**What is an Angular Component**

The Component is the main building block of an Angular Application.

The Component contains the data & user interaction logic that defines how the View looks and behaves. A view in Angular refers to a template (HTML).

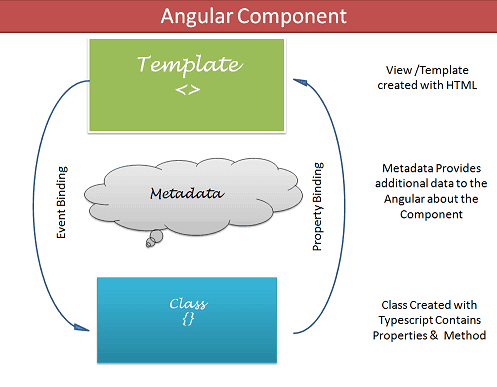
The Angular Components are plain [JavaScript](https://www.tektutorialshub.com/javascript-tutorial/) classes and defined using **@Component Decorator**. This Decorator provides the component with the View to display & Metadata about the Component

The Component is responsible to provide the data to the view. The Angular does this by using [data binding](https://www.tektutorialshub.com/angular/angular-data-binding/) to get the data from the Component to the View. This is done using the special HTML markup knows as the Angular Template Syntax. The Component can also get notified when the View Changes.

The Angular applications will have lots of components. Each component handles a small part of UI. These components work together to produce the complete user interface of the application

The Components consists of three main building blocks

* Template
* Class
* MetaData



**Building blocks of the Angular Components**

**Template (View)**

The template defines the layout and content of the View. Without the template,  there is nothing for Angular to render to the DOM.

The Templates are nothing but HTML codes along with the Angular specific special HTML markups ( knows as the Angular Template Syntax).

You can add [Angular directives](https://www.tektutorialshub.com/angular/angular-directives/) , [Angular Pipes](https://www.tektutorialshub.com/angular/angular-pipes/) & Other Angular Components on the template.

The data to Template comes from the Component, which in turn gets it from a [Angular Service](https://www.tektutorialshub.com/angular/angular-services/). Using the [data binding](https://www.tektutorialshub.com/angular/angular-data-binding/) techniques, we can keep the Template in sync with the Component. The templates can use the [Event Binding](https://www.tektutorialshub.com/angular/event-binding-in-angular/) or [two way data binding](https://www.tektutorialshub.com/angular/ngmodel-two-way-data-binding-in-angular/) to notify the component, when user changes something on the View.

There are two ways you can specify the Template in Angular.

1. Defining the Template Inline
2. Provide an external Template

**Class**

The Class provides the data & logic to the View. It contains the JavaScript code associated with Template (View). We use [TypeScript](https://www.tektutorialshub.com/typescript-tutorial/) to create the class, but you can also use JavaScript directly in the class.

Class Contains the Properties & Methods. The Properties of a class can be bind to the view using [Data Binding](https://www.tektutorialshub.com/angular/angular-data-binding/).

The simple Angular Class

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | export **class** AppComponent  {      title : **string** ="app"  } |

By convention we prefix the Component class with Component so as to easily identify them.

**Metadata**

Metadata Provides additional information about the component to the Angular. Angular uses this information to process the class. We use the @Component decorator to provide the Metadata to the Component.

**@Component decorator**

A decorator is a function that adds metadata to class, its methods & to its properties. The Components are defined with a

#### Important Component metadata properties

##### Selector

Selector specifies the simple CSS selector. The Angular looks for the CSS selector in the template and renders the component there.

##### Providers

The Providers are the [Angular Services](https://www.tektutorialshub.com/angular/angular-services/), that our component going to use. The Services provide service to the Components or to the other Services.

##### Directives

The[directives](https://www.tektutorialshub.com/angular/angular-directives/) that this component going to use are listed here.

