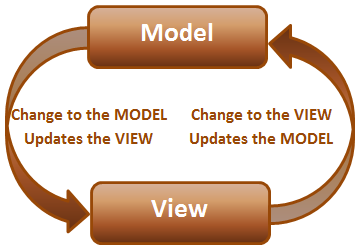
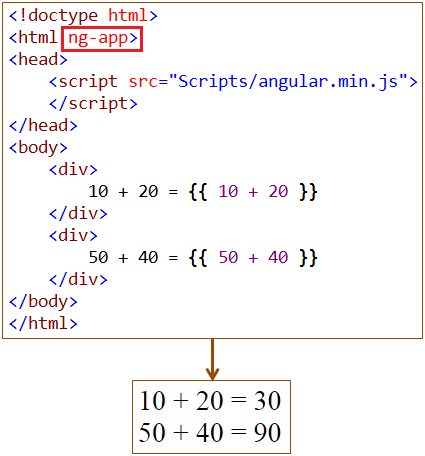
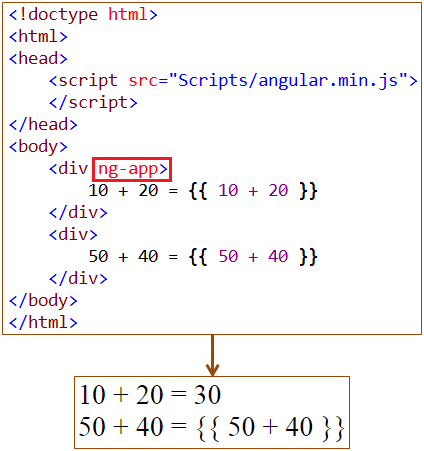
**What is AngularJS**  
AngularJS is a JavaScript framework that helps build applications that run in a web browser.    
  
**Who developed AngularJS**  
Google is the company that developed AngularJS. AngularJS is an open source project, which means it can be be freely used, changed, and shared by anyone.  
  
AngularJS is an excellent framework for building both Single Page Applications (SPA) and Line of Business Applications. Many companies are using Angular today, and there are many public facing web sites that are built with angular.  
  
There is a website, [https://www.madewithangular.com](https://www.madewithangular.com/), that has the list of web sites that are built using AngularJS. Within this list you can find many popular websites.  
  
**What are the benefits of using AngularJS**  
**1. Dependency Injection :** Dependency Injection is something AngularJS does quite well. If you are new to Dependency Injection, don't worry, we will discuss it in detail with examples in a later video.  
  
**2. Two Way Data-Binding :**One of the most useful feature in AngularJS is the Two Way Data-Binding. The Two Way Data-Binding, keeps the model and the view in sync at all times, that is a change in the model updates the view and a change in the view updates the model.   
   
  
**3. Testing :**Testing is an area where Angular really shines. Angular is designed with testing in mind right from the start. Angular makes it very easy to test any of it's components through both unit testing and end to end testing. So there's really no excuse for not testing any of your angular application code.  
  
**4. Model View Controller :**With angular it is very easy to develop applications in a clean MVC way. All you have to do is split your application code into MVC components. The rest, that is managing those components and connecting them together is done by angular.

**5. Many more benefits**like controlling the behaviour of DOM elements using **directives**and the flexibility that **angular filters**provide.  
  
We will discuss directives, filters, Modules, Routes etc with examples in our upcoming videos in this series.  
  
To build angular applications you only need one script file and that is angular.js.  
  
To get the script file visit [https://angularjs.org](https://angularjs.org/). From here   
1. You can download the angular script file   
2. CDN link - We discussed the benefits of using CDN in [Part 3](https://www.youtube.com/watch?v=hcpskGpuKaE&list=PL6n9fhu94yhVDV697uvHpavA3K_eWGQap&index=3) of [jQuery tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhVDV697uvHpavA3K_eWGQap).  
3. Various resources to learn angular - Here you will find videos, Free courses, Tutorials and Case Studies. You will also find API reference which is extremeley useful.  
  
**To get started with angular**  
1. Add a reference to the angular script  
2. Include ng-app attribute   
  
**What is ng-app**  
In angular, ng-app is called a directive. There are many directives in angular. You can find the complete list of directives on https://angularjs.org. The ng prefix in the directive stands for angular. The ng-app directive is a starting point of AngularJS Application. Angular framework will first check for ng-app directive in an HTML page after the entire page is loaded. If ng-app directive is found, angular bootstraps itself and starts to manage the section of the page that has the ng-app directive.   
  
So the obvious next question is, **where to place the ng-app directive on the page**  
It should be placed at the root of the HTML document, that is at the <html> tag level or at the <body> tag level, so that angular can control the entire page.   
  
However, there is nothing stopping you from placing it on any other HTML element with in the page. When you do this only that element and it's children are managed by angular.   
  
Double curly braces are called binding expressions in angular.

**Example :**In the example below, the **ng-app**directive is placed at the <html> tag level. So the binding expressions in both the div elements are evaluated and displayed as expected.   
   
  
**Example :**In the example below, the **ng-app** directive is placed on one of the <div> element. So the binding expressions in the <div> element that has the ng-app directive is evaluated but not the binding expression in the other <div> element.  
   
  
**All the following are valid expressions in angular**  
**{{** 1 == 1 **}}** - Evaluates to true  
**{{** { name: 'David', age : '30' }.name **}}** - Returns the name property value  
**{{** ['Mark', 'David', 'Sara'][2] **}}** - Returns the 2nd element from the array

**What is a module in AngularJS**  
A module is a container for different parts of your application i.e controllers, services, directives, filters, etc. In this video we will also discuss controllers. We will discuss services, filters and directives in a later video.   
  
