

## MODULE 4

### Introduction:

AngularJS is a JavaScript framework developed by Google to build dynamic, single-page web applications (SPAs). It extends HTML with additional features and makes web applications more interactive and responsive.

### Features

- **Data-binding** – It is the automatic synchronization of data between model and view components.
- **Scope** – These are objects that refer to the model. They act as a glue between controller and view.
- **Controller** – These are JavaScript functions bound to a particular scope.
- **Services** – AngularJS comes with several built-in services such as \$http to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.
- **Directives** – Directives are markers on DOM elements such as elements, attributes, css, and more. These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives such as ngBind, ngModel, etc.
- **Dependency Injection** – AngularJS has a built-in dependency injection subsystem that helps the developer to create, understand, and test the applications easily.

### Advantages

- It provides the capability to create Single Page Application in a very clean and maintainable way.
- It provides data binding capability to HTML. Thus, it gives user a rich and responsive experience.
- AngularJS code is unit testable.
- AngularJS uses dependency injection and make use of separation of concerns.
- AngularJS provides reusable components.
- With AngularJS, the developers can achieve more functionality with short code.
- In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.

### Disadvantages

- **Not Secure** – Being JavaScript only framework, application written in AngularJS are not safe. Server-side authentication and authorization is must to keep an application secure.
- **Not degradable** – If the user of your application disables JavaScript, then nothing would be visible, except the basic page.

## How to Set Up AngularJS?

You can set up AngularJS in two ways:

### 1. Using a CDN (Recommended)

Simply add the AngularJS script in your HTML file:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
```

### 2. Download and Use Locally

Download AngularJS from AngularJS Official Website.

Include it in your project:

```
<script src="angular.min.js"></script>
```

## Simple Angular JS Code

```
<html>
<head>
<title>AngularJS First Application</title>
<script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
</head>
<body>
<h1><center>Sample Application</center></h1>
<div ng-app = "">
<p>Enter your Name: <input type = "text" ng-model = "name"></p>
<p>Hello your name is: <span ng-bind = "name"></span>!</p>
</div>
</body>
</html>
```

### Sample Application

Enter your Name:

Hello your name is: Hareesha!

- `ng-app = ""` or simply `ng-app` refers to the `ng-app` directive which means that there is no module to assign controllers, directives, services attached to the code.
- If directive `"ng-app"` is removed from the code, HTML will simply display the expression without solving it.

## Expression

Expressions in AngularJS are used to bind application data to HTML. The expressions in AngularJS are written in double braces: `{{ expression }}`. They behave similar to *ng-bind* directives: `ng-bind="expression"`.

## Syntax

`{{ expression }}` or `ng-bind={{ expression }}`

**For example:**

`{{ 5 + 5 }}` or `{{ firstName + " " + lastName }}`

## Important features of AngularJS expressions

- Works similar to JavaScript expressions.
- Can use scope variables and perform calculations.
- Cannot contain control flow statements (if, for, while).
- Cannot access global objects (window, document).
- Can use filters for formatting

### 1. AngularJS Numbers

- In AngularJS, numbers can be used in expressions. Any expression using the number and the operators (like `+`, `-`, `*`, `%`, etc) then these are called number expressions.
- We may use `ng-init` directive to declare and initialize the variables in AngularJS. It's somewhat like defining local variables to code in any programming language.

For example: `ng-init="RateOfInterest=5;PrincipalAmount=20000"`

**Ex:**

```
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="">
  <p>This is an expression: {{ 2 * 5 }}</p>
</div>
</body>
</html>
```

This is an expression: 10

### 2. String Expressions in AngularJS

The string expression in AngularJS is a unit of code to perform operations on string values. It may be simply adding two strings i.e. concatenation etc.

