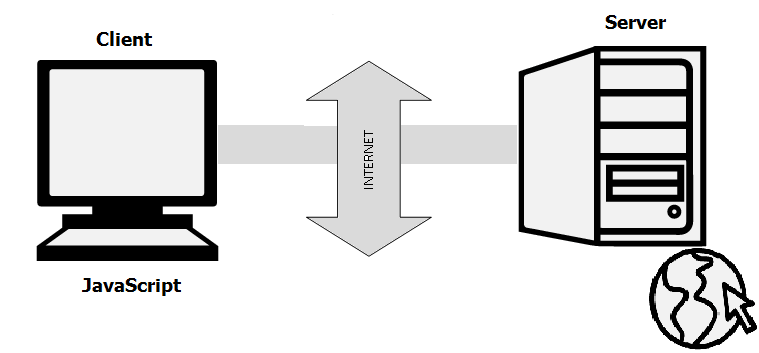
**Introduction to JavaScript**

JavaScript is a very powerful **client-side scripting language**. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and mobile application development.

****

**History**

JavaScript was developed by Brendan Eich in 1995, which appeared in Netscape, a popular browser of that time.

**Tools You Need**

To start with, you need a text editor to write your code and a browser to display the web pages you develop. You can use text editor of your choice including Notepad++, Komodo Edit or any other text editor you are comfortable with. You can use any web browser including Internet Explorer, Google Chrome, Firefox etc.

**A Simple JavaScript Code**

You should place all your JavaScript code within **<script> tags** (<script> and </script>) <script> tag and set its value to text/javascript like this.

<script type="text/javascript">

alert("Welcome!!! You are now learning JavaScript.");

</script>

# Using Variables in Javascript

Variables are used to **store values** (name = "John") **or expressions** (sum = x + y). Before using a variable, you first need to declare it. You have to use the keyword **var** to declare a variable like this:

var name;

You can assign a value to the variable either while declaring the variable or after declaring the variable.

var name = "John";

OR

var name;

name = "John";

# **Naming Variables**

# Though you can name the variables as you like, it is a good programming practice to give descriptive and meaningful names to the variables. Moreover, variable names should start with a letter and they are case sensitive. Hence the variables studentname and studentName are different because the letter n in name is different (n and N).

<script type="text/javascript">

var one = 22;

alert(one);

</script>

# Learn Arrays in JavaScript

An array is an object that can store a **collection of items**. Arrays become really useful when you need to store large amounts of data of the same type. Suppose you want to store details of 500 employees. If you are using variables, you will have to create 500 variables whereas you can do the same with a single array. You can access the items in an array by referring to its **indexnumber** and the index of the first element of any array is zero.

**Creating an Array**

You can create an array in JavaScript as given below.

var students = ["John", "Ann", "Kevin"];

Here, you are initializing your array as and when it is created with values “John”, “Ann” and “Kevin”.The index of “John”, “Ann” and “Kevin” is 0, 1 and 2 respectively. If you want to add more elements to the students array, you can do it like this:

students[3] = "Emma";

students[4] = "Rose";

You can also create an array using Array constructor like this:

var students = new Array("John", "Ann", "Kevin");

OR

var students = new Array();

students[0] = "John";

students[1] = "Ann";

students[2] = "Kevin";

**Properties and Methods**

The Array object has many properties and methods which help developers to handle arrays easily and efficiently. You can get the value of a property by specifying arrayname.property and the output of a method by specifying arrayname.method().

1. **length property** --> If you want to know the number of elements in an array, you can use the length property.
2. **prototype property** --> If you want to add new properties and methods, you can use the prototype property.
3. **reverse method** --> You can reverse the order of items in an array using reverse method.
4. **sort method -->** You can sort the items in an array using sort method.
5. **pop method** --> You can remove the last item of an array using pop method.
6. **shift method** --> You can remove the first item of an array using shift method.

<script>

var students = new Array("John", "Ann", "Aaron", "Edwin", "Elizabeth");

var arraylength=students.length;

alert(students+" "+'length:'+arraylength);

</script>

# How to use Loops in Javascript

# Loops are useful when you have to execute the same lines of code repeatedly, for a specific number of times or as long as a specific condition is true. Suppose you want to type a ‘Hello’ message 100 times in your webpage. Of course, you will have to copy and paste the same line 100 times. Instead if you use loops, you can complete this task in just 3 or 4 lines.

**Different Types of Loops**

There are mainly four types of loops in JavaScript.

1. for loop
2. for/in loop (explained later)
3. while loop
4. do…while loop

## for loop

Syntax:

for(statement1; statement2; statment3)

{

lines of code to be executed

}

Example:

<script type="text/javascript">

var students = new Array("John", "Ann", "Aaron", "Edwin", "Elizabeth");

document.write("<b>Using for loops </b><br />");

for (i=0;i<students.length;i++)

{

document.write(students[i] + "<br />");

}

</script>

## while loop:

The “while loop” is executed as long as the specified condition is true. Inside the while loop, you should include the statement that will end the loop at some point of time. Otherwise, your loop will never end and your browser may crash.

Syntax:

while(condition)

{

lines of code to be executed

}

Example

<script type="text/javascript">

document.write("<b>Using while loops </b><br />");

var i = 0, j = 1, k;

document.write("Fibonacci series less than 40<br />");

while(i<40)

{

document.write(i + "<br />");

k = i+j;

i = j;

j = k;

}

</script>

## do…while loop:

The do…while loop is very similar to while loop. The only difference is that in do…while loop, the block of code gets executed once even before checking the condition.

Syntax:

do{

block of code to be executed

} while (condition)

Syntax:

document.write("<b>Using do...while loops </b><br />");

var i = 2;

document.write("Even numbers less than 20<br />");

do

{

document.write(i + "<br />");

i = i + 2;

}while(i<20)

# Conditional Statements in JavaScript:

# Conditional statements are used to decide the flow of execution based on different conditions. If a condition is true, you can perform one action and if the condition is false, you can perform another action.

**Different Types of Conditional Statements**

There are mainly three types of conditional statements in JavaScript.

1. If statement
2. If…Else statement
3. If…Else If…Else statement

**If statement**

Syntax:

if (condition)

{

lines of code to be executed if condition is true

}

**Example:**

var age = prompt("Please enter your age");

if(age>=18)

document.write("You are an adult <br />");

if(age<18)

document.write("You are NOT an adult <br />");

**If…Else statement**

Syntax:

if (condition)

{

lines of code to be executed if the condition is true

}

else

{

lines of code to be executed if the condition is false

}

Example:

<script>

var hours = new Date().getHours();

if(hours<12)

document.write("Good Morning!!!<br />");

else

document.write("Good Afternoon!!!<br />");

</script>

**If…Else If…Else statement**

Syntax:

if (condition1)

{

lines of code to be executed if condition1 is true

}

else if(condition2)

{

lines of code to be executed if condition2 is true

}

else

{

lines of code to be executed if condition1 is false and condition2 is false

}

Example:

<script type="text/javascript">

var one = prompt("Enter the first number");

var two = prompt("Enter the second number");

one = parseInt(one);

two = parseInt(two);

if (one == two)

document.write(one + " is equal to " + two + ".");

else if (one<two)

document.write(one + " is less than " + two + ".");

else

document.write(one + " is greater than " + two + ".");

</script>

# Functions in JavaScript:

# Functions are very important and useful in any programming language as they make the code reusable A function is a block of code which will be executed only if it is called.  If you have a few lines of code that needs to be used several times, you can create a function including the repeating lines of code and then call the function wherever you want.

**Creating a Function**

1. Use the keyword **function** followed by the name of the function.
2. After the function name, open and close parentheses.
3. After parenthesis, open and close curly braces.
4. Within curly braces, write your lines of code.

Syntax:

function functionname()

{

lines of code to be executed

}

Example:

<script type="text/javascript">

function myFunction()

{

document.write("This is a simple function.<br />");

}

myFunction();

</script>

You can create functions with arguments as well. Arguments should be specified within parenthesis

function functionname(arg1, arg2)

{ lines of code to be executed}

Example:

<script type="text/javascript">

var count = 0;

function countVowels(name)

{

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

{

 if(name[i] == "a" || name[i] == "e" || name[i] == "i" || name[i] == "o" || name[i] == "u")

   count = count + 1;

}

document.write("Hello " + name + "!!! Your name has " + count + " vowels.");

}

var myName = prompt("Please enter your name");

countVowels(myName);

</script>

You can also create functions that return values. Inside the function, you need to use the keyword **return** followed by the value to be returned.

Syntax:

function functionname(arg1, arg2)

{

lines of code to be executed

return val1;

}

Example:

<script type="text/javascript">

function returnSum(first, second)

{

var sum = first + second;

return sum;

}

var firstNo = 78;

var secondNo = 22;

document.write(firstNo + " + " + secondNo + " = " + returnSum(firstNo,secondNo));

</script>

# Cookies in JavaScript:

A cookie is a piece of data that is stored on your computer to be accessed by your browser. You also might have enjoyed the benefits of cookies knowingly or unknowingly. Have you ever saved your facebook password so that you do not have to type it each and every time you try to login? If yes, then you are using cookies. Cookies are saved as key/value pairs.

**Relevance of Cookies**

The communication between a web browser and server happens using a stateless protocol named HTTP. Stateless protocol treats each request independent. So, the server does not keep the data after sending it to the browser. But in many situations, the data will be required again. Here comes cookies into picture. With cookies, the web browser will not have to communicate with the server each time the data is required. Instead, it can be fetched directly from the computer.

**Create, Access and Remove Cookies**

You can create cookies using document.cookie property like this.

document.cookie = "cookiename=cookievalue"

You can even add expiry date to your cookie so that the particular cookie will be removed from the computer on the specified date. The expiry date should be set in the UTC/GMT format. If you do not set the expiry date, the cookie will be removed when the user closes the browser.

document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2014 20:00:00 UTC"

You can also set the domain and path to specify to which domain and to which directories in the specific domain the cookie belongs to. By default, a cookie belongs to the page that sets the cookie.

document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2014 20:00:00 UTC; path=/ "

//create a cookie with domain to the current page and path to the entire domain.

You can access the cookie like this which will return all the cookies saved for the current domain.

var x = document.cookie

To delete a cookie, you just need to set the value of cookie to empty and set the value of expires to a passed date.

document.cookie = "cookiename= ; expires = Thu, 01 Jan 1970 00:00:00 GMT"

# Example:

<script type="text/javascript">

function createCookie(cookieName,cookieValue,daysToExpire)

{

var date = new Date();

date.setTime(date.getTime()+(daysToExpire\*24\*60\*60\*1000));

document.cookie = cookieName + "=" + cookieValue + "; expires=" + date.toGMTString();

}

function accessCookie(cookieName)

{

var name = cookieName + "=";

var allCookieArray = document.cookie.split(';');

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

  {

var temp = allCookieArray[i].trim();

  if (temp.indexOf(name)==0)

     return temp.substring(name.length,temp.length);

  }

return "";

}

function checkCookie()

{

var user = accessCookie("testCookie");

if (user!="")

  alert("Welcome Back " + user + "!!!");

else

  {

  user = prompt("Please enter your name");

  num = prompt("How many days you want to store your name on your computer?");

  if (user!="" && user!=null)

    {

createCookie("testCookie", user, num);

    }

  }

}

</script>

# Using DOM and Events in JavaScript:

# C:\Users\Harsha\Desktop\javascript8_1.png

**What can be achieved using DOM?**

Using DOM, JavaScript can perform multiple tasks. It can create new elements and attributes, change the existing elements and attributes and even remove existing elements and attributes. JavaScript can also react to existing events and create new events in the page.