**Why is a module required**  
You can think of a module as a Main() method in other types of applications. For example, a Dot Net console application has a Main() method which is the entry point into the application and it wires together the different parts of the application.  
  
Modules are the angular application's equivalent of the Main() method. Modules declaratively specify how the angular application should be bootstrapped.   
  
There are several benefits of the modular approach. It may be difficult to comprehend all those benefits right now, so we will defer the discussion of the benefits to a later video.  
  
**How to create a module**  
Creating a module in angular is staright forward. Use the angular object's module() method to create a module. The angular object is provided by angular script. The following example, creates a module.   
  
var myApp = angular.module("myModule", [])  
  
The first parameter specifies the name of the module.   
The second parameter specifies the dependencies for this module  
  
A module can depend on other modules. We will discuss an example of module dependencies in a later video. Right now, the module that we are creating is not dependent on any other external modules, so I am passing an empty array as the value for the second parameter.  
  
**What is a controller in angular**  
In angular a controller is a JavaScript function. The job of the controller is to build a model for the view to display. The model is the data. In a real world application, the controller may call into a service to retrieve data from the database.  
  
**How to create a controller in angular**  
Simple, create a JavaScript function and assign it to a variable  
  
var myController = function ($scope) {  
    $scope.message = "AngularJS Tutorial";  
}  
  
**What is $scope**  
$scope is a parameter that is passed to the controller function by angular framework. We attach the model to the $scope object, which will then be available in the view. With in the view, we can retrieve the data from the scope object and display.  
  
**How to register the controller with the module**  
Use module object's controller function to register the controller with the module  
  
myApp.controller("myController", myController);  
  
**Here is the complete code**

//Create the module

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

//Create the controller

var myController = function ($scope) {

    $scope.message = "AngularJS Tutorial";

}

// Register the controller with the module

myApp.controller("myController", myController);

**The above code can also be written as shown below**

//Create the module

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

// Creating the controller and registering with the module all done in one line.

myApp.controller("myController", function ($scope) {

    $scope.message = "AngularJS Tutorial";

});

**How to use the module that we created to bootstrap the angular application**  
Associate the module name with ng-app directive  
ng-app="myModule"  
  
Similarly associate the controller using ng-controller directive  
ng-controller="myController"  
  
**Here is the complete HTML**

<!doctype html>

<html ng-app="myModule">

<head>

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

    <script src="Scripts/Script.js"></script>

</head>

<body>

    <div ng-controller="myController">

**{{** message **}}**

    </div>

</body>

</html>

**Here is the complete JavaScript**

/// <reference path="angular.min.js" />

//Create module

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

// Register controller with the module

myApp.controller("myController", function ($scope) {

    $scope.message = "AngularJS Tutorial";

});

**The job of the controller is to build a model for the view**. The controller does this by attaching the model to the scope. The scope is not the model, it's the data that you attach to the scope is the model.   
  
In the following example, $scope is not the model. The message property that we have attached to the scope is the model.  
  
myApp.controller("myController", function ($scope) {  
    $scope.message = "AngularJS Tutorial";  
});  
  
The view will then use the data-binding expression to retrieve the model from the scope. This means the controller is not manipulating the DOM directly, thus keeping that clean separation between the model, view and the controller. So when you are developing controllers, make sure, you are not breaking that clean separation between the model, view and the controllers. The controller should only be used for setting up the $scope object and adding behavior it. We will discuss, when and why should we add behvior to the scope object in a later video.   
  
In the example above, **message is a simple property**. You can also attach a complex object to the scope. In the example below, we have an employee object which is a complex object with 3 properties attached to the view.   
  
myApp.controller("myController", function ($scope) {  
  
    var employee = {  
        firstName: 'Mark',  
        lastName: 'Hastings',  
        gender: 'Male'  
    };  
  
    $scope.employee = employee;  
});  
  
With in the view, we can then retrieve the employee properties and display them in the view as shown below

<div ng-controller="myController">

    <div>First Name : **{{** employee.firstName **}}**</div>

    <div>Last Name : **{{** employee.lastName **}}**</div>

    <div>Gender : **{{** employee.gender**}}**</div>

</div>

**What happens if the controller name is misspelled**  
When the controller name is misspelled, 2 things happen  
1. An error is raised. To see the error, use browser developer tools  
2. The binding expressions in the view that are in the scope of the controller will not be evaluated  
  
If you are using the minified version of the AngularJS script, the error messages may not be readable. To get readable error message  
1. In the developer tools, click on the link next to the error. This will lead you to a page, where you can see a much clean error message without all the encoding symbols.  
  
2. Another option you have is, if you are in the development environment, you may use the non-minified version of the AngularJS script, which produces readable error message.

**What happens if a property name in the binding expression is misspelled**  
Expression evaluation in angular is forgiving, meaning if you misspell a property name in the binding expression, angular will not report any error. It will simply return null or undefined.  
  
**How to create module, controller and register the controller with the module, all in one line**  
Use the method chaining mechanism as shown below  
  
var myApp = angular  
    .module("myModule", [])  
    .controller("myController", function ($scope) {  
        var employee = {  
            firstName: 'Mark',  
            lastName: 'Hastings',  
            gender: 'Male'  
        };  
        $scope.employee = employee;  
    });

AngularJS ng-src directive:

Let us understand this with an example : We want to display the name of the country, capital and it's flag.

