# Web development using Angular Js - CourseEra

Installations :

1. Install any Javascript editor. Here we are using Atom since it is free.
2. We will install **browser sync.** Use of Browser-sync is , whenever you save any code in the editor, the result will automatically synced with the web browser. You don’t need to run or reload the page in the browser every time you made any changes. Browser-sync need node js installation. So before installing Browser-sync, you need to install node js setup. Once the node Js is installed, you need to run the following command in the cmd : **npm install –g browser-sync .** However browser-sync installation is optional, you can also run the source code manually.
3. Once the browser-sync is installed. We will install the Git in our system. It will help us to make and manage repository. You can use console version as well as desktop GUI version of GIT. I will use the later one, since it is easy to use.
4. You also need to have a GIT account, if you don’t have, then create one.
5. After you have an account, create a new repository in github.
6. Once you created the repository, you want it to make public, so that it will act as server for your web pages. To do so, go ahead and click on setting. Scroll down to Github pages. Select source as Master and save it. As soon as you save it , it will gives us a link for our server. In my case it is : <https://vishwas1.github.io/AngularJs_Study/index.html>
7. Open the Git gui and clone the repository that you have created in the git account.
8. Once you created and configured the repository, go ahead and open cmd. Navigate to your repo. And type “**atom .**” It will open atom pointing to the directory. Now create a index.html page and write some simple html code.
9. To see the reflection with browser-sync, go and type this command in the cmd from the same location where atom was opened. :  **browser-sync start --server --directory –files= “\*\*/\*” .** It will open the directory in browser, browse index.html and type some code in atom editor. As soon as you commit/save the code, the result will get reflected in the browser.
10. Commit the changes and publish it from Git gui.
11. Done.

**MODULE 01**

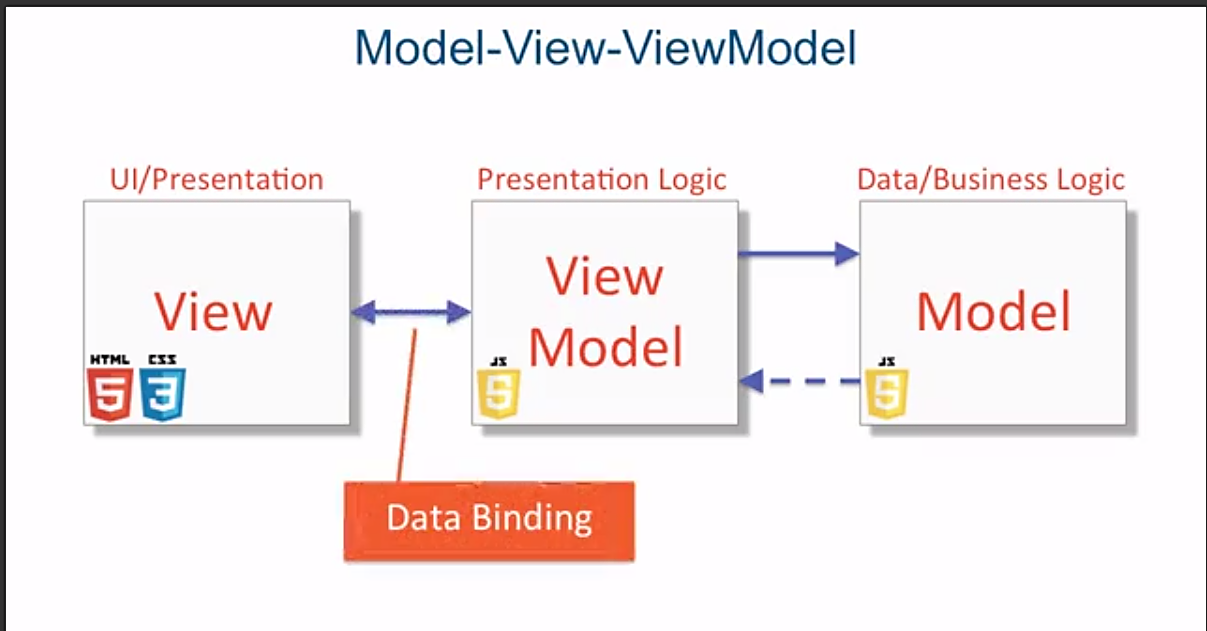
**Angular has many design pattern : one of them is, Model-View-ViewModel approach**

**Model : That has the actual data, either hard coded or from the server through some http request through Ajax call.**

**View : That shows those data as it is, without any modification. It also exposes the event but never responds to them.**

**ViewModel : It responds to view events. In other words it does the presentation logic. It also holds the data that is displayed in the view. It calls other functionaly for presentational logic processing. It is actually the data representation of the state of the View.**

**Architecture of Angular (MVVM):**

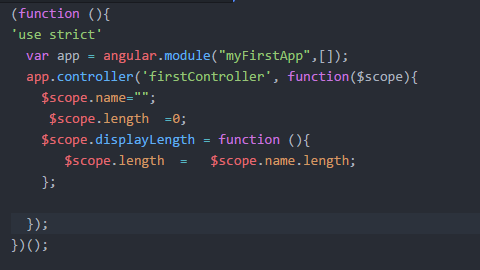
****

**For this Data Binding, you don’t have to write the code, as the manipulation is done itself by the Declarative Binder, i.e framework.**

**Declarative Binder : declaratively binds the ViewModel and View.**

**Installation : Angular Js Framework**

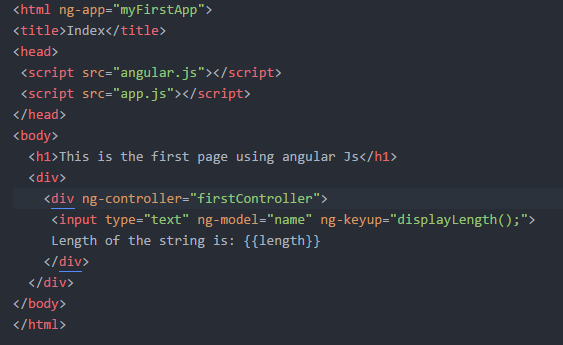
1. Just go to angular JS website and download the script of whatever version you want to use.
2. Add that script into your application.



