AngularJS

The Web Compilation

A Beginner's Guide to Learn AngularJS and JavaScript along with SemanticUI.

Basics of JavaScript
AngularJS
SemanticUI
Basics of PHP

Niraj Bhagchandani

ANGULARJS – THE WEB COMPILATION

A Beginner's Guide to Learn AngularJS and JavaScript along with SemanticUI

Basics of Javascript
AngularJS
Semantic-UI
Basics of PHP

AUTHOR: NIRAJ BHAGCHANDANI

SELF PUBLISHED

Title Verso

Title: AngularJS The Web Compilation

Author: Mr. Bhagchandani Niraj D.

Address of Publisher: “Manish”, SBS Society, Sahakar Nagar Main Road, Azad Chowk, Nr. Amrapali Cinema, Raiya Road, Rajkot – 360007

First Published – 30th December, 2019

This edition published 30th December, 2019 by Bhagchandani Niraj D.

Publisher: Self Published by Bhagchandani Niraj D.

Book Type: E-Book Publication – As this is ebook publication the Address of the printer is not required.

Printing Details: *This book is E-Book Published only. So does not require any Printing Details.* After the purchase of this E-Book, you may print the document at your end if required.

Copyright © Bhagchandani Niraj 30th December, 2019

Copyright held by: Bhagchandani Niraj.

ISBN - 978-93-5396-363-7

Dedicated To

Dedicated to my father , mother, brother, wife, sister-in-law who inspired me for converting the stuffs from a simple material to a nice and easy understanding book. I also dedicate this book to all those near and dear once who encouraged me to write something on this topic.

Acknowledgement

I would like to thank all the bloggers and youtubers who helped me understand various concepts regarding AngularJS. Also, This book is compilation of the various tutorials available on the internet. Thus, it became very much essential for me to pick up the right tutorial, understand it and make a simple demo from it. I would also say thanks to my family.

Table of Contents

Preface

Greets to all the programmers reading this section. Studying for any programming language needs dedication and patients. Thus, this book was designed to make the work more easier by collecting various attributes different sources and compiling it into one singleton material. Unlike the other books, this book follows the practical aspects which makes the work easier to understand and easier to program. The theory aspects are yet superficially covered to focus more on the programs that I had developed during my learning period. This would reduce the time and effort on the germane set of topics targeted at the right level of abstraction so they can confidently attempt various programs.

This study guide presents a set of topics needed to round out various edges of AngularJS programming. Here, I have developed my own set of programs to make you understand the concepts very easily and begin your journey towards AngularJS 1.x very smoothly and in natural way.

Chapter 1 JavaScript Basics

JavaScript (**JS**) is a lightweight interpreted or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles.[1]

Include – Scripts in a HTML Page

Including scripts on a Page is pretty interesting thing to do. To do this we use the `<script>` `tag.`

Program 1.1
File: program1-1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Javascript Inline</title>
   <script>
      console.log("Hello Niraj");
   </script>
</head>
<body>

</body>
</html>
```

```
Hello Niraj
```

You may include the <script> tag in <body> tag too.

Program 1.2
File: program1-2.html

```
<!DOCTYPE html>
<html lang="en">
```

[1] SphinxKnight. "JavaScript." *MDN Web Docs*, 13 June 2019, developer.mozilla.org/en-US/docs/Web/JavaScript.

```
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Javascript Inline</title>
</head>
<body>
   <script>
      console.log("Hello Niraj");
   </script>
</body>
</html>
```

```
Hello Niraj
```

You may also add the script using `src` attribute in the `<script>` tag.

Program 1.3

File: program1-3.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Javascript Inline</title>
   <script type="text/javascript" src="program1-3.js"></script>
</head>
<body>

</body>
</html>
```

program1-3.js

```
console.log("Hello! Niraj");
```

```
Hello! Niraj
```

Now, let us add the scripting file in body tag and see the difference.

Program 1.4.

File: program1-4.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorial</title>

</head>
```

```
<body>
    <script type="text/javascript" src="listing4.js"></script>
</body>
</html>
```

Listing4.js

```
console.log("Hello! Niraj");
```

```
Hello! Niraj
```

Okay, Now let us add more than 1 line in JavaScript file and see the difference.

Program 1.5

File: program1-5.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorial</title>

</head>
<body>
    <script type="text/javascript" src="program1-5.js"></script>
</body>
</html>
```

Okay, here is the **program1-5.js**

```
console.log("Hello! Niraj");
console.log(" was here");
```

```
Hello! Niraj
 was here
```

Okay now let's see the output without semicolons.

Program 1.6

File: program1-6.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorial</title>

</head>
<body>
    <script type="text/javascript" src="listing6.js"></script>
```

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

Okay, here is the listing6.js

```
console.log("Hello! Niraj")
console.log("was here")
```

```
Hello! Niraj
was here
```

So, you see still it executes the program.

Okay, now let us test the above by writing it in a single line.

Program 1.7

File: program1-7.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorial</title>

</head>
<body>
    <script type="text/javascript" src="listing7.js"></script>
</body>
</html>
```

Okay, here is the listing7.js

```
console.log("Hello! Niraj") console.log(" was here")
```

```
Uncaught SyntaxError: Unexpected identifier
```

JavaScript - Statements

Experessions and statements are the core part of the JavaSript without whose aid the other constructs such as braching and looping statements cannot be meaningful. We will later on see the looping and braching statements later in this chapter. These statements usually are specifically defined in the new line separted with semi colon as described below.

Program 1.8

File: program1-8.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorial</title>
</head>
<body>
```

```
   <script>
       console.log("I am a statement");
       console.log("I am also a statement");
   </script>
</body>
</html>
```

```
I am a statement
I am also a statement
```

We, can also observe the output by putting `<script>` tag in the `<body>` tag too.

Program 1.9

File: program1-9.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorial</title>
    <script>
            console.log("I am a statement");
            console.log("I am also a statement");
        </script>
</head>
<body>

</body>
</html>
```

```
I am a statement
I am also a statement
```

With all sorts of possibilities let us once again see one more possibility.

Program 1.10

File: program1-10.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Inline</title>
    <script>
            console.log("I am a statement inside Head tag");
            console.log("I am also a statement inside Head tag");
        </script>
</head>
<body>
    <!--an inline script-->
    <script>
            console.log("I am a statement inside body tag");
            console.log("I am also a statement inside body tag");
    </script>
</body>
</html>
```

```
I am a statement
I am also a statement
I am a statement inside body tag
I am also a statement inside body tag
```

With the observation of the above example you might have noticed that the `<script>` tag from the `<head>` tag is executed first and then the `<script>` tag from body is executed second.

All about Functions

The block of JavaScript code defined inside the script tag is known as function. It can be executed, or invoked, any number of times. Well, in JavaScript just type the keyword as function and you are ready to go with multiple lines, single line comments and all sorts of programming logics with multi line approach. Well, as we know that purpose of the comments is there to say something meaning and accurately explain the context of the code, also, it makes helpful for others to understand the code easily.
As, you can see that we are passing 1 and 2 as the parameter in the function with concatenation operator as + sign. It is quite obvious thing to see that the value returned here is x itself which is passed as parameter this returned value is then printed in console.log() function.

Program 1.11
File: program1-11.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorials</title>
    <script>
        function myParameterizedFunction(x){
            return x;
        }
        console.log("Hello!! Lecture 1 ! Statement "+ myParameterizedFunction
        (1));
        console.log("Hello!! Lecture 1 ! Statement "+ myParameterizedFunction
        (2));
    </script>
</head>
<body>
    <!--an inline script-->
    <script>
    </script>
</body>
</html>
```

```
Hello!! Lecture 1 ! Statement 1
Hello!! Lecture 1 ! Statement 2
```

As you can see that we have defined the function called parameterized function named as myParameterized(x) where in we have passed the value x to it. Here, we can name the function with lower and upper cases both but as a programming convention I am using camel case for it for readability.