**AngularJS Code :** The controller builds the country model. The flag property of the country object has the path of the image.

var myApp = angular

                .module("myModule", [])

                .controller("myController", function ($scope) {

                    var country = {

                        name: "United States of America",

                        capital: "Washington, D.C.",

                        flag: "/Images/usa-flag.png"

                    };

                    $scope.country = country;

                });

**HTML Code :** In the view we are binding country.flag property with the src attribute of the image element.

<!doctype html>

<html ng-app="myModule">

<head>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

</head>

<body>

    <div ng-controller="myController">

        <div>

            Country : {{ country.name }}

        </div>

        <div>

            Capital : {{ country.capital }}

        </div>

        <div>

            <img src="{{country.flag}}"

                 alt="{{ country.name + ' Flag' }}"

                 style="height:100px; width:200px" />

        </div>

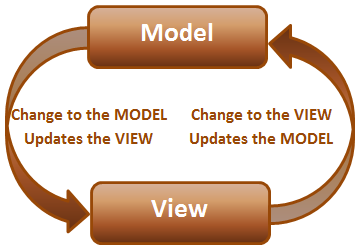
    </div>

</body>

</html>

When you view the page in the browser, the country details and the flag are displayed as expected. The problem with the img src attribute is that we get a 404 error. To see the error, launch the developer tools.  
  
**Let's now understand the reason for the 404 error**  
As soon as the DOM is parsed, an attempt is made is to fetch the image from the server. At this point, AngularJS binding expression that is specified with the src attribute is not evaluated. Hence 404 (not found) error.  
  
**To fix the 404 error use the ng-src directive :** ng-src attribute ensures that a request is issued only after AngularJS has evaluated the binding expression

### Two way databinding in AngularJS:

**Two way data binding in AngularJS**. Along the way we also discuss one of the very useful directive in angular **ng-model**.   
  
  
  
   
When the model changes the view is automatically updated. This is achieved using the data binding expression in the view.   
  
**Script.js :**The code in the controller attaches message property to the scope which is the model.  

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                $scope.message = "Hello Angular"

            });

**HtmlPage1.html :**Whenever the message property value changes, the data binding expression in the view updates the view automatically. 

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        {{ message }}

    </div>

</body>

</html>

**How about the other way round**. How to keep the model up to date when the view changes. That's exactly is the purpose of ng-model directive.   
  
In the html below, notice the input element is decorated with **ng-model** directive. This ensures that whenever the value in the textbox is changed, angular will automatically update the message property of the $scope object. This means the ng-model directive automatically takes the form values and updates the model. The binding expression does the opposite, i.e whenever the model changes the view is automatically updated.    
  
Because of the two way data binding provided by angular, as you type in the textbox, the value is immediately displayed on the view just below the textbox. This two way binding feature provided by angular, eliminates the need to write any custom code to move data from the model to the view or from the view to the model. 

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <input type="text" placeholder="Type your message here" ng-model="message" />

        <br /><br />

        {{ message }}

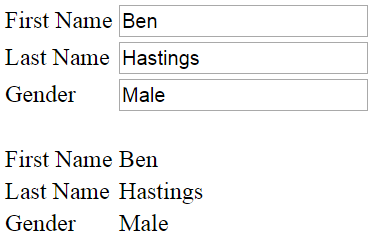
    </div>

</body>

</html>

**ng-model directive can be used with the following 3 html elements**

* input
* select
* textarea

**Two way binding example with complex object :**   
  
   
  
**Script.js code :** In the following example, the model is employee which is a complex object with properties like firstName, lastName and gender. 

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                var employee = {

                    firstName: "Ben",

                    lastName: "Hastings",

                    gender: "Male"

                };

                $scope.employee = employee;

            });

**HtmlPage1.html :**When the view loads, the model data is display in both, the textbox and td elements on the page. As you start to type in any of the textboxes, the respective employee model object property is updated, and the change is immediately reflected in the respective td element. 

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <tr>

                <td>

                    First Name

                </td>

                <td>

                    <input type="text" placeholder="Firstname"

                           ng-model="employee.firstName" />

                </td>

            </tr>

            <tr>

                <td>

                    Last Name

                </td>

                <td>

                    <input type="text" placeholder="Lastname"

                           ng-model="employee.lastName" />

                </td>

            </tr>

            <tr>

                <td>

                    Gender

                </td>

                <td>

                    <input type="text" placeholder="Gender"

                           ng-model="employee.gender" />

                </td>

            </tr>

        </table>

        <br />

        <table>

            <tr>

                <td>

                    First Name

                </td>

                <td>

                    {{ employee.firstName }}

                </td>

            </tr>

            <tr>

                <td>

                    Last Name

                </td>

                <td>

                    {{ employee.lastName }}

                </td>

            </tr>

            <tr>

                <td>

                    Gender

                </td>

                <td>

                    {{ employee.gender }}

                </td>

            </tr>

        </table>

    </div>

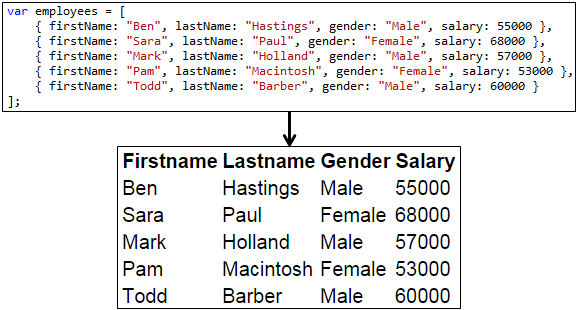
</body>

</html>

### AngularJS ng-repeat directive:

**ng-repeat** is similar to foreach loop in C#.    
  
Let us understand this with an example. Here is what we want to do.  
1. For each employee we have in the employees array we want a table row. With in each cell of the table row we to display employee