**Common Methods and Properties**

1. getElementById:  To access elements and attributes whose id is set.
2. innerHTML: To access the content of an element.

<script type="text/javascript">

function sample(){

var name = document.getElementById("one");

name.style.color = "red";

}

</script>

</head>

<form action="#" onSubmit="return sample();">

<p id="one">Harsha</p>

<input type="submit">

</form>

# Object Oriented JavaScript:

Many times, variables or arrays are not sufficient to simulate real life situations. JavaScript allows you to create objects that act like real life objects. A student or a home can be an object that have many unique characteristics of their own. You can create properties and methods to your objects to make programming easier. If your object is a student, it will have properties like first name, last name, id etc and methods like calculateRank, changeAddress etc. If your object is a home, it will have properties like number of rooms, paint color, location etc and methods like calculateArea, changeOwner etc.

**Create an Object**

You can create an object like this:

varobjName = new Object();

objName.property1 = value1;

objName.property2 = value2;

objName.method1 = function()

{

line of code

}

OR

varobjName= {property1:value1, property2:value2, method1: function()

{ lines of code} };

**Access Object Properties and Methods**

You can access properties of an object like this:

objectname.propertyname;

You can access methods of an object like this:

objectname.methodname();

<script type="text/javascript">

var student = new Object();

student.fName = "John";

student.lName = "Smith";

student.id = 5;

student.markE = 76;

student.markM = 99;

student.markS = 87;

student.calculateAverage = function()

{

return (student.markE + student.markM + student.markS)/3;

};

student.displayDetails = function(){

document.write("Student Id: " + student.id + "<br />");

document.write("Name: " + student.fName + " " + student.lName + "<br />");

varavg = student.calculateAverage();

document.write("Average Marks: " + varavg);

};

student.displayDetails();

</script>

# External JavaScript:

You plan to display the current date and time in all your web pages. Suppose you wrote the code and copied into all your web pages (say 100). But later, you want to change the format in which the date or time is displayed. In this case, you will have to make changes in all the 100 web pages. This will be a very time consuming and difficult task.

So, save the JavaScript code in a new file with the extension .js. Then, add a line of code in all your web pages to point to your .js file like this:

<script type="text/javascript" src="currentdetails.js">

# jQuery

# jQuery is a small, light-weight and fast JavaScript library. It is cross-platform and supports different types of browsers. it takes a lot of common tasks that requires many lines of JavaScript code to accomplish, and binds them into methods that can be called with a single line of code whenever needed. It is also very useful to simplify a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

## jQuery Features:

* HTML manipulation
* DOM manipulation
* DOM element selection
* CSS manipulation
* Effects and Animations
* Utilities
* AJAX
* HTML event methods
* JSON Parsing
* Extensibility through plug-ins

jQuery was first released in January 2006 by **John Resig** at BarCamp NYC. It is currently headed by Timmy Wilson and maintained by a team of developers.

# jQuery Selectors:

# jQuery Selectors are used to select and manipulate HTML elements. They are very important part of jQuery library. With jQuery selectors, you can find or select HTML elements based on their id, classes, attributes, types and much more from a DOM. All jQuery selectors start with a dollor sign and parenthesis e.g. $(). It is known as the factory function.

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">

</script>

<script type="text/javascript" language="javascript">

$(document).ready(function() {

Lines of code

});

</script>

## Different jQuery Selectors

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Description** |
| \* | $("\*") | It is used to select all elements. |
| #id | $("#firstname") | It will select the element with id="firstname" |
| .class | $(".primary") | It will select all elements with class="primary" |
| class,.class | $(".primary,.secondary") | It will select all elements with the class "primary" or "secondary" |

Like this we have various type of selectors are there. please visit official jquery website

# JQuery Effects

jQuery enables us to add effects on a web page. jQuery effects can be categorized into fading, sliding, hiding/showing and animation effects.

|  |  |
| --- | --- |
| **Various** | **Jquery Effects** |
| animate() | performs animation. |
| clearQueue() | It is used to remove all remaining queued functions from the selected elements. |
| delay() | sets delay execution for all the queued functions on the selected elements. |
| dequeue() | It is used to remove the next function from the queue, and then execute the function. |
| fadein() | shows the matched elements by fading it to opaque. In other words, it fades in the selected elements. |
| fadeout() | shows the matched elements by fading it to transparent. In other words, it fades out the selected elements. |
| fadeto() | adjusts opacity for the matched element. In other words, it fades in/out the selected elements. |
| fadetoggle() | shows or hides the matched element. In other words, toggles between the fadeIn() and fadeOut() methods. |
| finish() | It stops, removes and complete all queued animation for the selected elements. |
| hide() | hides the matched or selected elements. |
| queue() | shows or manipulates the queue of methods i.e. to be executed on the selected elements. |
| show() | displays or shows the selected elements. |
| slidedown() | shows the matched elements with slide. |
| slidetoggle() | shows or hides the matched elements with slide. In other words, it is used to toggle between the slideUp() and slideDown() methods. |

Please visit jquery official website for more effects.

# jQuery Events

jQuery events are the actions that can be detected by your web application. They are used to create dynamic web pages. An event shows the exact moment when something happens.

These are some examples of events.

* A mouse click
* An HTML form submission
* A web page loading
* A keystroke on the keyboard
* Scrolling of the web page etc.

These events can be categorized on the basis their types:

## Mouse Events

* click
* dblclick
* mouseenter
* mouseleave

## Keyboard Events

* keyup
* keydown
* keypress

## Form Events

* submit
* change
* blur
* focus

## Document/Window Events

* load
* unload
* scroll

## Syntax for event methods

Most of the DOM events have an equivalent jQuery method. To assign a click events to all paragraph on a page, do this:

1. $("p").click ();

The next step defines what should happen when the event fires. You must pass a function to the event.

1. $("p").click(function(){
2. // action goes here!!
3. });

# AngularJS

# Introduction:

# Angular JS is used for developing mostly Single Page applications. This framework has been developed by a group of developers from Google itself.

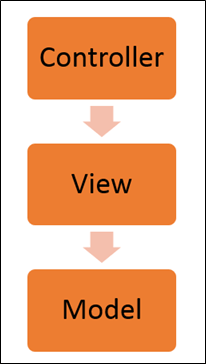
### What is AngularJS

On the onset, angularJS looks like a JavaScript framework, but it's more than that. It's based on a Model-View-Controller framework. It's a framework with a purpose.

Angular has the following key features which makes it one of the powerful frameworks in the market.

1. MVC – The framework is built on the famous concept of MVC (Model-View-Controller). This is a design pattern used in all modern day web applications. This pattern is based on splitting the business logic layer, the data layer, and presentation layer into separate sections. The division into different sections is done so that each one could be managed more easily.
2. Data Model Binding – You don't need to write special code to bind data to the HTML controls. This can be done by Angular by just adding a few snippets of code.
3. Writing less code – When carrying out DOM manipulation a lot of JavaScript was required to be written to design any application. But with Angular, you will be amazed with the lesser amount of code you need to write for DOM manipulation.
4. Unit testing ready – The designers at Google not only developed Angular but also developed a testing framework called "Karma" which helps in designing unit tests for AngularJS applications.

AngularJS Architecture: Angular.js follows the MVC architecture,



* The Controller represents the layer that has the business logic. User events trigger the functions which are stored inside your controller. The user events are part of the controller.
* Views are used to represent the presentation layer which is provided to the end users
* Models are used to represent your data. The data in your model can be as simple as just having primitive declarations. For example, if you are maintaining a student application, your data model could just have a student id and a name. Or it can also be complex by having a structured data model. If you are maintaining a car ownership application, you can have structures to define the vehicle itself in terms of its engine capacity, seating capacity, etc.

### AngularJS Advantages

* Since it's an open source framework, you can expect the number of errors or issues to be minimal.
* Two-way binding – Angular.js keeps the data and presentation layer in sync. Now you don't need to write additional JavaScript code to keep the data in your HTML code and your data later in sync. Angular.js will automatically do this for you. You just need to specify which control is bound to which part of your model.
* Routing – Angular can take care of routing which means moving from one view to another. This is the key fundamental of single page applications; wherein you can move to different functionalities in your web application based on user interaction but still stay on the same page.
* Angular supports testing, both unit testing, and integration testing.
* It extends HTML by providing its own elements called directives. At a high level, directives are markers on a DOM element (such as an attribute, element name, and comment or CSS class) that tell AngularJS's HTML compiler to attach a specified behavior to that DOM element. These directives help in extending the functionality of existing HTML elements to give more power to your web application.

# Directives:

They’re very, very critical and a kind of core concept in the AngularJS framework.

# From there we’re going to talk about filtering data and we’ll talk about data binding, so a lot of cool stuff in this particular section.

# What are Directives?

A directive is really a way to teach HTML new tricks.

The web when it first came out was really just designed to display static pages. As we all know it’s become very dynamic and we’ve dealt with that pretty well. jQuery came out many years ago and it provided a way to do it. Even before then we could use raw, vanilla JavaScript.

# Angular takes it up a whole notch and allows us to extend HTML very easily by simply adding attributes, elements or comments

# Using Directives and Data Binding:

# <html >

**<html >**

**<body>**

**<div ng-app>**

**Name:<input type="text" ng-model="sample"/>**

**{{ sample }}**

**</div>**

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

**</body>**

</html>

# <body>

# <div ng-app>

# Name:<input type="text" ng-model="sample"/>

# {{ sample }}

# </div>

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

# </body>

# </html>

Here’s an example of using a very basic, but important, Angular directive.

Notice at the top we have ng-app. Any time you see ng- that is an Angular directive. It’s a built-on directive. You can also write custom ones. You can get third party ones and things like that.

This particular directive is very important because the script that’s now loaded [at the bottom] is going to kick off and this will initialise the Angular app. Right now we don’t have any particular module associated or any other code but we can still do stuff just by adding ng-app.

So for example, this is an example of another directive called ng-model.

What ng-model does is behind the scenes it’s going to add a property up in the memory called “name” into what’s called “the scope”.

If you’ve ever dealt with the concept of a View’s model called a ViewModel - *Knockout* and some other libraries have this concept – then what this is really doing behind the scenes is making an empty ViewModel but then filling it with a name property. Now if I want to write out that value then I can simply come over and add a data binding expression.

