[NgFor](#)

Repeat a node for each item in a list.

[NgSwitch](#)

A set of directives that switch among alternative views.

For more information, see [Structural Directives](#).

Adding or removing an element with [NgIf](#)

Add or remove an element by applying an [NgIf](#) directive to a host element.

When [NgIf](#) is false, Angular removes an element and its descendants from the DOM. Angular then disposes of their components, which frees up memory and resources.

To add or remove an element, bind [*ngIf](#) to a condition expression such as `isActive` in the following example.

src/app/app.component.html

```
content_copy<app-item-detail *ngIf="isActive" [item]="item"></app-item-detail>
```

When the `isActive` expression returns a truthy value, [NgIf](#) adds the `ItemDetailComponent` to the DOM. When the expression is falsy, [NgIf](#) removes

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

the ItemDetailComponent from the DOM and disposes of the component and all of its subcomponents.

For more information on [NgIf](#) and NgIfElse, see the [NgIf API documentation](#).

Guarding against null

By default, [NgIf](#) prevents display of an element bound to a null value.

To use [NgIf](#) to guard a <div>, add `*ngIf="yourProperty"` to the <div>. In the following example, the currentCustomer name appears because there is a currentCustomer.

src/app/app.component.html

```
content_copy<div*ngIf="currentCustomer">Hello,
{{currentCustomer.name}}</div>
```

However, if the property is null, Angular does not display the <div>. In this example, Angular does not display the nullCustomer because it is null.

src/app/app.component.html

```
content_copy<div*ngIf="nullCustomer">Hello,
<span>{{nullCustomer}}</span></div>
```

Listing items with [NgFor](#)

Use the [NgFor](#) directive to present a list of items.

1. Define a block of HTML that determines how Angular renders a single item.
2. To list your items, assign the shorthand let item of items to `*ngFor`.

src/app/app.component.html

```
content_copy<div *ngFor="let item of items">{{item.name}}</div>
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

The string "let item of items" instructs Angular to do the following:

- Store each item in the items array in the local item looping variable
- Make each item available to the templated HTML for each iteration
- Translate "let item of items" into an [<ng-template>](#) around the host element
- Repeat the [<ng-template>](#) for each item in the list

For more information see the [Structural directive shorthand](#) section of [Structural directives](#).

Repeating a component view

To repeat a component element, apply [*ngFor](#) to the selector. In the following example, the selector is <app-item-detail>.

src/app/app.component.html

```
content_copy<app-item-detail *ngFor="let item of items"
[item]="item"></app-item-detail>
```

Reference a template input variable, such as item, in the following locations:

- Within the [ngFor](#) host element
- Within the host element descendants to access the item's properties

The following example references item first in an interpolation and then passes in a binding to the item property of the <app-item-detail> component.

src/app/app.component.html

```
content_copy<div *ngFor="let item of items">{{item.name}}</div>

<!-- ... -->

<app-item-detail *ngFor="let item of items" [item]="item"></app-item-
detail>
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

For more information about template input variables, see [Structural directive shorthand](#).

Getting the index of `*ngFor`

Get the index of `*ngFor` in a template input variable and use it in the template.

In the `*ngFor`, add a semicolon and let `i=index` to the shorthand. The following example gets the index in a variable named `i` and displays it with the item name.

src/app/app.component.html

```
content_copy<div *ngFor="let item of items; let i=index">{{i + 1}} -  
{{item.name}}</div>
```

The index property of the `NgFor` directive context returns the zero-based index of the item in each iteration.

Angular translates this instruction into an `<ng-template>` around the host element, then uses this template repeatedly to create a new set of elements and bindings for each item in the list. For more information about shorthand, see the [Structural Directives](#) guide.

Repeating elements when a condition is true

To repeat a block of HTML when a particular condition is true, put the `*ngIf` on a container element that wraps an `*ngFor` element.

For more information see [one structural directive per element](#).

