LAB 6

Name: Prasad Dhengale

Roll no: 212010069

Aim:

(a) Introduction to AngularJS
(b) Expressions and Data Binding
(c) Working with Directives and Controllers
(d) Filters
(e) Forms
(f) Modules and Services

(a)Introduction to AngularJS

AngularJS is a **JavaScript framework**. It can be added to an HTML page with a <script> tag.
AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.

AngularJS is a JavaScript Framework

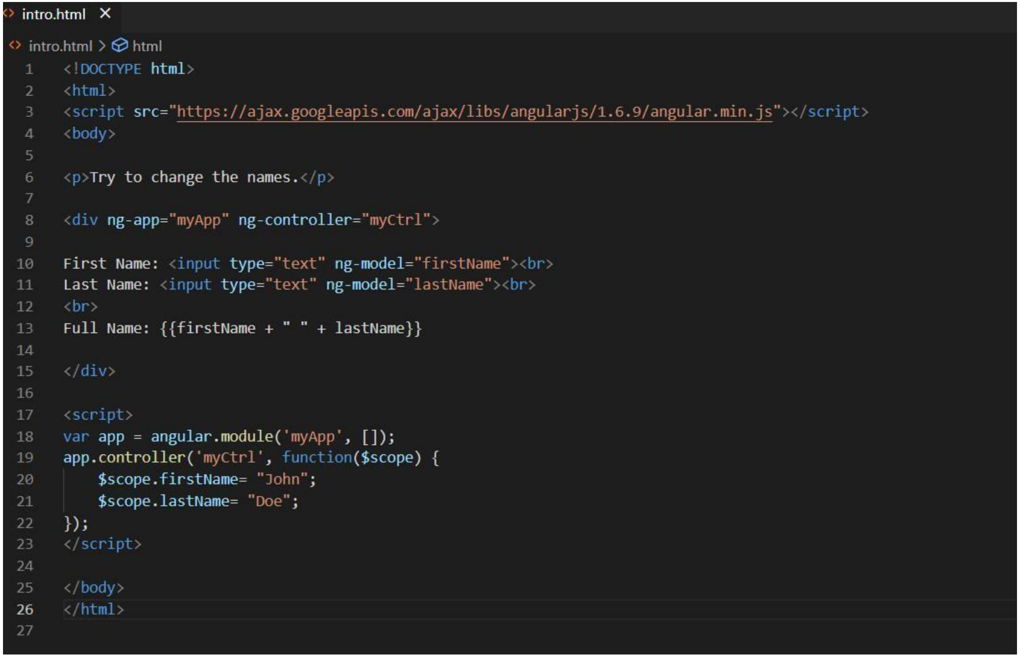
AngularJS is a JavaScript framework written in JavaScript.
AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag:
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

AngularJS Extends HTML

AngularJS extends HTML with **ng-directives**.
The **ng-app** directive defines an AngularJS application.
The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.
The **ng-bind** directive binds application data to the HTML view.

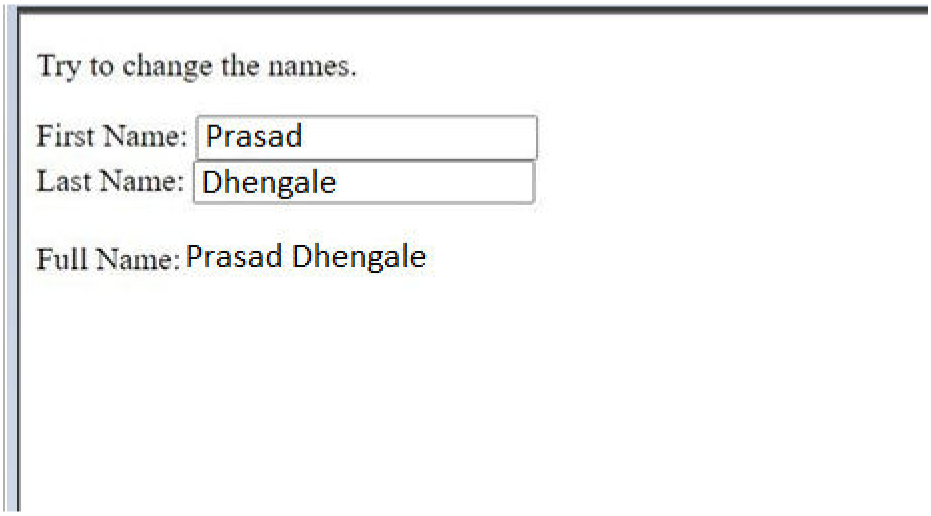
AngularJS Applications

AngularJS **modules** define AngularJS applications.
AngularJS **controllers** control AngularJS applications.
The **ng-app** directive defines the application, the **ng-controller** directive defines the controller.



```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<p>Try to change the names.</p>

<div ng-app="myApp" ng-controller="myCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.firstName= "John";
    $scope.lastName= "Doe";
});
</script>

</body>
</html>

```

Output:



Try to change the names.

First Name: Prasad
Last Name: Dhengale

Full Name: Prasad Dhengale

(b)Expressions and Data Binding

AngularJS Expressions vs. JavaScript

Expressions

Like JavaScript expressions, AngularJS expressions can contain literals, operators, and variables.

Unlike JavaScript expressions, AngularJS expressions can be written inside HTML.

AngularJS expressions do not support conditionals, loops, and exceptions, while JavaScript expressions do.

AngularJS expressions support filters, while JavaScript expressions do not.

AngularJS binds data to HTML using **Expressions**.

AngularJS Expressions

AngularJS expressions can be written inside double braces: {{ expression }}.

AngularJS expressions can also be written inside a directive: ng-bind="expression".

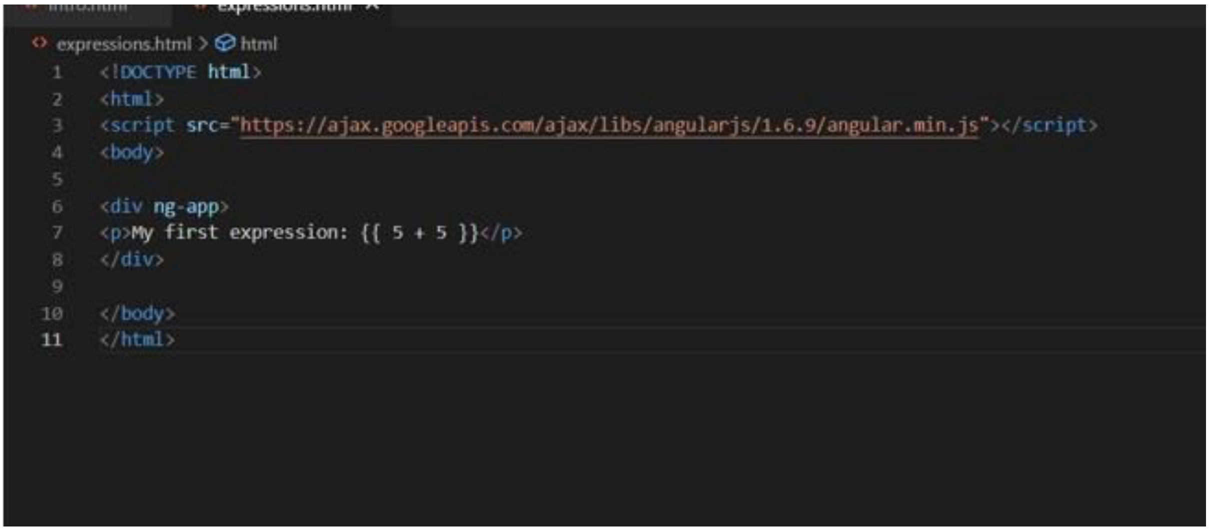
AngularJS will resolve the expression, and return the result exactly where the expression is

written.

AngularJS expressions are much like **JavaScript expressions:** They can contain

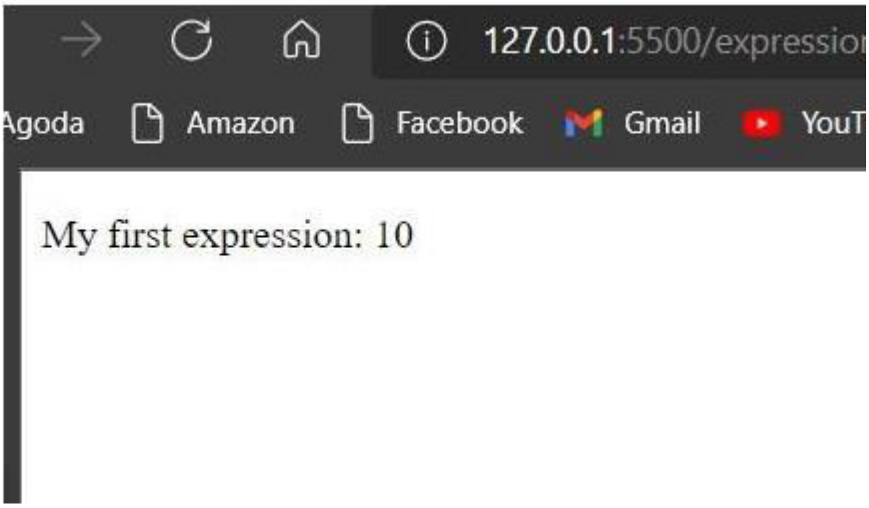
literals, operators, and variables.