##### Styles/styleUrls

The CSS styles or style sheets, that this component needs. Here we can use either external stylesheet (using styleUrls) or inline styles (using Styles). The styles used here are specific to the component

##### template/templateUrl

The HTML template that defines our View. It tells Angular how to render the Component’s view. The templates can be inline (using a template) or we can use an external template (using a templateUrl). The Component can have only one template. You can either use inline template or external template and not both

# **Data Binding in Angular**

[Angular Components](https://www.tektutorialshub.com/angular/angular-component/) are useless if they do not show any dynamic data. They also need to respond to user interactions and react to events. The data binding kees both component & view in sync with each other. We use techniques like [Interpolation](https://www.tektutorialshub.com/angular/interpolation-in-angular/), [Property Binding](https://www.tektutorialshub.com/angular/property-binding-in-angular/), [Event Binding](https://www.tektutorialshub.com/angular/event-binding-in-angular/) & [Two Way Binding](https://www.tektutorialshub.com/angular/ngmodel-two-way-data-binding-in-angular/) to bind data. We also learn how to use the [ngModel](https://www.tektutorialshub.com/angular/ngmodel-two-way-data-binding-in-angular/) directive to achieve the two-way binding in [Angular Forms](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/).

## What is Angular Data Binding

Whenever the user updates the data in the view, Angular updates the component. When the component gets new data, the Angular updates the view.

There are many uses of data binding. You can show models to the user, dynamically Change element style, respond to user events, etc

## One way binding

In one way binding data flows from one direction. Either from view to component or from component to view.

### From Component to View

To bind data from component to view, we make use of Interpolation & Property Binding.

#### Property binding

The [Property binding](https://www.tektutorialshub.com/angular/property-binding-in-angular/) allows us to bind HTML element property to a property in the component. Whenever the value of the component changes, the Angular updates the element property in the View. You can set the properties such as class, href, src, textContent, etc using property binding. You can also use it to set the properties of custom components or directives (properties decorated with @Input).

[binding-target]=”binding-source”

The binding-target (or target property) is enclosed in a square bracket []. It should match the name of the property of the enclosing element.

Binding-source is enclosed in quotes and we assign it to the binding-target. The Binding source must be a template expression. It can be property in the component, method in component, a template reference variable or an expression containing all of them.

##### [Class Binding](https://www.tektutorialshub.com/angular/class-binding-in-angular/)

##### [Style Binding](https://www.tektutorialshub.com/angular/angular-style-binding/)

## Syntax

The syntax of the style binding is similar to the [property binding](https://www.tektutorialshub.com/angular/angular-data-binding/#Property-Binding).

|  |  |
| --- | --- |
| 1  2  3 | [style.style-property] = "style-value"﻿ |

The Style Binding uses the [] brackets. Place the CSS Style property (binding target) inside the square bracket. The CSS Style property must begin with ‘Style’ followed by a dot (.) and then style name.

For Example, to set the color of p element.

|  |  |
| --- | --- |
| 1  2  3 | <p [style.color]="'red'">Give me red</p> |

## Style binding Example

Setting the background color of a paragraph

|  |  |
| --- | --- |
| 1  2  3 | <p [style.background-color]="'grey'">some paragraph with grey background</p> |

Setting the border style of a button.

|  |  |
| --- | --- |
| 1  2  3 | <button [style.border]="'5px solid yellow'">Save</button> |

[**BEST ANGULAR BOOKS**](https://www.tektutorialshub.com/angular/angular-best-books/)  
**The Top 8**[**Best Angular Books**](https://www.tektutorialshub.com/angular/angular-best-books/)**, which helps you to get started with Angular**

### Conditionally setting the styles

Define a variable status in the component

|  |  |
| --- | --- |
| 1  2  3 | status:**string**='error'; |

And use that variable in the template to set the color of button either to **red** and **blue** depending on the value of the status variable.

|  |  |
| --- | --- |
| 1  2  3 | <button [style.color]="status=='error' ? 'red': 'blue'">Button 1</button> |

Another way is to create the getColor() method and use it in the template as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5 | getColor() {  **return** 'yellow';  } |
| 1  2  3 | <button [style.color]="getColor()">Button 2</button> |

### Setting the units

The styles like font-size, width etc have unit extension. The following example conditionally sets the font-size in “px” unit

|  |  |
| --- | --- |
| 1  2  3 | <button [style.font-size.px]="'20'" >Big Button</button> |

The style property name can be written in either dash-case (font-size), as shown in above example, or camelCase (fontSize) as shown below.

|  |  |
| --- | --- |
| 1  2  3 | <button [style.fontSize.px]="'20'" >Big Button</button> |

### Setting Multiple styles

To change the multiple styles, we need to add each one separately as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <p [style.color]="getColor()"     [style.font-size.px]="'20'"     [style.background-color]="status=='error' ? 'red': 'blue'">     paragraph with multiple styles  </p> |

### From View to Component

#### Event Binding

Event binding allows us to bind events such as keystrokes, clicks, hover, touch, etc to a method in component. It is one way from view to component. By tracking the user events in the view and responding to it, we can keep our component in sync with the view.