**First we create a app called “myFirstApp” using module function. It accepts two parameters, one is the app name and second one in the dependencies. Here we don’t have any any dependencies, so we put the array to blank.**

**Once we created the app, we can use that object to call controller method. It also accepts two parameters, one the controller name and other is a method where all the functionality needs to be written.**

**Notice that we have used $scope variable as parameter to the controller functioin. We requires this object, to bind the data to the view (or the div or some html tag) where the controller is defined using ng-controller attribute. We can now defined as many properties to the $scope object. Even we can define any function to it. Please refer the image above. Here displayLength is used as a function.**

****

**Once we created our angular app, we can now use the declarative binder, which is nothing but some specific attributes defined by angular, to bind the ViewModel to view.**

1. **ng-app : We must put this attribute in the higher level tag. Its valued would be the appname defined in the script. Here we have used it in html tag. We can use it any top level tag we wanted to.**
2. **ng-controller : we can create any many controller we wanted to using app object. To bind the controller with the tag we have to use this attribute.**
3. **ng-model : It is used in the tag where the actual data will resides. The value of the model would the property of the $scope object.**
4. **There are many more attributes that acts as an event in javascript. Like, here we have userd, ng-keyup for onkeyup event, then we have ng-click for onclick event in javascript.**
5. **Other attributes are also there for the basic html attributes like, ng-src for src.**

**NOTE: Do read about Dependency Injection or Inversion of control.** Here is the link for it : [**https://docs.angularjs.org/guide/di**](https://docs.angularjs.org/guide/di)

**You must have noticed that $scope is nothing but a class or object or you can also say that it is acting as a service. And we can define as many properties as we want. Similarly, there are many other services exposed by Angular like, $filter, $injector. We can pass these services as parameter to the controller functions and use them.**

**MODULE 02**

**In this module we will talk about $filter service.**

**Ex. To convert into uppercase.**

**Var msg= “Lets cconvert this string into uppercase.”**

**Var output = $filter(‘uppercase’)(msg); // It will convert this msg into uppercase.**

**Similarly there are many more filter functions are there :**

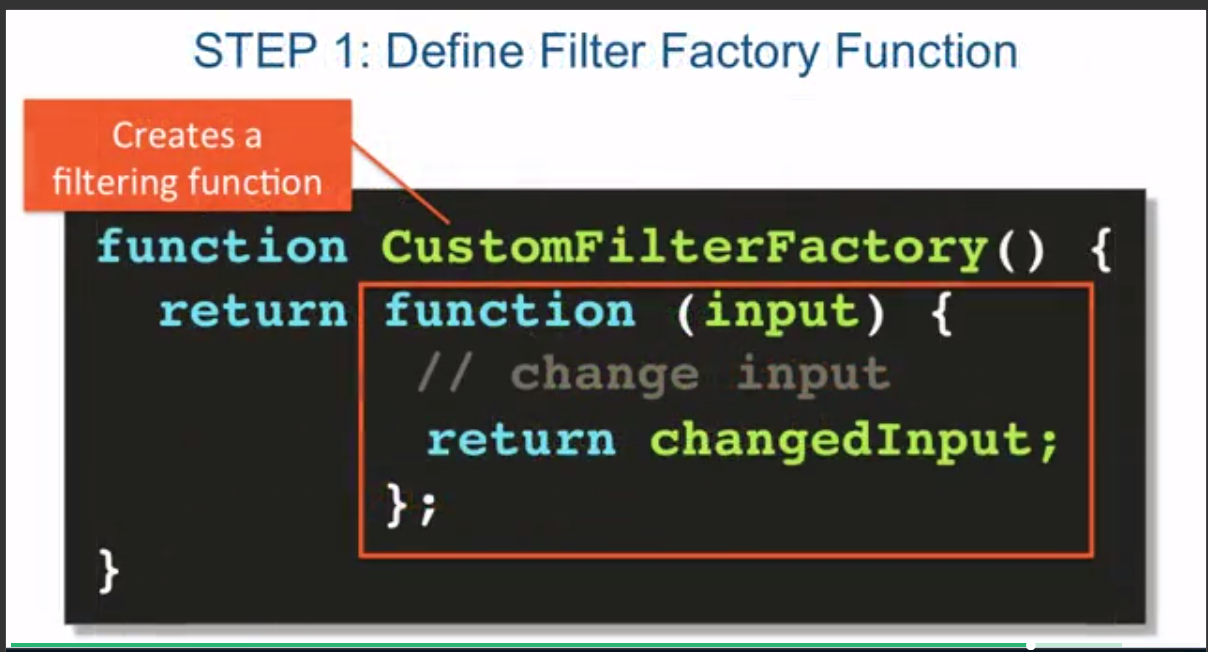
[**https://docs.angularjs.org/guide/filter**](https://docs.angularjs.org/guide/filter)