* Firstname
* Lastname
* Gender
* Salary

  
  
This can be achieved very easily using **ng-repeat directive**   
  
**Script.js :** The controll function builds the model for the view. The model employees has the list of all employees. 

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                var employees = [

                    { firstName: "Ben", lastName: "Hastings", gender: "Male", salary: 55000 },

                    { firstName: "Sara", lastName: "Paul", gender: "Female", salary: 68000 },

                    { firstName: "Mark", lastName: "Holland", gender: "Male", salary: 57000 },

                    { firstName: "Pam", lastName: "Macintosh", gender: "Female", salary: 53000 },

                    { firstName: "Todd", lastName: "Barber", gender: "Male", salary: 60000 }

                ];

                $scope.employees = employees;

            });

**HtmlPage1.html :** In the view, we are using ng-repeat directive which loops thru each employee in employees array. For each employee, we a get a table row, and in each table cell of the table row, the respective employee details (Firstname, Lastname, Gender, Salary) are retrieved using the angular binding expression. 

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <thead>

                <tr>

                    <th>Firstname</th>

                    <th>Lastname</th>

                    <th>Gender</th>

                    <th>Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> {{ employee.firstName }} </td>

                    <td> {{ employee.lastName }} </td>

                    <td> {{ employee.gender }} </td>

                    <td> {{ employee.salary }} </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Nested ng-repeat example :**The model contains an array of countries, and each country has an array of cities. The view must display cities nested under their respective country.

   
  
**Script.js :** The model is an array of countries. Each country contains an array of cities.

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                var countries = [

                    {

                        name: "UK",

                        cities: [

                            { name: "London" },

                            { name: "Birmingham" },

                            { name: "Manchester" }

                        ]

                    },

                    {

                        name: "USA",

                        cities: [

                            { name: "Los Angeles" },

                            { name: "Chicago" },

                            { name: "Houston" }

                        ]

                    },

                    {

                        name: "India",

                        cities: [

                            { name: "Hyderabad" },

                            { name: "Chennai" },

                            { name: "Mumbai" }

                        ]

                    }

                ];

                $scope.countries = countries;

            });

**HtmlPage1.html :** Notice that we are using two ng-repeat directives in the view, one nested inside the other. The outer ng-repeat directive loops thru each country in the model. The inner ng-repeat directive loops thru each city of a given country.

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <ul ng-repeat="country in countries">

            <li>

                {{country.name}}

                <ul>

                    <li ng-repeat="city in country.cities">

                        {{city.name}}

                    </li>

                </ul>

            </li>

        </ul>

    </div>

</body>

</html>

**Finding the index of an item in the collection :**

* To find the index of an item in the collection use $index property
* To get the index of the parent element
  + Use $parent.$index or
  + Use ng-init="parentIndex = $index"

The following example, shows how to retrive the index of the elements from a nested collection

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <ul ng-repeat="country in countries" ng-init="parentIndex = $index">

            <li>

                {{country.name}} - Index = {{ $index }}

                <ul>

                    <li ng-repeat="city in country.cities">

                        {{city.name}} - Parent Index = {{ parentIndex }}, Index = {{ $index }}

                    </li>

                </ul>

            </li>

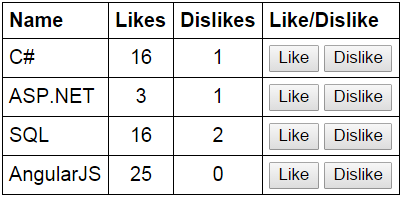
        </ul>

    </div>

</body>

</html>

### Handling events in AngularJS:

Let us understand with an example. Here is what we want to do.  
  


**1.** Display the list of technologies in a table  
**2.** Provide the ability to like and dislike a technology  
**3.** Increment the likes and dislikes when the respective buttons are clicked

**Script.js :**In the controller function we have 2 methods to increment likes and dislikes. Both the functions have the technology object that we want to like or dislike as a parameter. 

var app = angular

            .module("myModule", [])

            .controller("myController", function ($scope) {

                var technologies = [

                    { name: "C#", likes: 0, dislikes: 0 },

                    { name: "ASP.NET", likes: 0, dislikes: 0 },

                    { name: "SQL", likes: 0, dislikes: 0 },

                    { name: "AngularJS", likes: 0, dislikes: 0 }

                ];

                $scope.technologies = technologies;

                $scope.incrementLikes = function (technology) {

                    technology.likes++;

                };

                $scope.incrementDislikes = function (technology) {

                    technology.dislikes++;

                };

            });

**HtmlPage1.html :** Notice in the html below, we are associating **incrementLikes()** and **incrementDislikes()** functions with the respective button. When any of these buttons are clicked, the corresponsing technology object is automatically passed to the function, and the likes or dislikes property is incremented depending on which button is clicked.