Tracking items with `*ngFor` trackBy

Reduce the number of calls your application makes to the server by tracking changes to an item list. With the `*ngFor` trackBy property, Angular can change and re-render only those items that have changed, rather than reloading the entire list of items.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

1. Add a method to the component that returns the value [NgFor](#) should track. In this example, the value to track is the item's id. If the browser has already rendered id, Angular keeps track of it and doesn't re-query the server for the same id.

src/app/app.component.ts

```
content_copytrackByItems(index: number, item: Item): number {  
  return item.id; }
```

2. In the shorthand expression, set trackBy to the trackByItems() method.
src/app/app.component.html

```
content_copy<div *ngFor="let item of items; trackBy:  
  trackByItems">
```

```
  ({{item.id}}) {{item.name}}
```

```
</div>
```

Change ids creates new items with new item.ids. In the following illustration of the trackBy effect, Reset items creates new items with the same item.ids.

- With no trackBy, both buttons trigger complete DOM element replacement.
- With trackBy, only changing the id triggers element replacement.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)



Hosting a directive without a DOM element

The Angular [<ng-container>](#) is a grouping element that doesn't interfere with styles or layout because Angular doesn't put it in the DOM.

Use [<ng-container>](#) when there's no single element to host the directive.

Here's a conditional paragraph using [<ng-container>](#).

src/app/app.component.html (ngif-ngcontainer)

```
content_copy<p>
```

I turned the corner

```
<ng-container *ngIf="hero">
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

and saw {{hero.name}}. I waved

</ng-container>

and continued on my way.

</p>

1. Import the [ngModel](#) directive from [FormsModule](#).
2. Add [FormsModule](#) to the imports section of the relevant Angular module.
3. To conditionally exclude an <option>, wrap the <option> in an [<ng-container>](#).

src/app/app.component.html (select-ngcontainer)

content_copy<div>

Pick your favorite hero

(<label for="showSad"><input id="showSad" type="checkbox" checked (change)="showSad = !showSad">show sad</label>)

</div>

<select [(ngModel)]="hero">

<ng-container *ngFor="let h of heroes">

<ng-container *ngIf="showSad || h.emotion !== 'sad'">

<option [ngValue]="h">{{h.name}} ({{h.emotion}})</option>

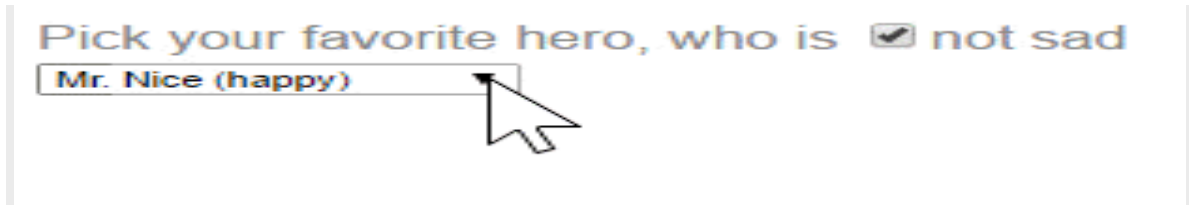
</ng-container>

</ng-container>

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

</select>



Switching cases with [NgSwitch](#)

Like the JavaScript switch statement, [NgSwitch](#) displays one element from among several possible elements, based on a switch condition. Angular puts only the selected element into the DOM.

[NgSwitch](#) is a set of three directives:

[NGSWITCH](#) DIRECTIVES

DETAILS

[NgSwitch](#)

An attribute directive that changes the behavior of its companion directives.

[NgSwitchCase](#)

Structural directive that adds its element to the DOM when its bound value equals the switch value and removes its bound value when it doesn't equal the switch value.

[NgSwitchDefault](#)

Structural directive that adds its element to the DOM when there is no selected [NgSwitchCase](#).

1. On an element, such as a <div>, add [[ngSwitch](#)] bound to an expression that returns the switch value, such as feature. Though the feature value in this example is a string, the switch value can be of any type.
2. Bind to *[ngSwitchCase](#) and *[ngSwitchDefault](#) on the elements for the cases.
src/app/app.component.html

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
content_copy<div [ngSwitch]="currentItem.feature">
```

```
  <app-stout-item *ngSwitchCase="'stout'"
  [item]="currentItem"></app-stout-item>
```

```
  <app-device-item *ngSwitchCase="'slim'"
  [item]="currentItem"></app-device-item>
```

```
  <app-lost-item *ngSwitchCase="'vintage'"
  [item]="currentItem"></app-lost-item>
```

```
  <app-best-item *ngSwitchCase="'bright'"
  [item]="currentItem"></app-best-item>
```

```
<!-- ... -->
```

```
  <app-unknown-item *ngSwitchDefault
  [item]="currentItem"></app-unknown-item>
```

```
</div>
```