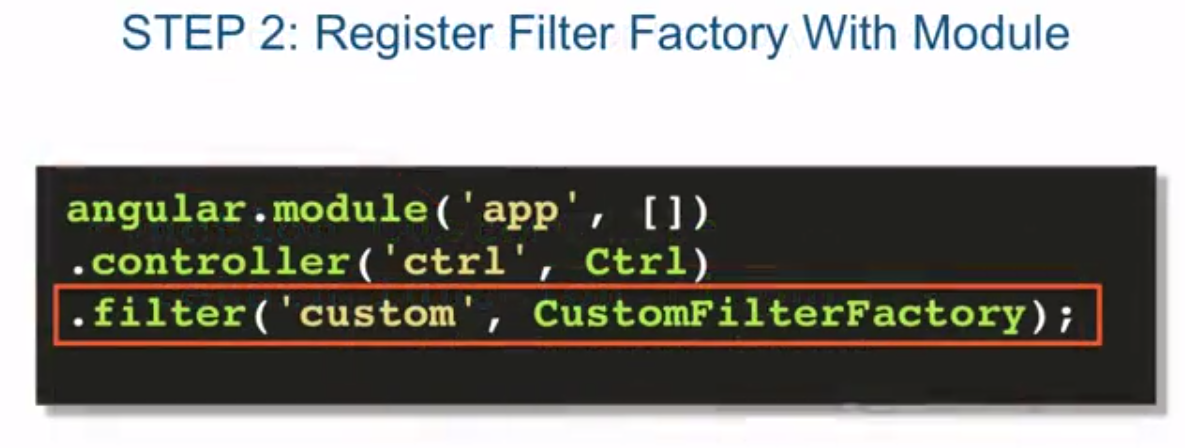
[**https://docs.angularjs.org/api/ng/filter**](https://docs.angularjs.org/api/ng/filter)

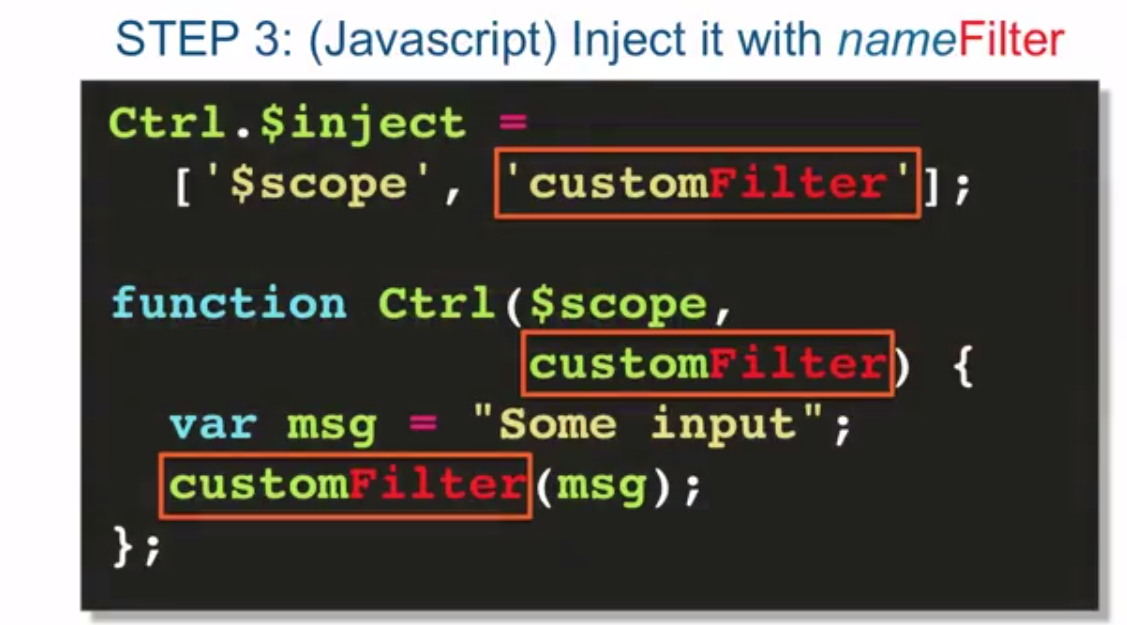
**We can also use filter into view itself: {{“ Lets cconvert this string into uppercase” | uppercase }}**

**WE CAN ALSO CREATE OUR OWN CUSTOM FILTER…..**

**There are 3 steps to do so.**

****

****

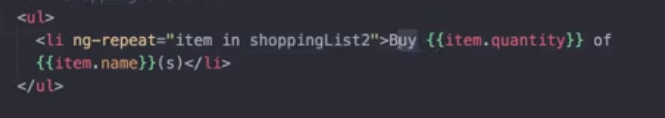
****

**Looping concepts:**

* **ng-repeat is used for looping. It is also a derictive of the other angular directives. It is similar to foreach loop. It can iterate over a list of objects or list of string.**

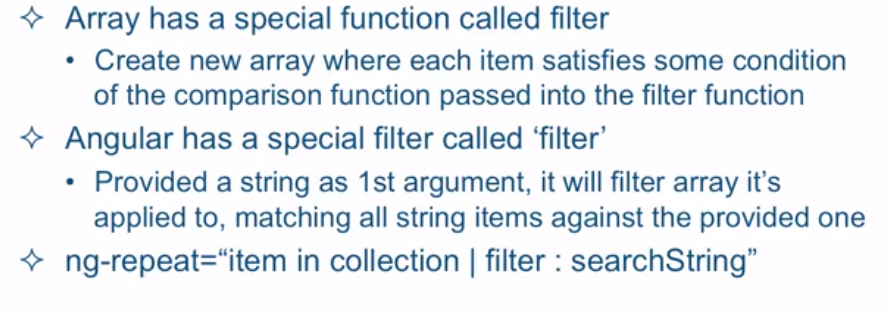
**Syntax: ng-repeat = “item in collection”**