Expressions are really cool because if I wanted to put “1 + 1” and try to write out the result I could do that. You can’t put conditional logic in here because you shouldn’t be putting that type of conditional logic in your views. But out of the box, just by adding the ng-app and ng-model with aproperty as they type into this text box I can actually bind to that value and that provides a very cool little feature.

You can see that as we type it automatically binds it, and that’s pretty damned easy, right?

Include the **ng-app** \

Include the **ng-model**

Bind to that model.

# This is pretty primitive and we’re going to go much deeper here, but that’s how we can get started.

The next thing we can do is we can actually iterate through data.

# So I have another directive here called ng-init and this isn’t one I use a lot in real life apps because we’re going to get into controllers and things like that later in the video, but this is going to give me some initialisation data that I want to actually bind to and display so we can come in and use another directive in Angular called ng-repeat.

# 

We’re going to say ng-repeat and then I’m going to give a variable here. For each name in the names variable write out that name.

In this case “name” is not the same thing as I just demonstrated: “name” is just a variable. If I put “foo” here then I would bind to “foo” here.

# This provides a very easy way to duplicate <li>s.

In this case we have four names and so we would get four <li>s with the name written out automatically.

So again, we have the ng-app, the ng-init: these are two directives. Then the third is ng-repeat which will simply loop through all the names, and then data-bind or apply the value into the <li>.

# Let’s look at an example of that really quickly.

# ng-Repeat Example:

# So we can come back into our web page and I’m going to do the ng-init.

# 

# I’m going to give it an array with a couple of names.

# 

This is a primitive way to initialise some variables with data.

# If we come down [below the input] I can do a <ul> and <li> and do an ng-repeat – yet another directive – for each name in names. I’ve already used “name” here:

# 

# So I’m going to call it something different. Let’s say for each personName in names:

# 

# Let’s go ahead and bind to personName:

# 

imply give it the name and it’s going to put that name into the variable when we bind to it

# If we go ahead and run this you’ll see that we just get a nice little list written out, nothing too fancy, but it does work, and if the name binding at the top still works as well.

# The AngularJS API Reference for Directives

# 

It gives us some info. It gives us some different samples of it and a look at how it works.

There’s even some tests [“End to end test” tab under “Source”] on how to test the repeater and do that kind of thing if you’d like as well.

# So there’s a lot of great stuff you can do with directives, and we’ll start to see more of these as we move along.

# Using Filters:

# 

The next thing we can do with Angular is apply filters.

# Let’s say that as we bind to, say a customer name, and we do that process we want to upper-case it. Now I could upper-case it in my data model, which we’ll get to in a little bit later, but an easy way to do this type of thing is to apply an AngularJS filter.

# 

All this will do is this pipe [|] is a separator between the data binding statement and something called a **filter**.

There’s a few filters built-in. We’ll look at that in the documentationonce I get into the demo and run off to the web page. “uppercase” says upper-case it, “lowercase” says lower-case it, you can restrict it if it’s an array and you want to output that array and you want to restrict it, limit it to say three out of the five or whatever it may be.

# Then when it comes to ng-repeat something that’s very cool: in this case we’re going to say for each cust in customers I want to filter by “nameText”.

# 

# Above this we have this ng-model, which we’ve already looked at.

# 

As they type, the value they type will automatically be applied to first filter down the customers based upon what was typed.

So if you type “da” and “dan” was in there then it’ll automatically pick me or any other people that start with “da” or have “da” in the name.

# Then we’re going to take those results and filter again – we’re going to order those results by a “name” property:

# 

What that will ultimately do is if we have 50 customers and we filter for all those that have “John” in the name then all the “John”s would be shown and we would then order those by “John Doe”, “John Smith” and that type of thing.

# What we’re going to do in this case is once that filtering goes through and we order those we’ll then write out the customer name and the customer city.

# 

# Back in our web page I’ve changed the ng-init a little bit. Instead of just having an array of strings, I have an array of objects.

# 

# You’ll notice that each objet has a name and a city property. I just have three of these in here: John Doe, John Smith and Jane Doe, from San Francisco, New York and Phoenix.

# 

# 

# I’m going to have to change this [contents of the body of the page] now. “names” still stays the same, but I’m going to go ahead and change that too. Let’s say this is a list of customers:

# 

# We’ll name it “customers” and we’ll change the <li> statement accordingly to let’s say “cust”:

# 

# Now what I’m going to have to do is write out the cust. – and now we can get into the properties and we can do “name” here:

# 

# Now if I wanted I could put in a space and maybe a dash or something and we could do cust.city and now we’re going to data-bind both those properties.

# 

# I could even come into here, just to show you, I could even do it this way if I wanted, and that would work too.

# 

But I’m going to break these out into two separate data-binding statements. So now we say for each cust in customers let’s go ahead and write out cust.name and cust.city.

# Let me go ahead and just make sure this works and then we’ll apply some filters.

It looks like it does [work]. You can see the cities now being written out, but notice as I type nothing really happens that’s useful or interesting.

# What I’m going to do is come in and let’s do a filter by and whenever they type a name instead of data-binding to it I want to use it as a filter. So we’re going to filter by the name property that’s in our model.

# 

# Let’s go ahead and test this. You’ll notice that when we type “s” the “S” from “San Francisco” also pulls up because I didn’t tell it anything specific – just enter everything:

# You can see all that works and it’s all live. We can also use “orderBy”. Let’s see if we have things in the right area. It looks like if I do “John” and we order by say city then obviously Phoenix is out of place with New York, so we can come in and do another pipe and “orderBy” and then in quotes I give it the property. Let’s order by city and then it will bind those remaining customers and order them by city.

# 

# Let’s go ahead and search for “John” here, and notice that “John” in New York now appears first, and Phoenix follows which of course was not the case with the data.

So that’s an example of applying not only some data-binding and some **ng-repeat** –type of directives, but also how we can apply filters and orderBys and I could even do upper-case if I wanted.

# Let’s say we wanted the name to be uppercase and the city to be lowercase.

# 

Notice though that as I type if I do “John” [using mixed lower-case] it still works. It still does the filters – it still filters and sorts. “Jane” still works and all that.

So these are some of the built-in sorts and filters you get out of the box.

# Again, if you go off to http://angularjs.org , go to “API Reference” and then scroll on down a little bit you’ll see a whole list of the filters.

# 

If I had a number that I wanted to convert into a currency with a $ sign for instance or a £ sign or whatever currency you’re in then I would just say “| currency” and it would automatically do that.

If I had a date and I wanted to format it a certain way, and you can control that by the way, you could do |date. There’s a lot of different things you can do here.

What’s really nice about Angular is not only can we write our own custom directives but I can even write my own custom filters if I want to get a little more advanced with this.

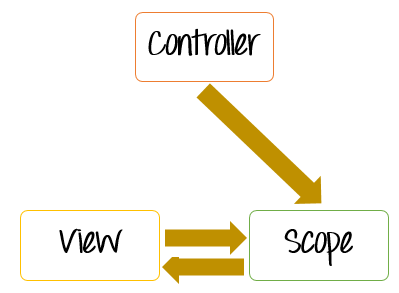
# So, a very powerful framework and we’ve only scratched the surface so far.

# What we’re going to do next is start talking about the MVC part of Angular – the Model, the View, and the Controller, and we’ll see how all that fits in. In this part of the tutorial we’re going to talk about Views, Controllers and a really integral part of Angular called Scope, which is really another term for ViewModel if you’ve used that term before.

# AngularJS Controller:

Controllers will have your core business logic. The primary purpose of controller in Angular.js is to take the data from the View, process the data accordingly and then send that data across to the view which is displayed to the end user.

It will use the data model, carry out the required processing and then pass the output to the view which in turn is displayed to the end user.



* The controller's primary responsibility is to control the data which gets passed to the view. The scope and the view have two way communication.
* The properties of the view can call "functions" on the scope. And events on the view can call "methods" on the scope. The below code snippet gives a simple example of a function.
* The function($scope) which is defined when defining the controller and an internal function which is used to return the concatenation of the $scope.firstName and $scope.lastName.
* In AngularJS when you define a function as a variable, it is known as a Method.

**App.controller(‘ctrl’,function($scope)){**

**$scope.firstName=”Harsha”;**

**$scope.secondName=”Vardhan”;**

**$scope.fullName=function(){**

**return $scope.firstName+””+$scope.lastName;**

**}}**

* Data in this way passes from the controller to the scope and then the data passes back and forth from the scope to the view.
* The scope is used to expose the model to the view. The model can be modified via methods defined in the scope which could be triggered via events from the view. We can define two way model binding from the scope to the model.
* Controllers should not ideally be used for manipulating the DOM. This should be done by the directives which we will see later on.
* Best practice is to have controller's based on functionality. For example, if you have a form for input and you need a controller for that, create a controller called "form controller".

### Building a basic controller:

**<div ng-app=”demo” ng-controller=”MyController”>**

**Name:<input type = “text” ng-model=”details”>**

**</div>**

# Views:

Nowadays, everyone would have heard about the "Single Page Applications". Many of the well-known websites such as Gmail use the concept of Single Page Applications (SPA's).

SPA's is the concept wherein when a user requests for a different page, the application will not navigate to that page but instead display the view of the new page within the existing page itself.

It just gives the feeling to the user that he never left the page in the first place. The same can be achieved in the Angular with the help of Views in conjunction with Routes.

### Usage of views

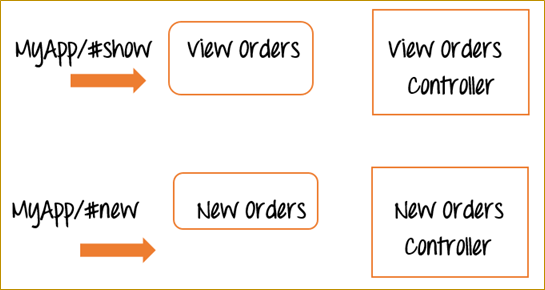
A view is basically the content which is shown to the user. Basically what the user wants to see, accordingly that view of the application will be shown to the user.

The combination of views and Routes helps one in dividing an application in logical views and bind different views to Controllers.

Dividing the application into different views and using Routing to load different part of application helps in logically dividing the application and making it more manageable.

Let's assume that we have an ordering application, wherein a customer can view orders and place new ones.

The below diagram and subsequent explanation demonstrates how to make this application as a single page application.



Now, instead of having two different web pages, one for "View orders" and another for "New Orders", in AngularJS, you would instead create two different views called "View Orders" and "New Orders" in the same page.

We will also have 2 reference links in our application called #show and #new.

* So when the application goes to MyApp/#show, it will show the view of the View Orders, at the same time it will not leave the page. It will just refresh the section of the existing page with the information of "View Orders". The same goes for the "New Orders" view.

So in this way it just becomes simpler to separate the application into different views to make it more manageable and easy to make changes whenever required.

And each view will have a corresponding controller to control the business logic for that functionality.