3. In the parent component, define `currentItem`, to use it in the [\[ngSwitch\]](#) expression.
src/app/app.component.ts

```
content_copycurrentItem!: Item;
```

4. In each child component, add an item [input property](#) which is bound to the `currentItem` of the parent component. The following two snippets show the parent component and one of the child components. The other child components are identical to `StoutItemComponent`.
In each child component, here `StoutItemComponent`

```
content_copyexport class StoutItemComponent {
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
@Input() item!: Item;  
  
}
```



Switch directives also work with built-in HTML elements and web components. For example, you could replace the `<app-best-item>` switch case with a `<div>` as follows.

src/app/app.component.html

```
content_copy<div *ngSwitchCase="'bright'"> Are you as bright as  
{{currentItem.name}}?</div>
```

ng-for loop

The ng-for loop is a structural directive. We will see how to change the structure of the dom.

Point is to repeat given HTML ones for each value in an array[]. Context is each time passing the array value for string interpolation or binding.

The syntax is `*for =` let `<value>` of collection."

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

<value> is a variable name, <collection.> is a property on your component which holds a collection, usually an array but anything that can be iterated.

Ng-For - local variable

- Index - used to provide the index for the current element while iteration.
- First - return true if the current element is the last in the iteration otherwise it is false.
- Even - return true if current elements are even elements based on the index in the iteration otherwise false.

It is used to execute the block of code to a specific item number. A for loop is contains the initialization, condition, and increment/ decrement in a single line, which provides easy debug, and structure of looping.

- The for loop is used to control the repeated statements.
- A for loop is worked till the condition is (found)/matches successfully.
- A for-loop id is used to repeat a portion of part of the HTML template once per item from an iterable list.

Example

First, you have to create an application using the command " ng serve". Then open this project and then create a component using the command " ng g component loops". Go to the .ts file and take a variable array type. and then put the few values. Then go to the HTML file and make a list. Take a list item and then put the *ng-for loop. Then take put this variable name in the interpolation with the variable name like - {{student.name}}.

Now save all the files.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

for-loop.html

<h2><p>Understanding with Ng For loop</p></h2>

<h3><p>For loop in Angular</p></h3>

<li *ngFor='let student of students;'>

{{i+1}} - {{student.name}} - {{f}} - {{l}} - {{ev}} - {{od}}

foor-loop.ts

import {

Component,

OnInit

} from '@angular/core';

@Component({

selector: 'app-loops',

templateUrl: './loops.component.html',

styleUrls: ['./loops.component.css']

})

export class LoopsComponent {

students: any[] = [{

'name': 'Chaman Gautam'

}, {

'name': 'Ravi Gautam'

}, {

'name': 'Mohit sharma'

}, {

'name': 'Gaurav Sharma'

}, {

'name': 'Gaurav Kumar'

}, {

'name': 'Sandeep singh'

}, {

'name': 'Sumit Nimmi'

}, {

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
'name': 'Deepak'  
}, {  
'name': 'Vikas'  
}};  
}
```

module.ts

```
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
  
import { AppRoutingModule } from './app-routing.module';  
import { HttpClientModule } from '@angular/common/http';  
import { AppComponent } from './app.component';  
import { AddComponent } from './add/add.component';  
import { RouterModule, Routes } from '@angular/router';  
import { BookService } from './book.service';  
import { InMemoryWebApiModule } from 'angular-in-memory-web-api';  
import { TestData } from './testdata';  
  
import { FormsModule, ReactiveFormsModule } from '@angular/forms';  
import { ShowdataComponent } from './showdata/showdata.component';  
import { CommonModule } from '@angular/common';  
import { IfelseComponent } from './ifelse/ifelse.component';  
import { SwitchComponent } from './switch/switch.component';  
import { LoopsComponent } from './loops/loops.component';  
const routes: Routes = [  
  {path:'', component:AddComponent},  
  { path:'add', component:AddComponent}  
];  
  