```

<!DOCTYPE html>
<html>
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="" ng-init="firstName='ATME';lastName='College'">
<p>My full name is: {{ firstName + " " + lastName }}</p>
</div>
</body>
</html>

```

**My full name is:** ATME College

### 3. AngularJS Objects

AngularJs expressions can have objects as well. The object expressions in AngularJs hold object properties in them and the same way then evaluates at the view where they are used.

#### For Example,

lets define one object as a personname object having 2 key value pairs of "firstName" and "lastName".

personname object may be defined using ng-init directive

likenginit="personname={firstName: ,lastName: }"

```

<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
<body>
<h2> Using Object in AngularJS Expression </h2>
<div ng-app="" ng-init="personname={firstName:' ',lastName:'C'}">
<p>My First Name is : {{ personname.firstName }}</p>
<p>My Last Name is : {{ personname.lastName }}</p>
</div>
</body>
</html>

```

#### Using Object in AngularJS Expression

My First Name is :

My Last Name is :

#### 4. AngularJS Array Expressions

Expressions can be used to work with arrays as well. Array expressions in AngularJs are the expression that hold an array and use those array objects while evaluating array expressions.

**For Example**, submarks array may hold Marks value of various subject(here 4 subjects) and it may be defined using ng-init directive as ng-init="submarks=[43,45,50,37]"

```
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
<body>
<h2> Using Arrays in AngularJS</h2>
<div ng-app="" ng-init="submarks=[43,45,50,37]">
  <p>My First Subject Marks is : {{ submarks[0]}}</p>
  <p>My Second Subject Marks is : {{ submarks[1]}}</p>
  <p>My Third Subject Marks is : {{ submarks[2]}}</p>
  <p>My Fourth Subject Marks is : {{ submarks[3]}}</p>
</div>
</body>
</html>
```

#### Using Arrays in AngularJS

My First Subject Marks is : 43

My Second Subject Marks is : 45

My Third Subject Marks is : 50

My Fourth Subject Marks is : 37

#### AngularJS modules

In AngularJS, a **module** is a container for different parts of an application, such as controllers, directives, filters, and services. It helps in organizing code and keeping it modular and reusable.

It helps to link many components. So, it is just a group of related components.

##### Syntax:

```
angular.module('moduleName', [dependencies])
```

##### Ex:

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

Where,

- "myApp" is the **module name**.
- [] is an **array of dependencies** (empty for now)

**Creating a Module in AngularJS:**

```

<body>
  <div ng-app="Module-name">
    <div ng-controller="Controller-name">
      {{variable-name}}
    </div>

    <!-- This wont get printed since its not part of the div in which controller is included -->
    {{variable-name}}
  </div>
</body>

```

```

<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
  {{ firstName + " " + lastName }}
</div>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
  $scope.firstName = "ATME";
  $scope.lastName = "College";
});
</script>
</body>
</html>

```

**AngularJS directives**

AngularJS directives are used to extend HTML. Directives are markers on HTML DOM element that tell AngularJS to attach a specified behaviour to that HTML element.

Directives in AngularJS are special attributes starting with ng- prefix where ng stands for Angular. AngularJS includes various built-in directives; you may also create your own directive in AngularJS. Some built-in directives are listed here.

Directives	Description
ng-app	Start of AngularJS application.
ng-init	It is used to initialize a variable
ng-model	It is used to bind to the HTML controls
ng-controller	Attaches a controller to the view
ng-bind	Binds the value with an HTML element
ng-repeat	Repeats HTML template once per each item in the specified collection.
ng-show	Shows or hides the associated HTML element
ng-hide	Conditionally hides an HTML element based on the truthiness of an expression.
ng-readonly	Makes HTML element read-only
ng-disabled	Use to disable or enable a button dynamically
ng-if	Removes or recreates HTML element
ng-click	Custom step on click
ng-class	Conditionally applies CSS classes to an element based on the evaluation of expressions.
ng-submit	Binds a function to the submit event of an HTML form, allowing execution of custom behaviour on form submission.

### 1. ng-app directive

The ng-app directive defines the root element of an AngularJS application and starts an AngularJS Application. The ng-app directive will auto-bootstrap (automatically initialize) the application when a web page is loaded. It is also used to load various AngularJS modules in AngularJS Application.

**Ex:**

```
<div ng-app = "">
...
</div>
```

In the above example code, we have defined a default AngularJS application using ng-app attribute of a <div> element.

### 2. ng-init

The ng-init directive is used to initialize an AngularJS Application data. It defines the initial value for an AngularJS application and assigns values to the variables.

**Ex:**

```
<div ng-app = "" ng-init="firstName='';lastName=' ">
...
</div>
```

### 3. ng-model directive

The ng-model directive is used for two-way data binding in AngularJS. It is used to get value of input controls like textbox, label, etc and use these values in web pages.

example, we define a model named myname.