**Example:**

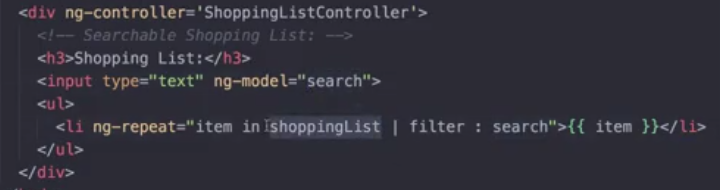
****

**NOTE: $index is the index, that is exposed by the ng-repeat directive. So, there is an array instead of list then, you can iterate over it by using , array[$index]. $index starts with 0.**

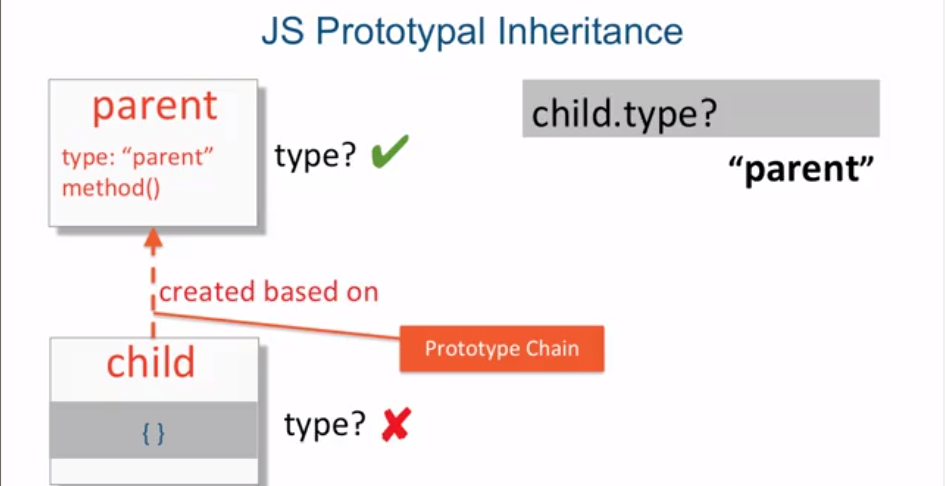
**Filter Concepts:**

****

**Example:**

****

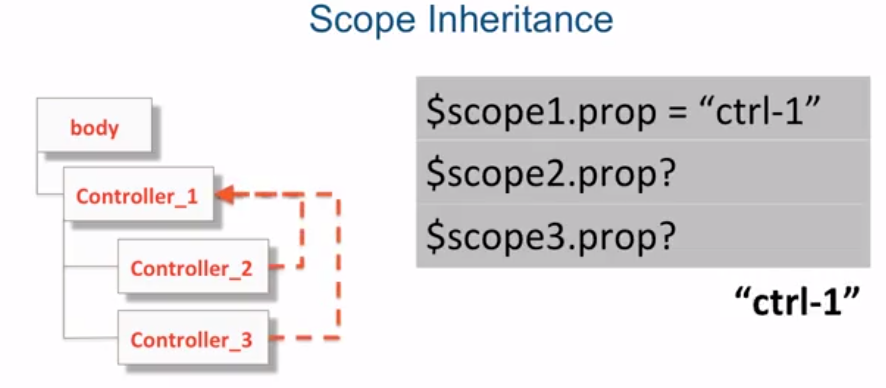
**Prototypal Inheritance:**

****

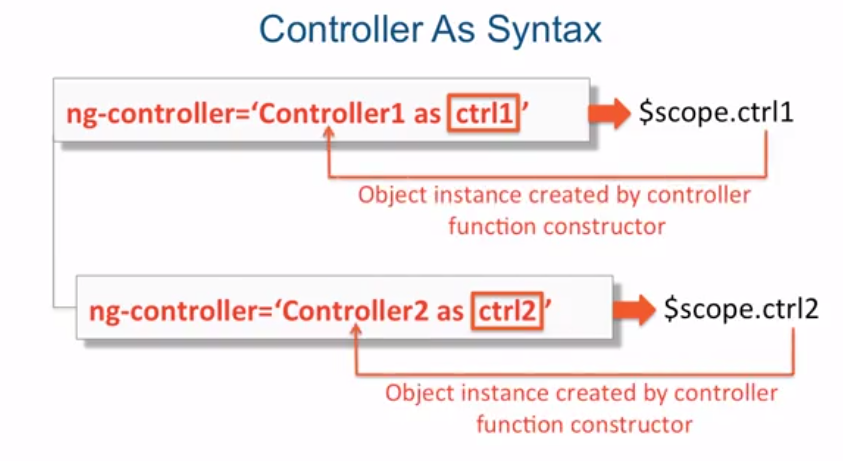
**Scope Inheritance:**

Now that we understand how prototypal inheritance works, let's talk about another concept called scope inheritance. It's not common, nor is it a good practice to have one controller that handles everything in your page. It's much easier to code smaller pieces of functionality that are responsible for smaller parts of your page. As you know, document object model, or the DOM, is a tree of nested HTML elements and nodes.