@NgModule({  
  declarations: [  
    AppComponent,  
    AddComponent,  
    ShowdataComponent,  
    IfelseComponent,
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
SwitchComponent,  
LoopsComponent  
],  
imports: [  
  BrowserModule,  
  AppRoutingModule,RouterModule.forRoot(routes),ReactiveFormsModule, Forms  
Module,  
  HttpClientModule, CommonModule,  
  InMemoryWebApiModule.forRoot(TestData)  
],  
providers: [BookService],  
bootstrap: [ LoopsComponent]  
})  
export class AppModule { }
```

mains.ts

```
import { enableProdMode } from '@angular/core';  
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  
import { AppModule } from './app/app.module';  
import { environment } from './environments/environment';  
if (environment.production) {  
  enableProdMode();  
}  
platformBrowserDynamic().bootstrapModule(AppModule)  
  .catch(err => console.error(err));
```

index.html

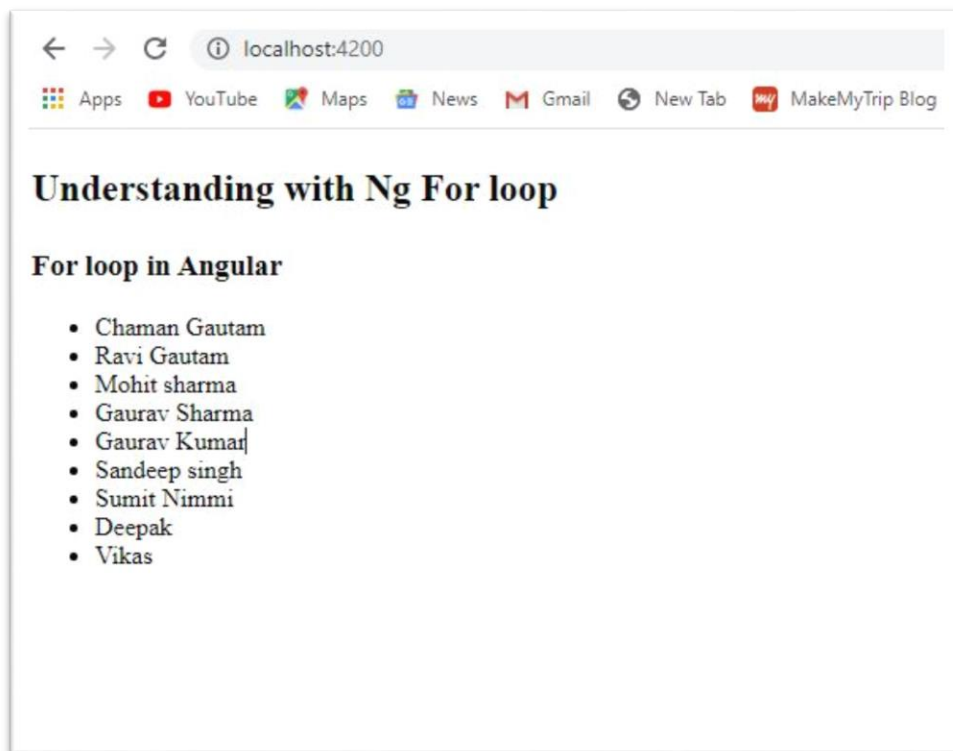
```
<!doctype html>  
<html lang="en">  
<head>  
  <meta charset="utf-8">  
  <title>Dhwani</title>  
  <base href="/">  
  <meta name="viewport" content="width=device-width, initial-scale=1">  
  <link rel="icon" type="image/x-icon" href="favicon.ico">  
</head>
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
<body>
  <app-loops></app-loops>
</body>
</html>
```

Now compile the project using the command " ng serve", after compiling this project you have to open the web browser and hit " *localhost:4200*" after few seconds you can see that your system shows the output like,



Now you can see that all data will be display.

Now use the indexing in the list.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

index.html

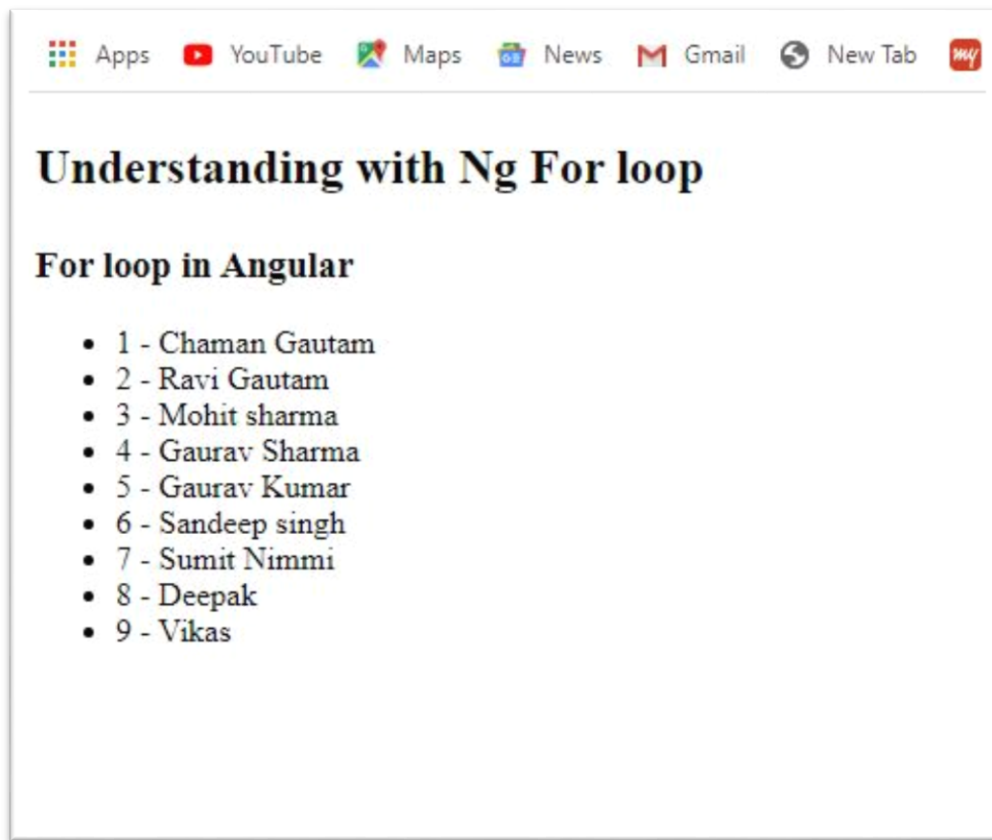
```
<h2><p>Understanding with Ng For loop</p></h2>
<h3><p>For loop in Angular</p></h3>
<ul>
  <li *ngFor='let student of students; let i=index;'>
    {{i+1}}
  </li>
</ul>
```

Now save all the data and refresh the browser, and after refresh, the browser the output will be shown like,

Output

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)



Now you can see that there is a hit numbering that will be shown the number with each item.

Now use the true false and even odd for the last and first in the index like,