```
<div ng-app = " "> ... <input type = "text" ng-model = "myname">
<p>My name is(enter it): {{myname}} </p>
</div>
```

### 4. ng-repeat directive

The ng-repeat directive repeats HTML elements for each item in a collection. Or simply say that it is used to loop through items in collection element and it will act as for loop.

In the following example, we iterate over the name of the students:

```
<div ng-app="" ng-init="students=['Bharath','Divya','Sachin']">
<ul>
<li ng-repeat="name in students"> {{ 'Student Name: ' +name }}
</li>
</ul>
</div>
```

### 5. ng-bind

The ng-bind directive binds the model property declared via ng-model directive or the result of an expression to the HTML element. It also updates an element if the value of an expression changes.

```
<html>
<head>
<title> ng-bind Example </title>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
</head>
<h1> Showing ng-bind Directive</h1>
<body ng-app="">
<div> Sum of 10 + 5 = <span ng-bind="10 + 5"></span> <br />
Please Enter your Mobile no: <input type="text" ng-model="mobilenno" /><br />
Hi, my Mobile no is <span ng-bind="mobilenno">
</span>
</div>
</body>
</html>
```



## 6. ng-controller

The ng-controller Directive in AngularJS is used to add the controller to the application. It can be used to add methods, functions, and variables that can be called on some event like click, etc to perform certain actions.

Ex:

```
<!DOCTYPE html>
<html>
<head>
  <title>ng-controller Directive</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.2/angular.min.js">
  </script>
</head>
<body ng-app="app"
  style="text-align:center">
  <h2>ng-controller Directive</h2><br>
  <div ng-controller="control">
    Name: <input class="form-control" type="text" ng-model="name"> <br><br>
    You entered: <b><span>{{name}}</span></b>
  </div>

  <script>
    var app = angular.module('app', []);
    app.controller('control', function ($scope) {
      $scope.name = " ATME";
    });
  </script>
</body>
</html>
```

### ng-controller Directive

Name:

You entered:

## 7. ng-repeat

Angular-JS ng-repeat directive is a handy tool to repeat a set of HTML code a number of times or once per item in a collection of items. ng-repeat is mostly used on arrays and objects.

The ng-repeat is similar to a loop that we have in C, C++. Angular maintains a \$index variable as a key to the element which is currently being accessed and the user can also access this variable.

**Ex:** Create an app.js file for the app.

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

app.controller('MainCtrl', function($scope){

    $scope.names = ['Adam','Steve','George','James','Armin'];

    console.log($scope.names);

});
```

### Create index.html page

```
<script type="text/javascript" src="jquery-3.2.1.min.js"> </script>
<script type="text/javascript" src="angular.js"> </script>
<script type="text/javascript" src="app.js"> </script>
</head>

<body ng-controller="MainCtrl">
<ul>
    <li ng-repeat="name in names"> {{name}} </li>
</ul>
</body>
```

## 8. ng-show

The ng-show Directive in AngularJS is used to show or hide the specified HTML element. If the given expression in the ng-show attribute is *true* then the HTML element will display otherwise it hides the HTML element.

```
<!DOCTYPE html>
<html>
<head>
    <title>ng-show Directive</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
    </script>
</head>
<body>
    <div ng-app="app" ng-controller="show">
        <h2>ng-show Directive</h2>
        <input id="chshow" type="checkbox" ng-model="show" />
```

```
<label for="chshow"> Show Paragraph </label>
```

```
<p ng-show="show"
  style="background: green;
        color: white;
        font-size: 14px;
        width: 35%;
        padding: 10px;">
```

Show this paragraph using ng-show

```
</p>
```

```
</div>
```

```
<script>
```

```
var myapp = angular.module("app", []);
myapp.controller("show", function ($scope) {
  $scope.show = false;
});
```

```
</script>
```

```
</body>
```

```
</html>
```

## ng-show Directive

☒ Show Paragraph

Show this paragraph using ng-show

### 9. ng-disabled

The ng-disabled Directive in AngularJS is used to enable or disable HTML elements. If the expression inside the ng-disabled attribute returns true then the form field will be disabled or vice versa. It is usually applied on the form field (input, select, button, etc).

```
<div ng-controller="app"
  ng-init="disable=false">
  <button ng-click="geek(disable)"
    ng-disabled="disable">
    Click to Disable
  </button>

  <button ng-click="geek(disable)"
    ng-show="disable">
```

```

    Click to Enable
  </button>
</div>