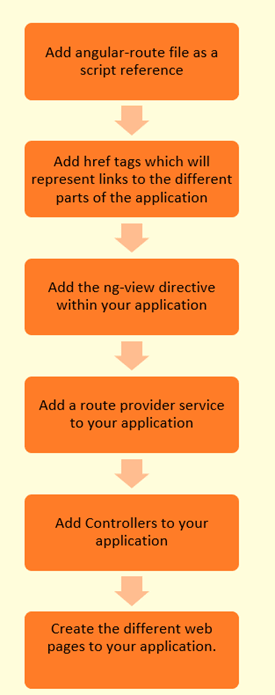
### Angularjs Views

The "ngView" is a directive that complements the $route service by including the rendered template of the current route into the main layout (index.html) file.

Every time the current route changes, the view included changes with it according to the configuration of the $route service without changing the page itself.

We will be covering routes in a later chapter, for now, we will focus on adding multiple views to our application.

Below is the entire flowchart of how the entire process works. We will go through in detail for every process in our example shown below.



### Explain Angular.js Expressions with example

AngularJS expressions are those that are written inside double braces {{expression}}.

**Syntax:**

A simple example of an expression is {{5 + 6}}.

Angular.JS expressions are used to bind data to HTML the same way as the ng-bind directive. AngularJS displays the data exactly at the place where the expression is placed.

Let's look at an example of Angular.JS expressions.

In this example, we just want to show a simple addition of numbers as an expression.

**AngularJS Module**

Modules are one of the most important concepts of Angular JS.

In all of our earlier tutorials, you would have noticed the ng-app directive used to define your main Angular application. This is one of the key concepts of modules in Angular.JS.

Overall modules in Angular have 2 roles to play;

The first is to define the application functionality that is applied to the entire HTML page using the ng-app directive.

The second use of a module is to define functionality, such as services, directives, and filters, in a way that makes it easy to reuse it in different applications.

In the duration of this tutorial, we will look at modules in detail and how they affect the working of Controllers.

### What is a AngularJS module

Before we start off with what is a module, let's look at an example of an AngularJS application without a module and then understand the need of having modules in your application.

Let's consider creating a file called "DemoController.js" and adding the below code to the file

Function Democontroller($scope) {

$scope.a=1;

$scope.b=2;

$scope.c=$scope.b + $scope.a;

});

In the above code, we are created a function called "DemoController" which is going to act as a controller within our application.

In this controller we are just performing the addition of 2 variables a and b and assigning the addition of these variables to a new variable, c, and assigning it back to the scope object.

Now let's create our main Sample.html, which will be our main application. Let's insert the below code snippet in our HTML page.

<body ng-app="">

<div ng-controller="DemoController">

<h3> Global Event</h3>

{{c}}

In the code above, we are including our DemoController and then invoking the value of the $scope.c variable via an expression.

But notice our ng-app directive, it has a blank value.

* This basically means that all controllers which are called within the context of the ng-app directive can be accessed globally. There is no boundary which separates multiple controllers from each other.
* Now in modern day programming, this is a bad practice to have controllers not attached to any modules and making them globally accessible. There has to be some logically boundary defined for controllers.

And this is where modules come in. Modules are used to create that separation of boundaries and assist in separating controllers based on functionality.

Let's change the code above to implement modules and attach our controller to this module

var sampleApp = angular.module('sampleApp',[]);

sampleApp.controller('DemoController', function($scope) {

$scope.a=1;

$scope.b=2;

$scope.c=$scope.b + $scope.a;

});

Let's note the key differences in the code written above

1. var sampleApp = angular.module('sampleApp',[]);

We are specifically creating a AngularJS module called 'sampleApp'. This will form a logical boundary for the functionality that this module will contain. So in our above example, we have a module which contains a controller that performs the role of the addition of 2 scope objects. Hence, we can have one module with a logical boundary which says that this module will only perform the functionality of mathematical calculations for the application.

1. sampleApp.controller('DemoController', function($scope)

We are now attaching the controller to our AngularJS module "SampleApp". This means that if we don't reference the module 'sampleApp' in our main HTML code, we will not be able to reference the functionality of our controller.

Our main HTML code will not look as shown below

<body ng-app="'sampleApp'">

<div ng-controller="DemoController">

<h3> Global Event</h3>

{{c}}

Let's note the key differences in the code written above and our previous code

<body ng-app="'sampleApp'">

In our body tag,

* Instead of having an empty ng-app directive, we are now calling the module sampleApp.
* By calling this application module, we can now access the controller 'DemoController' and the functionality present in the demo controller.

### Modules and Controllers

In Angular.JS, the pattern used for developing modern day web applications is of creating multiple modules and controllers to logically separate multiple levels of functionality.

Normally modules will be stored in separate[Java](http://www.guru99.com/java-tutorial.html)script files, which would be different from the main application file.

Let's look at an example of how this can be achieved.

In the example below,

* We will create a file called Utilities.html which will hold 2 modules, one for performing the functionality of addition and the other for performing the functionality of subtraction.
* We are then going to create 2 separate application files and access the Utilities file from each application file.
* In one application file we will access the module for addition and in the other we will access the module for subtraction.

**Step1)** Define the code for the multiple modules and controllers.

var AdditionApp = angular.module('AdditionApp',[]);

AdditionApp.controller('DemoAddController', function($scope) {

$scope.a=5;

$scope.b=6;

$scope.c=$scope.a + $scope.b;

});

var SubractionApp = angular.module('SubtractionApp',[]);

SubractionApp.controller('DemoSubtractController', function($scope) {

$scope.a=8;

$scope.b=6;

$scope.d=$scope.a - $scope.b;

});

Let's note the key points in the code written above

var AdditionApp = angular.module('AdditionApp',[]);

var SubractionApp = angular.module('SubtractionApp',[]);

1. There are 2 separate Angular Module created, one which is given the name 'AdditionApp' and the second one is given the name 'SubtractionApp'.
2. AdditionApp.controller('DemoAddController', function($scope)
3. SubractionApp.controller('DemoSubtractController', function($scope)

There are 2 separate controllers defined for each module , one is called the DemoAddController and the other is the DemoSubtractController. Each controller has separate logic for addition and subtraction of numbers.

**Step2)** Create your main application files. Let's create a file called ApplicationAddition.html and add the below code

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Addition</title>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<script src="/lib/Utilities.js"></script>

</head>

<body>

<div ng-app = "AdditionApp" ng-controller="DemoAddController">

{{c}}

</div>

</script>

</body>

</html>

Let's note the key points in the code written above

1. <script src="/lib/Utilities.js"></script>

We are referencing our Utilities.js file in our main application file. This allows us to reference any AngularJS modules defined in this file.

1. <div ng-app = "AdditionApp" ng-controller="DemoAddController">

We are accessing the 'AdditionApp' module and DemoAddController by using the ng-app directive and the ng-controller respectively.

1. {{c}}

Since we are referencing the above mentioned module and controller we are able to reference the $scope.c variable via an expression. The expression will be the result of the addition of the 2 scope variables a and b which was carried out in the 'DemoAddController' Controller

The same way we will do for subtraction function.

**Step3)** Create your main application files. Let's create a file called "ApplicationSubtraction.html" and add the below code

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Subtraction</title>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<script src="/lib/Utilities.js"></script>

</head>

<body>

<div ng-app = "SubtractionApp" ng-controller="DemoSubtractController">

{{d}}

</div>

</script>

</body>

</html>

Let's note the key points in the code written above

1. <script src="/lib/Utilities.js"></script>

We are referencing our Utilities.js file in our main application file. This allows us to reference any modules defined in this file.

1. <div ng-app = " SubtractionApp " ng-controller=" DemoSubtractController ">

We are accessing the 'SubtractionApp module and DemoSubtractController by using the ng-app directive and the ng-controller respectively.

1. {{d}}

Since we are referencing the above mentioned module and controller we are able to reference the $scope.d variable via an expression. The expression will be the result of the subtraction of the 2 scope variables a and b which was carried out in the 'DemoSubtractController' Controller

# AngularJS Events:

When creating web based applications, sooner or later your application will need to handle DOM events like mouse clicks, moves, keyboard presses, change events etc.

AngularJS has the ability to add functionality which can be used to handle such events.

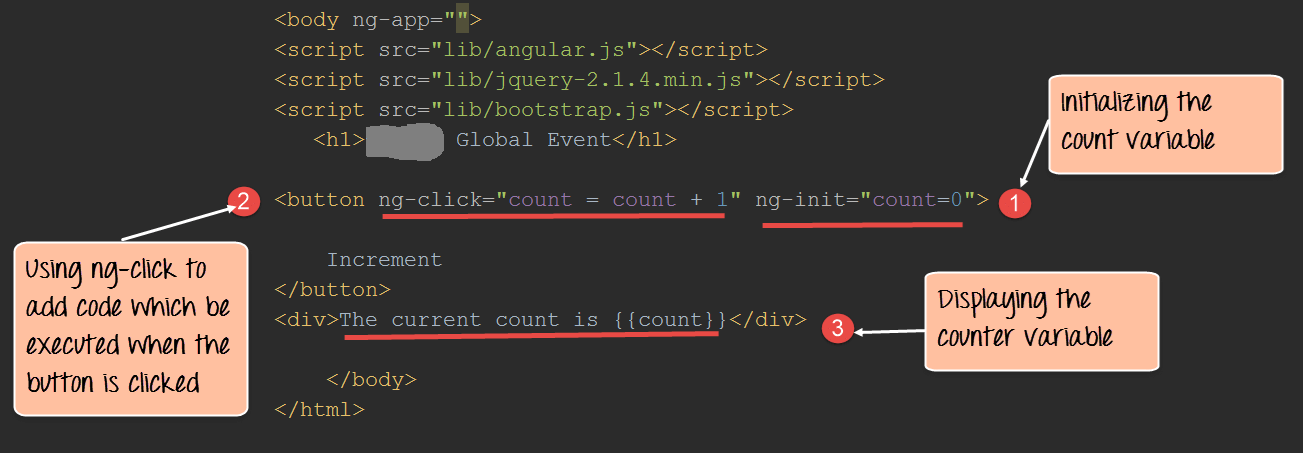
For example, if there is a button on the page and you want to process something when the button is clicked, we can use the ng-click event directive.

### The ng-click directive

The "ng-click directive" is used to apply custom behavior to when an element in HTML clicked. This is normally used for buttons because that is the most common place for adding events which respond to clicks performed by the user.

Let's look a simple example of how we can implement the click event.

In this example, we will have a counter variable which will increment in value when the user clicks a button.



**Code Explanation:**

1. We are first using the ng-init directive to set the value of a local variable count to 0.
2. We are then introducing the ng-click event directive to the button. In this directive, we are writing code to increment the value of the count variable by 1.
3. Here we are displaying the value of the count variable to the user.