When you declare that a controller is responsible for part of your page, some natural nesting of controllers will occur. Therefore, the scope service, or simply the scope of the outer controller, is available to the inner controllers. However, Angular makes it even nicer than that. The scope of the inner controller prototypically inherits from the scope of the outer controller. Since you now know what prototypal inheritance means, you understand that the properties declared in the outer controller scope, Controller\_1, in this case, are accessible to the inner controller scope, Controller\_2 and 3, without any extra effort. That is, of course, if the inner controller doesn't mask those outer parent controller scope properties by declaring those same properties on its own scope. So if we declare a property called prop on the scope of the first controller, and make it equal to a string ctrl-1, to signify that it comes from the first controller. And then try to access that same property on the scope of the second controller and the third controller, which are inheriting from the Controller\_1. The JavaScript engine will go up the prototype chain and will look up the value of the property in the first controller, which is going to be again, ctrl-1. But what would happen if first, inside of our Controller\_2, we retrieve the prop value inherited from scope1, evaluate it somehow and then wanted to change it in such a way that both scope1's prop value and scope2's prop value would be identical? Better yet, we would want them pointing to the same memory location. However, without manually going up the prototype chain and retrieving the instance of scope1 object, we can't achieve this with primitive types, every time we set the prop property on scope2, we end up masking the prop property of scope1 from the view of scope2. So in this scenario, $scope1.prop would still be unchanged and be equal to that same ctrl-1. However, as you may remember from the previous video, when we're dealing with prototypically inherited objects, things work out very differently.

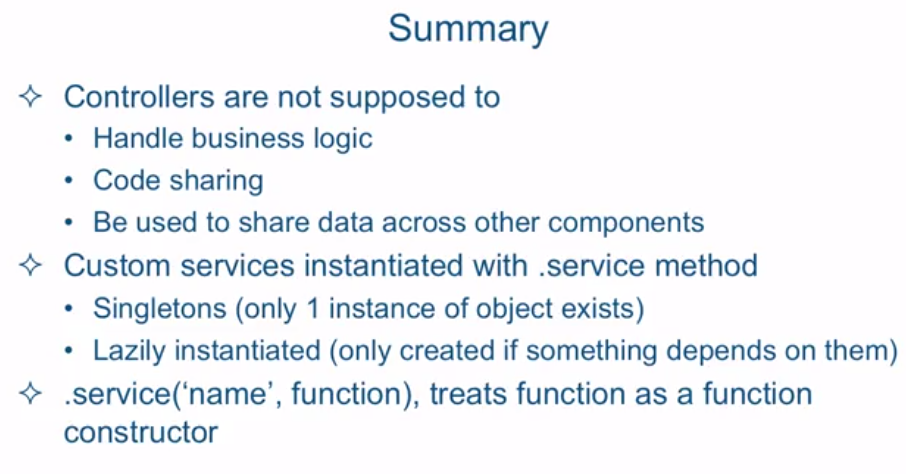
****

The reason for this is that just to get to the property of the inherited object, we already have to walk up the prototype chain. So when we change the property of such an object, we're changing it at its original source, therefore all controllers reflect the change and no masking occurs. As you could see in this case, even though we changed the object's property prop through the reference of scope2, which is an inner controller scope, scope1, with the same object property prop changed its value as well. So you can see, it's very advantageous when we have scope inheritance to work with objects that are properties of the scope, and not with primitive properties that are directly sitting on the scope itself. And this is where controller as syntax, comes to help us quite a bit. The controller as syntax provides us with a very convenient object we can use to attach all of our properties to, so property masking does not get in the way, it let's us specify a label in our HTML template. The name of this label is attached to the corresponding scope service as a property. So in this case for the first controller, it's $scope.ctrl1 and for the second $scope.ctrl2. Those properties, in this case, ctrl1 and ctrl2, point to objects that are instances of the controllers themselves. In other words, the controller instance gets attached as a property on the $scope. That means, that when it comes to implementing the control function, or the function that is responsible to implement the functionality of our controller, we can attach our properties directly to the this keyword inside of our controller function and not even have to inject the $scope into the controller. So the this keyword becomes synonymous with the name of the label given in the controller as syntax in our HTML template. The reason we're able to accomplish all that without much code is because Angular does a lot for us behind the scenes. Here's about what it does, obviously, what Angular actually does is much more involved. So this is just a rough idea of what happens behind the scenes. So in our case, we have two nested controllers. The outer controller, Controller1, declared as ctrl1 and the inner controller, Controller2, declared as ctrl2. After AngularJS creates the scope for our Controller1, which we'll call $scope1 for now, AngularJS attaches a property called ctrl1, the same as our label, and then instatiates it using our controller function, as a function constructor. Which means ctrl1 is now equivalent to the this keyword inside of our function constructor represented by a function that we specified as our controller function. Since we're on prototypical inheritance, $scope2 which is the inner scope is going to get created based on $scope1, which is the outer scope. Then we're going to go through a similar process where we're going to create a property for scope2 with the name of the label that we assigned our Controller2 as, which is ctrl2. And we'll initialize it with the instance of the Controller2 function that is created using the new keyword, therefore, using Controller2 function as a function constructor. So, yet again, the ctrl2 is going to be pointing to the same thing, that the this keyword will point to inside of the Controller2 function constructor. But things don't just get better and simpler inside of our JavaScript, the controller as syntax makes the HTML templates simpler as well. Not only does it allow me to access properties that are named the same from the parent scope without masking any properties, it also makes for a much more readable code. So now when I look at this interpolation, ctrl2.myProp, I know that this property is instantiated and belongs to Controller2. And ctrl1.myProp is a property that is instantiated on a Controller1 and that's where it belongs.