```

## 10. ng-click:

The ng-click Directive in AngularJS is used to apply custom behavior when an element is clicked. It can be used to show/hide some element or it can pop up an alert when the button is clicked.

```

<div ng-controller="app">
  <button>
    <a href="" ng-click="alert()">
      Click Here
    </a>
  </button>
</div>

```

## Model in AngularJS

In AngularJS, a model represents the data and state of an application. It is responsible for storing and managing application data and is connected to the view (HTML) using two-way data binding.

AngularJS uses \$scope as the model, which acts as a bridge between the controller (logic) and view (UI).

### Example:

```

html
<html lang="en" ng-app="myApp">
<head>
  <title>AngularJS Model Example</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-controller="myController">
  <p>Enter Name: <input type="text" ng-model="name"></p>
  <p>Hello, {{ name }}!</p>
  <script>
    var app = angular.module("myApp", []);
    app.controller("myController", function($scope) {
      $scope.name = "ATME";
    });
  </script>
</body>
</html>

```

Where,

- `ng-model="name"` → Binds input to the `$scope.name`.
- `{{ name }}` → Displays the updated value dynamically.

## Data Binding

Data binding in AngularJS is the synchronization of data between the model (JavaScript) and the view (HTML). This means when data changes in the model, the view updates automatically, and vice versa.

Data binding can be categorized into 2 types, ie.,

1. One-Way Data Binding (View → Model or Model → View)

2. Two-Way Data Binding (View ↔ Model)

**1. One-way Binding:** This type of binding is unidirectional, i.e. this binds the data flow from either component to view (DOM) or from the view (DOM) to the component.

### Syntax:

`class="{{variable_name}}"`

```
<html lang="en" ng-app="myApp">
<head>
  <title>AngularJS Model Example</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-controller="myController">

  <div ng-app="myApp" ng-controller="myController">
    <p ng-bind="message"></p>
  </div>

  <script>
    var app = angular.module("myApp", []);
    app.controller("myController", function($scope) {
      $scope.message = "Hello, AngularJS!";
    });
  </script>
</body>
</html>
```

Hello, AngularJS!

**2. Two-way Binding:** In this type of binding, the immediate changes to the view & component, will be reflected automatically, i.e. when the changes made to the component or model then the *view* will render the changes simultaneously. Similarly, when the data is altered or modified in the view then the model or component will be updated accordingly.

```
<html lang="en" ng-app="myApp">
<head>
  <title>AngularJS Model Example</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-controller="myController">
  <div ng-app="myApp" ng-controller="myController">
    <input type="text" ng-model="name">
    <p>Hello, {{ name }}!</p>
  </div>
  <script>
    var app = angular.module("myApp", []);
    app.controller("myController", function($scope) {
      $scope.name = "ATME";
    });
  </script>
</body>
</html>
```

Hello, ATME

Note: When you type in the input, both the model (`$scope.name`) and view update.

## Controllers

AngularJS controller controls the flow of data from the model part to the view part. Data is taken from the view part as an input by the controller, then it processes the data and sends back to the view part (HTML part) that is being displayed to the users.

Controllers are responsible for initializing data, handling user interactions, and defining the business logic of an AngularJS application.

### Syntax:

```
<element ng-controller="expression">
```

Contents...

```
</element>
```

```

<!DOCTYPE html>
<html>

<head>
  <title>ng-controller Directive</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
  </script>
  <script>
    var app = angular.module("myApp", []);
    app.controller("myController", function ($scope) {
      $scope.message = "Hello, AngularJS!";
    });
  </script>
</head>

<body>
  <body ng-app="myApp">
    <div ng-controller="myController">
      <h1>{{ message }}</h1>
    </div>
  </body>
</body>
</html>

```

**Hello, AngularJS!**

### Explanation:

- `angular.module("myApp", [])`: Creates an AngularJS module named `myApp`.
- `.controller("myController", function($scope) {...})`: Defines a controller named `myController`.
- `$scope.message`: Adds a variable `message` to the `$scope`, which binds data to the view.
- `ng-app="myApp"`: Declares an AngularJS application.
- `ng-controller="myController"`: Binds the controller to the `<div>`.
- `{{ message }}`: Uses AngularJS **expression** to display data from the controller.