```
<h2><p>Understanding with Ng For loop</p></h2>
```

```
<h3><p>For loop in Angular</p></h3>
```

```
<ul>
```

```
  <li *ngFor='let student of students; let i=index; let f=first;
  let l=last; let ev=even;let od=odd'>
```

```
    {{i+1}} - {{student.name}} - {{f}} - {{l}}
```

```
</ul>
```

Now save all the data and refresh the browser, and after the refresh, the browser shows the output like below,

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

OUTPUT



Condition with ngIf,

❖ Angular ngIf: Complete Guide

Learn all the features available in ngIf, learn the best way to use it to consume Observables, avoid a common anti-pattern.

- In this post, we are going to cover *all* the features that we have available for using the Angular `ngIf` core directive.

Besides the most commonly used features, we are going to learn how to avoid a **potential** `ngIf` **anti-pattern** that we might run into while developing more complex UI screens that consume a lot of Observable data coming from different sources (backend, Observable services, stores, etc.).

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

❖ Table Of Contents

In this post, we will cover the following topics:

- What is Angular ngIf?
- How does ngIf compare to hiding elements using CSS?
- What type of expressions can ngIf accept?
- The ngIf else syntax
- The ngIf if then else syntax
- Consuming observable data with ngIf and the async pipe
- A potential anti-pattern while consuming Observable data with ngIf
- The Single Data Observable pattern
- How does ngIf work under the hood?

❖ Summary

This post is part of our ongoing series on Angular Core features, you can find all the articles available [here](#).

So without further ado, let's get started learning everything that we need to know about Angular ngIf!

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

Passing Data into a Component

There are two ways to pass data into a component, with 'property binding' and 'event binding'. In Angular, data and event change detection happens top-down from parent to children. However for Angular events we can use the DOM event mental model where events flow bottom-up from child to parent. So, Angular events can be treated like regular HTML DOM based events when it comes to cancellable event propagation.

The `@Input()` decorator defines a set of parameters that can be passed down from the component's parent. For example, we can modify the `HelloComponent` component so that `name` can be provided by the parent.