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



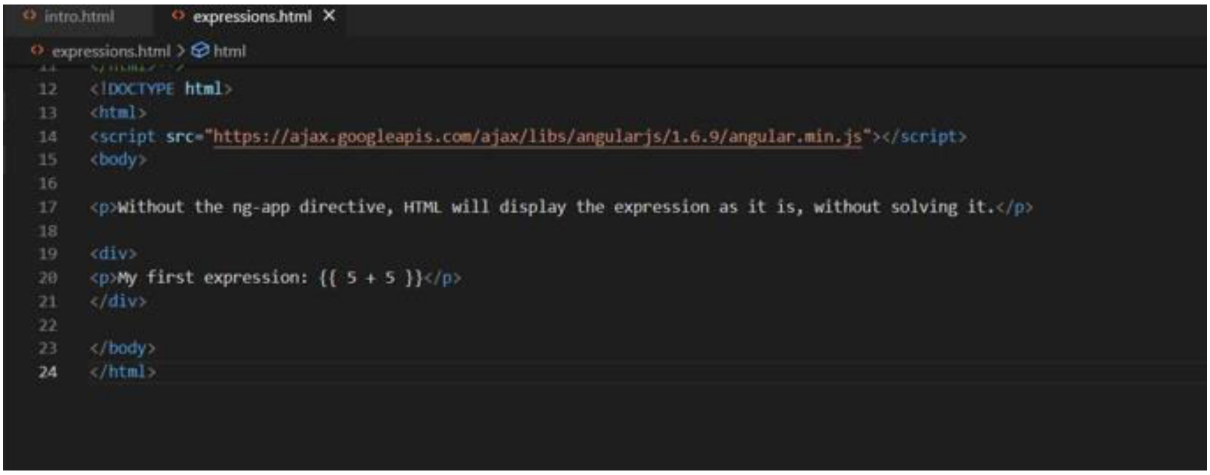
```html
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app>
<p>My first expression: {{ 5 + 5 }}</p>
</div>

</body>
</html>
```

Output:



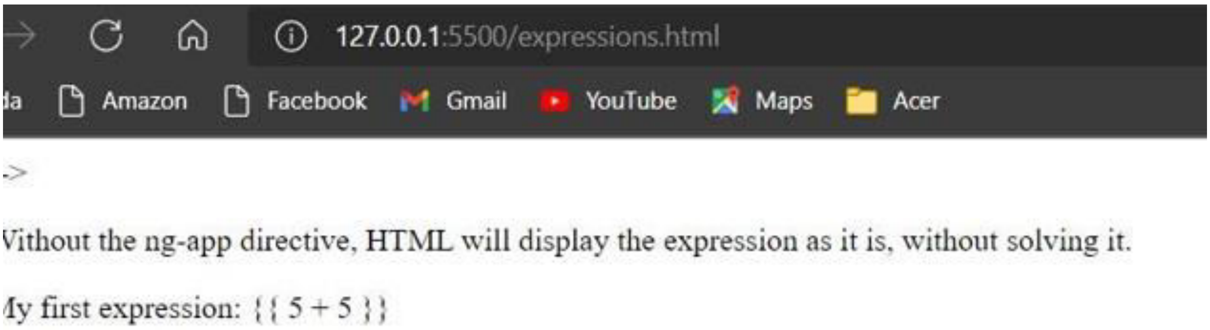
My first expression: 10

If you remove the ng-app directive, HTML will display the expression as it is, without solving it.



```html
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<p>Without the ng-app directive, HTML will display the expression as it is, without solving it.</p>

<div>
<p>My first expression: {{ 5 + 5 }}</p>
</div>

</body>
</html>
```

Output:



->

Vithout the ng-app directive, HTML will display the expression as it is, without solving it.

Iy first expression: {{ 5 + 5 }}

AngularJS Data Binding

Data binding in AngularJS is the synchronization between the model and the view.

Data Model

AngularJS applications usually have a data model. The data model is a collection of data available for the application.

HTML View

The HTML container where the AngularJS application is displayed, is called the view.
The view has access to the model, and there are several ways of displaying model data in the view.
You can use the ng-bind directive, which will bind the innerHTML of the element to the specified model property:

Example

```
<p ng-bind="firstname"></p>
```

You can also use double braces {{ }} to display content from the model:

Example

```
<p>First name: {{firstname}}</p>
```

The ng-model Directive

Use the ng-model directive to bind data from the model to the view on HTML controls (input, select, textarea)

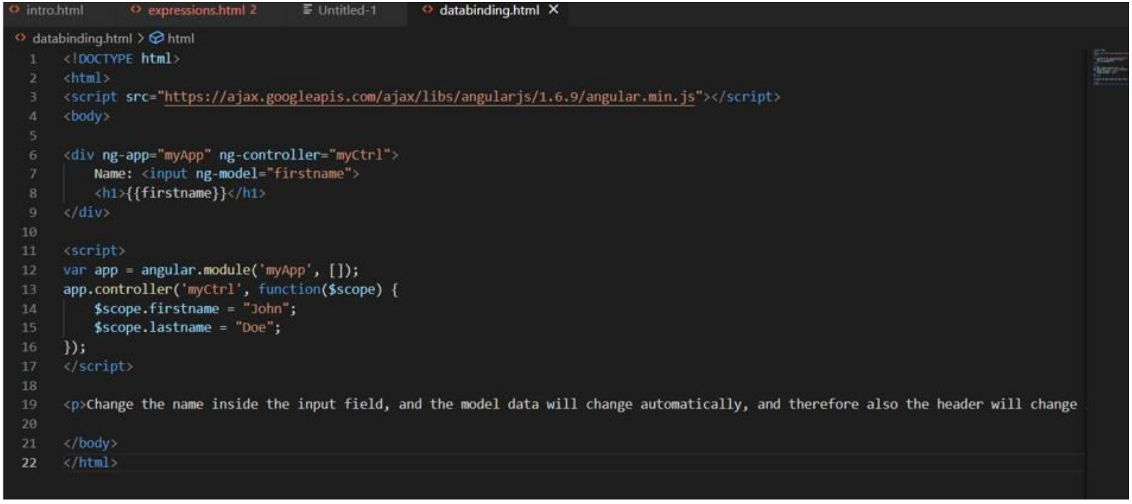
Example

```
<input ng-model="firstname">
```

The ng-model directive provides a two-way binding between the model and the view.

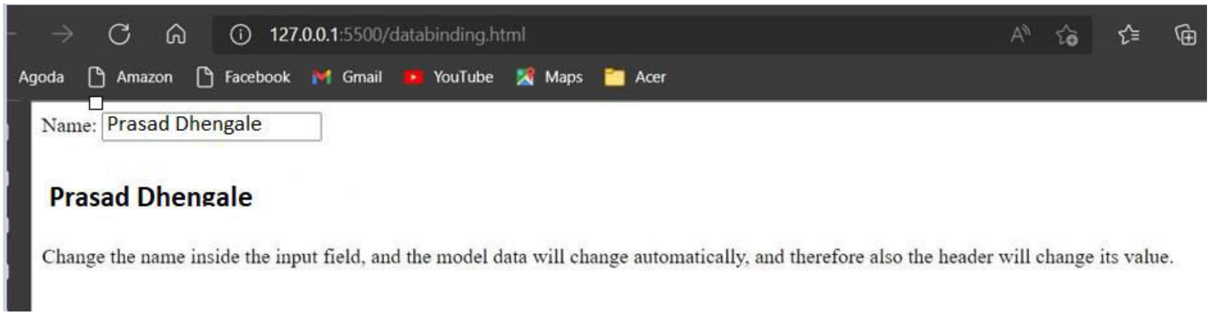
Two-way Binding

Data binding in AngularJS is the synchronization between the model and the view. When data in the model changes, the view reflects the change, and when data in the view changes, the model is updated as well. This happens immediately and
automatically, which makes sure that the model and the view is updated at all times



```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
    Name: <input ng-model="firstname">
    <h1>{{firstname}}</h1>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.firstname = "John";
    $scope.lastname = "Doe";
});
</script>

<p>Change the name inside the input field, and the model data will change automatically, and therefore also the header will change

</body>
</html>
```

Output:



Name: Prasad Dhengale

Prasad Dhengale

Change the name inside the input field, and the model data will change automatically, and therefore also the header will change its value.