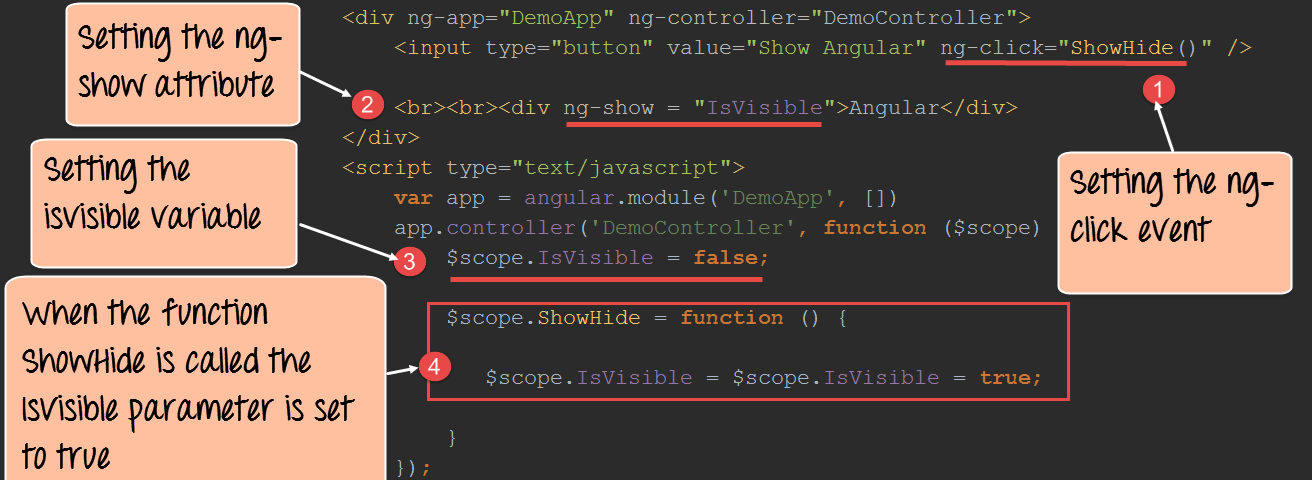
If the code is executed successfully, the following Output will be shown when you run your code in the browser.

### Showing HTML Elements using ng-show

The ngShow directive is used to show or hide a given HTML element based on the expression provided to the ngShow attribute.

In the background, the element is shown or hidden by removing or adding the .ng-hide CSS class onto the element.

Let's look at an example of how we can use the "ngshow event" directive to display a hidden element.



**Code Explanation:**

1. We are attaching the ng-click event directive to the button element. Over here we are referencing a function called "ShowHide" which is defined in our controller – DemoController.
2. We are attaching the ng-show attribute to a div tag which contains the text Angular. This is the tag which we are going to show/hide based on the ng-show attribute.
3. In the controller, we are attaching the "IsVisible" member variable to the scope object. This attribute will be passed to the ng-show angular attribute (step#2) to control the visibility of the div control. We are initially setting this to false, so that when the page is first displayed the div tag will be hidden.

**Note:-** When the attributes ng-show is set to true, the subsequent control which in our case is the div tag will be shown to the user. When the ng-show attribute is set to false the control will be hidden from the user.

1. We are adding code to the ShowHide function which will set the IsVisible member variable to true so that the div tag can be shown to the user.

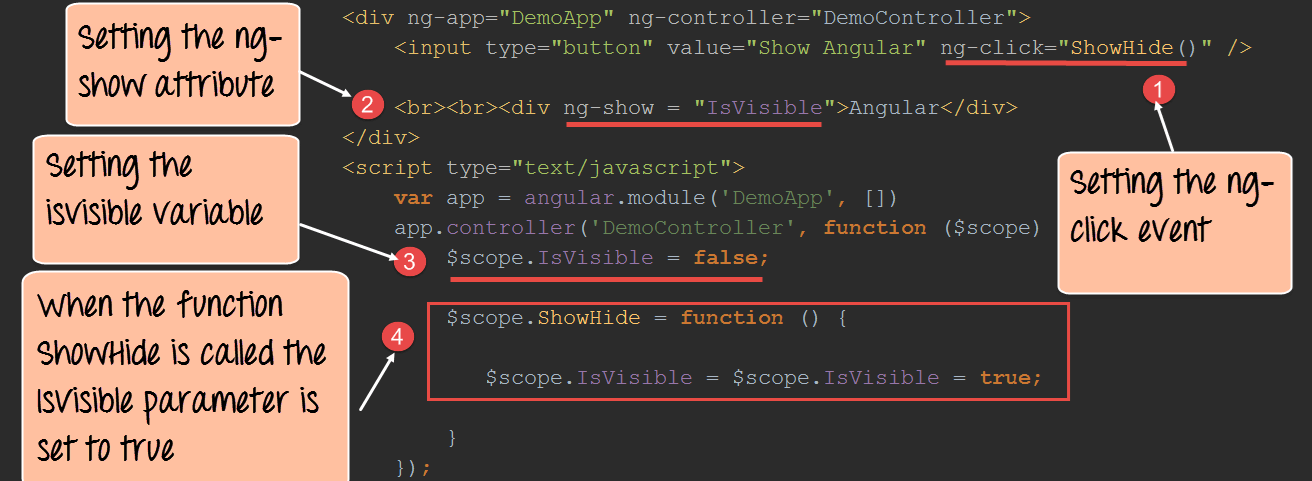
If the code is executed successfully, the following Output will be shown when you run your code in the browser.

### Hiding HTML Elements using ng-hide

Just like the ngShow directive, there is also the ng-hide directive. With ng-show the element is shown if the expression is true, it will hide if it is false.

On the other hand with ng-hide the element is hidden if the expression is true and it will be shown if it is false.

Let's look at the similar example as the one for shown for ngShow to showcase how the ngHide attribute can be used.



**Code Explanation:**

1. We are attaching the ng-click event directive to the button element. Over here we are referencing a function called ShowHide which is defined in our controller – DemoController.
2. We are attaching the ng-hide attribute to a div tag which contains the text Angular. This is the tag, which we are going to show/hide based on the ng-show attribute.
3. In the controller, we are attaching the isVisible member variable to the scope object. This attribute will be passed to the ng-show angular attribute to control the visibility of the div control. We are initially setting this to false, so that when the page is first displayed the div tag will be hidden.
4. We are adding code to the ShowHide function which will set the IsVisible member variable to true so that the div tag can be shown to the user.

If the code is executed successfully, the following Output will be shown when you run your code in the browser.

# AngularJS Routes ($route):

Before we start with routing, let's just have a quick overview on Single-Page Applications.

Single page applications or (SPAs) are web applications that load a single HTML page and dynamically update the page based on the user interaction with the web application.

SPAs are now practically implemented in every modern web page application. In AngularJS, routing is what allows you to create Single Page Applications.

* AngularJS routes enable you to create different URLs for different content in your application.
* AngularJS routes allow one to show multiple content depending on which route is chosen.
* A route is specified in the URL after the # sign.

Let's take an example of a site which is hosted via the URL **http://example.com/index.html**.

On this page you would host the main page of your application. Suppose if the application was organizing an Event and one wanted to see what the various events on display are, or wanted to see the details of a particular event or delete an event. In a Single Page application, when routing is enabled, all of this functionality would be available via the following links

**http://example.com/index.html#ShowEvent**

**http://example.com/index.html#DisplayEvent**

**http://example.com/index.html#DeleteEvent**

The # symbol would be used along with the different routes (ShowEvent, DisplayEvent, and DeleteEvent).

* So if the user wanted to see all Events, they would be directed to the link (**http://example.com/index.html#ShowEvent**), else
* If they wanted to just see a particular event they would be re-directed to the link ( **http://example.com/index.html#DisplayEvent**) or
* If they wanted to delete an event, they would be directed to the link **http://example.com/index.html#DeleteEvent**.

Note that the main URL stays the same.

### Adding angular route:

So as we discussed earlier, routes in AngularJS are used to route the user to a different view of your application. And this routing is done on the same HTML page so that the user has the experience that he has not left the page.

In order to implement routing the following main steps have to be implemented in your application in any specific order.

1. Reference to angular-route.js. This is a[JavaScript](http://www.guru99.com/interactive-javascript-tutorials.html)file developed by Google that has all the functionality of routing. This needs to be placed in your application so that it can reference all of the main modules which are required for routing.
2. The next important step is to add a dependency to the ngRoute module from within your application. This dependency is required so that routing functionality can be used within the application. If this dependency is not added, then one will not be able to use routing within the angular.JS application.

Below is the general syntax of this statement. This is just a normal declaration of a module with the inclusion of the ngRoute keyword.

var module = angular.module("sampleApp", ['ngRoute']);

1. The next step would be to configure your $routeProvider. This is required for providing the various routes in your application.

Below is the general syntax of this statement which is very self-explanatory. It just states that when the relevant path is chosen, use the route to display the given view to the user.

when(path, route)

1. Links to your route from within your HTML page. In your HTML page, you will add reference links to the various available routes in your application.

<a href="#/route1">Route 1</a><br/>

1. Finally would be the inclusion of the ng-view directive, which would normally be in a div tag. This would be used to inject the content of the view when the relevant route is chosen.

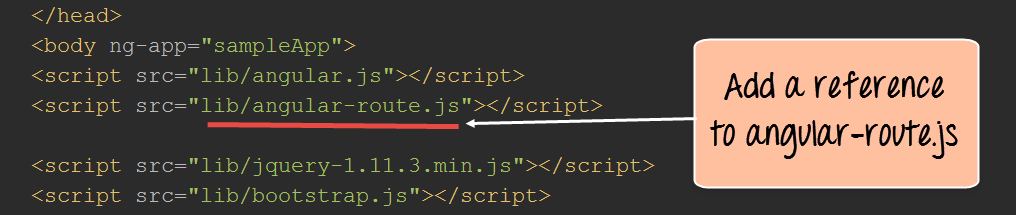
Now, let's look at an example of routing using the above-mentioned steps.

In our example,

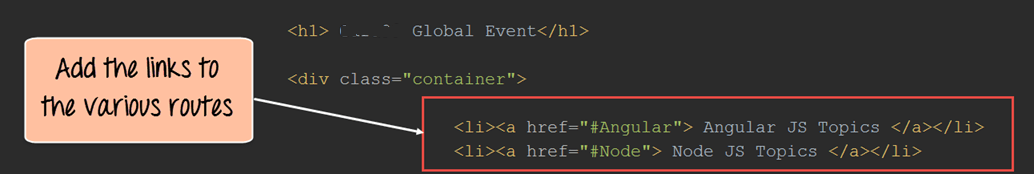
We will present 2 links to the user,

* One is to display the topics for an **Angular JS** course, and the other is for the **Node.js** course.
* When the user clicks either link, the topics for that course will be displayed.