## wo Way binding

Two-way binding means that changes made to our model in the component are propagated to the view and that any changes made in the view are immediately updated in the underlying component

Two-way binding is useful in data entry forms. Whenever a user makes changes to a form field, we would like to update our model. Similarly, when we update the model with new data, we would like to update the view as well

The two-way binding uses the special syntax known as a banana in a box [()]

<someElement [(someProperty)]="value"></someElement>.

The above syntax sets up both property binding & event binding. But to make use of it, the property must have the change event with the name <propertyName>Change

But, angular has a special directive ngModel, which sets up the two-way binding

### ngModel

The Angular uses the ngModel directive to achieve the two-way binding on HTML Form elements. It binds to a form element like input, select, selectarea. etc.

The ngModel directive is not part of the Angular Core library. It is part of the @angular/forms. You need to import the FormsModule package into your Angular module.

|  |  |
| --- | --- |
| 1  2  3 | import { FormsModule } from '@angular/forms'; |

Then you can use it using the two-way binding syntax as shown below

|  |  |
| --- | --- |
| 1  2  3 | <input type="text" name="value" [(ngModel)]="value"> |

When you bind to a ngModel directive, behind the scene it sets up property binding & event binding. It binds to the value property of the element using property binding. It then uses the ngModelChange event to sets up the event binding to listen to the changes to the value.

## Examples of interpolation

You can use interpolation to invoke a method in the component, Concatenate two string, perform some mathematical operations or change the property of the DOM element like color, etc.

### Invoke a method in the component

We can invoke the component’s methods using interpolation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | *//Template*    {{getTitle()}}      *//Component*  title = 'Angular Interpolation Example';  getTitle(): **string** {  **return** **this**.title;  } |

### Concatenate two string

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <p>Welcome to {{title}}</p>  <p>{{ 'Hello & Welcome to '+ ' Angular Interpolation '}}</p>  <p>Welcome {{firstName}}, {{lastName}}</p>  <p>Welcome {{getFirstName()}}, {{getLastName()}}</p> |

### Perform some mathematical operations

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <h2>Mathematical Operations</h2>    <p>100x80 = {{100\*80}}</p>  <p>Largest: {{max(100, 200)}}</p>    *//Component*  max(first: number, second: number): number {  **return** Math.max(first, second);  } |

### Bind to an element property

We can use it to bind to a property of the HTML element, a component, or a directive. in the following code, we bind to the style.color property of the <p> element. We can bind to any property that accepts a string.

|  |  |
| --- | --- |
| 1  2  3  4 | <p>Show me <span **class** = "{{giveMeRed}}">red</span></p>  <p style.color={{giveMeRed}}>**This** **is** red</p> |

### Use a template reference variable

You can also use the [template reference variable](https://www.tektutorialshub.com/angular/template-reference-variable-in-angular/). The following example creates a template variable #name to an input box. You can use it get the value of the input field {{name.value}}

|  |  |
| --- | --- |
| 1  2  3  4  5 | <label>Enter Your Name</label>  <input (keyup)="0" #name>  <p>Welcome {{name.value}} </p> |

We also use (keyup)="0" on the input element. It does nothing but it forces the angular run the change detection, which in turn updates the view.

The Angular updates the view, when it runs the change detection. The change detection runs only in response to asynchronous events, such as the arrival of HTTP responses, raising of events, etc. In the example above whenever you type on the input box, it raises the keyup event. It forces the angular run the change detection, hence the view gets the latest values.