Working with Directives and Controllers

AngularJS Directives

AngularJS lets you extend HTML with new attributes called **Directives**.

AngularJS has a set of built-in directives which offers functionality to your applications. AngularJS also lets you define your own directives.

AngularJS directives are extended HTML attributes with the prefix ng-. The ng-app directive initializes an AngularJS application.

The ng-init directive initializes application data.

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

The ng-app Directive

The ng-app directive defines the **root element** of an AngularJS application.

The ng-app directive will **auto-bootstrap** (automatically initialize) the application when a web page is loaded.

The ng-init Directive

The ng-init directive defines **initial values** for an AngularJS application. Normally, you will not

use ng-init. You will use a controller or module instead. You will learn more about controllers

and modules later.

The ng-model Directive

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

The ng-model directive can also:

- Provide type validation for application data (number, email, required).
- Provide status for application data (invalid, dirty, touched, error).
- Provide CSS classes for HTML elements.
- Bind HTML elements to HTML forms.

Read more about the ng-model directive in the next chapter.

Create New Directives

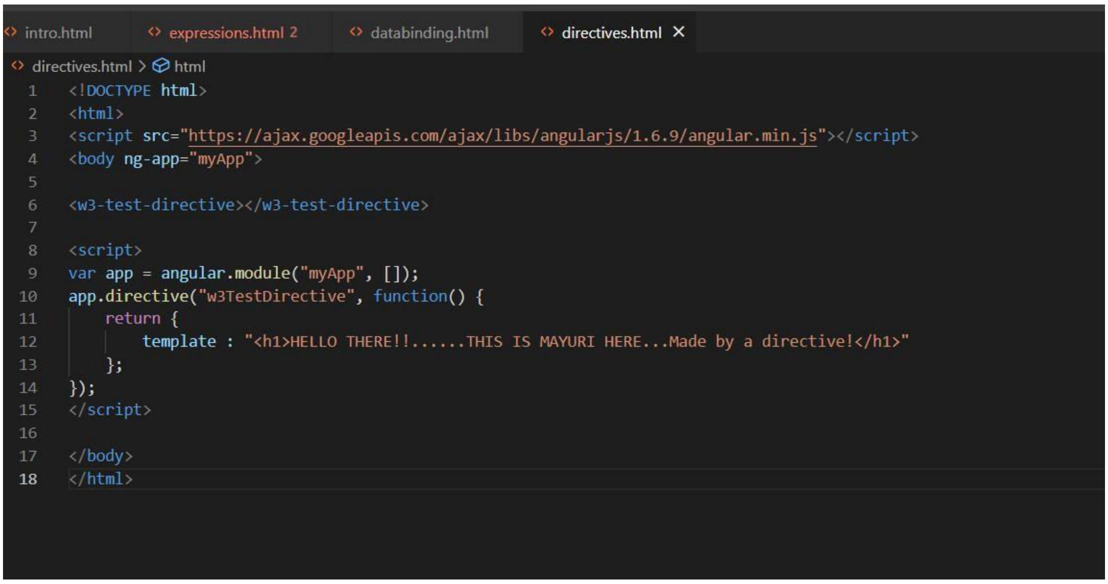
In addition to all the built-in AngularJS directives, you can create your own directives. New directives are

created by using the .directive function.

To invoke the new directive, make an HTML element with the same tag name as the new directive.

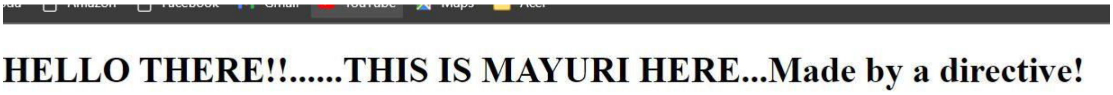
When naming a directive, you must use a camel case name, w3TestDirective, but when invoking it, you must use - separated name, w3-test-directive:

Example



```html
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="myApp">

<w3-test-directive></w3-test-directive>

<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
    return {
        template : "<h1>HELLO THERE!!......THIS IS MAYURI HERE...Made by a directive!</h1>"
    };
});
</script>

</body>
</html>
```

OUTPUT:



HELLO THERE!!......THIS IS MAYURI HERE...Made by a directive!

You can invoke a directive by using:

- Element name
- Attribute
- Class
- Comment

The examples below will all produce the same result:

Element name

```html
<w3-test-directive></w3-test-directive>
```

Attribute

```html
<div w3-test-directive></div>
```

Class

```html
<div class="w3-test-directive"></div>
```

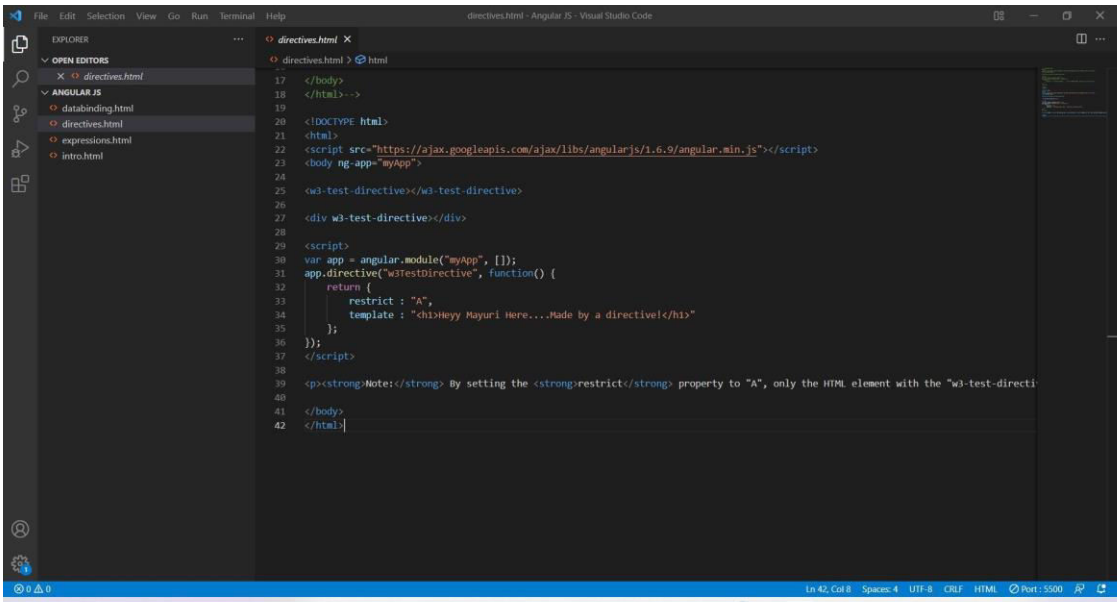
Comment

```
<!-- directive: w3-test-directive -->
```

Restrictions

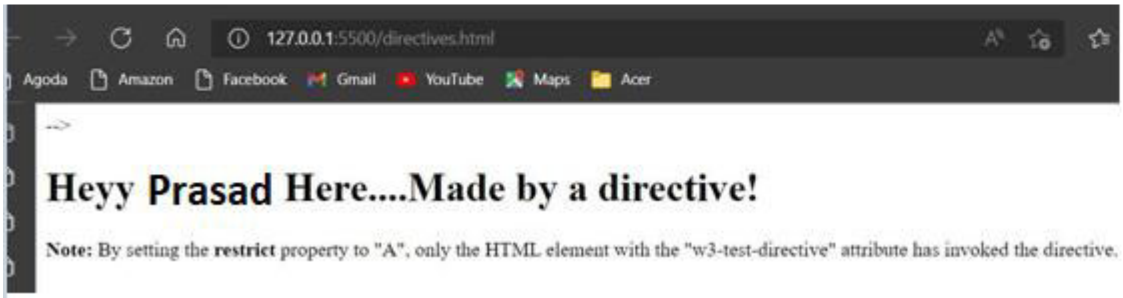
You can restrict your directives to only be invoked by some of the methods.

Example



```
</body>
</html>-->

<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body ng-app="myApp">

<w3-test-directive></w3-test-directive>

<div w3-test-directive></div>

<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
    return {
        restrict : "A",
        template : "<h1>Heyy Mayuri Here....Made by a directive!</h1>"
    };
});
</script>

<p><strong>Note:</strong> By setting the <strong>restrict</strong> property to "A", only the HTML element with the "w3-test-directi

</body>
</html>
```

OUTPUT:



-->

Heyy Prasad Here....Made by a directive!