Let, us preview one more example for how to use the function by finding the radius of the circle.

Program 1.12

File: program1-12.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorials</title>
    <script>
        function areaOfCircle(radius){
            return 2*3.14*radius;
        }
        console.log("Area of Circle is "+areaOfCircle(5));
    </script>
</head>
<body>
    <!--an inline script-->
    <script>

    </script>
</body>
</html>
```

```
Area of Circle 31.4
```

You may have seen that the floating points described here are not too many. Let us reduce the decimal point to 2.

Program 1.13

File: program1-13.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorials</title>
    <script>
        function areaOfCircle(radius){
            return 2*3.14*radius;
        }
        console.log("Area of Circle is "+Math.round(areaOfCircle(5)*100)/100);
    </script>
</head>
<body>
    <!--an inline script-->
    <script>

    </script>
</body>
</html>
```

```
Area of Circle is 31.4
```

In the above example we are calling areaOfCircle() function by passing the parameter value as 5. Thus, for converting this up to 2 decimal places we are multiplying and dividing by 100 also using Math.round() to rounding off the returned value.

Variables and its Data type

Containers that hold the data by which the application works with different values and which are interchangeable during the compile as well as execution time is known as variables. The variables are stored in computer memory which can also be used to retrieve the values whenever needed. The below example shows how to declare a variable.

Program 1.14

File: program1-14.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorials</title>
</head>
<body>
    <script>
        var color = "red";
        console.log("The color is " + color);
    </script>
</body>
</html>
```

```
The color is red
```

You can see in the above example that we have used `var` keyword to declare the variable. Here, we have declared the variable named as color whose value is “red” as string. Well, this gives the following output as shown below.

Program 1.15

File:program1-15.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorials</title>
</head>
<body>
    <script>
        //declaring some variables
        var color, size, shape;

        //assigning values
        color = "blue";
        size = "large";
        shape = "circular";
        console.log("Your body color is "+ color + " size is "+size+ " shape
        is "+shape);
    </script>
</body>
</html>
```

```
Your body color is blue size is large shape is circular
```

You can see in the above example that we have declared multiple variables and then assigned values to it after which, we have printed the output on console.

Primitive Types

Boolean

Well, in the regular terms Boolean is meant to spread two possibilities: true or false. Lets, see an example.

```
var isSignedIn = true;
var isRegistered = false;
```

Here, you may watch that the value is not single or double quoted. Rather they are interpreted as it is. Also, “true” or “false” are not the actual value but rather they are treated as string. Thus, the later type is not Boolean type. Thus, if you assign the string value to “false” to a variable, in Boolean terms, that value is treated as true. Consider the following example:

```
var isSignedIn = “false”;
var isSignedIn = 1;
var isSignedIn = “Hello”
```

Thus, in the above example all the values are treated as true. Conversely, let us view the falsy statement.

```
var isSignedIn = "";
var isSignedIn = 0;
var isSignedIn = -0;
```

Strings

To store the series or multiple characters in a variable, such as “Hello! Niraj Bhagchandani”.
You have two choices when creating strings:

a. You may use single quotation marks or
b. You may use double quotation marks.

Let us view the example here.

```
var firstName = "Niraj";
var lastName = 'Bhagchandani';
```

You may also view or set the string as

```
var option = "It's Good";
```

This, can work in both the ways as demonstrated below.

```
var message = 'He said,"I am fine today" and then gone away';
```

If you want to use either single quote or double quote entirely, then we can set that too in this context by using escape sequence ‘\’ as described below.

```
var message = 'He said,\'I am fine today\' and then gone away';
```

or

```
var message = "He said, \"I am fine today\" and then gone away";
```

Well, but wrong way of writing the string would be as described below.

```
var message = 'He said, 'I am fine today' and then gone away';
```

Well, this goes in the wrong way of representation because grammatically it may be correct but syntactically it is incorrect.

```
Uncaught SyntaxError: Unexpected identifier
```

Well, we should also keep in mind that the string started with double quotes must be ended with the double quotes itself and vice versa. Interchanging the quotes some how would result in syntax error.

Numbers

The number type is used to represent integers and floating-point numbers in JavaScript. JavaScript will look at the value and treat it accordingly. Let us view a simple example of it.

Program 1.16
File: program1-16.html

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript Tutorials</title>
</head>
<body>
    <script>
        var val1 = 22;
        var val2 = 23;
        console.log(val1+val2);

        var val3=22.5;
        var val4=23.5;
        console.log(val3+val4);

        var val5=50;
        var val6=.6;
        console.log(val5+val6);

        var val7=25;
        var val8="25";
        console.log(val7+val8);
    </script>
</body>
</html>
```

```
45
46
50.6
2525
```

In the last statement you can see that the JavaScript written here coerced with val7 by converting val7 variable to string and then concatenated it with variable val8.

Missing Values - Undefined and null types.

JavaScript defines the idea of missing values in 2 different types i.e. undefined and null values.
For ex.

```
var myLine;
console.log(myLine);
```

In the example, shown above, we have declared the variable myLine but have not been given any type of value to it. Thus the output of the above program would be as

```
Undefined
```

Here, JavaScript defines the value as undefined that means the program is declared but is not given any value would be undefined.

```
null
```

From the above demonstrations we can come to the conclusion that undefined and null are two distinct types. Thus, undefined is of undefined type, while null can be termed as an object. So, the concept of null and undefined are a little bit tricky. This leads us to one more step that whenever we define a variable, we can either assign a value to it or assign it null rather than keep it undefined.

Conditional Operators in JavaScript

Let's see the JavaScript Operators in tabular form and understand the meaning of it in a very short and sweet manner.

Sr#	Operator	Description
1.	++,--	Increment and Decrement Operator. These operators can be used as prefix or postfix.
2.	+,-,*,/,%	Arithmetic operators for various calulations.
3.	<,<=,>,>=	These are comparison operators to check the values are less than, less than or equal to, greater than and greater than or equal to respectively.
4.	==, !=	These operators are used to check the equality or inequality of the values.
5.	===,!==	The following operator are used to check the Identity and non-identity tests.
6.	=	This is an assignment operator used to assign values to variable.
7.	+	This operator is used to concatenate 2 strings.

Okay, now let us prepare a simple program to deal with the entire operators and see how they are working.

Program 1.17

File: program1-17.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorial</title>
</head>
<body>
    <script>
        console.log("Doing assignment");
        var fName="Niraj";
        console.log(fName);

        console.log("Let's do Maths");
        console.log(15+15); //30
        console.log(15-15); //0
        console.log(15*15); //225
        console.log(15/15); //1
        console.log(15%15); //0
        console.log(15%14); //1

        console.log("Doing comparision");
        console.log(12>10); //true
        console.log(12<10); //false
        console.log(12>=10); //true
        console.log(12<=10); //false

        console.log("Boolean Logic");
        console.lot(true && true); //true
        console.log(false && true); //false
        console.log(true || true); //true
        console.log(true || false); //true
    </script>
</body>
</html>
```

Well, to save the space and page I have tried to put the output of each line in the comment section above. But you may verify it by running this program in your browser and check it from the console tab.