## Property Binding Example

**Binding to innerHTML with HTML tags**

Here the Angular parses the b &p tags and renders it in the view.

|  |  |
| --- | --- |
| 1  2  3  4  5 | *//Template*  <p [innerHTML]="text1"></p>  <div [innerHTML]="text2"></div> |
| 1  2  3  4  5 | *//Component*  text1="The <b>Angular</b> is printed in bold"  text2="<p>This is first para</p><p>This is second para</p> " |

**img**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | *//Template*    <img [src]="itemImageUrl">  <img bind-src="itemImageUrl"> |
| 1  2  3  4 | *//Component*  itemImageUrl="https://angular.io/assets/images/logos/angular/logo-nav@2x.png" |

**Concatenate two string**

|  |  |
| --- | --- |
| 1  2  3 | <p [innerText]="'Hello & Welcome to '+ ' Angular Data binding '"></p> |

**Mathematical expressions**

|  |  |
| --- | --- |
| 1  2  3 | <p [innerText]="100\*80"></p> |

**setting the color**

|  |  |
| --- | --- |
| 1  2  3  4 | *//template*  <p [style.color]="color">**This** **is** red</p> |
| 1  2  3  4 | *//Component*  color='red' |

## Event Binding

Event binding allows us to bind events such as keystroke, clicks, hover, touche, etc to a method in component. It is one way from view to

component. By tracking the user events in the view and responding to it, we can keep our component in sync with the view. For Example, when the user changes to an input in a text box, we can update the model in the component, run some validations, etc. When the user submits the button, we can then save the model to the backend server.

## Syntax

The Angular event binding consists of two parts

|  |  |
| --- | --- |
| 1  2  3 | (target-**event**)="TemplateStatement |

Copy the following code to app.component.html

***app.component.html***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <h1 [innerText]="title"></h1>    <h2>Example 1</h2>  <button (click)="clickMe()">Click Me</button>  <p>You have clicked {{clickCount}}</p> |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-1)

Add the following code to the app.component.ts

***app.component.ts***

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | clickCount=0    clickMe() {  **this**.clickCount++;    } |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-1)

In the above example, the component listens to the click event on the button. It then executes the clickMe() method and increases the clickCount by one.

## on-

Instead of parentheses, you can also use the on- syntax as shown below.

|  |  |
| --- | --- |
| 1  2  3 | <button on-click="clickMe()">Click Me</button> |

## $event Payload

DOM Events carries the event payload. I.e the information about the event. We can access the event payload by using $event as an argument to the handler function.

|  |  |
| --- | --- |
| 1  2  3  4 | <input (input)="handleInput($event)">  <p>You have entered {{value}}</p> |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-3?file=src/app/app.component.ts)

And in the component

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | value=""  handleInput(**event**) {  **this**.value = (**event**.target **as** HTMLInputElement).value;  } |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-3?file=src/app/app.component.ts)

The properties of a $event object vary depending on the type of DOM event. For example, a mouse event includes different information than an input box editing event.

Remember you need to use the variable as $event in the Template statement. Example handleInput($event). Otherwise, it will result in an error

## Template reference variable

We can also make use of the template reference variable to pass the value instead of $event.

In the template

|  |  |
| --- | --- |
| 1  2  3  4  5 | <h2>Template Reference Variable</h2>  <input #el (input)="handleInput1(el)">  <p>You have entered {{val}}</p> |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-4?file=src/app/app.component.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | val="";  handleInput1(element) {  **this**.val=element.value;  } |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-4?file=src/app/app.component.html)

## Key event filtering (with key.enter)

We use keyup/keydown events to listen for keystrokes. The following example does that

|  |  |
| --- | --- |
| 1  2  3  4 | <input (keyup)="value1= $any($event.target).value" />  <p>You entered {{value1}}</p> |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-4-zg4ifv?file=src/app/app.component.html)

But Angular also offers a feature, where it helps to filter out certain keys. For Example, if you want to listen only to the enter keys you can do it easily

|  |  |
| --- | --- |
| 1  2  3  4 | <input (keyup.enter)="value2=$any($event.target).value">  <p>You entered {{value2}}</p> |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-4-zg4ifv?file=src/app/app.component.html)

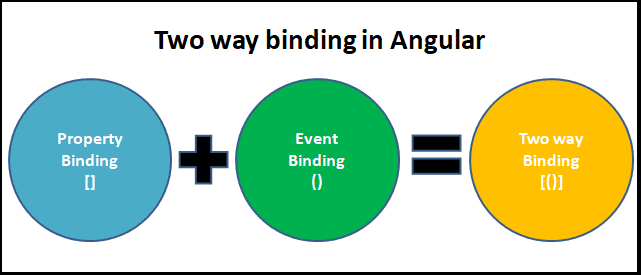
Here is an interesting example. On pressing enter key it updates the value3 variable and on escape clears the variable.