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Likes</th>

                    <th>Dislikes</th>

                    <th>Like/Dislike</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="technology in technologies">

                    <td> **{{** technology.name **}}** </td>

                    <td style="text-align:center"> **{{** technology.likes **}}** </td>

                    <td style="text-align:center"> **{{** technology.dislikes **}}** </td>

                    <td>

                        <input type="button" ng-click="incrementLikes(technology)" value="Like"/>

                        <input type="button" ng-click="incrementDislikes(technology)"value="Dislike" />

                    </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Styles.css :**Styles for table, td and th elements

table {

    border-collapse: collapse;

    font-family:Arial;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

    border: 1px solid black;

    padding: 5px;

    text-align: left;

### }

### AngularJS filters:

**Filters in angular can do 3 different things**  
1. Format data  
2. Sort data  
3. Filter data   
  
Filters can be used with a binding expression or a directive  
  
To apply a filter use pipe (|) character  
  
**Syntax :** **{{** expression | filterName:parameter **}}**

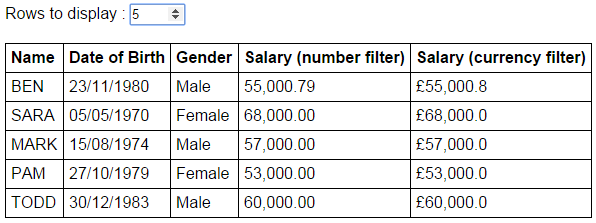
**Angular filters for formatting data**

|  |  |
| --- | --- |
| **Filter** | **Description** |
| lowercase | Formats all characters to lowercase |
| uppercase | Formats all characters to uppercase |
| number | Formats a number as text. Includes comma as thousands separator and the number of decimal places can be specified |
| currency | Formats a number as a currency. $ is default. Custom currency and decimal places can be specified |
| date | Formats date to a string based on the requested format |

**Angular Date formats**

|  |  |
| --- | --- |
| **Format** | **Result** |
| yyyy | 4 digit year. Exampe 1998 |
| yy | 2 digit year. Example 1998 => 98 |
| MMMM | January - December |
| MMM | Jan - Dec |
| MM | 01 - 12 |
| M | 1 - 12 (No leading ZEROS) |
| dd | 01 - 31 |
| d | 1 - 31 (No leading ZEROS) |

**Angular date format documentation**  
<https://docs.angularjs.org/api/ng/filter/date>  
  
**limitTo filter :** Can be used to limit the number of rows or characters in a string.  
  
**Syntax :** **{{** expression | limitTo : limit : begin**}}**

**The following example uses all the above filters**  
  
  
**Script.js**

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                {

                    name: "Ben", dateOfBirth: new Date("November 23, 1980"),

                    gender: "Male", salary: 55000.788

                },

                {

                    name: "Sara", dateOfBirth: new Date("May 05, 1970"),

                    gender: "Female", salary: 68000

                },

                {

                    name: "Mark", dateOfBirth: new Date("August 15, 1974"),

                    gender: "Male", salary: 57000

                },

                {

                    name: "Pam", dateOfBirth: new Date("October 27, 1979"),

                    gender: "Female", salary: 53000

                },

                {

                    name: "Todd", dateOfBirth: new Date("December 30, 1983"),

                    gender: "Male", salary: 60000

                }

            ];

            $scope.employees = employees;

            $scope.rowCount = 3;

        });

**HtmlPage1.html** 

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        Rows to display : <input type="number" step="1"

                                 ng-model="rowCount" max="5" min="0" />

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Date of Birth</th>

                    <th>Gender</th>

                    <th>Salary (number filter)</th>

                    <th>Salary (currency filter)</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees | limitTo:rowCount">

                    <td> **{{** employee.name | uppercase **}}** </td>

                    <td> **{{** employee.dateOfBirth | date:"dd/MM/yyyy" **}}** </td>

                    <td> **{{** employee.gender **}}** </td>

                    <td> **{{** employee.salary | number:2 **}}** </td>

                    <td> **{{** employee.salary | currency : "£" : 1 **}}** </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Styles.css** 

body {

    font-family: Arial;

}

table {

    border-collapse: collapse;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

    border: 1px solid black;

    padding: 5px;

    text-align: left;

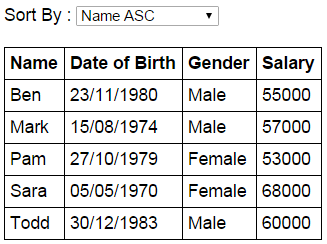
}

### Sorting data in AngularJS:

**To sort the data in Angular**  
**1.** Use orderBy filter  
    **{{** orderBy\_expression | orderBy : expression : reverse**}}**

**Example :** ng-repeat="employee in employees | orderBy:'salary':false"

**2.** To sort in ascending order, set reverse to false  
**3.** To sort in descending order, set reverse to true  
**4.** You can also use + and - to sort in ascending and descending order respectively  
    **Example :**ng-repeat="employee in employees | orderBy:'+salary'"

Let us understand sorting data with an example.  
  
  
The dropdownlist shows the columns and the direction we want to sort  
When a dropdownlist item is selected, the table data should be sorted by the selected column  
  
**Script.js :** The controller function builds the model. Also sortColumn property is added to the $scope object. Notice sortColumn property is initialized to name. This ensures that the data is sorted by name column in ascending order, when the form first loads.

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                {

                    name: "Ben", dateOfBirth: new Date("November 23, 1980"),

                    gender: "Male", salary: 55000

                },

                {

                    name: "Sara", dateOfBirth: new Date("May 05, 1970"),

                    gender: "Female", salary: 68000

                },

                {

                    name: "Mark", dateOfBirth: new Date("August 15, 1974"),

                    gender: "Male", salary: 57000

                },

                {

                    name: "Pam", dateOfBirth: new Date("October 27, 1979"),

                    gender: "Female", salary: 53000

                },

                {

                    name: "Todd", dateOfBirth: new Date("December 30, 1983"),

                    gender: "Male", salary: 60000

                }

            ];

            $scope.employees = employees;

            $scope.sortColumn = "name";

        });

**HtmlPage1.html :** The select element, has the list of columns by which the data should be sorted. + and - symbols control the sort direction. When the form initially loads notice that the data is sorted by name column in ascending order, and name option is automatically selected in the select element. Notice the orderBy filter is using the sortColumn property that is attached to the $scope object. When the selection in the select element changes, the sortColumn property of the $scope object will be updated automatically with the selected value, and in turn the updated value is used by the orderBy filter to sort the data.