Difference: Equality vs Identity

Well, to compare any two data we can use Equality or Identity comparison, but they both have some difference between them.

Equality (==) Operator: Well, this operator actually attempts to compare the data on both the side of the operator, while it will skip the data type constraint to be checked out. Thus, if we say that ***(2 == "2")*** i.e. number comparison with the string it will return ***true*** as both of them are having the same values apart from the data type.

Identity (===) Operator: Hmm, Well, this operator will attempt to compare the data as well as data type both. Thus, if we say that ***(2 === "2")*** then, it will return ***false*** as JavaScript now knows that it has to compare both data as well as data type in it. Let's view an small example.

Program 1.18

File: program1-18.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <script>
    var varOne = 2;
    var varTwo = "2";
    //Example for Equality
    if(varOne == varTwo){
        console.log("varOne and varTwo are the same");
    }else{
        console.log("varOne and varTwo are NOT the same");
    }
    //Example for Identity
    if(varOne === varTwo){
        console.log("varOne and varTwo are the same");
    }else{
        console.log("varOne and varTwo are NOT the same");
    }
    //One more experiment
    console.log(1+"1")
    </script>
</body>
</html>
```

```
varOne and varTwo are the same
varOne and varTwo are NOT the same
11
```

You see, in the last line when I wrote ***console.log(1+"1")*** it is quite a wonder that JavaScript automatically ***converted 1 to string*** and ***then the (+) operator works as concatenation*** operator and ***not as (+) addition operator.***

Function / Operator	Description
typeof()	It returns the datatype of the operands. It may return any one of the following. "number", "string", "boolean", "object", "undefined", null
toString()	Converts a value,such as number to string.
parseInt()	This function will convert the string to number and thus, return number back. If it is unable to convert the string then it will return the NaN(Not a number).
isNaN()	This is function which checks if the given argument is a number or not. For example, isNaN('three') will return true; isNaN(3) will return false.

Let us demonstrate one example on the above functions:

Program 1.19

File: program1-19.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorial</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Demonstration of typeof(), toString(), parseInt() and isNaN()</h3>
    <script>
```

```
        var myString = "33";
        //typeof() demonstration
        console.log("type of myString is "+typeof(myString));
        var myNumber = 44;
        console.log("type of myString is "+typeof(myNumber));
        //Checking if the variable contains a number or not.
        myString = "Hello";
        console.log("myNumber isNaN: "+isNaN(myNumber)+" myString isNaN: 
        "+isNaN(myString));
        myString = "33";
        //converting to Number
        myNumber = parseInt(myString);
        console.log("Converting Number from String: "+ myNumber + "Type: "+ 
        typeof(myNumber));
        //converting to String
        myString = myNumber.toString();
        console.log("Converting Number from String: "+ myString + "Type: "+ 
        typeof(myString));
    </script>
</body>
</html>
```

```
type of myString is string
type of myString is number
myNumber isNaN: false myString isNaN: true
Converting Number from String: 33Type: number
Converting Number from String: 33Type: string
```

Pre- and Post-Increment

Okay then, now we came with a lot of programming practice lets' view how to do the increment and decrements with different styles. We can achieve these operations with and without operators as we can illustrate here.

Sr	Increments	Decrements
1	`myVariable = myNumber+1;`	`myVariable = myNumber-1;`
2	`myVariable += myNumber;`	`myVariable -= myNumber;`
3	`myVariable = ++myNumber;`	`myVariable = --myNumber;`
4	`myVariable = myNumber++;`	`myVariable = myNumber--;`

Well, with the above table we can say that the first two operators are without increment / decrement operators and the last two operators are with increment / decrement operators. Well, there is difference between pre increment / decrement and post increment / decrement operators.

Pre- Increment / Decrement Operator: In this type of operator first the increment / decrement operation is performed and then the value is returned back. For example,

```
            myVariable = ++myNumber; //myVariable = 2, myNumber = 2
```

Let us assume that the value for variable myNumber is 1, then myVariable will store the value 2. Similarly, the same thing applies for the decrement operator.

Post- Increment / Decrement Operator: In this type of operator first the value is returned back from where it is called and then the increment or decrement operation is performed. For example,

```
            myVariable = myNumber++; //myVariable = 1, myNumber=2
```

Let us again assume, the value for the myNumber as 1, then myVariable would store the value as 1 and then the value is incremented for myNumber and it becomes 2. Similar operation is also performed for the decrement operator.

Program 1.20

File: program1-20.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorial</title>
</head>
<body>
    <h1>JavaScript Tutorial</h1>
    <h3>Pre-Increment / Decrement and Post-Increment / Decrement</h3>
    <script>
        //Pre Increment
        var myNumber = 1;
        myNumber = myNumber+1;
        myNumber = ++myNumber;
        console.log("Pre-Increment result is "+myNumber);

        //Post Increment
        var myNumber = 1;
        myNumber = myNumber+1;
        myNumber = myNumber++;
        console.log("Post-Increment result is "+myNumber);

        //Pre Decrement
        var myNumber = 2;
        myNumber = myNumber - 1;
        myNumber = --myNumber;
        console.log("Pre-Decrement result is "+ myNumber);

        //Post Decrement
        var myNumber = 2;
        myNumber = myNumber - 1;
        myNumber = myNumber--;
        console.log("Post Decrement result is "+ myNumber);
    </script>
</body>
</html>
```

```
Pre-Increment result is 3
Post-Increment result is 2
```

```
Pre-Decrement result is 0
Post Decrement result is 1
```

Playing with objects

The container of the data and home to the functions can be termed as an Object in JavaScript. These are such a versatile aspect of JavaScript that it very important to get a decent handle on this concept.

Program 1.21

File: program1-21.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Working with Objects</h3>
    <script>
        // Illustration 1 -First way
        var myObject = {};
        myObject.firstName = "Niraj";
        myObject.lastName = "Bhagchandani";
        console.log("Firstname: " + myObject.firstName + " Lastname: 
        "+myObject.lastName);

        //Illustration 2 - Second way
        var myObject = {
            firstName : "Niraj",
            lastName : "Bhagchandani"
        };
        console.log("Firstname: " + myObject.firstName + " Lastname: 
        "+myObject.lastName);

        //Illustration 3 - Third way
        var myObject = new Object();
        myObject.firstName = "Niraj";
        myObject.lastName = "Bhagchandani";
        console.log("Firstname: " + myObject.firstName + " Lastname: 
        "+myObject.lastName);
    </script>
</body>
</html>
```

```
Firstname: Niraj Lastname: Bhagchandani
Firstname: Niraj Lastname: Bhagchandani
Firstname: Niraj Lastname: Bhagchandani
```

The above example defines 3 different ways of initializing and declaring of the object.

Reading and Modifying an Object's Properties

Now, we will learn one more example on how to read and modify the object properties.