```
import { Component, Input } from "@angular/core";
```

```
@Component({  
  selector: "rio-hello",  
  template: "<p>Hello, {{name}}!</p>",  
})  
export class HelloComponent {  
  @Input() name: string;  
}
```

The point of making components is not only encapsulation, but also reusability. Inputs allow us to configure a particular instance of a component.

We can now use our component like so:

```
<!-- To bind to a raw string -->  
<rio-hello name="World"></rio-hello>  
<!-- To bind to a variable in the parent scope -->  
<rio-hello [name]="helloName"></rio-hello>
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)



Two-way Data Binding in Angular

Here you will learn how to do two-way data binding in Angular.

Two-way data binding refers to sharing data between a component class and its template. If you change data in one place, it will automatically reflect at the other end. For example, if you change the value of the input box, then it will also update the value of the attached property in a component class.

Two-way data binding performs the following actions:

1. Sets a property of a component class
2. Listens for a DOM element change event

Angular v2+ supports two-way data binding using ngModel directive and also by having getter and setter methods.

❖ ngModel Directive

The `ngModel` directive with `[()]` syntax (also known as banana box syntax) syncs values from the UI to a property and vice-versa. So, whenever the user changes the value on UI, the corresponding property value will get automatically updated.

`[()]` = `[() + ()]` where `[()]` binds attribute, and `()` binds an event.

Example: Banana Box `[()]`

Copy

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

```
@Component({
  selector: 'app-greet',
  template: `
    User Name: <input type="text" [(ngModel)]="userName" ><br/>
    {{userName}}
  `,
})
export class GreetComponent implements OnInit {
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
constructor() { }  
  
userName: string = "jbond";  
  
ngOnInit(): void {  
}  
  
}
```

The `[(ngModel)]` syntax is the recommended way of two-way data binding.

The `ngModel` directive with `[]` syntax is used for one-way data binding. `[ngModel]` binds a value to a property to UI control.

❖ Getter and Setter Methods

For two-way data binding, declare a private property and access its value using get and set methods in the component class. Then, assign that property to `[(ngModel)]`.

For example, declare a private property with a get and set method, as shown below.

Example: Private Property

Copy

```
private _userName: string = "bill gates";
```

```
get userName(): string {  
    return this._userName;  
}
```

```
set userName(val: string) {  
    //do some extra work here  
    this._userName = val;  
}
```

Now, assign `userName` to `[(ngModel)]` in the template.

Example: `[(ngModel)]`

Copy

```
<input type="text" [(ngModel)]="userName" >
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

The following is a full example of two-way data binding using get and set methods.

Example: Two-way Data Binding

Copy

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

@Component({
  selector: 'app-greet',
  template: `
    User Name: <input type="text" [(ngModel)]="userName" ><br/>
    {{userName}}`
})
export class GreetComponent implements OnInit {

  constructor() { }

  private _userName: string = "bill gates";

  get userName(): string {
    return this._userName;
  }
  set userName(val: string) {
    //do some extra work here
    this._userName = val;
  }
  ngOnInit(): void {
  }
}
```

❖ What is the use of ngOnInit?

ngOnInit: ngOnInit is a lifecycle hook in Angular that is called after the constructor is called and after the component's inputs have been initialized. It is used to perform any additional initialization that is required for the component. ngOnInit is commonly used to call services or to set up subscriptions

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

❖ ngOnInit Example | Angular

home.component.ts

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

```
@Component({  
  selector: 'app-home',  
  templateUrl: './home.component.html',  
  styleUrls: ['./home.component.css']  
})
```

```
export class HomeComponent implements OnInit {  
  constructor() {}  
  ngOnInit() {  
    console.log("component has been initialized!")  
  }  
}
```

ngOnInit() executes once when a component is initialized.

ngOnInit() executes after data-bound properties are displayed and input properties are set.

ngOnInit() executes once after the first `ngOnChanges`.

When you create a new Angular component via the Angular CLI, the `ngOnInit()` method is included by default.