****

**Creating Custom Service:**

Controllers are not supposed to handle business logic. They're also not supposed to be used to share codes. So if you have common code between different controllers, it's not a good idea to try to reuse the controllers. Rather, you should Factor that code into a service and then share the service within the Controllers. Controllers are also not supposed to be used to share data across different components. Now custom services instantiated with a .service method are Singletons Which means, there's only one instance of the object that ever exists, which means you could share data between different controllers or other components using that service because again only one exists so therefore it's going to hold the same data Across all your components of your application. Also, services instantiated with a .service method are lazily instantiated, which means they're only created if something actually depends on them. If nothing depends on them. They're simply not instantiated at all. Ans last but not least, the .service method that you supply the function to as your custom service, treats that function as a function constructor and we spoken before, that what it means is that the angular JS internally will actually new up your function using in new keyword Which has ramifications for what this keyword means inside of your service function.



Note:

* <http://stackoverflow.com/questions/22401553/what-are-all-the-difference-between-function-and-constructor-function-in-javascr>
* Single ton design pattern.

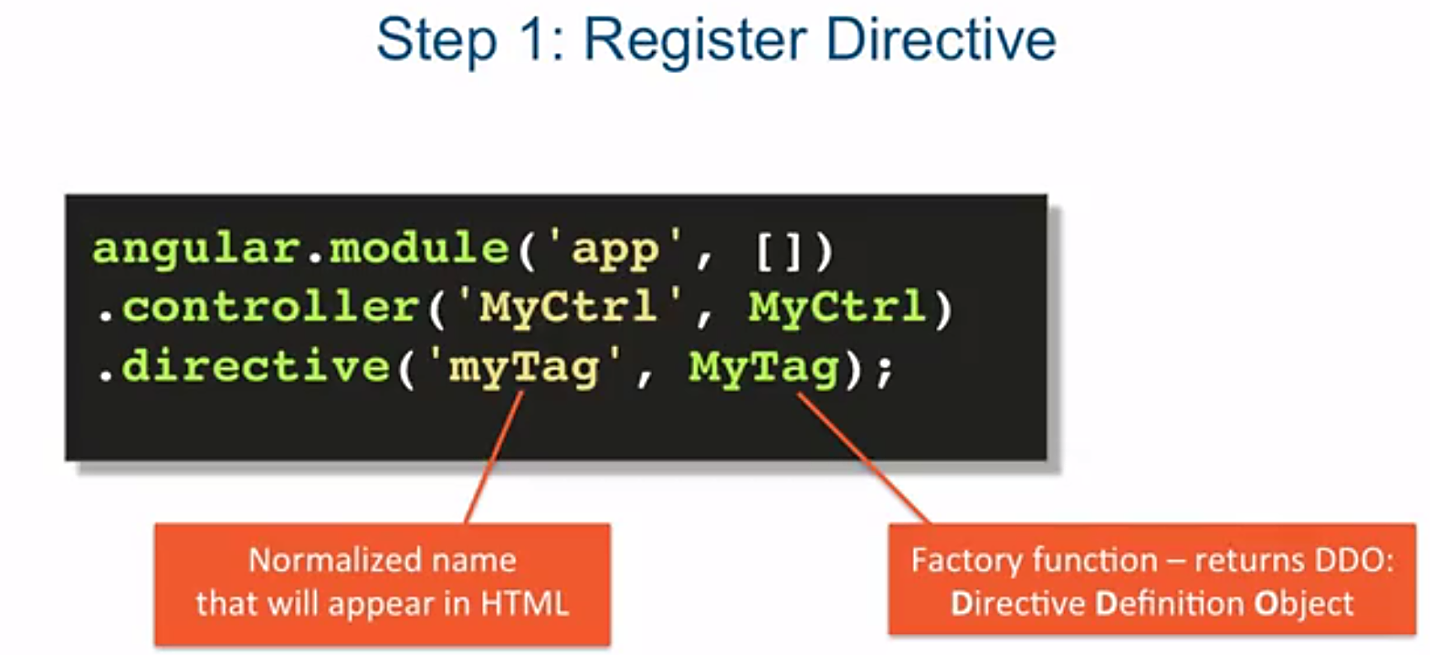
**HTTP Service:**

**$http Service:**

**The HTTP service is itself a function, so you can actually call it directly. It takes just one argument, a configuration object which Angular expects to have some pretty fine properties such as method, URL, and so on. As I mentioned, it returns a promise so that's why you can invoke the familiar then method.** 