Program 1.22

File: program1-22.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Reading and Modifying an Object's Properties</h3>
    <script>
        var myObject = {};
        myObject.firstName = "Niraj";
        console.log(myObject.firstName);
        myObject.firstName = "Manish";
        console.log(myObject.firstName);
        myObject["firstname"] = "Dhiraj";
        console.log(myObject["firstname"]);
    </script>
</body>
</html>
```

```
Niraj
Manish
Dhiraj
```

In the example above, we defined the value myObject.firstname as “Niraj” and accessed the value on the console.log after which we once again modified the object and changed the value to “Manish”. In the end we also defined the associative array to the object and assigned the value as “Dhiraj”. Well, associative array is also very useful and preferred way of handling the value of the object.
Thus, let us view one more example on the associative array and see how it works.

Program 1.23

File: program1-23.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
```

```
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Associative Arrays</h3>
    <script>
        var myObject = {};
        myObject["firstName"] = "Niraj";
        console.log(myObject["firstName"]);
        //We can also define the array as
        var propertyName = "firstName";
        myObject[propertyName] = "Manish";
        console.log(myObject[propertyName]);
        //or
        console.log(myObject["firstName"]);
    </script>
</body>
</html>
```

```
Niraj
Manish
Manish
```

What are JSON and XML?

JavaScript Object Notation, or *JSON*, is a lightweight data-interchange format. Essentially, it is way of representing data in a way that is much more compact than XML yet still relatively human and totally machine-readable. If you need to send data from place to place, or even store it somewhere, JSON is often a good choice.
Because JSON is JavaScript (well, a subset of JavaScript, to be precise), it is easy to work with. Unlike XML, it is considerably faster over the wire. I won’t labor too much on JSON, but I will show you what it looks like.

JSON Data

```
{
    "company": "McD",
    "employees": "100",
    "address": {
    "city": "Ahmedabad",
    "state": "Gujarat",
    },
    "phoneNumbers": [
    "111 222-3333-44444",
    "999 888-7777-66666"
    ]
}
```

This is essentially a JavaScript object with a bunch of properties representing contact data for a company named as McD. *Company name* and *total no. of employees* have simple string values.
The *address* property is itself represented as an object, and the *phoneNumbers* property is an array.

Program 1.37

File: program1-37.html

```
<?xml version="1.0" encoding="UTF-8" ?>
<contact>
    <company>McD</company>
    <employees>Smith</employees>
    <address>
        <city>Ahmedabad</city>
        <state>Gujarat</state>
    </address>
    <phoneNumbers>
        <phoneNumber>111 222-3333-44444</phoneNumber>
        <phoneNumber>999 888-7777-66666</phoneNumber>
    </phoneNumbers>
</contact>
```

Unlike XML, JSON is having one key concept i.e. transferring and receiving data in easier way. But it cannot be replaced with XML. XML is in human readable format compared to JSON, this makes XML one step ahead of JSON. But, looking at the perspective of AngujlarJS and JavaScript we will working more on the JSON for the communication with the back-end servers.

Adding Methods to Objects

Let us add a methods to object and see one more example to it.

Program 1.24

File: program1-24.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Adding Methods to Objects</h3>
    <script>
        var myObject = {
            firstName: "Niraj",
            lastName: "Bhagchandani",
            myInfo: function(){
            console.log("My first name is "+ this.firstName +" and last name
            is "+this.lastName);
            }
        };
        myObject.myInfo();
    </script>
</body>
</html>
```

```
My first name is Niraj and last name is Bhagchandani
```

Well, after looking at the output let me explain you that the object contains the firstName, lastName and a property named myInfo with value as a function. The last line shows that the method is being called by the reference variable myObject.
Well, with all sorts of things you may have find one more thing is that ***this keyword.*** Well, the other properties of the same object can be called using **this keyword** in my case I have **this.firstName and this.lastName**. Thus, we can term that **this** is the reference to the current object.

Okay, let us view the example by removing this keyword in the above example and view the output.

Program 1.25
File: program1-25.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Adding Methods to Objects</h3>
    <script>
        var myObject = {
            firstName: "Niraj",
            lastName: "Bhagchandani",
            myInfo: function(){
            console.log("My first name is "+ firstName +" and last name is
            "+lastName);
            }
        };
        myObject.myInfo();
    </script>
</body>
</html>
```

```
Uncaught ReferenceError: firstName is not defined at Object.myInfo
```

Displaying / Enumerating JSON properties.

Well, enumerating an object Properties can be termed with other meaning as computing or collecting the required information from the object. This can be achieved by for loop. Well, we will see the for loop in different form too in coming part of this document but as of now let us illustrate the example.

Program 1.26
File: program1-26.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Enumerating the array</h3>
    <script>
        var myObject = {
            firstName: "Niraj",
            lastName: "Bhagchandani",
            gender: "Male"
        };
        for(var property in myObject){
            console.log("property "+property+": "+myObject[property]);
        }
    </script>
</body>
</html>
```

```
property firstName: Niraj
property lastName: Bhagchandani
property gender: Male
```

Well, in the example above the for loop iterates through all the myObject property in the console and prints them. Well, this example actually prints the property of the object inside and then using associative array we print the value of that property inside stored as an array.

Control Flow

Generally, JavaScript is the language which is read line by line by the browser. For example, a loop or a branch statement. Well, a block of code wanted to get repeatedly executed number of times is known as looping. While, branching is the ability to jump the block of code to another block of code with its potential to execute the code.
Let's understand them one by one.

Loops

Hmm!!, with looping structure let us learn about for loop and see how it goes. The for loop may seem some what complicated at first, but it's not yet that difficult to use. Thus, there are 4 key parts of the for loop.

For Loop:

1. **Initialization variable**: this part of the code is actually where the initial values of the variables are to be provided say for example.
   `var i = 0;`
2. **Conditional logic:** One of the important parts of the for loop, where the decision has to be made whether or not the loop should continue executing the code of block provided.

3. **Counter variable:** This is usually incremented, or otherwise altered, after every loop.
4. **Code block:** The actual code of the block that is executed each pass through the loop.

With these explanation in mind let us demonstrate one example to see how it works.

Program 1.27

File: program1-27.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Control Flow - for Loop</h3>
    <script>
        console.log("Loop Started");
        for(var i=0;i<5;i++){
            console.log("Current Value: "+i+". I will get executed again as
            "+i+" is less than 5.");
        }
        console.log("Loop Ended. ")
    </script>
</body>
</html>
```

```
Loop Started
Current Value: 0. I will get executed again as 0 is less than 5.
Current Value: 1. I will get executed again as 1 is less than 5.
Current Value: 2. I will get executed again as 2 is less than 5.
Current Value: 3. I will get executed again as 3 is less than 5.
Current Value: 4. I will get executed again as 4 is less than 5.
Loop Ended.
```

While loop:

The while loop is pretty simpler version of the for loop, while it has very few arguments as compared to the for loop. Also, we require some extra effort by adding certain statements inside the code block to complete its execution.

```
while (/*some value is true*/) {
   // execute the block of code
}
```

Okay, as now we have seen the syntax above. Let us view the code on how to achieve a loop using while ().