**Step 1)** Include the angular-route file as a script reference.

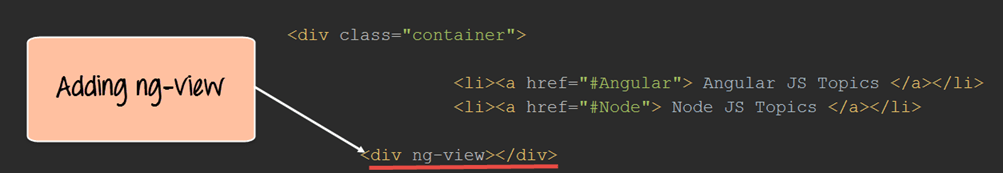


This route file is necessary in order to make use of the functionalities of having multiple routes and views. This file can be downloaded from the angular.JS website.

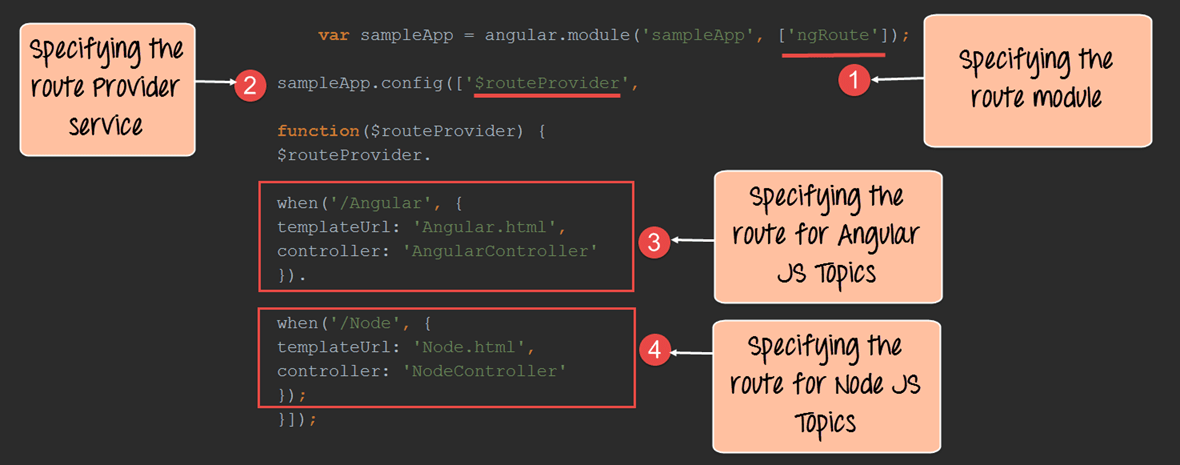
**Step 2)**Add href tags which will represent links to "Angular JS Topics" and "Node JS Topics."

**Step3)**Add a div tag with the ng-view directive which will represent the view.

This will allow the corresponding view to be injected whenever the user clicks on either the "Angular JS Topics" or "Node JS Topics."



**Step 4)** In your script tag for AngularJS, add the "ngRoute module" and the "$routeProvider" service.

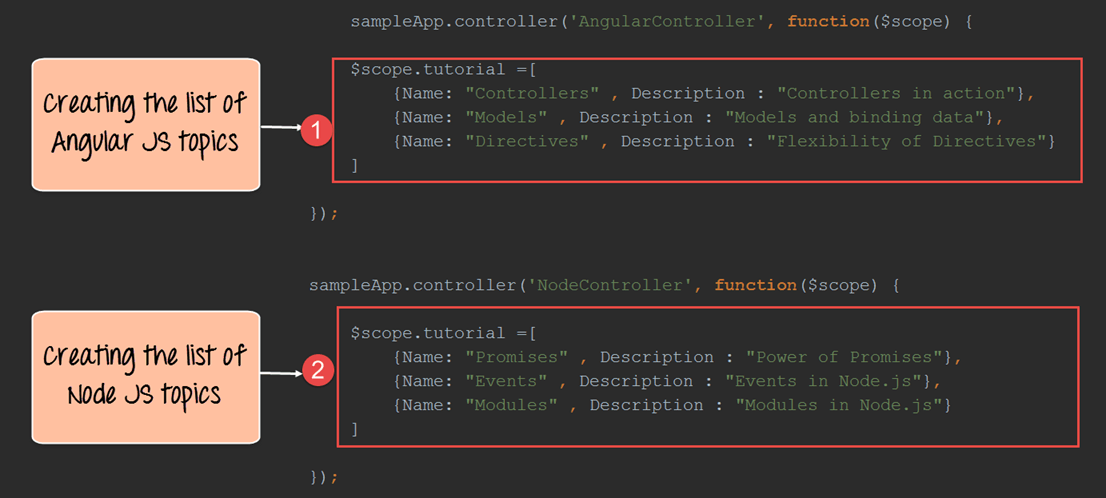


**Code Explanation:**

1. The first step is to ensure to include the "ngRoute module." With this in place, Angular will automatically handle the routing in your application. The ngRoute module which is developed by Google has all of the functionality which allows for routing to be possible. By adding this module, the application will automatically understand all of the routing commands.
2. The $routeprovider is a service that angular uses to listen in the background to the routes which are called. So when the user clicks a link, the routeprovider will detect this and then decide on which route to take.
3. Create one route for the Angular link – This block means that when the Angular link is clicked, inject the file Angular.html and also use the Controller 'AngularController' to process any business logic.
4. Create one route for the Node link – This block means that when the Node link is clicked, inject the file Node.html and also use the Controller 'NodeController' to process any business logic.

**Step 5)** Next is to add controllers to process the business logic for both the AngularController and NodeController.

In each controller, we are creating an array of key-values pairs to store the Topic names and descriptions for each course. The array variable 'tutorial' is added to the scope object for each controller.

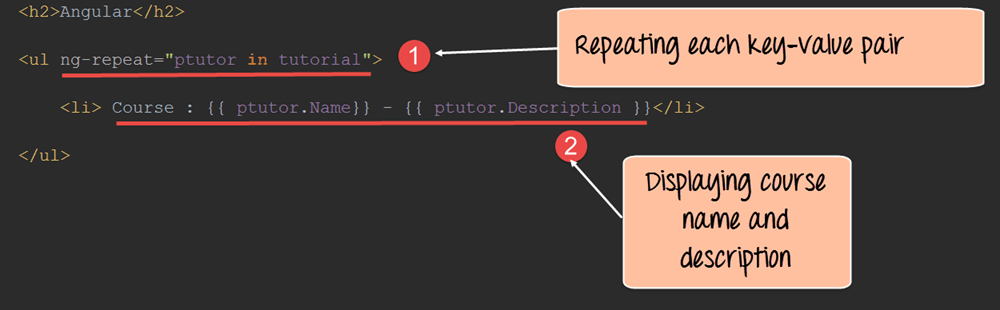


**Step 6)** Create pages called Angular.html and Node.html. For each page we are carrying out the below steps.

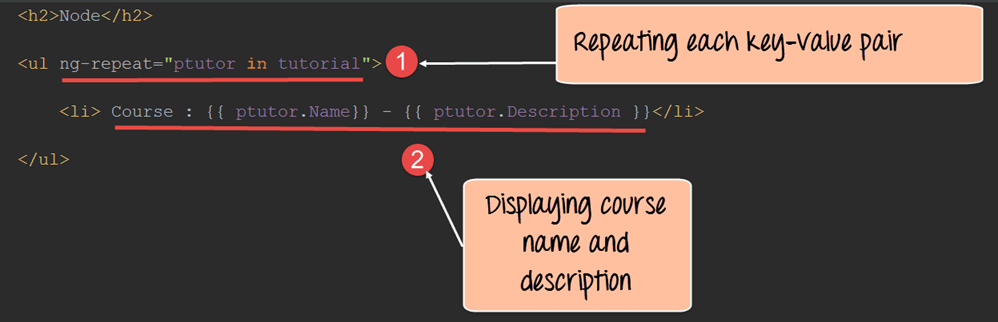
These steps will ensure that all of the key-value pairs of the array are displayed on each page.

1. Using the ng-repeat directive to go through each key-value pair defined in the tutorial variable.
2. Displaying the name and description of each key-value pair.

* **Angular.html**



* **Node.html**



### Creating a default route

Routing in AngularJS also provides the facility to have a default route. This is the route which is chosen if there is no match for the existing route.

The default route is created by adding the following condition when defining the $routeProvider service.

The below syntax just simply means to redirect to a different page if any of the existing routes doesn't match.

otherwise ({

redirectTo: 'page'

});

Let's use the same example above and add a default route to our $routeProvider service.

# C:\Users\Harsha\Desktop\010616_0524_AngularJSRo11.png

**Code Explanation:**

1. Here we are using the same code as above with the only difference is that we are using the otherwise statement and the "redirectTo" option to specify which view should be loaded if no route is specified. In our case we want the '/Angular' view to be shown.

### Accessing parameters from the route

Angular also provides the functionality to provide parameters during routing. The parameters are added to the end of the route in the URL, for example, **http://example/index.html#/Angular/1**. In this example

1. **http://example/index.html** is our main application URL
2. The # symbol is the separator between the main application URL and the route.
3. Angular is our route
4. And finally '1' is the parameter which is added to our route

The syntax of how parameters look in the URL is shown below:

**HTMLPage#/route/parameter**

Here you will notice that the parameter is passed after the route in the URL.

So in our example, above for the Angular JS topics, we can pass a parameter's as shown below

Sample.html#/Angular/1

Sample.html#/Angular/2

Sample.html#/Angular/3

Here the parameters of 1, 2 and 3 can actually represent the topicid.

Let's look in detail of how we can implement this.

**Step 1)**Add the following code to your view

1. Add a table to show all the topics for the Angular JS course to the user
2. Add a table row for showing the topic "Controllers." For this row, change the href tag to "Angular/1" which means that when the user clicks this topic, the parameter 1 will be passed in the URL along with the route.
3. Add a table row for showing the topic "Models." For this row, change the href tag to "Angular/2" which means that when the user clicks this topic, the parameter 2 will be passed in the URL along with the route.
4. Add a table row for showing the topic "Directives." For this row, change the href tag to "Angular/3" which means that when the user clicks this topic, the parameter 3 will be passed in the URL along with the route.

# C:\Users\Harsha\Desktop\010616_0524_AngularJSRo13.png

# **Step 2)** In the routeprovider service function add the:topicId to denote that any parameter passed in the URL after the route should be assigned to the variable topicId.C:\Users\Harsha\Desktop\010616_0524_AngularJSRo14.png

**Step 3)**Add the necessary code to the controller

1. Make sure to first add the "$routeParams" as a parameter when defining the controller function. This parameter will have access to all of the route parameters passed in the URL.
2. The "routeParams" parameter has a reference to the topicId object, which is passed as a route parameter. Here we are attaching the '$routeParams.topicId' variable to our scope object as the variable '$scope.tutotialid'. This is being done so that it can be referenced in our view via the tutorialid variable.