### Scope

The **Scope** in AngularJS is the binding part between HTML (view) and JavaScript (controller) and it is a built-in object. It contains application data and objects. It is available for both the view and the controller.

There are two types of scopes in Angular JS.

1. `$Scope` (Local Scope)
2. `$rootScope` (Global Scope)

## 1. \$Scope (Local Scope)

Each controller has its own \$scope, which is local to that controller. Thus, the data and methods attached to \$scope inside one controller cannot be accessed on another controller.

```
<!DOCTYPE html>
<html>
<head>
  <title>ng-controller Directive</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
  </script>
  <script>
    var app = angular.module("myApp", []);
    app.controller("myController", function ($scope) {
      $scope.message = " Local Scope";
    });
  </script>
</head>
<body>
  <body ng-app="myApp">
    <div ng-controller="myController">
      <h1>{{ message }}</h1>
    </div>
  </body>
</html>
```

## Local Scope

## 2. \$rootScope (Global Scope)

In AngularJS, the global scope typically refers to the \$rootScope, which is the top-level scope available throughout the application. Every AngularJS app has a \$rootScope, and all other scopes (such as those in controllers, directives, and services) inherit from it.

- The \$rootScope is the top-most scope and is shared across all controllers.
- If a variable is defined on \$rootScope, it can be accessed anywhere in the application.

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
  <title>AngularJS Global Scope Example</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-controller="MainCtrl">
```



```
<p>Global Message: {{ globalMessage }}</p>
```

```
<p>Controller Message: {{ localMessage }}</p>
```

```
<button ng-click="updateGlobalMessage()">Change Global Message</button>
```

```
<script>
```

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

```
app.run(function($rootScope) {
```

```
    $rootScope.globalMessage = "Hello from $rootScope!";
```

```
});
```

```
app.controller("MainCtrl", function($scope, $rootScope) {
```

```
    $scope.localMessage = "Hello from MainCtrl!";
```

```
    $scope.updateGlobalMessage = function() {
```

```
        $rootScope.globalMessage = "Global Message Updated!";
```

```
    };
```

```
});
```

```
</script>
```

```
</body>
```

```
</html>
```

Global Message: Hello from \$rootScope!

Controller Message: Hello from MainCtrl!

Change Global Message

Global Message: Global Message Updated!

Controller Message: Hello from MainCtrl!

Change Global Message

## Filters

Filters can be added in AngularJS to format (Modify) data to display on UI without changing the original format. Filters can be added to an expression or directives using the pipe | symbol. They can be applied in expressions, directives, and controllers.

### Syntax:

```
{{ expression | filterName }}
```

<b>currency</b>	It is used to convert a number into a currency format.
<b>Date</b>	It is used to convert a date into a specified format.
<b>Filter</b>	It is used to filter the array and object elements and return the filtered items.
<b>Json</b>	To convert a JavaScript object into JSON.
<b>limitTo</b>	It is used to return an array or a string that contains a specified number of elements.
<b>lowercase</b>	It is used to convert a string into lowercase letters.
<b>uppercase</b>	It is used to convert a string into uppercase letters.
<b>number</b>	It is used to convert a number into a string or text.
<b>orderBy</b>	It sorts an array based on specified predicate expressions.

**1. Currency Filter:** This filter is used to convert a number into a currency format. When no currency format is implemented, the currency filter uses the local currency format.

**Syntax:**

```
{{ currency_expression | currency : symbol : fractionSize }}
```

**Parameters:** It contains 2 parameters as mentioned above and described below:

- **symbol:** It is an optional parameter. It is used to specify the currency symbol. The currency symbol can be any character or text.
- **fractionsize:** It is an optional parameter. It is used to specify the number of decimals.

```
<div ng-app="gfgApp" ng-controller="currencyCntrl">
  <h3>
    Currency filter with currency
    symbol and fraction size.
  </h3>
  <p>Amount : {{ amount | currency : "₹" :2}}</p>
</div>
```

**2. Date Filter:** This filter is used to convert a date into a specified format. If the date format is not specified, then the default format of date is used 'MM dd yyyy'.

**Syntax:**

```
{{ date | date : format : timezone }}
```

**Parameter Values:** The date filter contains format and timezone parameters which is optional.