Note: By setting the **restrict** property to "A", only the HTML element with the "w3-test-directive" attribute has invoked the directive.

The legal restrict values are:

- E for Element name
- A for Attribute
- C for Class
- M for Comment

By default the value is EA, meaning that both Element names and attribute names can invoke the directive.

AngularJS controllers **control the data** of AngularJS applications.

AngularJS controllers are regular **JavaScript Objects**.

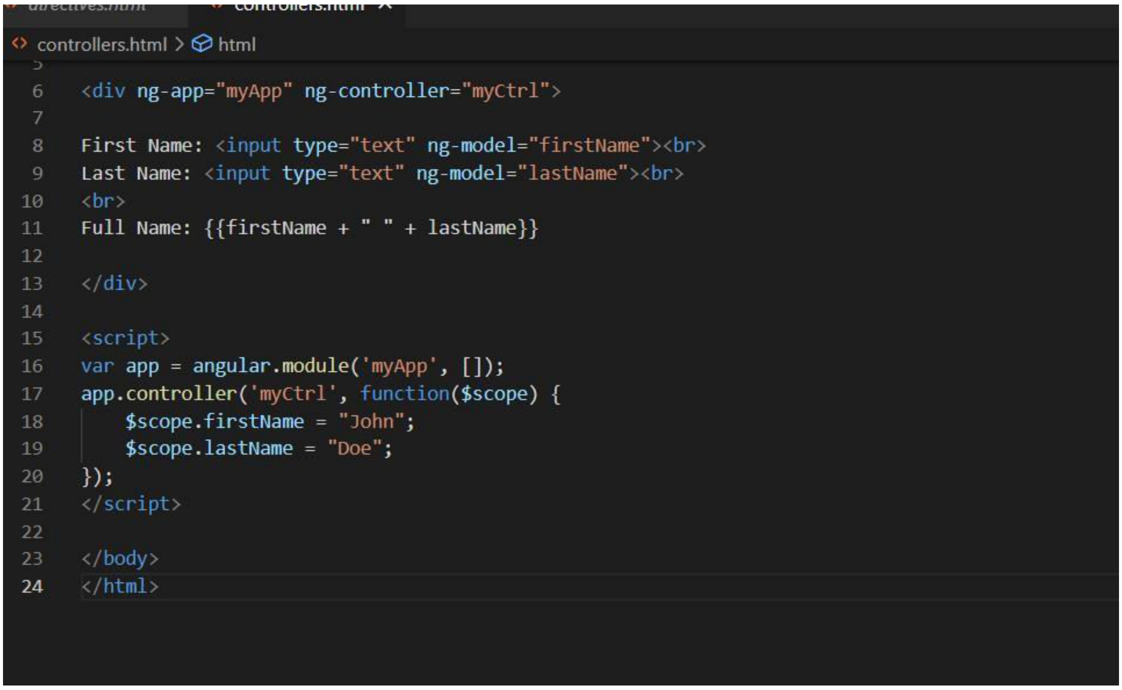
AngularJS Controllers

AngularJS applications are controlled by controllers.

The **ng-controller** directive defines the application controller.

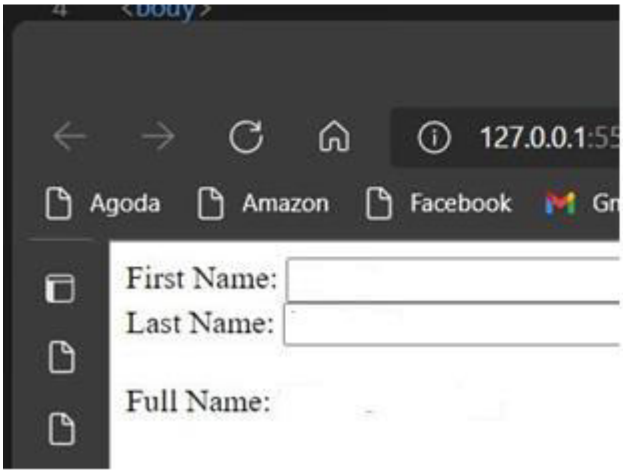
A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

AngularJS Example



```
<div ng-app="myApp" ng-controller="myCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.firstName = "John";
    $scope.lastName = "Doe";
});
</script>

</body>
</html>
```

OUTPUT:



First Name:
Last Name:

Full Name:

Application explained:

The AngularJS application is defined by **ng-app="myApp"**. The application runs inside the <div>.

The **ng-controller="myCtrl"** attribute is an AngularJS directive. It defines a controller. The **myCtrl** function is a JavaScript function.

AngularJS will invoke the controller with a **$scope** object.

In AngularJS, $scope is the application object (the owner of application variables and functions).

The controller creates two properties (variables) in the scope (**firstName** and **lastName**).

The **ng-model** directives bind the input fields to the controller properties (firstName and lastName).

Controller Methods

The example above demonstrated a controller object with two properties: lastName and firstName.

A controller can also have methods (variables as functions):

Controllers In External Files

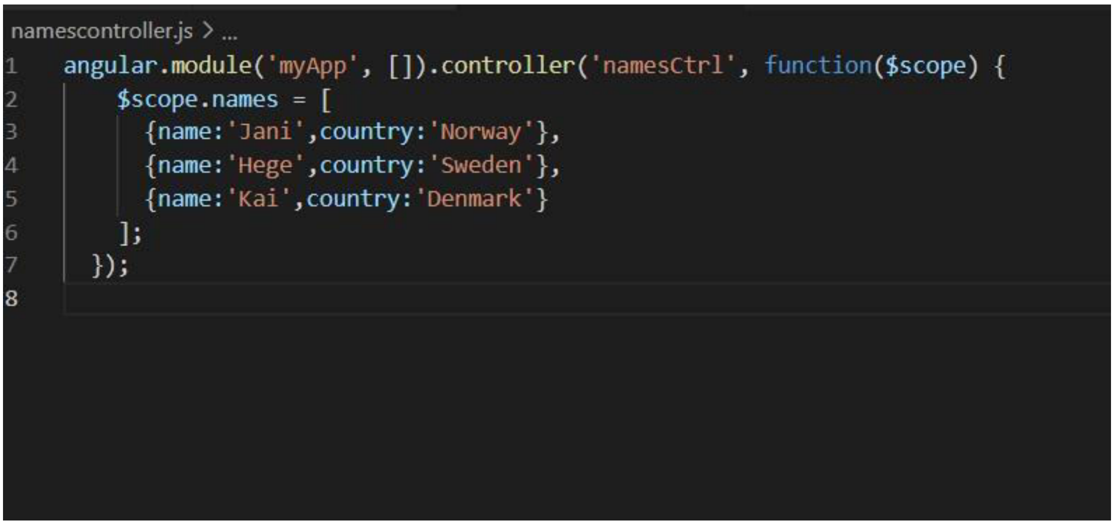
In larger applications, it is common to store controllers in external files.

Just copy the code between the <script> tags into an external file named personController.js:

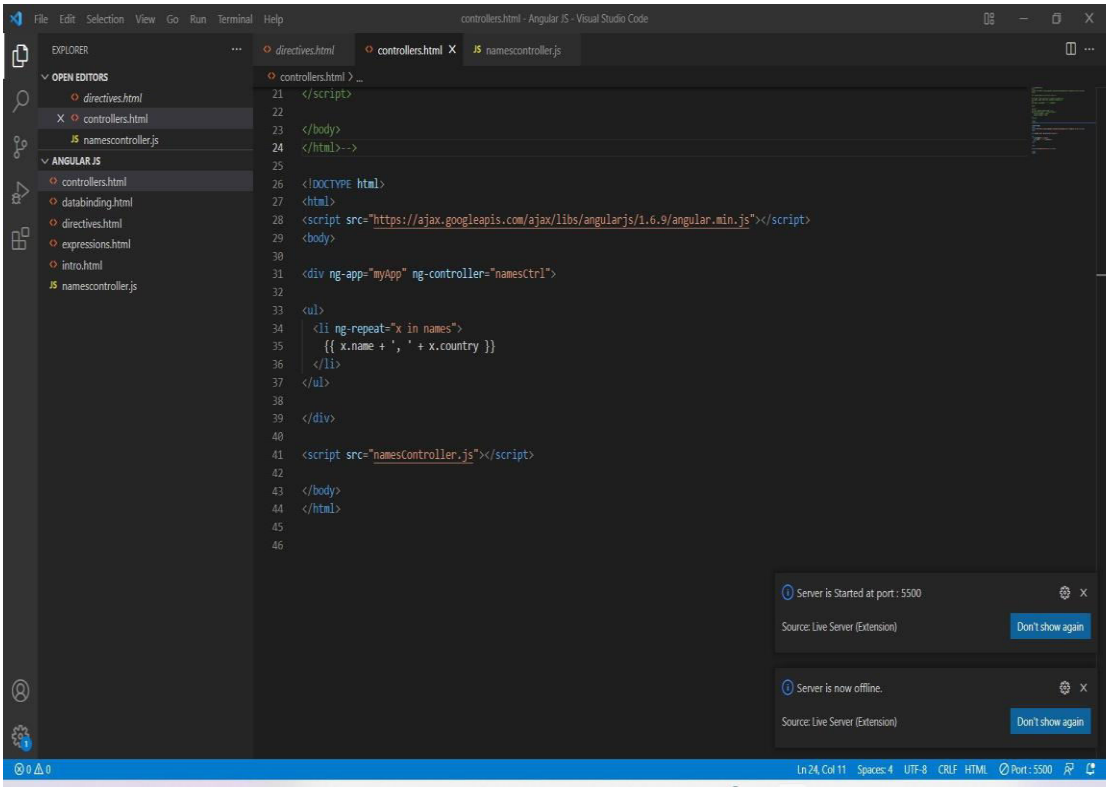
Save the file as namesController.js:

And then use the controller file in an application: MY

JAVASCRPIT FILE:



```
angular.module('myApp', []).controller('namesCtrl', function($scope) {
    $scope.names = [
      {name:'Jani',country:'Norway'},
      {name:'Hege',country:'Sweden'},
      {name:'Kai',country:'Denmark'}
    ];
  });
```

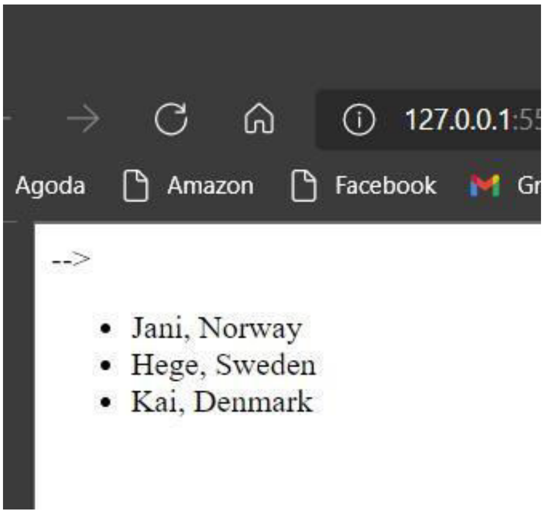


```
</script>

</body>
</html>-->

<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="namesCtrl">

<ul>
  <li ng-repeat="x in names">
    {{ x.name + ', ' + x.country }}
  </li>
</ul>

</div>

<script src="namesController.js"></script>

</body>
</html>
```

Server is Started at port : 5500
Source: Live Server (Extension)

Server is now offline.
Source: Live Server (Extension)

OUTPUT:



```
-->

  • Jani, Norway
  • Hege, Sweden
  • Kai, Denmark
```

Filters

Filters can be added in AngularJS to format data.

AngularJS Filters

AngularJS provides filters to transform data:

- currency Format a number to a currency format.
- date Format a date to a specified format.
- filter Select a subset of items from an array.
- json Format an object to a JSON string.
- limitTo Limits an array/string, into a specified number of elements/characters.
- lowercase Format a string to lower case.
- number Format a number to a string.
- orderBy Orders an array by an expression.
- uppercase Format a string to upper case.

Adding Filters to Expressions

Filters can be added to expressions by using the pipe character |, followed by a filter. The uppercase filter format strings to upper case:

Example

```
<div ng-app="myApp" ng-controller="personCtrl">

<p>The name is {{ lastName | uppercase }}</p>

</div>
```

The lowercase filter format strings to lower case:

Example

```
<div ng-app="myApp" ng-controller="personCtrl">

<p>The name is {{ lastName | lowercase }}</p>

</div>
```

Adding Filters to Directives

Filters are added to directives, like ng-repeat, by using the pipe character |, followed by a filter:

Example

The orderBy filter sorts an array:

```
<div ng-app="myApp" ng-controller="namesCtrl">

<ul>
  <li ng-repeat="x in names | orderBy:'country'">
    {{ x.name + ', ' + x.country }}
  </li>
</ul>

</div>
```

The currency Filter

The currency filter formats a number as currency:

Example

```
<div ng-app="myApp" ng-controller="costCtrl">

<h1>Price: {{ price | currency }}</h1>

</div>
```

The filter Filter

The `filter` filter selects a subset of an array.

The `filter` filter can only be used on arrays, and it returns an array containing only the matching items.

Example

Return the names that contains the letter "i":

```
<div ng-app="myApp" ng-controller="namesCtrl">

<ul>
  <li ng-repeat="x in names | filter : 'i'">
    {{ x }}
  </li>
</ul>

</div>
```

Filter an Array Based on User Input

By setting the ng-model directive on an input field, we can use the value of the input field as an expression in a filter.

Type a letter in the input field, and the list will shrink/grow depending on the match:

- Jani
- Carl
- Margareth
- Hege
- Joe
- Gustav
- Birgit
- Mary
- Kai

Example

```
<div ng-app="myApp" ng-controller="namesCtrl">

<p><input type="text" ng-model="test"></p>

<ul>
  <li ng-repeat="x in names | filter : test">
    {{ x }}
  </li>
</ul>

</div>
```

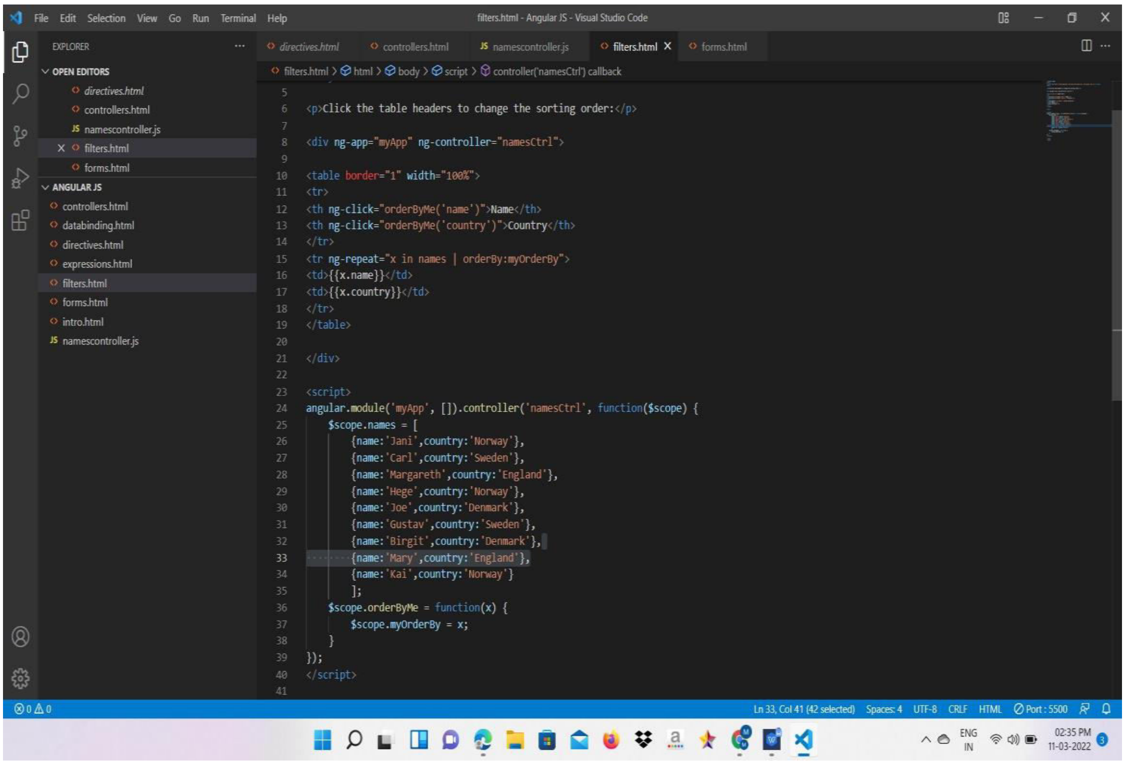
Sort an Array Based on User Input

Click the table headers to change the sort order::

Name	Country
Jani	Norwa
Margareth	Englan
Jo	Denmar
Gustav	Sweden
Birgit	Denmar
Mary	England
Kai	Norwa