## What is Two way data binding

Two way data binding means that changes made to our model in the component are propagated to the view and that any changes made in the view are immediately updated in the underlying component data.

Two way data binding is useful in data entry forms. Whenever a user makes changes to a form field, we would like to update our model. Similarly, when we update the model with new data, we would like to update the view as well

The two way data binding is nothing but both property binding & event binding applied together. Property Binding is one way from component to view. The event binding is one way from view to component. If we combine both we will get the Two-way binding.



## Two way using property & Event Binding

The following example shows how we can achieve two-way binding using the combination of property binding & event binding

Create a new Angular application

copy the following code to app.component.html

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | h2>Example 1</h2>  <input type="text" [value]="name" (input)="name=$event.target.value">  <p> You entered {{name}}</p>  <button (click)="clearName()">Clear</button> |

Update the app.component.ts with the following code.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | name=""  clearName() {  **this**.name="";  } |

We bind the name property to the input element ([value]="name"). We also use the event binding (input)="name=$event.target.value". It updates the name property whenever the input changes. The Angular interpolation updates the {{name}}, so we know the value of name property.

$event.target.value raises the error Property ‘value’ does not exist on type ‘EventTarget’ if fullTemplateTypeCheck is set to true under angularCompilerOptions in the tsconfig.json.  
  
The error is due to the fact that the value property is not guaranteed to exist in the $event.target.  
  
To solve this problem either you can use the $any typecast function ($any($event.target).value) to stop the type checking in the template or set fullTemplateTypeCheck to false in tsconfig.json

## Component Directive

## Structural Directives

Structural directives can change the DOM layout by adding and removing DOM elements. All structural Directives are preceded by Asterix symbol

### Commonly used structural directives

#### ngFor

The [ngFor](https://www.tektutorialshub.com/angular/angular-ngfor-directive/) is an Angular structural directive, which repeats a portion of the HTML template once per each item from an iterable list (Collection). The [ngFor](https://www.tektutorialshub.com/angular/angular-ngfor-directive/) is similar to [ngRepeat](https://docs.angularjs.org/api/ng/directive/ngRepeat) in AngularJS

##### Example of ngFor

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <tr \*ngFor="let customer of customers;">      <td>{{customer.customerNo}}</td>      <td>{{customer.name}}</td>      <td>{{customer.address}}</td>      <td>{{customer.city}}</td>      <td>{{customer.state}}</td>  </tr>  ﻿ |

You can read more about the [Angular ngFor Directive](https://www.tektutorialshub.com/angular/angular-ngfor-directive/) tutorial.

#### ngSwitch

The [ngSwitch](https://www.tektutorialshub.com/angular/angular-ngswitch-directive/) directive lets you add/remove HTML elements depending on a match expression. [ngSwitch](https://www.tektutorialshub.com/angular/angular-ngswitch-directive/) directive used along with [ngSwitchCase](https://www.tektutorialshub.com/angular/angular-ngswitch-directive/" \l "ngswitchcase) and [ngSwitchDefault](https://www.tektutorialshub.com/angular/angular-ngswitch-directive/" \l "ngswitchdefault)

##### The example of ngSwitch

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <div [ngSwitch]="Switch\_Expression">      <div \*ngSwitchCase="MatchExpression1”> First Template</div>      <div \*ngSwitchCase="MatchExpression2">Second template</div>      <div \*ngSwitchCase="MatchExpression3">Third Template</div>      <div \*ngSwitchCase="MatchExpression4">Third Template</div>      <div \*ngSwitchDefault?>**Default** Template</div>  </div> |

You can read more about the [Angular ngSwitch Directive](https://www.tektutorialshub.com/angular/angular-ngswitch-directive/) tutorial.