Program 1.28

File: program1-28.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Control Flow - while loop</h3>
    <script>
        var i = 0;
        while(i<10){
            console.log("The value of i is "+i);
            i++; //incrementing the value -> this is some extra work. :)
        }
    </script>
</body>
</html>
```

```
The value of i is 0
The value of i is 1
…
The value of i is 8
The value of i is 9
```

Note: The above output has been reduced to save the space

Conditional Statements

Conditional Statements allows us to code either or type of situation. Indeed, it's like a fork type of situation on the road. For example, you may drive to a single road at a time. Thus, conditional statements help us to select either or type of statements.
Let us view one sweet and short example in JavaScript.

Program 1.29

File: program1-29.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Conditional Statements - if condition</h3>
    <script>
        var isLoggedIn = false;
        if(isLoggedIn){
            console.log('Welcome back!! - Redirecting you to your profile');
        }
        else{
            console.log("Please Login- Access Denied");
        }
    </script>
</body>
</html>
```

```
Please Login- Access Denied
```

What will be the output of the nested if conditional statements if we desire to write the following code? Thus, let’s see the output alongside of the code.

Program 1.30

File: program1-30.html

Nested Conditional Logic

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Conditional Statements - nested if</h3>
    <script>
        var isUserLogged = true;
        var isUserAdmin = true;
        if(isUserLogged){
            console.log("Well, you are logged in. ");
            if(isUserAdmin){
                console.log("You are an admin too");
            }
            else{
                console.log("You are are a just normal user");
            }
        }
    </script>
</body>
</html>
```

```
Well, you are logged in.
You are an admin too
```

Ways to work with Array!

JavaScript arrays are flexible to store the multiple values in a single variable. But before that let me explain you the array. Array are the continuous block of the storage memory wherein you can define one variable to access multiple values with different indices or associations. Well, the definition is quite looking complex but in actual the concept within is quite simple. Yes, JavaScript can store the different types of values within itself. Let's have a look at it.

Program 1.31

File: program1-31.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Working with Arrays</h3>
    <script>
        var myArray = [];
        myArray[0] = "Niraj"; //my first name
        myArray[1] = "Bhagchandani"; //my lastname
        myArray[2] = 32; //my Age
        myArray[3] = '03-03-1986'; //my Birthday
        myArray[4] = 5.9; //my height
        //Now let us print all the values to the console.
        for(var i = 0;i<myArray.length;i++){
            console.log("myArray["+i+"] is "+myArray[i]);
        }
    </script>
</body>
</html>
```

```
myArray[0] is Niraj
myArray[1] is Bhagchandani
myArray[2] is 32
myArray[3] is 03-03-1986
myArray[4] is 5.9
```

You may wonder here that `myArray.length` is used in the for loop. Yes, that is the property of the object which returns the total length of the array and loops until all the values are been printed. Also, you might have noticed that a single array contains float as number, string, integer as number etc.

Also, you may also trick the above example by using length property to store the values. Okay then let us once again code it but this time using length property as the index value. ☺

Program 1.32

File: program1-32.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Working with Arrays</h3>
    <script>
        var myArray = [];
        myArray[myArray.length] = "Niraj"; //my first name
        myArray[myArray.length] = "Bhagchandani"; //my lastname
        myArray[myArray.length] = 32; //my Age
        myArray[myArray.length] = '03-03-1986'; //my Birthday
        myArray[myArray.length] = 5.9; //my height
        //Now let us print all the values to the console.
        for(var i = 0;i<myArray.length;i++){
            console.log("myArray["+i+"] is "+myArray[i]);
        }
    </script>
</body>
</html>
```

```
myArray[0] is Niraj
myArray[1] is Bhagchandani
myArray[2] is 32
myArray[3] is 03-03-1986
myArray[4] is 5.9
```

Here, you will see that instead of indices such as 0,1,2, etc. I have completely used here `myArray.length` which will return the current value of the index and Eureka!! We do not need to remember the index value.

Array Literals

The alternative way of defining the static array is as follows.

```
        myArray = ["Niraj", "Bhagchandani", 32, '03-03-1986', 5.9];
```

This, can be termed as one of the easy way of defining the entire array in one single line and is more concise.

Modifying and displaying array values.

There are several ways of computing the entire array or modifying them one of them is using for loop. Thus, let us have an example on how to enumerate the entire array and then modify them at our ease.

Program 1.33
File: program1-33.html
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Working with Arrays</h3>
    <script>
        var myArray = ["Niraj", "Manish", "Avinash", "Dhiraj", "Prakash"];
        for(var i = 0;i<myArray.length;i++){
            console.log("myArray["+i+"] is "+myArray[i]);
        }
    </script>
</body>
</html>
```

```
myArray[0] is Niraj
myArray[1] is Manish
myArray[2] is Avinash
myArray[3] is Dhiraj
myArray[4] is Prakash
```

As once again mentioning and seeing that we are using length property to scroll the entire array. Modifying the array values is same as modifying the value of any single variable but the difference lies in the loop or logic as described above. Okay then, above program can be used to enumerate the entire array with a loop. But now we need to go one step further by modifying the array in the loop. Okay!! Let's process some of the things in the array and display the results.

Program 1.34
File: program1-34.html
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Modifying Array Values</h3>
    <script>
        var Name = ["Niraj", "Manish", "Avinash", "Dhiraj", "Prakash"];
        var lastName = ["Bhagchandani", "Bhagchandani", "Gurubaxani", "Kaka",
        "Navani"];
        console.log("Before Change: "+Name);
        for(var i = 0;i<Name.length;i++){
            Name[i] = Name[i] +" "+ lastName[i];
        }
        console.log("After change: "+ Name);
    </script>
</body>
</html>
```

```
Before Change: Niraj,Manish,Avinash,Dhiraj,Prakash
After change: Niraj Bhagchandani,Manish Bhagchandani,Avinash Gurubaxani,Dhiraj
Kaka,Prakash Navani
```

So, in the above example I have taken the two arrays which I concatenated one by one and replaced it with the older one. The index values are important in this aspect to carry out the concatenate operation. You may also have noticed that I have logged the variable value before and after the change. Thus, passing the entire array to the `console.log()` is a handy way to dump the contents of the entire array for inspection.

Callbacks

Simple Callbacks: *Another function can only begin its executing **after** its previous function has been finished its execution – hence named as 'call back'*
Complex Callbacks: The functions in JavaScript are known as objects. Thus, the functions which can be passed as arguments inside another functions are known as **call backs** in **higher-order.**

Functions in JavaScript are nothing but objects.Thus, as we can pass the other objects and data types in functions such as strings, arrays etc. we can also pass the functions as the argument in JavaScript. These functions if passed as the argument but is called later in the internal code, then we call such functions as call back functions.

JavaScript code to show the working of callback:

Program 1.35
File: program1-35.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Callbacks in JavaScript - I</h3>
    <script>
        // add() function is called with arguments a, b
        // and callback, callback will be executed just
        // after ending of add() function
        function add(num1, num2 , callback){
        document.write(`The sum of ${num1} and ${num2} is ${num1+num2}.`
        +"<br>");
        callback();
        }
        // disp() function is called just
        // after the ending of add() function
        function disp(){
        document.write('This must be printed after addition');
        }
        // Calling add() function
        add(5,6,disp);
     </script>
</body>
</html>
```

Output

```
JavaScript Tutorials
Callbacks in JavaScript - I
The sum of 5 and 6 is 11.
This must be printed after addition
```

Explanation:

* Well, to continue with the above explanation, we have described 2 different functions containing, add() and the disp() methods.
* The add() method contains 3 arguments, containing num1, num2 and the disp() method as argument, Here, disp() is the functions which is passed as the argument but not called. As shown, add(5,6,disp).
* Thus, the entire operation prints the output using write() method inside the add() function. After completion of the above stuffs, the disp() function is called with different name, as callback() because, we are taking with the name callback().
* We, can also anynymously call the function by initializing and creating method directly while passing the argument.