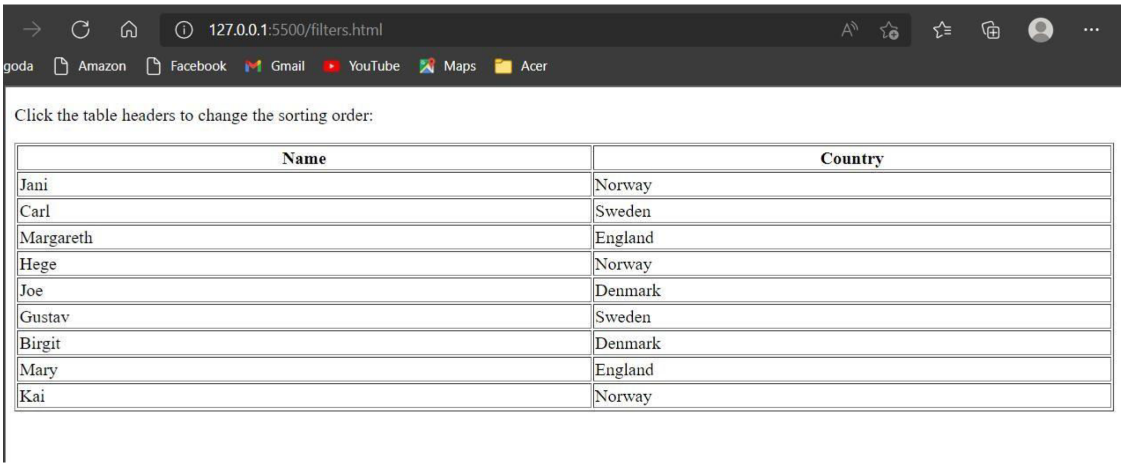
By adding the `ng-click` directive on the table headers, we can run a function that changes the sorting order of the array:

Example



```
<p>Click the table headers to change the sorting order:</p>

<div ng-app="myApp" ng-controller="namesCtrl">

<table border="1" width="100%">
<tr>
<th ng-click="orderByMe('name')">Name</th>
<th ng-click="orderByMe('country')">Country</th>
</tr>
<tr ng-repeat="x in names | orderBy:myOrderBy">
<td>{{x.name}}</td>
<td>{{x.country}}</td>
</tr>
</table>

</div>

<script>
angular.module('myApp', []).controller('namesCtrl', function($scope) {
    $scope.names = [
        {name:'Jani',country:'Norway'},
        {name:'Carl',country:'Sweden'},
        {name:'Margareth',country:'England'},
        {name:'Hege',country:'Norway'},
        {name:'Joe',country:'Denmark'},
        {name:'Gustav',country:'Sweden'},
        {name:'Birgit',country:'Denmark'},
        {name:'Mary',country:'England'},
        {name:'Kai',country:'Norway'}
        ];
    $scope.orderByMe = function(x) {
        $scope.myOrderBy = x;
    }
});
</script>
```

OUTPUT:



Click the table headers to change the sorting order:

Name	Country
Jani	Norway
Carl	Sweden
Margareth	England
Hege	Norway
Joe	Denmark
Gustav	Sweden
Birgit	Denmark
Mary	England
Kai	Norway

Custom Filters

You can make your own filters by registering a new filter factory function with your module:

Example

Make a custom filter called "myFormat":

```
<ul ng-app="myApp" ng-controller="namesCtrl">
  <li ng-repeat="x in names">
    {{x | myFormat}}

  </li>
</ul>

<script>


var app = angular.module('myApp', []);
app.filter('myFormat', function() {
  return function(x) { var
    i, c, txt = "";
    for (i = 0; i < x.length; i++) { c
      = x[i];
      if (i % 2 == 0) {
        c = c.toUpperCase();
      }
      txt += c;
    }
    return txt;
  };
});
app.controller('namesCtrl', function($scope) {
  $scope.names =
['Jani', 'Carl', 'Margareth', 'Hege', 'Joe', 'Gustav', 'Birgit', 'Mary', 'Kai'];
});


</script>
```

The myFormat filter will format every other character to uppercase.

Forms

AngularJS Forms

Forms in AngularJS provides data-binding and validation of input controls.

Input Controls

Input controls are the HTML input elements:

- input elements
- select elements
- button elements
- textarea elements

Data-Binding

Input controls provides data-binding by using the ng-model directive.

```
<input type="text" ng-model="firstname">
```

The application does now have a property named firstname.

The ng-model directive binds the input controller to the rest of your application. The property

firstname, can be referred to in a controller:

Example

```
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
  $scope.firstname = "John";
});
</script>
```

It can also be referred to elsewhere in the application:

Example

```
<form>
  First Name: <input type="text" ng-model="firstname">
</form>

<h1>You entered: {{firstname}}</h1>
```

Checkbox

A checkbox has the value true or false. Apply the ng-model directive to a checkbox, and use its value in your application.

Example

Show the header if the checkbox is checked:

```
 <form>
  Check to show a header:
  <input type="checkbox" ng-model="myVar">
</form>

<h1 ng-show="myVar">My Header</h1>
```

Radiobuttons

Bind radio buttons to your application with the ng-model directive.

Radio buttons with the same ng-model can have different values, but only the selected one will be used.

Example

Display some text, based on the value of the selected radio button:

```
<form>
  Pick a topic:
  <input type="radio" ng-model="myVar" value="dogs">Dogs
  <input type="radio" ng-model="myVar" value="tuts">Tutorials
  <input type="radio" ng-model="myVar" value="cars">Cars
</form>
```

The value of myVar will be either dogs, tuts, or cars.

Selectbox

Bind select boxes to your application with the ng-model directive.

The property defined in the ng-model attribute will have the value of the selected option in the selectbox.

Example

Display some text, based on the value of the selected option:

```
<form>
  Select a topic:
  <select ng-model="myVar">
    <option value="">
    <option value="dogs">Dogs
    <option value="tuts">Tutorials
    <option value="cars">Cars
  </select>
</form>
```

The value of myVar will be either dogs, tuts, or cars.

An AngularJS Form Example

First Name:

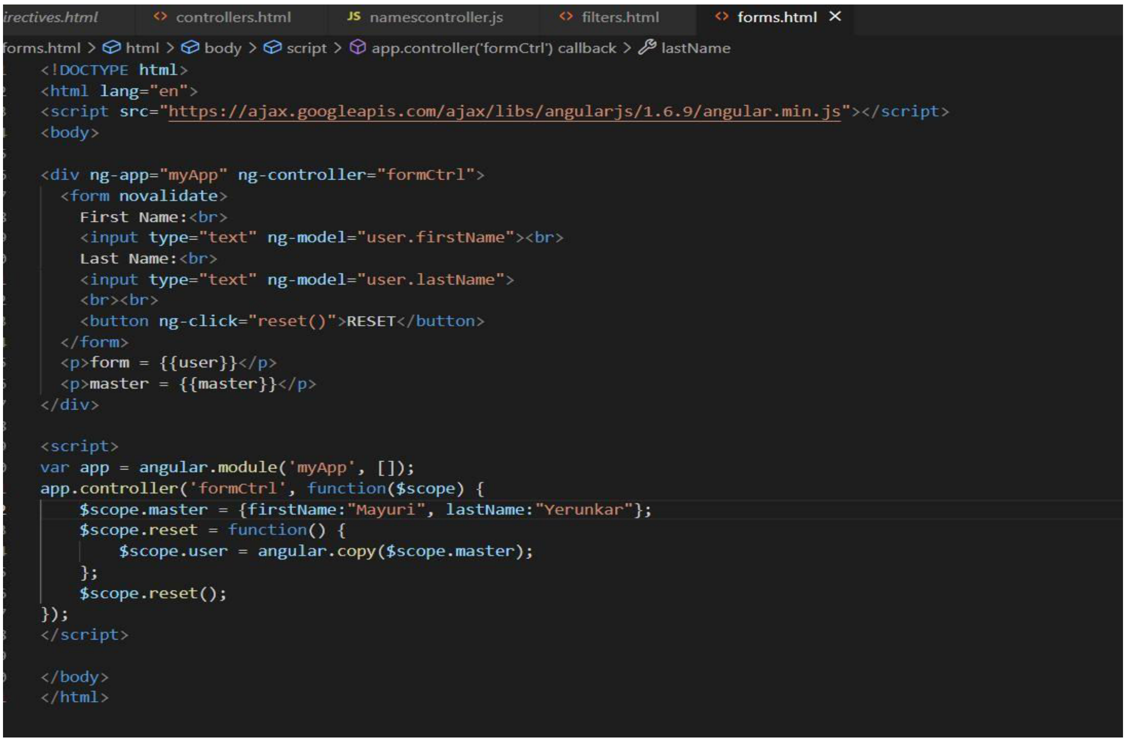
Last Name:

RESET

form = {"firstName":"John","lastName":"Doe"} master =

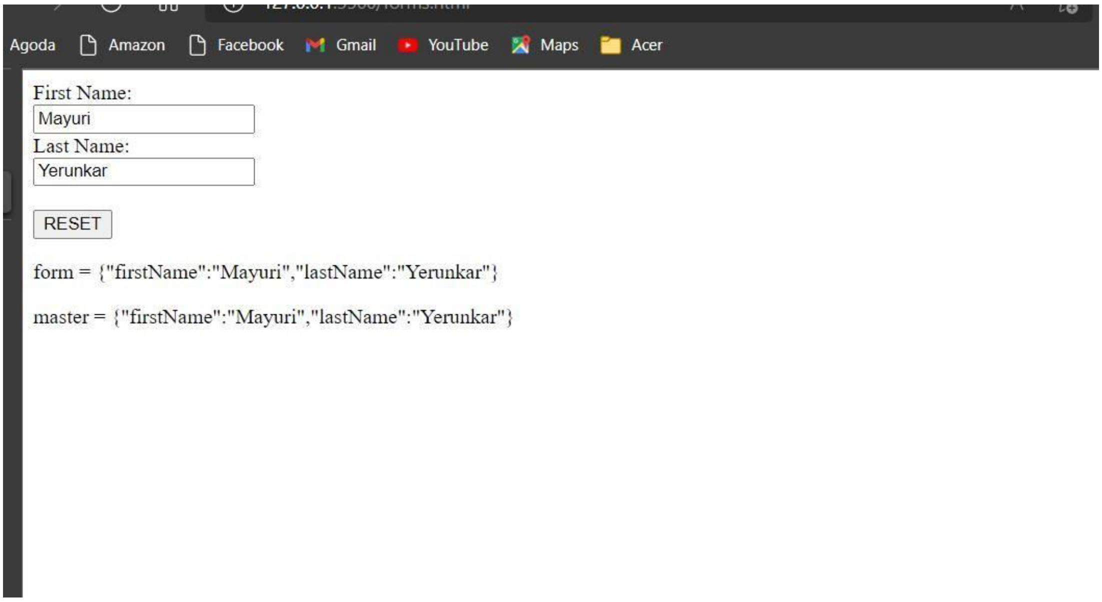
{"firstName":"John","lastName":"Doe"}

Application Code


irectives.html
controllers.html
namescontroller.js
filters.html
forms.html
forms.html > html > body > script > app.controller('formCtrl') callback > lastName

```
<!DOCTYPE html>
<html lang="en">
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="formCtrl">
  <form novalidate>
    First Name:<br>
    <input type="text" ng-model="user.firstName"><br>
    Last Name:<br>
    <input type="text" ng-model="user.lastName">
    <br><br>
    <button ng-click="reset()">RESET</button>
  </form>
  <p>form = {{user}}</p>
  <p>master = {{master}}</p>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
    $scope.master = {firstName:"Mayuri", lastName:"Yerunkar"};
    $scope.reset = function() {
        $scope.user = angular.copy($scope.master);
    };
    $scope.reset();
});
</script>

</body>
</html>
```

OUTPUT:



First Name:
Mayuri
Last Name:
Yerunkar

RESET

form = {"firstName":"Mayuri","lastName":"Yerunkar"}

master = {"firstName":"Mayuri","lastName":"Yerunkar"}

The **novalidate** attribute is new in HTML5. It disables any default browser validation.

Example Explained

The **ng-app** directive defines the AngularJS application.

The **ng-controller** directive defines the application controller.

The **ng-model** directive binds two input elements to the **user** object in the model.

The **formCtrl** controller sets initial values to the **master** object, and defines the **reset()** method.

The **reset()** method sets the **user** object equal to the **master** object.

The **ng-click** directive invokes the **reset()** method, only if the button is clicked.

The novalidate attribute is not needed for this application, but normally you will use it in AngularJS forms, to override standard HTML5 validation.

Modules and Services

AngularJS Modules

An AngularJS module defines an application.

The module is a container for the different parts of an application. The module is

a container for the application controllers.

Controllers always belong to a module.

Creating a Module

A module is created by using the AngularJS function `angular.module`

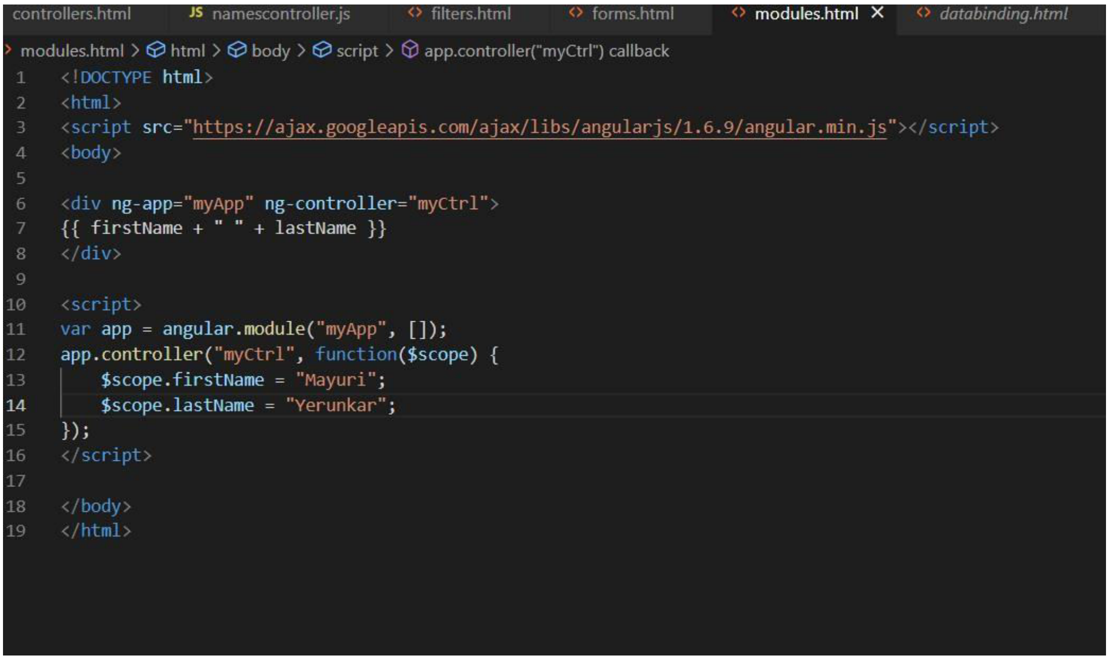
```
<div ng-app="myApp">...</div>

<script>

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

</script>
```

The "myApp" parameter refers to an HTML element in which the application will run. Now you can add controllers, directives, filters, and more, to your AngularJSapplication.

Adding a Controller

Add a controller to your application, and refer to the controller with the ng- controller directive:

Example



```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/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 = "Mayuri";
    $scope.lastName = "Yerunkar";
});
</script>

</body>
</html>
```

OUTPUT:

Adding a Directive

AngularJS has a set of built-in directives which you can use to add functionality to your application.

In addition you can use the module to add your own directives to your applications:

Example

```
<div ng-app="myApp" w3-test-directive></div>

<script>
var app = angular.module("myApp", []);

app.directive("w3TestDirective", function() {
  return {
    template : "I was made in a directive constructor!"
  };
});
</script>
```

Modules and Controllers in Files

It is common in AngularJS applications to put the module and the controllers in JavaScript files.