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        Sort By :

        <select ng-model="sortColumn">

            <option value="name">Name ASC</option>

            <option value="+dateOfBirth">Date of Birth ASC</option>

            <option value="+gender">Gender ASC</option>

            <option value="-salary">Salary DESC</option>

        </select>

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Date of Birth</th>

                    <th>Gender</th>

                    <th>Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees | orderBy:sortColumn">

                    <td>

**{{** employee.name **}}**

                    </td>

                    <td>

**{{** employee.dateOfBirth | date:"dd/MM/yyyy" **}}**

                    </td>

                    <td>

**{{** employee.gender **}}**

                    </td>

                    <td>

**{{** employee.salary  **}}**

                    </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>  
  
**Styles.css :** CSS styles to make the form look pretty.

body {

    font-family: Arial;

}

table {

    border-collapse: collapse;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

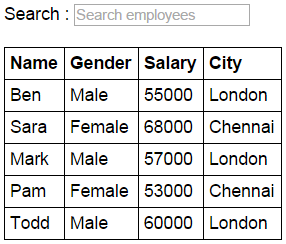
    border: 1px solid black;

    padding: 5px;

    text-align: left;

### }

### Search filter in AngularJS:

**how to implement search in Angular** using search filter.   
  
  
  
   
  
As we type in the search textbox, all the columns in the table must be searched and only the matching rows should be displayed.    
  
**Script.js :**

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: "Male", salary: 55000, city: "London" },

                { name: "Sara", gender: "Female", salary: 68000, city: "Chennai" },

                { name: "Mark", gender: "Male", salary: 57000, city: "London" },

                { name: "Pam", gender: "Female", salary: 53000, city: "Chennai" },

                { name: "Todd", gender: "Male", salary: 60000, city: "London" },

            ];

            $scope.employees = employees;

        });

**HtmlPage1.html :**

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        Search : <input type="text" placeholder="Search employees"

                        ng-model="searchText" />

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>Salary</th>

                    <th>City</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees | filter:searchText">

                    <td> **{{** employee.name **}}** </td>

                    <td> **{{** employee.gender **}}** </td>

                    <td> **{{** employee.salary  **}}** </td>

                    <td> **{{** employee.city  **}}** </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

**Styles.css :**

body {

    font-family: Arial;

}

table {

    border-collapse: collapse;

}

td {

    border: 1px solid black;

    padding: 5px;

}

th {

    border: 1px solid black;

    padding: 5px;

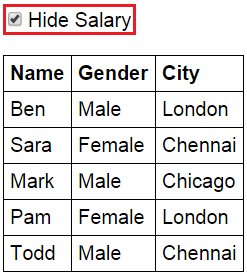
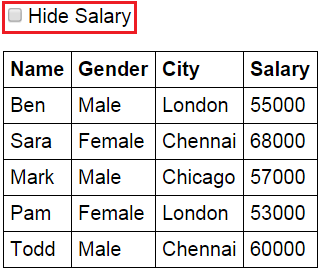
    text-align: left;

}

At the moment, the search is being done across all columns. If you want to search only one specific column, then change **ng-model** directive value on the search textbox as shown below. **With this change only city column is searched.**

<input type="text" ng-model="searchText.city" placeholder="Search employees" />

### ng-hide and ng-show in AngularJS:

ng-hide and ng-show directives are used to control the visibility of the HTML elements. Let us understand this with an example   
  
When Hide Salary checkbox is checked, the Salary column should be hidden.   
   
  
When it is unchecked the Salary column should be unhidden   
   
  
**Script.js :** The controller function builds the model 

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            var employees = [

                { name: "Ben", gender: "Male", city: "London", salary: 55000 },

                { name: "Sara", gender: "Female", city: "Chennai", salary: 68000 },

                { name: "Mark", gender: "Male", city: "Chicago", salary: 57000 },

                { name: "Pam", gender: "Female", city: "London", salary: 53000 },

                { name: "Todd", gender: "Male", city: "Chennai", salary: 60000 }

            ];

            $scope.employees = employees;

        });

**HtmlPage1.html :** Notice **ng-model** directive on the checkbox is set to **hideSalary**. hideSalary variable is then used as the value for ng-hide directive on the th and td elements that displays Salary. When the page is first loaded, hideSalary variable will be undefined which evaluates to false, as a result Salary column will be visible. When the checkbox is checked, hideSalary variable will be attached to the $scope object and true value is stored in it. This value is then used by the ng-hide directive to hide the salary td and it's th element. When the checkbox is unchecked, false value is stored in the hideSalary variable, which is then used by the ng-hide directive to display the Salary column. 

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <input type="checkbox" ng-model="hideSalary" />Hide Salary

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>City</th>

                    <th ng-hide="hideSalary">Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> {{ employee.name }} </td>

                    <td> {{ employee.gender}} </td>

                    <td> {{ employee.city}} </td>

                    <td ng-hide="hideSalary"> {{ employee.salary  }} </td>

                </tr>

            </tbody>

        </table>

    </div>

</body>

</html>

With the above example we can also use **ng-show** directive instead of **ng-hide**directive. For this example to behave the same as before, we will have to negate the value of hideSalary variable using ! operator. 