Program 1.36

File: program1-36.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Callbacks in JavaScript - II</h3>
    <script>
        // a, b are passed as the argument in add() function
        // along with callback function as object.
        // after add() function reads the last line, it will call callback()
       // function
        function add(num1, num2 , callback){
        document.write(`The sum of ${num1} and ${num2} is ${num1+num2}.`
        +"<br>");
        callback();
        }
        // add() function is called with arguments given below
        add(5,6,function disp(){
        document.write(‘Phew!! I will be printed after addition.');
        });
     </script>
</body>
</html>
```

Output

```
JavaScript Tutorials
Callbacks in JavaScript - II
The sum of 5 and 6 is 11.
Phew!! I will be printed after addition.
```

JavaScript Immediately-invoked Function Expressions (IIFE)

In reality, there are many such situations, where in one needs to execute the functions immediately, as soon as they are getting created. Such functions which execute immediately, as soon as they are created are known as Immediately-invoked Function Expression. They are useful to complete an important amount of task as an anonymous function object. After the completion they diminish leaving back the global objects untouched.

This is the syntax that defines an IIFE:

Syntax:

```
(function() {
  /* the code goes here */
})()
```

IIFEs can be defined with arrow functions as well:

Syntax:

```
(() => {
    /* the code goes here */
  })()
```

We basically have a function defined inside parentheses, and then we append () to execute that function: (/* function */)().

Those wrapping parentheses are actually what make our function, internally, be considered an expression. Otherwise, the function declaration would be invalid, because we didn’t specify any name:

Program executed in console:

```
function(){
    /* the code goes here */
}
SyntaxError: function statement requires a name [Learn More]
(function() {
  /* */
})()
undefined
```

To check it out, function declaration contains name, while that is not the case with the function expression. Then, we come to the common conclusion, and that is, function expression does not need the function name, they can be directly invoked by the parenthesis beside the expression parenthesis.

Unary operators along with the IIFE
There is some weirder syntax that you can use to create an IIFE, but it's very rarely used in the real world, and it relies on using any unary operator:
There are some rarely used syntaxes which you may come across, where in you can use it in the real world. These syntax utilizes unary operators as shown below:

```
  !(function() {
    /* the code goes here */
  })()

  ~(function() {
    /* the code goes here */
  })()

  -(function() {
    /* the code goes here */
  })() + (function() {
      /* the code goes here */
    })()
```

(does not work with arrow functions)

IIFE with name
We can pass the function as the argument too in IIFE just like normal function. Thus, it can only be invoked once and it does not leak to the global scope too.

```
(function doSomething() {
    /* the code goes here */
  })()
```

Semicolon Start - IIFE
You might see this in the wild:

```
;(function() {
    /* the code goes here */
})()
```

If you have more than one statements, adjoining each other, then separating 2 different JavaScript statement leads to correct execution. Thus, semicolon speartor would actually end the previous statement and execute our IIFE too. The above IIFE would also concatenate with a file with some other statements without causing any error for minified files.

Closure in JavaScript

Definition of Closure:

When the child function keeps the environment of the parent scope even after when the parent function has already finished it's execution is knowne as Closure function in JavaScript. JavaScript developers use the closure functions knowingly or unknowingly to achieve the better control over the code.

Let's see and understand closure through an example.

Program 1.38

File: program1-38.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Closure in JavaScript</h3>
    <script>
        function foo()
        {
            var b = 1;
            function inner(){
                return b;
            }
            return inner;
        }
        var get_func_inner = foo();
        console.log(get_func_inner());
        console.log(get_func_inner());
        console.log(get_func_inner());
     </script>
</body>
</html>
```

```
1
1
1
```

Now let's look at another example.

Program 1.39

File: program1-39.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Closure in JavaScript</h3>
    <script>
        function foo(outerArg) {
            function bar(innerArg) {
                return outerArg + innerArg;
            }
            return inner;
        }
        var inner_var = foo(5);
        console.log(inner_var(4));
        console.log(inner_var(3));
     </script>
</body>
</html>
```

Explanation: In the above example we used a parameter function rather than a default one. Note even when we are done with the execution of **foo(5)** we can access the **outerArg** variable from the inner function. And on execution of bar function produce the summation of **outerArg** and **innerArg** as desired.

Output:

```
9
8
```

Now let's see an example of closure within a loop. In this example we would to store a anonymous function at every index of an array.

Program 1.40

File: program1-40.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Closure in JavaScript</h3>
    <script>
        // Outer function
    function outer_func()
    {
        var numbers = [], i;
        for (i = 0; i < 4; i++) {
            // storing anonymus function
            numbers[i] = function () { return i; }
        }
        // returning the array.
        return numbers;
    }
    var getNumbers = outer();
    console.log(getNumbers[0]());
    console.log(getNumbers[1]());
    console.log(getNumbers[2]());
    console.log(getNumbers[3]());
     </script>
</body>
</html>
```

```
4
4
4
4
```

Explanation: Got surprised by the answer? You should be! Because in the actual scenario **Closure does not store the value of the variable it only points to the variable** or stores the reference of the variable thus, it **returns the current value** stored at that memory location.

Let's see a correct way to write the above code so as to get different values of i at different index.

Program 1.41
File: program1-41.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>Closure in JavaScript</h3>
    <script>
        // Outer function
        function exterior(){
            function createClosure(val) {
                return function(){
                    return val;
                }
            }
            var arr = [];
            var i;
            for (i = 0; i < 4; i++) {
                array[i] = createClosure(i);
            }
            return array;
        }
        var my_array = exterior ();
        console.log(my_array[0]());
        console.log(my_array[1]());
        console.log(my_array[2]());
        console.log(my_array[3]());
    </script>
</body>
</html>
```

```
0
1
2
3
```

Explanation: The above code depicts that createClosure function actually updates the arguments every time it is called. So, we see over here in this example, that we gain the different value of *i*.

Note: *In a single notice it may be difficult to understand the closure property but you may try experimenting with closure with changed circumstances and setups by creating getter setter methods or callbacks etc.*

Bind, call and apply

As we have learned that functions are also objects in JavaScript,well taking this into consideration there are 3 different methods can we can invoke and they are call(), apply() and bind() out of which call() and apply() functions are the part of ECMAScript3 while bind() got introduced in ECMAScript 5.

Uses

In certain situations where in we need to invoke the function immediately, call() and app() functions would be one of the best available options for us. While bind() function returns a bounded function which would be executed later. Thus, we can have correct context “this” as object by which we can call the original function. This proves that bind() can be used when the function needs to be called later in certain events.

call() or Function.prototype.call()

Check the code sample below for call()

Program 1.42

File: program1-42.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>call() or Function.prototype.call()</h3>
    <script>
        //Demo with javascript .call()
        var obj = {name:"Niraj Bhagchandani"};
        var greeting = function(a,b,c){
            return "Welcome "+this.name+" to "+a+" "+b+" in "+c;
        };
        console.log(greeting.call(obj,"New World of ","Programming ",
        "AngularJS" ));
// returns output as Welcome Niraj Bhagchandani to new World of Programming
AngularJS
     </script>
</body>
</html>
```

The above program states the call() function which sets “this” value as an object. On invoking this object the “obj” and its properties gets its value inside the greeting function. The rest parameters such as a, b and c are given the values “New World of”, “Programming “, and “AngularJS” respectively.

```
Welcome Niraj Bhagchandani to new World of Programming AngularJS
```

apply() or Function.prototype.apply()