#### ngIf

The [ngIf](https://www.tektutorialshub.com/angular/angular-ngif-directive/) Directives is used to add or remove HTML elements based on an expression. The expression must return a boolean value. If the expression is false then the element is removed, else the element is inserted

##### Example of ngIf

|  |  |
| --- | --- |
| 1  2  3  4  5 | <div \*ngIf="condition">  **This** **is** shown **if** condition **is** **true**  </div> |

Angular ngFor directive iterates over a collection of data like an array, list, etc, and creates an HTML element for each of the items from an HTML template. It helps us to build lists or tables to display tabular data in a nice way. In this tutorial, we will look at the syntax and how to use ngFor to display a list of movies using example code. The ngFor also exports several local variables like Index, First, Last, odd, even & trackby.etc. In this article, we will learn the following

Open the app.component.ts and add the following code. The code contains a list of Top 10 movies. Let us build a template to display the movies using ngFor.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | import { Component } from '@angular/core';    @Component({    selector: 'app-root',    templateUrl: './app.component.html',  })  export **class** AppComponent {    title: **string** ="Top 5 Movies" ;        movies: Movie[] =[      {title:'Zootopia',director:'Byron Howard, Rich Moore',cast:'Idris Elba, Ginnifer Goodwin, Jason Bateman',releaseDate:'March 4, 2016'},    {title:'Batman v Superman: Dawn of Justice',director:'Zack Snyder',cast:'Ben Affleck, Henry Cavill, Amy Adams',releaseDate:'March 25, 2016'},    {title:'Captain American: Civil War',director:'Anthony Russo, Joe Russo',cast:'Scarlett Johansson, Elizabeth Olsen, Chris Evans',releaseDate:'May 6, 2016'},    {title:'X-Men: Apocalypse',director:'Bryan Singer',cast:'Jennifer Lawrence, Olivia Munn, Oscar Isaac',releaseDate:'May 27, 2016'},    {title:'Warcraft',director:'Duncan Jones',cast:'Travis Fimmel, Robert Kazinsky, Ben Foster',releaseDate:'June 10, 2016'},  ]      }    **class** Movie {    title : **string**;    director : **string**;    cast : **string**;    releaseDate : **string**;  } |

[***Source code***](https://stackblitz.com/edit/angular-ngfor-directive-example1?)

### Using ngFor

To use ngFor,

1. Create a block of HTML elements, which can display a single movie.
2. Use the ngFor to repeat the block for each movie in the movies.

Open the app.component.html and add the following code.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <h1> {{title}} </h1>      <ul>      <li \*ngFor="let movie of movies">        {{ movie.title }} - {{movie.director}}      </li>    </ul> |

[***Source code***](https://stackblitz.com/edit/angular-ngfor-directive-example1?)

We use the ul to display the movies. The li element displays a single movie. We need to repeat the li for each movie. Hence we apply the ngFor on the li element.

let movie of movies will iterate over the movies collection, which is a property on the [component class](https://www.tektutorialshub.com/angular/angular-component/). movie is the [*Template input variable*](https://www.tektutorialshub.com/angular/template-reference-variable-in-angular/#template-input-variable), which represents the currently iterated movie from the collection. We use [Angular Interpolation](https://www.tektutorialshub.com/angular/interpolation-in-angular/) to display the movie title & name of the director

Here is the output

The Angular generates the following code. You can see li element for every movie.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <ul \_ngcontent-gop-c0="">    <li \_ngcontent-gop-c0=""> Zootopia - Byron Howard, Rich Moore </li>    <li \_ngcontent-gop-c0=""> Batman v Superman: Dawn of Justice - Zack Snyder </li>    <li \_ngcontent-gop-c0=""> Captain American: Civil War - Anthony Russo, Joe Russo </li>    <li \_ngcontent-gop-c0=""> X-Men: Apocalypse - Bryan Singer </li>    <li \_ngcontent-gop-c0=""> Warcraft - Duncan Jones </li>  </ul> |

## Local Variables

ngFor exposes several values, which help us to fine-tune display. We assign these values to the local variable and use it in our template

The list of exported values provided by ngFor directive

* index: number: The zero-based index of the current element in the collection.
* count: number: The total no of items in the collection
* first: boolean: True when the item is the first item in the collection.
* last: boolean: Is set to True, when the item is the last item in the collection.
* even: boolean: True when the item has an even index in the collection.
* odd: boolean: is set to True when the item has an odd index in the collection.