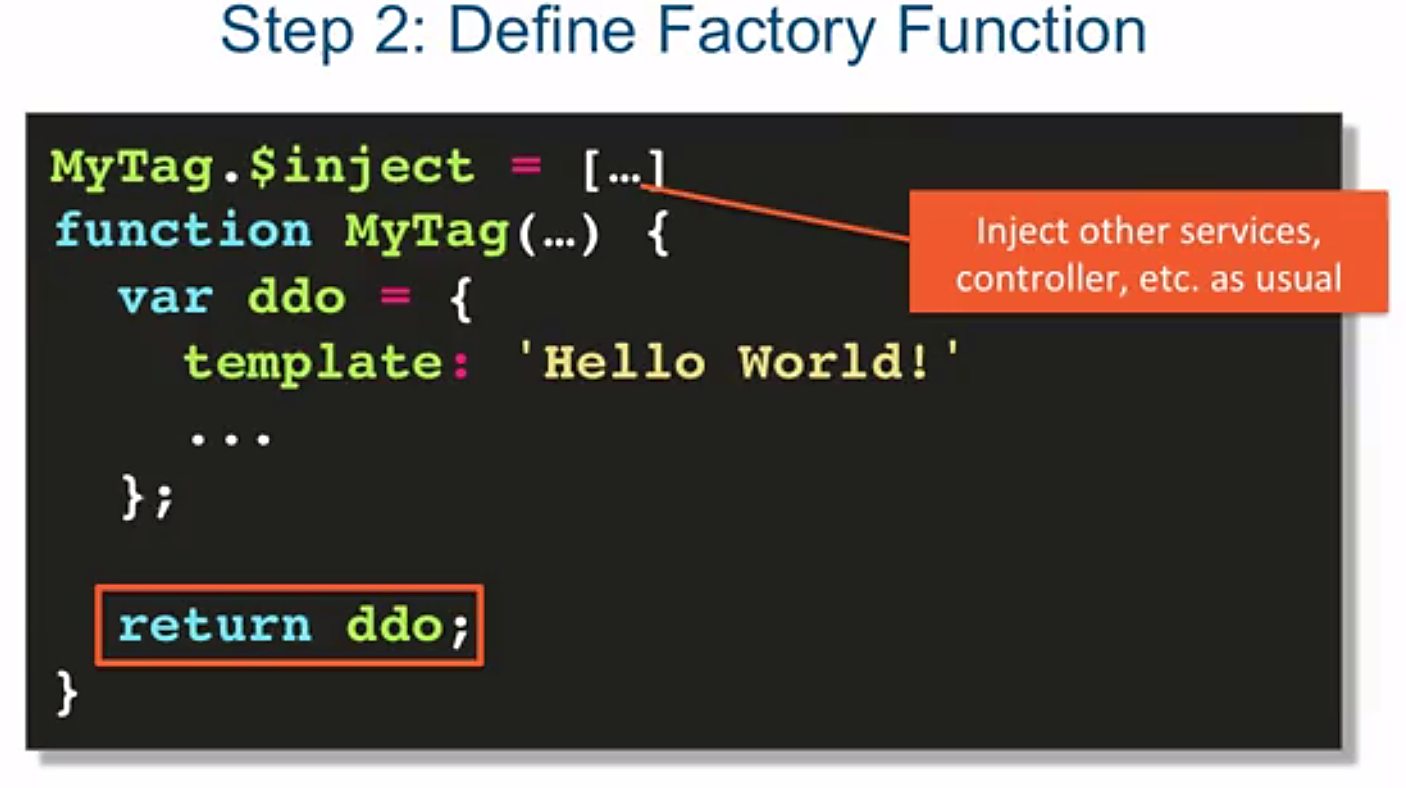
****

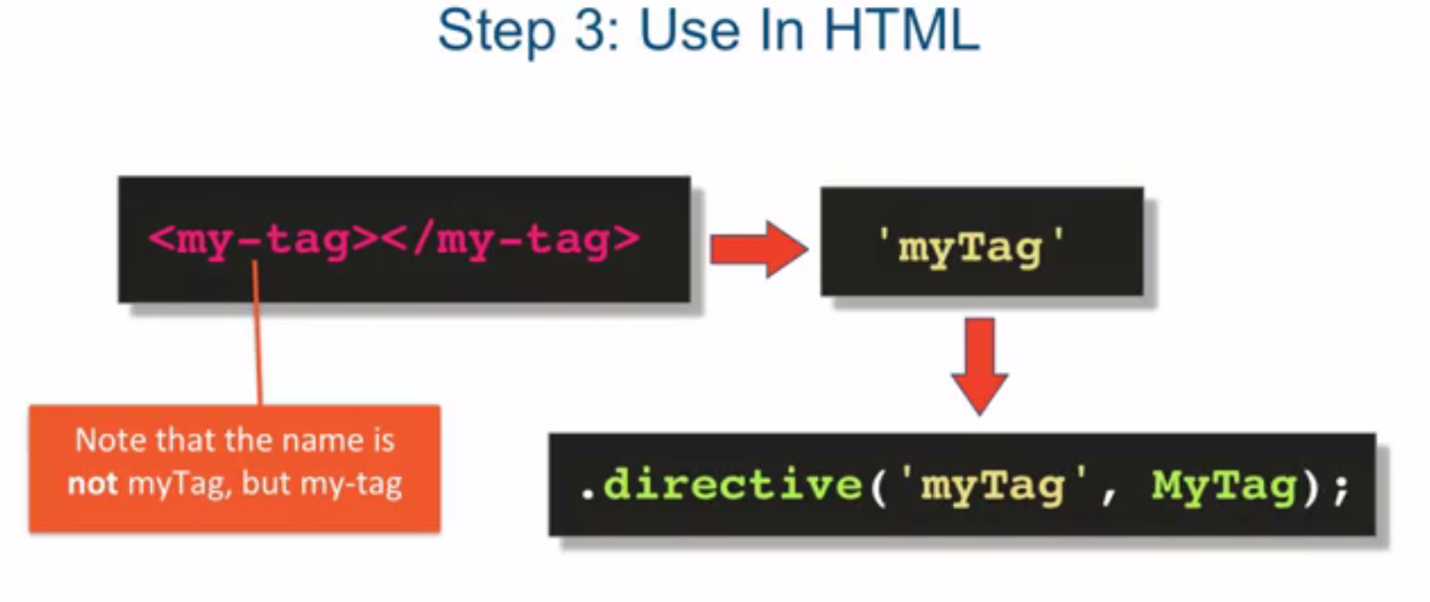
**Custom Angular Directives:**

****

**DDO (Directive Definition Language): is nothing but an object, that tells Angular compiler, how this directive should behave when it finds this directive/tag/attribute in the page.**