```

<body>
  <div ng-app="gfgApp" ng-controller="dateCntrl">
    <p>{{ today | date : "dd.MM.y" }}</p>
  </div>

  <script>
    var app = angular.module('gfgApp', []);
    app.controller('dateCntrl', function($scope) {
      $scope.today = new Date();
    });
  </script>
</body>

```

**3. filter Filter:** This filter is used to filter the array and object elements and return a new array.

**Syntax:**

```
{{ expression | filter : filter_criteria }}
```

**Parameter Values:**

- **arrayexpression:** The source array on which the filter will be applied.
- **expression:** It is used to select the items from the array after the filter conditions are met. It can be of String type, Function type, or Object type.
- **comparator:** It is used to determine the value by comparing the expected value from the filter expression, and the actual value from the object array.
- **anyPropertyKey:** It is an optional parameter having the special property that is used to match the value against the given property. It is of String type & its default value is \$.

```

$scope.items = ["Apple", "Banana", "Cherry", "Mango"];
<input type="text" ng-model="searchText">
<ul>
<li ng-repeat="item in items | filter:searchText">{{ item }}</li>
</ul>

```

**4. json Filter:** This filter is used to convert a JavaScript object into JSON.

**Syntax:**

```
{{ object | json : spacing }}
```

**Parameter value:**

- **json:** It is used to specify that the object should be displayed in JSON format.
- **spacing:** It is an optional parameter with a default value of 2 that specifies the number of spaces per indentation.

**5. limitTo Filter:** This filter is used to return an array or a string that contains a detailed number of elements. It is used for strings and numbers. It returns a string containing only the specified number of digits and characters.

**Syntax:**

```
{{ object | limitTo : limit : begin }}
```

**Parameters:**

- **limit:** Specifies the length of the returned array or strings.
- **Begin:** Specifies the index with which the limitation begins. By default, its value is zero.

```
angular.module("myApp", []).controller("myCtrl", function($scope) {
$scope.items = ["Apple", "Banana", "Cherry", "Date", "Fig"];
});
```

```
<div ng-app="myApp" ng-controller="myCtrl">
<ul>
<li ng-repeat="item in items | limitTo:3">{{ item }}</li>
</ul>
</div>
```

**6. lowercase:** This filter is used to convert a string into lowercase letters.

**Syntax:**

```
{{expression|lowercase}}
```

```
angular.module("myApp", []).controller("myCtrl", function($scope) {
$scope.message = "Hello AngularJS!";
});
```

```
<div ng-app="myApp" ng-controller="myCtrl">
<p>Original: {{ message }}</p>
<p>Lowercase: {{ message | lowercase }}</p>
</div>
```

**7. uppercase:** This filter is used to convert a string into an uppercase letter.

**Syntax:**

```
{{ string | uppercase}}
```

```
angular.module("myApp", []).controller("myCtrl", function($scope) {
$scope.message = "Hello AngularJS!";
});
```

```
<div ng-app="myApp" ng-controller="myCtrl">
  <p>Original: {{ message }}</p>
  <p>Uppercase: {{ message | uppercase }}</p>
</div>
```

**8. number Filter:** This filter is used to convert a number into a string or text.

**Syntax:**

```
{{ string| number : fractionSize }}
```

- `number_expression` → The number to be formatted.
- `fractionSize` (optional) → Number of decimal places (default is 3).

```
angular.module("myApp", []).controller("myCtrl", function($scope) {
  $scope.amount = 9876.54321;
});
<div ng-app="myApp" ng-controller="myCtrl">
  <p>Original: {{ amount }}</p>
  <p>Formatted: {{ amount | number:2 }}</p>
</div>
```

**9. orderBy Filter:** This filter sorts an array based on specified predicate expressions.

**Syntax:**

```
{{ orderBy_expression | orderBy : expression : reverse }}
```

- `expression` → Property name or custom function to sort by.
- `reverse` (optional) → If true, sorts in descending order (default is ascending)

```
angular.module("myApp", []).controller("myCtrl", function($scope) {
  $scope.numbers = [5, 3, 8, 1, 4];
});
<div ng-app="myApp" ng-controller="myCtrl">
  <p>Sorted Numbers: {{ numbers | orderBy }}</p> </div>
```