## Finding the Index

To Find the index, we create another local variable i and use the let to make it equal to index.

|  |  |
| --- | --- |
| 1  2  3 | let i=index; |

The following code shows the list of movies along with the index.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <tr \*ngFor="let movie of movies; let i=index;">      <td> {{i}} </td>      <td>{{movie.title}}</td>      <td>{{movie.director}}</td>      <td>{{movie.cast}}</td>      <td>{{movie.releaseDate}}</td>  </tr> |

[***Source Code***](https://stackblitz.com/edit/angular-ngfor-directive-example4?)

## Formatting odd & even rows

We can use the odd & even values to format the odd & even rows alternatively. To do that create two local variables o & e. Assign the values of odd & even values to these variables using the let statement. Then use the[ngClass](https://www.tektutorialshub.com/angular/angular-ngclass-directive/) to change the class name to either odd or even. The example code is shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <tr \*ngFor="let movie of movies; let i=index; let o= odd; let e=even;"  [ngClass]="{ odd: o, even: e }">      <td> {{i}} </td>      <td>{{movie.title}}</td>      <td>{{movie.director}}</td>      <td>{{movie.cast}}</td>      <td>{{movie.releaseDate}}</td>  </tr> |

[***Source Code***](https://stackblitz.com/edit/angular-ngfor-directive-example5?)

Add the appropriate background color to the odd and even classes as shown below in app.component.css

|  |  |
| --- | --- |
| 1  2  3  4 | .even { background-color: azure; }  .odd { background-color: floralwhite; } |

[***Source Code***](https://stackblitz.com/edit/angular-ngfor-directive-example5?)

## First and the Last element of a list

Similarly, you can use the first & last values to style the first & last element. The code below will add CSS classes first & last to the first and last movie using the [ngClass](https://www.tektutorialshub.com/angular/angular-ngclass-directive/).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <div **class**='table-responsive'>    <table **class**='table table-bordered table-sm '>      <thead **class**="thead-dark">        <tr>          <th>Index</th>          <th>Title</th>          <th>Director</th>          <th>Cast</th>          <th>Release Date</th>        </tr>      </thead>      <tbody>        <tr \*ngFor="let movie of movies; let i=index; let first= first; let last=last;" [ngClass]="{ first: first, last: last }">          <td> {{i}} </td>          <td>{{movie.title}}</td>          <td>{{movie.director}}</td>          <td>{{movie.cast}}</td>          <td>{{movie.releaseDate}}</td>        </tr>      </tbody>    </table>  </div> |

[***Source Code***](https://stackblitz.com/edit/angular-ngfor-directive-example6?)

Remember to add the CSS classes to app.component.css

|  |  |
| --- | --- |
| 1  2  3  4 | .first { background-color: azure; }  .last { background-color: floralwhite; } |

[***Source Code***](https://stackblitz.com/edit/angular-ngfor-directive-example6?)

## Track By

The angular includes Track By clause, just like AngularJS did. Track By clause allows you to specify your own key to identify objects.

Angular uses the object identity to compare the elements in the collection to the DOM nodes. Hence when you add an item or remove an item, the Angular will track it and update only the modified items in the DOM. It does not render the entire list.

But this fails if we update the list from the backend server. That is because the retrieved objects cannot be compared with the existing objects in the list as the reference has changed. The Angular simply removes these elements from DOM and recreates the new elements from the new data. This has a huge performance implication.

[Angular trackBy](https://www.tektutorialshub.com/angular/angular-track-by-to-improve-ngfor-performance/)clause eliminates this problem, by telling angular how to identify similar elements. The Angular will use the value returned by the trackBy function to match the elements returned by the database and update the DOM Elements without recreating them.

We should always specify the primary key or unique key as the trackBy clause.

### Example

In our movie list example, let us make the title of the movie as the identifier.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <tr \*ngFor="let movie of movies; trackBy:trackByFn;">      <td>{{movie.title}}</td>      <td>{{movie.director}}</td>      <td>{{movie.cast}}</td>      <td>{{movie.releaseDate}}</td>  </tr> |

[***Source Code***](https://stackblitz.com/edit/angular-ngfor-directive-example7?)

In the [Component](https://www.tektutorialshub.com/angular/angular-component/) Class create a trackByFn. It gets the index and the current item as its argument. It should return the unique id

|  |  |
| --- | --- |
| 1  2  3  4  5 | trackByFn(index, item) {  **return** item.title;    } |