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <input type="checkbox" ng-model="hideSalary" />Hide Salary

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>City</th>

                    <th ng-show="!hideSalary">Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> {{ employee.name }} </td>

                    <td> {{ employee.gender}} </td>

                    <td> {{ employee.city}} </td>

                    <td ng-show="!hideSalary"> {{ employee.salary  }} </td>

                </tr>

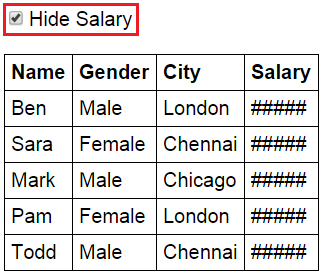
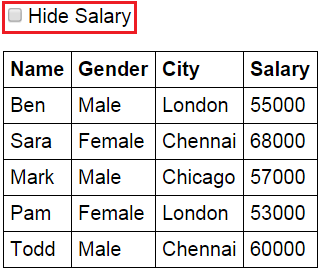
            </tbody>

        </table>

    </div>

</body>

</html>

The following example masks and unmasks the Salary column values using **ng-hide** and **ng-show**directives, depending on the checked status of the Hide Salary checkbox.  
  
   
  
 

<!DOCTYPE html>

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

<head>

    <title></title>

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

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <input type="checkbox" ng-model="hideSalary" />Hide Salary

        <br /><br />

        <table>

            <thead>

                <tr>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>City</th>

                    <th ng-hide="hideSalary">Salary</th>

                    <th ng-show="hideSalary">Salary</th>

                </tr>

            </thead>

            <tbody>

                <tr ng-repeat="employee in employees">

                    <td> {{ employee.name }} </td>

                    <td> {{ employee.gender}} </td>

                    <td> {{ employee.city}} </td>

                    <td ng-hide="hideSalary"> {{ employee.salary  }} </td>

                    <td ng-show="hideSalary"> ##### </td>

                </tr>

            </tbody>

        </table>

    </div>

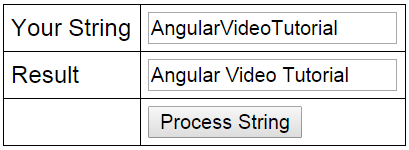
</body>

</html>

### AngularJS Services:

### What is a service in AngularJS Before we talk about what a service is in Angular. Let's talk about a service in web development.   If you have any experience developing web applications 1. You might have heard about Web Services and WCF Services 2. You might have also created objects that provide some services. For example, a Math object may provide services to add numbers.  So, a service in Angular is simply an object that provide some sort of service that can be reused with in an angular application. The angular service object has properties and methods, just like any other JavaScript object.   AngularJS has lot of built in services. We discussed two of the built in services - $http & $log, in our previous video. $http service is used to make AJAX calls. $log service is useful to log an object to the console, which is very useful when debugging applications. We can also create our own custom services, which we will discuss in a later video.  For now let's understand the need for services.  Why do we need services in an angular application The primary responsibility of the controller is to build the model for the view.  The controller should not be doing too many things. For example, if the controller also has the logic to compute Age from Date of Birth, it violates one of the SOLID principles i.e the Single Responsibility Principle. The Single Responsibility Principle states that an object should only have a Single Responsibility. So this kind a logic belongs in it's own service, which can then be injected into the object that needs that service.  In our previous video session, we have injected 2 of the angular built in services i.e $http and $log service into the controller function that needs them. In general, if the logic with in your controller, is becoming too large or too complex, then it is time, to take a step back, and think if anything can be abstracted into it's own service. Services can be used by controllers, directives and filters. What are the benefits of using services Reusability : In a service you usually have a logic that you want to reuse with in your entire application. For example, any time you want to make AJAX calls, you can use one of the built in angular service - $http, simply by injecting it into the object that needs that service. The application is also easier to maintain when the reusable components are encapsulated into their own services. Dependency Injection : Another benefit of services, is that, they can simply be injected into controllers or other services that need them. Testability : Since services are injected into controllers or other services that need them, it becomes very easy to test them. Depending on which service you are testing, you can pass mock implementations or real implementations. If you are new to unit testing and mocking,

### Create custom service in AngularJS:

Whenever the case changes from lower to upper, a single space character should be inserted. This means the string **"AngularVideoTutorial"** should be converted to **"Angular Video Tutorial"**.   
  
   
  
Let us first see, how to achieve this without using a custom service.   
  
**HtmlPage1.html :**  

<!DOCTYPE html>

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

<head>

    <title></title>

    <script src="Scripts/angular.js"></script>

    <script src="Scripts/Script.js"></script>

    <link href="Styles.css" rel="stylesheet" />

</head>

<body ng-app="myModule">

    <div ng-controller="myController">

        <table>

            <tr>

                <td>Your String</td>

                <td><input type="text" ng-model="input" /> </td>

            </tr>

            <tr>

                <td>Result</td>

                <td><input type="text" ng-model="output" /></td>

            </tr>

            <tr>

                <td></td>

                <td>

                    <input type="button" ng-click="transformString(input)"

                           value="Process String" />

                </td>

            </tr>

        </table>

    </div>

</body>

</html>

**Script.js :**Notice, all the logic to insert a space when the case changes is in the controller. There are 2 problems with this  
1. The controller is getting complex  
2. This logic cannot be reused in another controller. If you have to use this logic in another controller, we will have to duplicate this same code with in that controller.   
  
When we use our own custom service to encapsulate this logic, both of these problems go away. The custom service can be injected into any controller where you need this logic. 