In this example, "myApp.js" contains an application module definition, while "myCtrl.js" contains the controller:

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">

{{ firstName + " " + lastName }}
</div>

<script src="myApp.js"></script>

<script src="myCtrl.js"></script>


</body>
</html>
```

myApp.js

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

The [] parameter in the module definition can be used to define dependent modules.

Without the [] parameter, you are not creating a new module, but retrieving an existing one.

myCtrl.js

```
app.controller("myCtrl", function($scope) {
  $scope.firstName = "John";
  $scope.lastName= "Doe";
});
```

Functions can Pollute the Global Namespace

Global functions should be avoided in JavaScript. They can easily be overwritten or destroyed by other scripts.

AngularJS modules reduces this problem, by keeping all functions local to the module.

When to Load the Library

While it is common in HTML applications to place scripts at the end of the <body> element, it is recommended that you load the AngularJS library either in the <head> or at the start of the <body>.

This is because calls to angular.module can only be compiled after the library has been loaded.

Example

```
<!DOCTYPE html>
<html>
<body>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
  $scope.firstName = "John";
  $scope.lastName = "Doe";
});
</script>

</body>
</html>
```

AngularJS Services

In AngularJS you can make your own service, or use one of the many built-in services.

What is a Service?

In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.

AngularJS has about 30 built-in services. One of them is the `$location` service.

The `$location` service has methods which return information about the location of the current web page:

Example

Use the `$location` service in a controller:

```
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $location) {
    $scope.myUrl = $location.absUrl();
});
```

Why use Services?

For many services, like the `$location` service, it seems like you could use objects that are already in the DOM, like the `window.location` object, and you could, but it would have some limitations, at least for your AngularJS application.

AngularJS constantly supervises your application, and for it to handle changes and events properly, AngularJS prefers that you use the `$location` service instead of the `window.location` object.

The $http Service

The `$http` service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

Example

Use the `$http` service to request data from the server:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm").then(function (response) {
    $scope.myWelcome = response.data;
  });
});
```

This example demonstrates a very simple use of the `$http` service. Learn more about the `$http` service in the AngularJS Http Tutorial.

The $timeout Service

The `$timeout` service is AngularJS' version of the `window.setTimeout` function.

Example

Display a new message after two seconds:

```
var app = angular.module('myApp', []); app.controller('myCtrl',
function($scope, $timeout) {
  $scope.myHeader = "Hello World!";
  $timeout(function () {
    $scope.myHeader = "How are you today?";
  }, 2000);
});
```

The $interval Service

The `$interval` service is AngularJS' version of the `window.setInterval` function.

Example

Display the time every second:

```
var app = angular.module('myApp', []); app.controller('myCtrl',
function($scope, $interval) {
  $scope.theTime = new Date().toLocaleTimeString();
  $interval(function () {
    $scope.theTime = new Date().toLocaleTimeString();
  }, 1000);
});
```

Create Your Own Service

To create your own service, connect your service to the module:

Create a service named `hexafy`:

```
app.service('hexafy', function() {
  this.myFunc = function (x) {
    return x.toString(16);
  }
});
```

To use your custom made service, add it as a dependency when defining the controller:

Example

Use the custom made service named hexafy to convert a number into a hexadecimal number:

```
app.controller('myCtrl', function($scope, hexafy) {
  $scope.hex = hexafy.myFunc(255);
});
```

Use a Custom Service Inside a Filter

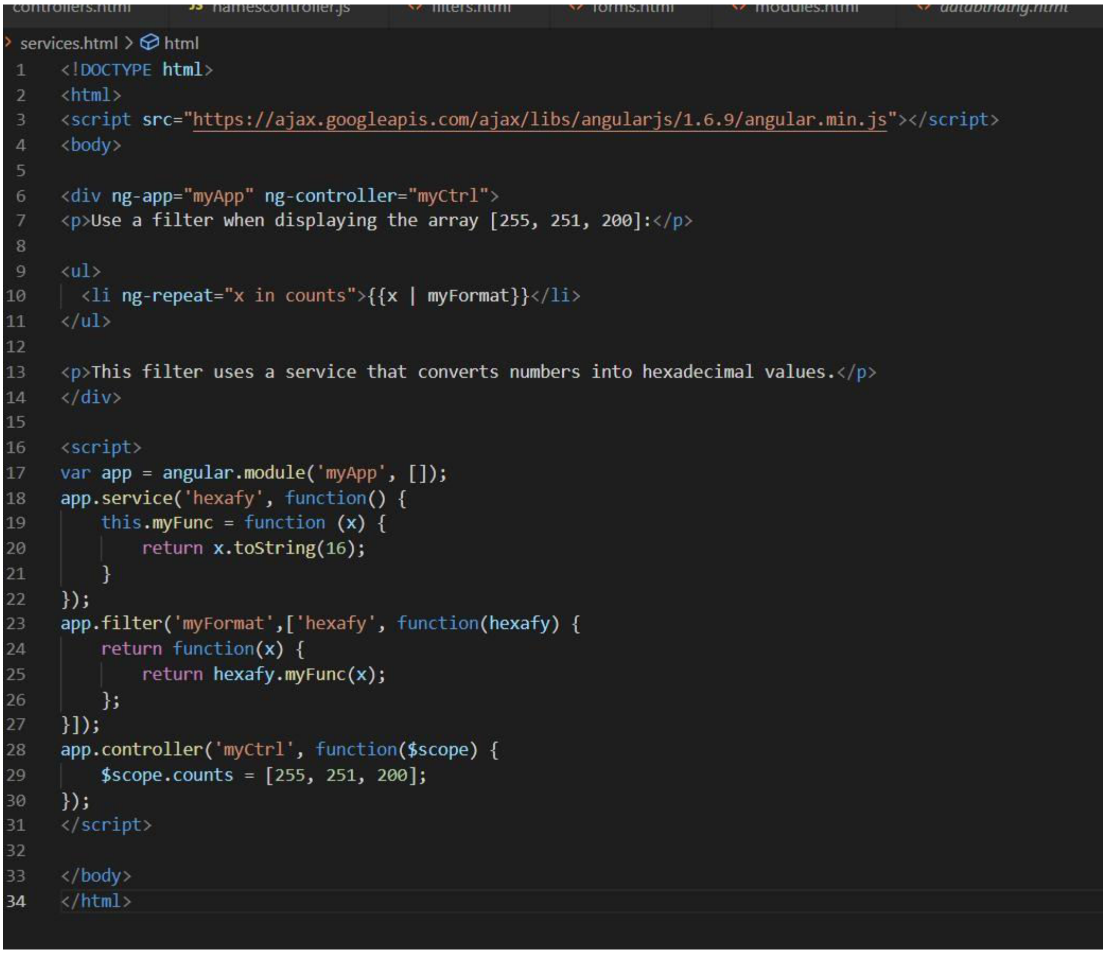
Once you have created a service, and connected it to your application, you can use the service in any controller, directive, filter, or even inside other services.

To use the service inside a filter, add it as a dependency when defining the filter:

The service hexafy used in the filter myFormat:

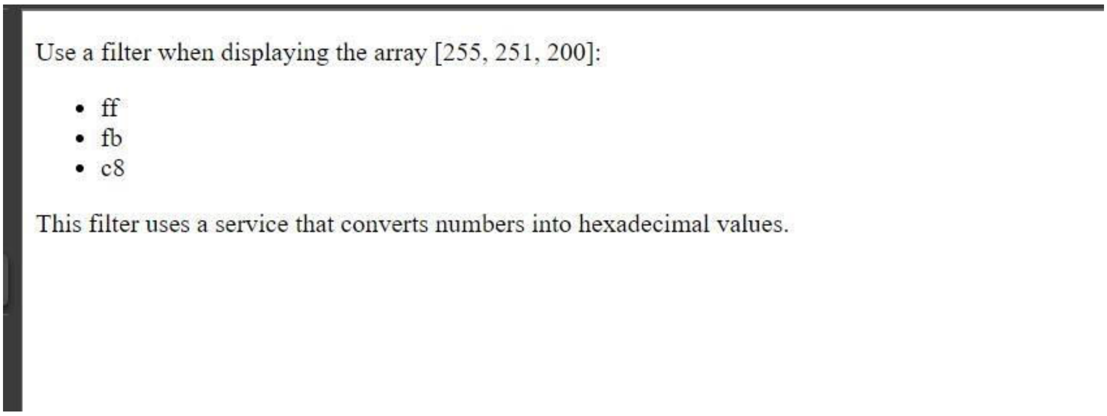
```
app.filter('myFormat',['hexafy', function(hexafy) {
  return function(x) {
    return hexafy.myFunc(x);

  };
}]);
```

You can use the filter when displaying values from an object, or an array:



```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
<p>Use a filter when displaying the array [255, 251, 200]:</p>

<ul>
  <li ng-repeat="x in counts">{{x | myFormat}}</li>
</ul>

<p>This filter uses a service that converts numbers into hexadecimal values.</p>
</div>

<script>
var app = angular.module('myApp', []);
app.service('hexafy', function() {
    this.myFunc = function (x) {
        return x.toString(16);
    }
});
app.filter('myFormat',['hexafy', function(hexafy) {
    return function(x) {
        return hexafy.myFunc(x);
    };
}]);
app.controller('myCtrl', function($scope) {
    $scope.counts = [255, 251, 200];
});
</script>

</body>
</html>
```

OUTPUT:



Use a filter when displaying the array [255, 251, 200]:

- ff
- fb
- c8

This filter uses a service that converts numbers into hexadecimal values.