Notice how `implements OnInit` is added to the class definition. While not required, this is considered best practice as it provides the type checking benefits of TypeScript.

ngOnInit() will still execute regardless of whether or not `implements OnInit` is included in the class definition.

Component styles

Angular applications are styled with standard CSS. That means you can apply everything you know about CSS stylesheets, selectors, rules, and media queries directly to Angular applications.

Additionally, Angular can bundle component styles with components, enabling a more modular design than regular stylesheets.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

This page describes how to load and apply these component styles.
Using component styles

For every Angular component you write, you can define not only an HTML template, but also the CSS styles that go with that template, specifying any selectors, rules, and media queries that you need.

One way to do this is to set the `styles` property in the component metadata. The `styles` property takes an array of strings that contain CSS code. Usually you give it one string, as in the following example:

```
src/app/hero-app.component.ts
content_copy@Component({
  selector: 'app-root',
  template: `
    <h1>Tour of Heroes</h1>
    <app-hero-main [hero]="hero"></app-hero-main>
  `,
  styles: ['h1 { font-weight: normal; }']
})
export class HeroAppComponent {
  /* ... */
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

}

❖ **Component styling best practices**

See View Encapsulation for information on how Angular scopes styles to specific components.

You should consider the styles of a component to be private implementation details for that component. When consuming a common component, you should not override the component's styles any more than you should access the private members of a TypeScript class. While Angular's default style encapsulation prevents component styles from affecting other components, global styles affect all components on the page. This includes `::ng-deep`, which promotes a component style to a global style.

Authoring a component to support customization

As component author, you can explicitly design a component to accept customization in one of four different ways.

1. Use CSS Custom Properties (recommended)

You can define a supported customization API for your component by defining its styles with CSS Custom Properties, alternatively known as CSS Variables. Anyone using your component can consume this API by defining values for these properties, customizing the final appearance of the component on the rendered page.

While this requires defining a custom property for each customization point, it creates a clear API contract that works in all style encapsulation modes.

2. Declare global CSS with @mixin

While Angular's emulated style encapsulation prevents styles from escaping a component, it does not prevent global CSS from affecting the entire page. While component consumers should avoid directly overwriting the CSS internals of a component, you can offer a supported customization API via a CSS preprocessor like Sass.

For example, a component may offer one or more supported mixins to customize various aspects of the component's appearance. While this approach uses global

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

styles in its implementation, it allows the component author to keep the mixins up to date with changes to the component's private DOM structure and CSS classes.

3. Customize with CSS ::part

If your component uses [Shadow DOM](#), you can apply the part attribute to specify elements in your component's template. This allows consumers of the component to author arbitrary styles targeting those specific elements with [the::partpseudo-element](#).

While this lets you limit the elements within your template that consumers can customize, it does not limit which CSS properties are customizable.

4. Provide a TypeScript API

You can define a TypeScript API for customizing styles, using template bindings to update CSS classes and styles. This is not recommended because the additional JavaScript cost of this style API incurs far more performance cost than CSS.

Creating multiple modules

'How to Use Multiple Modules on the Same Page.'

While you are working with AngularJS, you might find a situation where you are having some scopes which are available in some controller, but those controllers belong to different modules; in such a case you might have introduced new scope in controller so that you can use it. This is because **you can't declare multiple modules in "ng-app"**, this is a limitation of ng-app, it restricts you to only one module per page. To get rid of this problem you can **choose "ng-module" which is available in "angular.ng-modules"**. angular.ng-modules is a new directive which needs to be provided where you had provided the AngularJS directive, you can download it from my source code.

creating a sample application to show how you can use this feature.

Step 1: Firstly, download the "angular.ng-modules" using the above links.

After this provide where you had provided the "angularjs" directive.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
<head>
  <title></title>
  <script src="angular.js"></script>
  <script src="angular.ng-modules.js"></script>
  <script src="AngularController.js"></script>
</head>
```

Step 2: Now I am creating a new JavaScript file where we will declare our modules, controller, and scopes.

```
var firstModule = angular.module('firstModule', []);

firstModule.controller('firstController', function ($scope) {
  $scope.UserName = 'Anubhav Chaudhary';
});

firstModule.controller('secondController', function ($scope) {
  $scope.MobileNumber = '00000000';
});

var secondModule = angular.module('secondModule', []);

secondModule.controller('thirdController', function ($scope) {
  $scope.EmailId = 'anu@test.com';
});
```

Here you can see that I declared two modules named "**firstModule**" and "**secondModule**," inside the firstModule I created two controllers and in those controllers I created some scope properties, inside secondModule only one controller is created and in the controller one scope property is created. To all these scopes some default value is also provided.