# C:\Users\Harsha\Desktop\010616_0524_AngularJSRo15.png

# **Step 4)** Add the expression to display the tutorialid variable in the Angular.htmlC:\Users\Harsha\Desktop\010616_0524_AngularJSRo16.png

### Using Angular $route service

The $route service allows you to access the properties of the route. The $route service is available as a parameter when the function is defined in the controller. The general syntax of how the $route parameter is available from the controller is shown below;

myApp.controller('MyController',function($scope,$route)

1. myApp is the angular.JS module defined for your applications
2. MyController is the name of the controller defined for your application
3. Just like the $scope variable is made available for your application, which is used to pass information from the controller to the view. The $route parameter is used to access the properties of the route.

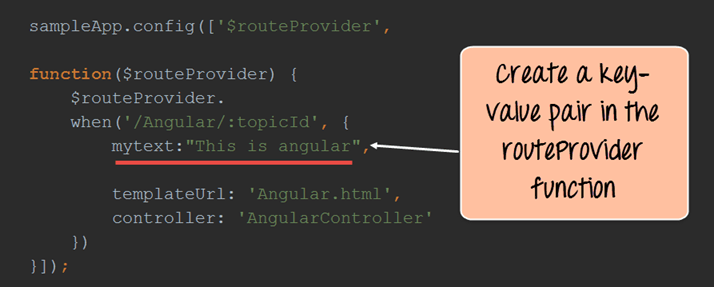
Let's have a look on how we can use the $route service.

In this example,

* We are going to create a simple custom variable called "mytext," which will contain the string "This is angular."
* We are going to attach this variable to our route. And later we are going to access this string from our controller using the $route service and then subsequently use the scope object to display that in our view.

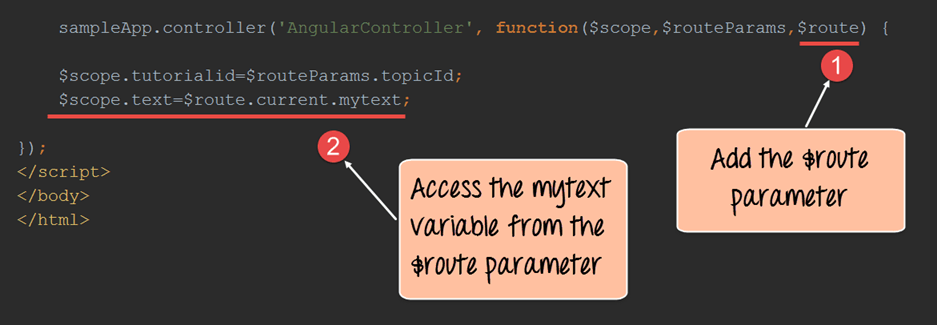
So, let's see the steps which we need to carry out to achieve this.

**Step 1)**Add a custom key-value pair to the route. Here, we are adding a key called 'mytext' and assigning it a value of "This is angular."



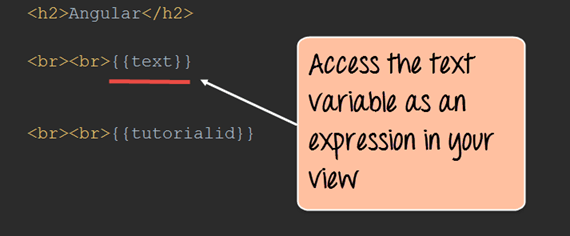
**Step 2)**Add the relevant code to the controller

1. Add the $route parameter to the controller function. The $route parameter is a key parameter defined in angular, which allows one to access the properties of the route.
2. The "mytext" variable which was defined in the route can be accessed via the $route.current reference. This is then assigned to the 'text' variable of the scope object. The text variable can then be accessed from the view accordingly.



**Step 3)**Add a reference to the text variable from the scope object as an expression. This will be added to our Angular.html page as shown below.

This will cause the text "This is angular" to be injected into the view. The {{tutorialid}} expression is the same as that seen in the previous topic and this will display the number '1'.



# AngularJS http, Ajax:

AJAX is the short form of Asynchronous[JavaScript](http://www.guru99.com/interactive-javascript-tutorials.html)and XML. AJAX was primarily designed for updating parts of a web page, without reloading the whole page.

The reason for designing this technology was to reduce the number of round trips which were made between the client and the server and the number of entire page refresh which used to take place whenever a user required certain information.

AJAX allowed web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This meant that it was possible to update parts of a web page, without reloading the whole page.

Many modern day web applications actually follow this technique for refreshing the page or getting data from the server.

### High level interactions with servers using $resource

Angular provides two different APIs to handle Ajax requests. They are

* $resource
* $http

We will look at the "$resource" property in Angular, which is used to interact with servers at a high level.

When we talk about interacting at a higher level, it means that we will only be bothered about what the server has to offer in terms of functionality, and not about what exactly the server does in detail with regards to this functionality.

For example, if the server was hosting an application which maintains the employee information of a certain company, the server might expose functionality to clients such as GetEmployeeDetails, SetEmployeeDetails, etc.

So at a high level, we know what these 2 functions can do, and we can invoke them using the $resource property. But then we don't know exactly the details of the "GetEmployeeDetails" and the "SetEmployeeDetails functions", and how it is implemented.

The above explanation explains what is known as a REST based architecture.

* REST is basically known as Representational State Transfer, which is an architecture followed in many modern web based systems.
* It means that you can use the normal HTTP verbs of GET, POST, PUT and DELETE to work with a web-based application.

So let's assume, we have a web application that maintains a list of Events.

If we wanted to get to the list of all of the events,

* The web application can expose a URL such as **http://example/events** and
* We can use the "GET" verb to get the entire list of events if the application is following a REST based architecture.

So for example, if there was an Event with an ID of 1, then we can get the details of this event via the URL. **http://example/events/1**

**What is the $resource object**

The $resource object in Angular helps in working with servers that serve applications on a REST based architecture.

The basic syntax of the @resource statement along with the various functions are given below

**Syntax of @resource statement**

var resource Object = $resource(url);

Once you have the resourceObject at hand, you can use the below functions to make the required REST calls.

1. get([params], [success], [error]) – This can be used to make the standard GET call.

2. save([params], postData, [success], [error]) - This can be used to make the standard POST call.

3. query([params], [success], [error]) - This can be used to make the standard GET call, but an array is returned as part of the response.

4. remove([params], postData, [success], [error]) - This can be used to make the standard DELETE call.

In all of the functions given below the 'params' parameter can be used to provide the required parameters, which need to be passed in the URL.

For example,

* Suppose you pass a value of Topic: 'Angular' as a 'param' in the get function as
* get('**http://example/events**' ,'{ Topic: 'Angular' }')
* The URL **http://example/events?Topic=Angular** gets invoked as part of the get function.

**How to use $resource property**

In order to use the $resource property, the following steps need to be followed:

**Step 1)**The file "angular-resource.js" needs to be downloaded from the Angular.JS site and has to place in the application.

**Step 2)** The application module should declare a dependency on the "ngResource" module in order to use the $resource.

In the following example, we are calling the "ngResource" module from our 'DemoApp' application.

angular.module(DemoApp,['ngResource']); //DemoApp is our main module

**Step 3)** Calling the $resource() function with your REST endpoint, as shown in the following example.

If you do this, then the $resource object will have the ability to invoke the necessary functionality exposed by the REST endpoints.

The following example calls the endpoint url: **http://example/events/1**

angular.module('DemoApp.services').factory('Entry', function($resource)

{

return $resource('/example/Event/:1); // Note the full endpoint address

});

In the example above the following things are being done

1. When defining the Angular application, a service is being created by using the statement 'DemoApp.services' where DemoApp is the name given to our Angular application.
2. The .factory method is used to create the details of this new service.
3. 'Entry' is the name we are giving to our service which can be any name you want to provide.
4. In this service, we are creating a function which is going to call the $resource api
5. $resource('/example/Event/:1).

The $resource function is a service which is used to call a REST endpoint. The REST endpoint is normally a URL. In the above function, we are using the URL (/example /Event/:1) as our REST endpoint. Our REST endpoint(/example /Event/:1) basically denotes that there is an Event application sitting on our main site "example". This Event application is developed by using a REST based architecture.

1. The result of the function call is a resource class object. The resource object will now have the functions ( get() , query() , save() , remove(), delete() ) which can be invoked.

**Step 4)** We can now use the methods which were returned in the above step( which are the get() , query() , save() , remove(), delete() ) in our controller.

In the below code snippet , we are using the get() function as an example

Let's look at the code which can make use of the get() function.

angular.module('DemoApp.controllers',[]);

angular.module('DemoApp.controllers').controller('DemoController',function($scope, MyFunction) {

var obj = MyFunction.get({ 1: $scope.id }, function() {

console.log(obj);

});

In the above step,

* The get() function in the above snippet issues a GET request to / example /Event/:1.
* The parameter:1 in the URL is replaced with $scope.id.
* The function get() will return an empty object which is populated automatically when the actual data comes from the server.
* The second argument to get() is a callback which is executed when the data arrives from the server. The possible output of the entire code would be a JSON object which would return the list of Events exposed from the "example" website.

An example of what can be returned is shown below

{

{ 'EventName' : 'Angular , 'EventDescription' : 'Angular Basics'},

{ 'EventName' : 'Node , 'EventDescription' : 'Node Basics'},

{ 'EventName' : 'Programming in C++ , 'EventDescription' : 'C++ Basics'}

}

### Low-level server interactions with $http

The $http is another Angular JS service which is used to read data from remote servers. The most popular form of data which is read from servers is data in the JSON format.

Let's assume, that we have a[PHP](http://www.guru99.com/php-tutorials.html)page ( **http://example/angular/getTopics.PHP** ) that returns the following JSON data

"Topics": [

{

"Name" : "Controllers",

"Description" : "Controllers in action"

},

{

"Name" : "Models",

"Description" : "Binding data using Models"

},

{

"Name" : "Directives",

"Description" : "Using directives in Angular"

}

]

Let's look at the AngularJS code using $http, which can be used to get the above data from the server

<script>

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

app.controller('AngularCtrl', function($scope, $http) {

$http.get("[http://example/angular/getTopics.PHP](http://examplexyz/angular/getTopics.PHP)")

.then(function(response)

{

$scope.names = response.data.records;});

});

</script>

In the above example

1. We are adding the $http service to our Controller function, so that we can use the "get" function of the $http service.
2. We are now using the get function from the HTTP service to get the JSON objects from the URL **http://example /angular/getTopics.PHP**
3. Based on the 'response' object, we are creating a callback function which will return the data records and subsequently we are storing them in the $scope object.
4. We can then use the $scope.names variable from the controller and use it in our view to display the JSON objects accordingly.

### Fetching data from server running SQL and MySQL

Angular can also be used to fetch data from a server running MySQL or SQL.

The idea is that if you have a PHP page connecting to a MySQL database or an[Asp.Net](http://www.guru99.com/asp-net-tutorial.html)page connecting to an MS[SQL](http://www.guru99.com/sql.html)server database then you need to ensure both the PHP and ASP.Net page renders the data in JSON format.

Let's assume, we have a PHP site (**http://example /angular/getTopics.PHP**) serving data from either a MySQL or SQL database.