Program 1.43

File: program1-43.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>apply() or Function.prototype.apply()</h3>
    <script>
        //Demo with javascript .apply()
        var obj = {name:"Niraj Bhagchandani"};
        var greeting = function(a,b,c){
            return "welcome "+this.name+" to "+a+" "+b+" in "+c;
        };
        // array of arguments to the actual function
        var args = ["new world of ","Programming ","AngularJS."];
        console.log("Output using .apply() below ")
        console.log(greeting.apply(obj,args));
     </script>
</body>
</html>
```

```
Output using. apply () below
welcome Niraj Bhagchandani to new world of Programming AngularJS.
```

Check the below code sample for apply ()

Well, in the apply() function, the first argument here in the sample is the object passed to it which can be accessed it via “this” object in JavaScript, while the second argument is nothing but combination of the 3 variables a,b and c in terms of an array. Thus, second argument is accepted as an array which would be initialized by the array respectively.

bind() or Function.prototype.bind()

Check the below code sample for bind()

Program 1.44

File: program1-44.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript Tutorials</title>
</head>
<body>
    <h1>JavaScript Tutorials</h1>
    <h3>bind() or Function.prototype.bind()</h3>
    <script>
        //Use .bind() javascript
        var obj = {name:"Niraj Bhagchandani"};
        var greeting = function(a,b,c){
            return "welcome "+this.name+" to "+a+" "+b+" in "+c;
        };
        //creates a bound function that has same body and parameters
        var bound = greeting.bind(obj);
        console.dir(bound); ///returns a function
        console.log("Output using. bind() below ");
        console.log(bound("new world of ","Programming ","AngularJS")); //call
        the bound function
     </script>
</body>
</html>
```

Chapter 3 Introduction to AngularJS

What is AngularJS?

AngularJS is a JavaScript framework that helps build web applications. It is an open source project which is used to build both Single Page Applications (SPA) and Line of Business Applications (LBA).
There is a website too which lists most famous project built using AngularJS ie. *www.madewithangular.com*

Benefits of AngularJS

1. **Dependency Injection**

The Dependency Injection or DI is design of the code or component in such a way that instead of hard coding them they are written in separate component. This makes them reusable and configurable every time such code is used. Thus, making it maintainable and testable whenever modified.
AngularJS provides a supreme Dependency Injection mechanism. It provides following core components which can be injected into each other as dependencies.

- Value
- Factory
- Service
- Provider
- Constant

2. **Two Way Data-Binding**

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

3. **Testing**

Testing is one of the most rich features of AngularJS. It is known that when AngularJS was getting developed in Google Labs, they also kept testing in mind and made sure the entire framework was also testable.
Thus, Karma a testing framework came into existance. This Karma framework is one of the best testing framework for AnguarlJS. Well, one can also test the AngularJS without Karma too in normal way but Karma with its coding functionality is to its best way to test it.

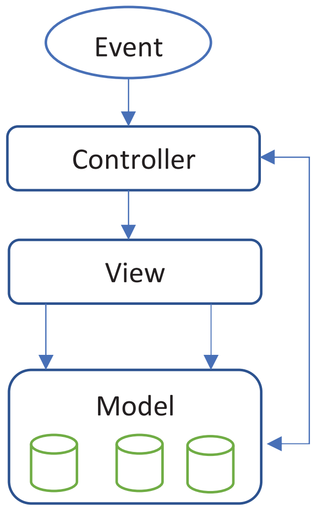
- In AngularJS, we can perform Unit Testing separately for controllers and directives.
- We can also perform end of end testing of AngularJS, which is testing from a user perspective.

4. **MVC Architecture.**

One of the most important aspect in the software design patters for developing any web application is **M**odel **V**iew **C**ontroller or in other words, it is MVC. Thus, as the name suggest, this pattern is made up of three parts.

- **Model**: This part of the MVC maintains the data and responsible to store and fetch the data.
- **View**: The part of the web application which is useful to display the data to user is nothing but view.
- **Controller**: There is one intermediate part of the code which maintans and controls the entire interaction between Model and View. This part of the code is known as Controller.

In MVC Architecture as described above, the controller will receive request from the application and then asks the model to prepared the data for the view. This in turn, uses the application logic from the user interface layer. After all the above operation has been performed, the data would then, get prepared by the controller to retrieve back to the user via view. Graphical representation of the MVC is as shown below.



The Model

The model is responsible for managing application data. It responds to the request from view and to the instructions from controller to update itself.

The View

A presentation of data in a particular format, triggered by the controller's decision to present the data. They are script-based template systems such as JSP, ASP, PHP and very easy to integrate with AJAX technology.

The Controller

The controller responds to user input and performs interactions on the data model objects. The controller receives input, validates it, and then performs business operations that modify the state of the data model.

AngularJS is a MVC based framework. In the coming chapters, we will see how AngularJS uses MVC methodology.

5. Directives, filters, etc.

Directives:

Well, Directives are one of the important concepts when learning AngularJS applications. Well, when we think about component which are said to be most-used unit in AngularJS are nothing but directives too.

Directive combinely with a template can form an AngularJS component. Thus, when we say that AngularJS has the building block as component. In actual case, we are referring

AngularJS

to directives as the building blocks of AngularJS applications.

Filters:

AngularJS provides filters to transform data:

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

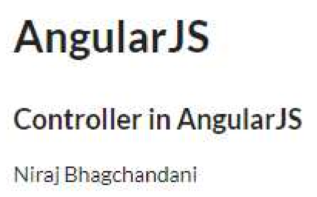
First AngularJS Application

Okay!! Now let us prepare and understand the integration of AngularJS script, semantic-ui framework and jQuery into the web application and display the hello world program using AngularJS.

Program 3.1

File: program3-1.html

```
<!DOCTYPE html>
<html lang="en" ng-App>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS</title>
</head>
<body>
    <div class="ui container">
        <br/><br/>
        <div class="row">
            <h1 class="ui header">AngularJS Tutorials</h1>
            <h3 class="ui header">First Applications</h3>
            <p>Hello {{ 'Wor' + 'ld' }}</p>
        </div>
    </div>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
     src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"><
     /script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
     ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
     href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
     ui/2.4.1/semantic.min.css" />
    <script
     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js
     "></script>
</body>
</html>
```



AngularJS

Controller in AngularJS

Niraj Bhagchandani

Looking at the above example, you will notice that I have included HTML5 doctype to it. Although it is not necessary to include HTML5 for AngularJS to work.
The second line of the code includes the directive named as ng-App. This will initialize and tell AngularJS to start the application from that particular position and work as the root of the application. Here, we have declared ng-app inside the html element tag, the entire document is under control of AngularJS.

Moving down the the line where we have mentioned the <p> Hello {{ 'Wor' + 'ld' }} </p> we have declared the AngularJS expression <- *that's the interesting part*. Thus, we keep nice and interesting thing here we concatenated two different string and then AngularJS examined these inside the expression which is then compiled by it. Thus, giving us the final output as Hello World.