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope) {

            $scope.transformString = function (input) {

                if (!input)

                    return input;

                var output = "";

                for (var i = 0; i < input.length; i++) {

                    if (i > 0 && input[i] == input[i].toUpperCase()) {

                        output = output + " ";

                    }

                    output = output + input[i];

                }

                $scope.output = output;

            };

        });

Now let's create a custom service. Here are the steps  
1. Add a JavaScript file to the Scripts folder in the project. Name it stringService.js.  
2. Copy and paste the following code. Notice we are using the factory method to create and register the service with Angular. 

app.factory('stringService', function () {

    return {

        processString: function (input) {

            if (!input)

                return input;

            var output = "";

            for (var i = 0; i < input.length; i++) {

                if (i > 0 && input[i] == input[i].toUpperCase()) {

                    output = output + " ";

                }

                output = output + input[i];

            }

            return output;

        }

    };

});

3. Copy and paste the following code in Script.js. Notice that we have injected stringService into the controller function. 

var app = angular

        .module("myModule", [])

        .controller("myController", function ($scope, stringService) {

            $scope.transformString = function (input) {

                $scope.output = stringService.processString(input);

            };

        });

4. On HtmlPage1.html, only one change is required and that is to reference the stringService.js script file

<script src="Scripts/stringService.js"></script>

### AngularJS routing tutorial:

### In general, as the application becomes complex you will have more than one view in the application. Let's say you are building a single page application for a training institute and you have the following views  - Home  - Courses  - Students  We can take advantage of the Angular routing feature, to have a single layout page, and then inject and swap out different views depending on the URL the end user has requested.  So in our application we will have the following views AngularJS routing tutorial  index.html is the layout view home.html, courses.html & students.html will be injected into the layout view(index.html) depending on the URL the end user has requested  For example, if the user has requested http://localhost:51983/home, then home.html will be injected into the layout view (index.html). Similarly if the user has requested http://localhost:51983/courses, then courses.html will be injected into the layout view (index.html).  Preparing the angular application application to use routing : The AngularJS Route module is present in a separate JavaScript file. You can either download it from AngularJs.org and use it in the application or you can use the CDN link.

### Angular 1 was released in October 2010, and by far the most popular JavaScript framework available for creating web applications. Many developers are already using Angular 1, so the obvious question that comes to our mind is why should we use Angular 2.  Angular 2 is not a simple upgrade from angular 1. Angular 2 is completely rewritten, so it has lot of improvements when compared with Angular 1. Let's look at a few of these improvements.  Performance : From a performance standpoint, Angular 2 has faster initial loads, change detection, and improved rendering time. Not just performance, we also have improved modularity, Dependency injection and testability. According to angular conference meetup, Angular 2 is 5 times faster compared to AngularJS 1.  Mobile Support : Angular 1 was not built for mobile devices. It is possible to run Angular 1 on mobile but we will have to use other frameworks. Angular 2 on the other hand is designed from the ground up with mobile support. Mobile device features and limitations like touch interfaces, limited screen real estate, and mobile hardware have all been considered in Angular 2. So with Angular 2 we can build a single application that works across mobile and desktop devices. Component Based Development : Component based web development is the future of web development. In Angular 2, "everything is a component". Components are the building blocks of an Angular application. The advantage of the component-based approach is that, it facilitates greater code reuse. From unit testing standpoint, the use of components make Angular2 more testable. We will discuss what a component is and how to build components with examples in detail, in our upcoming videos.  More language choices : There are several languages that we can use to develop Angular applications. To name a few, we have 1. ECMAScript 5 2. ECMAScript 6 (also called ES 2015) 3. TypeScript etc.  Besides these 3 languages we can also use Dart, PureScript, Elm, etc, but among all these, TypeScript is the most popular language.   Angular 2 itself, is built using TypeScript. TypeScript has great support of ECMAScript 6 standard. So the obvious questions that come to our mind at this point are  1. What is ECMAScript  2. Wha is Type Script What is ECMAScript : The JavaScript language standard is officially called ECMAScript. Over the past several years many versions of ECMAScript were released starting with ECMAScript version 1 all the way till ECMAScript version 7. Most of the modern browsers available today support ECMAScript 5. The browser support for ECMAScript 6 is still incomplete. However, using a process called Transpilation, ECMAScript 6 can be converted to ECMAScript 5 which is supported by all the modern browsers. ECMAScript 6 is officially known as ECMAScript 2015. ECMAScript 2015 introduced several new features like classes, modules, arrow functions etc. If you are interested in reading more about the ECMAScript standard and what these different versions of ECMAScript have to offer, please refer to the the following Wikipedia article. <https://en.wikipedia.org/wiki/ECMAScript> Wha is Type Script : TypeScript is a free and open-source programming language developed by Microsoft. It is a superset of JavaScript and compiles to JavaScript through a process called transpilation. Using TypeScript to build angular applications provides several benefits. 1. Intellisense  2. Autocompletion 3. Code navigation 4. Advanced refactoring 5. Strong Typing 6. Supports ES 2015 (also called ES 6) features like classes, interfaces and inheritance. If you have any experience with object oriented programming languages like C# and Java, learning TypeScript is easy. Because of all these benefits writing, maintaining and refactoring applications can be an enjoyable experience. So obviously TypeScript has become the number one choice of many developers for developing Angular applications. For this course we will be using Visual Studio as the code editor. Besides Visual Studio, TypeScript is supported by several other editors like 1. Visual Studio Code 2. Eclipse 3. WebStorm 4. Atom 5. Sublime Text etc. So you can use any favourite editor of your choice to build Angular 2 applications using TypeScript.