**The Factory function is defined and declared only once, so it can be used for any initializations that we need besides returning this DDO.**

****

****

Interview question related to angular Js :

Topics need to be study :

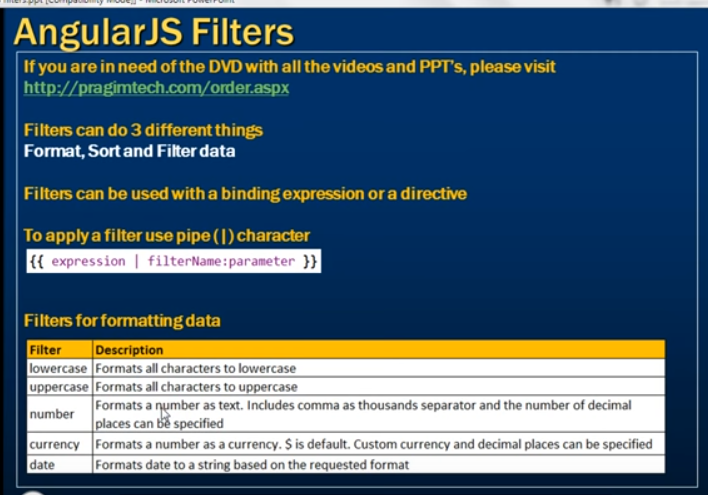
* **Filters**
* **Providers**
* **Factory**
* **Service**
* **.config()**

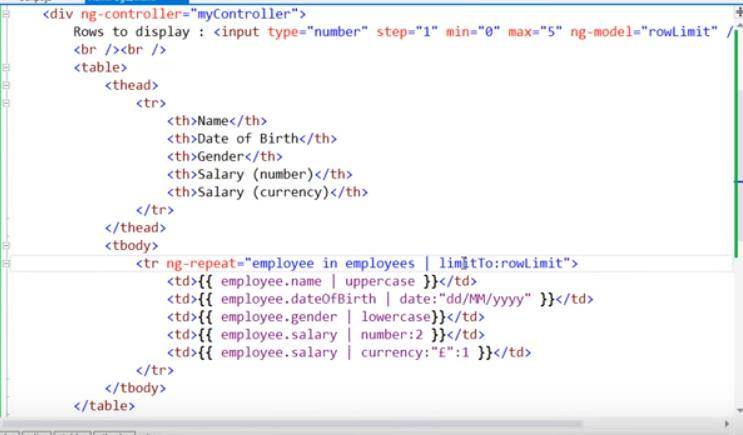
Factory : Factory produces service.

1. What is the difference between Service, Factory and Provider and their usage ?

Ans: Service and factory are basically providers.

1. Filter : Filters can be used to basically doing 3 things , a) Formatting b) Sorting c) Filtering Data

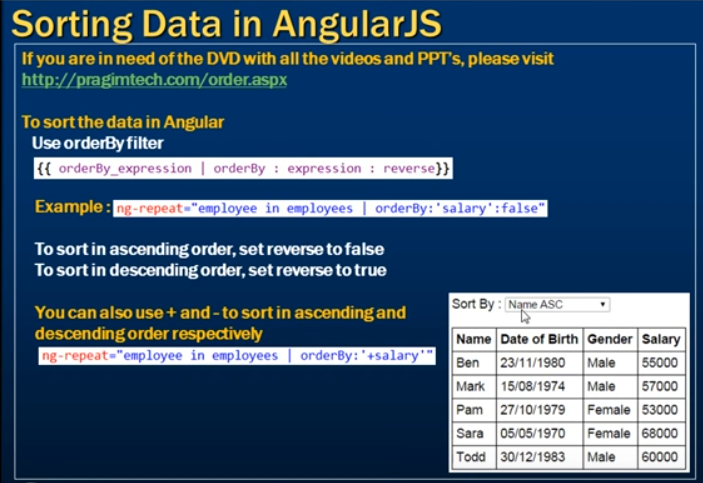




Pipe (|) symbol is used here, followed by filter name(i.e. date,lowercase,number,currency,uppercase) followed semicolon followed by format

There is another filter called “limitTo” filter which can be used in the ‘ng-repeat’. Like this:

< tr ng-repeat =”employee in employeeLst | **limitTo** : rowLimit ”> If we put rowLimit = 3 then only 3 rows will be displayed in the table, irrespective of number of employee in the list.



< tr ng-repeat =”employee in employeeLst |**orderBy** :’name’:false ”>

So, it will sort the table into Asc order by column ‘name’. If we change false to true then t will sort the table into Desc order by column ‘name’.

**Search Implementation Using Filters:**

< tr ng-repeat =”employee in employeeLst | **filter** : ‘searchText’ ”>

**How to create a custom Filter ?**

**app.js :**

angular.module(‘myApp’,[])

.controller (‘ControllerName’,function($scope){

$scope.names = [‘Vishwas’,’Amit’,’Annu’];

})

.filter(‘FilterName’,function(){

Return function(input){

//process this input and return the output

Return output;

};

});

**Index.html :**

<div ng-app=”myApp”>

<div ng-controller=”ControllerName”>

<ul><li ng-repeat=”name in names ”>{{ name | FilterName }}</li></ul>

</div>

</div>