**Step 1)** The first step is to ensure that the PHP page takes the data from a MySQL database and serves the data in JSON format.

**Step 2)** Write the similar code shown above to use the $http service to get the JSON data.

Let's look at the AngularJS code using $http which can be used to get the above data from the server

<script>

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

app.controller('AngularCtrl', function($scope, $http) {

$http.get("http://example /angular/getTopics.PHP")

.then(function(response)

{

$scope.topics = response.data.records;});

});

</script>

**Step 3)** Render the data in your view using the ng-repeat directive.

Below we are using the ng-repeat directive to go through each of the key-value pairs in the JSON objects returned by the "get" method of the $http service.

For each JSON object , we are displaying the key which is "Name" and the value is "Description".

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

<table>

<tr ng-repeat="x in topics">

<td>{{ x.Name }}</td>

<td>{{ x.Description }}</td>

</tr>

</table>

</div>

# ng-include in AngularJS:

By default, HTML does not provide the facility to include client side code from other files. It's normally a good practice in any programming language to distribute functionality across various files for any application.

For example, if you had logic for numeric operations, you would normally want to have that functionality defined in one separate file. That separate file could then be re-used across multiple applications by just including that file.

This is normally the concept of **Include statements** which are available in programming languages such as .Net and Java.

This chapters looks at other ways files (files which contain external HTML code) can be included in the main HTML file.

### Client Side includes

One of the most common ways is to include HTML code is via Javascript.[JavaScript](http://www.guru99.com/interactive-javascript-tutorials.html)is a programming language which can be used to manipulate the content in an HTML page on the fly. Hence, Javascript can also be used to include code from other files.

The below steps shows how this can be accomplished.

**Step1)** Define a file called Sub.html and add the following code to the file.

<div>

This is an included file

</div>

**Step 2)**Create a file called Sample.html, which is your main application file and add the below code snippet.

Below are the main aspects to note about the below code,

1. In the body tag, there is a div tag which has an id of Content. This is the place where the code from the external file 'Sub.html' will be inserted.
2. There is a reference to a jquery script. JQuery is a scripting language built on top of Javascript which makes DOM manipulation even easier.
3. In the Javascript function, there is a statement '$("#Content").load("Sub.html");' which causes the code in the file Sub.html to be injected in the div tag which has the id of Content.
4. <html>
5. <head>
6. <script src="/jquery.js"></script>
7. <script>
8. $(function(){
9. $("#Content").load("Sub.html");
10. });
11. </script>
12. </head>
13. <body>
14. <div id="Content"></div>
15. </body>

</html>

### Server Side Includes

Server Side Includes are also available for including a common piece of code throughout a site. This is normally done for including content in the below parts of an HTML document.

1. Page header
2. Page footer
3. Navigation menu.

In order for a web server to recognize a Server Side Includes, the file names have special extensions. They are usually accepted by the web server such as .shtml, .stm, .shtm , .cgi.

The directive used for including content is the "include directive". An example of the include directive is shown below;

<!--#include virtual="navigation.cgi" -->

* The above directive allows the content of one document to be included in another.
* The "virtual" command in above code is used to specify the target relative to the domain root of the application.
* In addition, to the virtual parameter, there is also the file parameter which can be used. The "file" parameters are used when one needs to specify the path relative to the directory of the current file.

**Note:**

The virtual parameter is used to specify the file (HTML page, text file, script, etc.) that needs to be included. If the web server process does not have access to read the file or execute the script, the include command will fail. The 'virtual' word is a keyword that is required to be placed in the include directive.

### AngularJS Includes

Angular provides the function to include the functionality from other AngularJS files by using the ng-include directive.

* The primary purpose of the "ng-include directive" is used to fetch, compile and include an external HTML fragment in the main AngularJS application.

Let's look at the below code base and explain how this can be achieved using Angular.

**Step 1)** let's write the below code in a file called Table.html. This is the file which will be injected in our main application file using the ng-include directive.

The below code snippet assumes that there is a scope variable called "tutorial." It then uses the ng-repeat directive, which goes through each topic in the "Tutorial" variable and displays the values for the 'name' and 'description' key-value pair.

<table>

<tr ng-repeat="Topic in tutorial">

<td>{{ Topic.Name }}</td>

<td>{{ Topic.Country }}</td>

</tr>

</table>

**Step 2)** let's write the below code in a file called Main.html. This is a simple angular.JS application which has the following aspects

1. Use the "ng-include directive" to inject the code in the external file 'Table.html'. The statement has been highlighted in bold in the below code. So the div tag ' **<div ng-include="Table.html"></div>'**will be replaced by the entire code in the 'Table.html' file.
2. In the controller, a "tutorial" variable is created as part of the $scope object. This variable contains a list of key-value pairs.

In our example, the key value pairs are

1. Name – This denotes the name of a topic such as Controllers, Models, and Directives.
2. Description – This gives a description for each topic

The tutorial variable is also accessed in the 'Table.html' file.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Event Registration</title>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

</head>

<body ng-app="sampleApp">

<div ng-controller="AngularController">

<h3> Global Event</h3>

<div ng-include="Table.html"></div>

</div>

<script>

var sampleApp = angular.module('sampleApp',[]);

sampleApp.controller('AngularController', function($scope) {

$scope.tutorial =[

{Name: "Controllers" , Description : "Controllers in action"},

{Name: "Models" , Description : "Models and binding data"},

{Name: "Directives" , Description : "Flexibility of Directives"}

];

});

</script>

</body>

</html>

# AngularJS - Dependency Injection:

In software engineering, dependency injection is a software design pattern that implements inversion of control for resolving dependencies.

**Inversion of Control**: It means that objects do not create other objects on which they rely to do their work. Instead, they get these objects from an outside source. This actually forms the basis of dependency injection wherein if one object is dependent on another, the primary object does not take the responsibility of creating the dependent object and then use its methods. Instead, an external source (which in AngularJS, is the AngularJS framework itself) creates the dependent object and gives it to the source object for further usage.

So let's first understand what a dependency is.

# C:\Users\Harsha\Desktop\010616_1118_AngularJSDe1.png

The above diagram shows a simple example of an everyday ritual in database programming.

* The 'Model' box depicts the "Model class" which is normally created to interact with the database. So now the database is actually a dependency for the "Model class" to function.
* By dependency injection, we actually create a service to grab all the information from the database and get into the model class.

In the remainder of this tutorial, we will look more at dependency injection and how this is accomplished in AngularJS.

### What is dependency injection

In Angular.JS, dependencies are injected by using an "injectable factory method" or "constructor function".

These components can be injected with "service" and "value" components as dependencies. We have actually seen this in an earlier topic with the $http service.

We've already seen that the $http service can be used within AngularJS to get data from a MySQL or MS[SQL](http://www.guru99.com/sql.html)Server database via a[PHP](http://www.guru99.com/php-tutorials.html)web application.

The $http service is normally defined from within the controller in the following manner.

sampleApp.controller ('AngularJSController', function ($scope, $http)

Now when the $http service is defined in the controller as shown above. It means that the controller now has a dependency on the $http service.

So when the above code gets executed, AngularJS will perform the following steps;

1. Check to see if the "$http service" has been instantiated. Since our "controller" now depends on the "$http service", an object of this service needs to be made available to our controller.
2. If AngularJS finds out that the $http service is not instantiated, AngularJS uses the 'factory' function to construct an $http object.
3. The injector within Angular.JS then provides an instance of the $http service to our controller for further processing.

Now that the dependency is injected in our controller, we can now invoke the necessary functionality within the $http service for further processing.

### Example of Dependency Injection:

Dependency injection can be implemented in 2 ways

1. One is through the "Value Component"
2. Another is through a "Service"

Let's look at the implementation of both ways in more detail.

1. **Value component** – This concept is based on the fact of creating a simple[JavaScript](http://www.guru99.com/interactive-javascript-tutorials.html)object and pass it to the controller for further processing.

This is implemented using the below 2 steps

**Step 1)** Create a JavaScript object by using the value component and attach it to your main AngularJS.JS module.

The value component takes on 2 parameters; one is the key, and the other is the value of the javascript object which is created.

**Step 2)** Access the JavaScript object from the Angular.JS controller

<! DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Event Registration</title>

<script src="http://ajax.googleapis.com/ajax/libs/AngularJS/1.4.8/AngularJS.min.js"></script>

</head>

<body ng-app="sampleApp">

<div ng-controller="AngularJSController">

<h3> Guru99 Global Event</h3>

{{ID}}

</div>

<script>

var sampleApp = AngularJS.module('sampleApp',[]);

sampleApp.value("TutorialID", 5);

sampleApp.controller('AngularJSController', function($scope,TutorialID) {

$scope.ID =TutorialID;

});

</script>

</body>

</html>

In the above code example, the below main steps are being carried out

1. sampleApp.value("TutorialID", 5);

The value function of the Angular.JS JS module is being used to create a key value pair called "TutorialID" and the value of "5".

1. sampleApp.controller('AngularJSController', function ($scope,TutorialID)

The TutorialID variable now becomes accessible to the controller as a function parameter.

1. $scope.ID =TutorialID;

The value of TutorialID which is 5, is now being assigned to another variable called ID in the $scope object. This is being done so that value of 5 can be passed from the controller to the view.

1. {{ID}}

The ID parameter is being displayed in the view as an expression. So the output of '5' will be displayed on the page.

1. **Service** - Service is defined as a singleton JavaScript object consisting of a set of functions that you want to expose and inject in your controller.

For example, the "$http" is a service in Angular.JS which when injected in your controllers provides the necessary functions of

( get() , query() , save() , remove(), delete() ).

These functions can then be invoked from your controller accordingly.

Let's look at a simple example of how you can create your own service. We are going to create a simple addition service which adds 2 numbers.

var mainApp = AngularJS.module("mainApp", []);

mainApp.service('AdditionService', function(){

this.Addition = function(a,b) {

return a+b;

}

});

mainApp.controller('DemoController', function ($scope, AdditionService) {

$scope.result = AdditionService.Addition(5,6);

});

In the above example, the following steps are carried out

1. mainApp.service('AdditionService', function()

Here we are creating a new service called 'AdditionService' using the service parameter of our main AngularJS JS module.

1. this.Addition = function(a,b)

Here we are creating a new function called Addition within our service. This means that when AngularJS instantiates our AdditionService inside of our controller, we would then be able to access the 'Addition' function. In this function definition, we are saying that this function accepts 2 parameters, a and b.

1. return a+b;

Here we are defining the body of our Addition function which just simply adds the parameters and returns the added value.

1. mainApp.controller('DemoController', function($scope, AdditionService)

This is the main step which involves dependency injection. In our controller definition, we are now referencing our 'AdditionService' service. When AngularJS see's this, it will instantiate an object of type 'AdditionService.'

1. $scope.result = AdditionService.Addition(5,6);

We are now accessing the function 'Addition' which is defined in our service and assigning it to the $scope object of the controller.

So this is a simple example of how we can define our own service and inject the functionality of that service inside of our controller.