Step 3: Now it's time to work on the main section i.e. HTML section.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.
(AFFILIATED TO SAURASHTRA UNIVERSITY)

```
<head>
  <title></title>
  <script src="angular.js"></script>
  <script src="angular.ng-modules.js"></script>
  <script src="AngularController.js"></script>
</head>
<body>
  <form ng-modules="firstModule,secondModule">
    <fieldset>
      <legend>We are having both the modules</legend>
      <div ng-controller="firstController">
        Provide Your Name :
        <input type="text" ng-model="UserName" />
      </div>
      <div ng-controller="secondController">
        Provide Your Mobile Number :
        <input type="text" ng-model="MobileNumber" />
      </div>
      <div ng-controller="thirdController">
        Provide Your Email ID :
        <input type="text" ng-model="EmailId" />
      </div>
    </fieldset>
  </form>
  <form ng-module="secondModule">
    <fieldset>
      <legend>I am having only second module</legend>
      <div ng-controller="thirdController">
        Provide Your Email ID :
        <input type="text" ng-model="EmailId" />
      </div>
    </fieldset>
  </form>
</body>
</html>
```

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

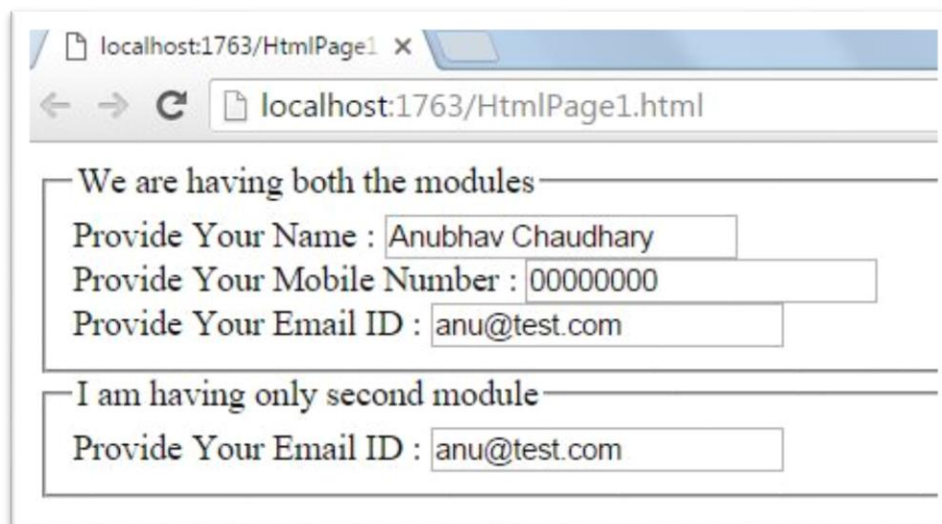
Here you can see that I created two forms, let's firstly talk about the first form.

In the first form you can see that I had provided both the modules using "**ng-modules**" directive. When you are using "ng-modules" then it allows you to use multiple modules at same time, but if you don't want to use multiple modules then you can use "**ng-module**" directive.

Inside the first form all three controllers are used in three different div's and their scopes are bind to some input fields.

In the second form you can see that I have used "ng-module" because I need to use only one module here. Inside the form div is bound to a corresponding controller and an input element is bound to scope property.

Now our application is created and it's time to see the output.



The screenshot shows a web browser window with the address bar displaying 'localhost:1763/HtmlPage1.html'. The page content is divided into two sections. The first section, titled 'We are having both the modules', contains three input fields: 'Provide Your Name : Anubhav Chaudhary', 'Provide Your Mobile Number : 00000000', and 'Provide Your Email ID : anu@test.com'. The second section, titled 'I am having only second module', contains one input field: 'Provide Your Email ID : anu@test.com'.

Output: On running the application you will see an output like this,

All the input fields have their scope value and this means that our code has run successfully and both the modules got bound at same time.