Well, to describe more the AngularJS expressions are too much simple and handy to use thus, giving it more flexibility to display it in an easier and understandable way.
When we put anything between two curly braces AngularJS treats them that as expression which is bound to display some values on the website. Anytime it changes or updates the value the AngularJS would directly display its value onto website where it is called. Here is the output:

Summary

1. The ngApp directive was used to tell the AngularJS to consider the code under it's control and process it accordingly.
2. We then use script tag to include the AngularJS library to the HTML document.
3. We also included the AngularJS expression to display the concatenated string to do some basic operation and display the binding value on the web.

Program 3.2

HTML File: program3-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS Tutorial</title>
</head>
<body>
    <div class="ui container" ng-App="myModule">
        <br/><br/><br/>
        <h1 class="ui header">AngularJS Tutorial</h1>
        <h3>Simple MVC Program</h3>
        <div class="ui grid">
            <div class="four wide column">
                <div class="ui floating message" ng-controller="myController">
                    <p>{{ message }}</p>
                </div>
                </div>
                <div class="four wide column"></div>
                <div class="four wide column"></div>
                <div class="four wide column"></div>
            </div>
        </div>
        <!-- Semantic UI / JQuery / AngularJS script added here-->
        <script
         src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"><
         /script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
         ui/2.4.1/semantic.js"></script>
        <link rel="stylesheet"
         href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
         ui/2.4.1/semantic.min.css" />
        <script
         src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js
         "></script>
        <script src="program3-2.js"></script>
</body>
</html>
```

JavaScript File: program3-2.js

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

var myController = function($scope){
    var message = "Hello World";
    $scope.message = message;
}

myApp.controller("myController", myController);
```

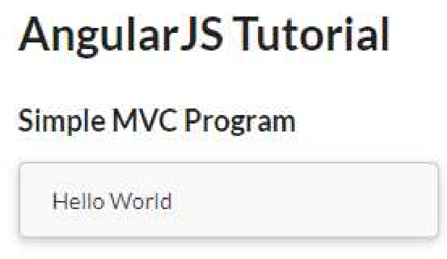
In the above program, let us start understanding the program2.js file first.

* Line 1. We have declared and initialized the angular Module named as myModule and stored it as object in myApp variable.
* Line2. We are declaring a function with argument $scope which is used as global variable in the AngularJS. This function is stored as the object in myController.
* Line 3. We are declaring the simple Object having string value as “Hello World” and stored as message variable.
* Line 4. The message variable is then stored as $scope variable as property.

Now, let us look at the program2.html file and understand it.

* We have initialized and bootstrapped ng-App=”myModule” where in we define the AngularJS engine to start the compilation from myModule itself.
* This is then translated to other line when ng-controller is found as the directive in one of the HTML tag. This directive will search for the controller from the given module and then try to execute its’ value as function.
* Also, the controller contains the $scope.message which would be globally assigned the value.
* The {{ }} expressions are used to fetch the value from the global variable $scope and

print its property as variable there in the view.



AngularJS Tutorial

Simple MVC Program

Hello World

You may also write the program3-3.js in the following way to minimize the code as described below.

Program 3.3
JavaScript File: program3-3.js

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

myApp.controller("myController", function($scope){
    var message = "Hello World";
    $scope.message = message;
});
```

Here, you may see that rather than assigning the value to the object as function we directly passed the value of the function in the controller's 2nd argument as shown. Still there won't be any change in the output.

Also, we can use JavaScript chaining method to develop the code above and reduce the number of lines in the above code. Let me show you.

Program 3.4
File: program3-4.js

```
var myApp = angular.module("myModule", [])
            .controller("myController", function($scope){
                var message = "Hello World";
                $scope.message = message;
            });
```

Here, as described the output for Program 3-2, 3-3 and 3-44 remains the same and there is no change in it.

Okay, now enough of MVC architecture let us program some simpler ones without MVC and see what comes out.

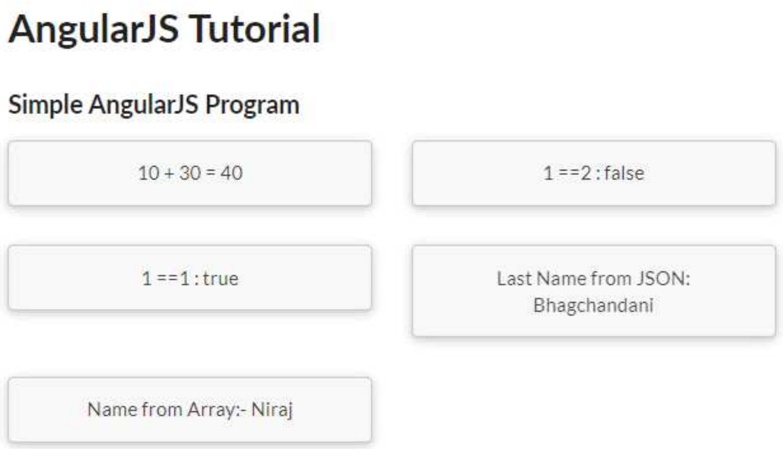
Program 3.5
File: program3-5.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS Tutorial</title>
</head>
<body>
    <div class="ui container" ng-App>
        <br/><br/><br/>
        <h1 class="ui header">AngularJS Tutorial</h1>
        <h3>Simple AngularJS Program</h3>
        <div class="ui grid">
            <div class="row">
            <div class="four wide column">
                <div class="ui floating message">
                    <p align="center">10 + 30 = {{ 10 + 30 }}</p>
                </div>
            </div>
            <div class="four wide column">
                <div class="ui floating message">
                        <p align="center">1 ==2 : {{ 1 == 2 }}</p>
                </div>
            </div>
            </div>
            <div class="row">
            <div class="four wide column">
                <div class="ui floating message">
                        <p align="center">1 ==1 : {{ 1 == 1 }}</p>
                </div>
            </div>
            <div class="four wide column">
                <div class="ui floating message">
                        <p align="center">Last Name from JSON: {{ {name:
                        "Niraj", lastName: "Bhagchandani"}.lastName }}</p>
                </div>
            </div>
            </div>
            <div class="row">
            <div class="four wide column">
                <div class="ui floating message">
                        <p align="center">Name from Array:- {{ ["Niraj",
                        "Bhagchandani"][0] }}</p>
                </div>
            </div>
            </div>
        </div>
    </div>

    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></scri
pt>
```

```
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
ui/2.4.1/semantic.min.css" />
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></s
cript>
</body>
</html>
```


AngularJS Tutorial

Simple AngularJS Program

10 + 30 = 40

1 ==2 : false

1 ==1 : true

Last Name from JSON: Bhagchandani

Name from Array:- Niraj

What is Module?

A module in AngularJS is a container of the different parts of an application such as controller, service, filters, directives, factories etc. It supports separation of concern using modules. AngularJS stops polluting global scope by containing AngularJS specific functions in a module.

Application Module

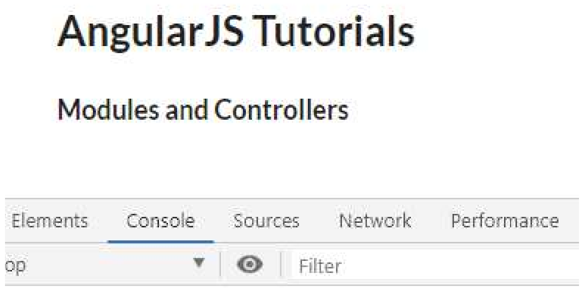
An AngularJS application must create a top-level application module. This application module can contain other modules, controllers, filters, etc.

Program 3.6

File: program3-6.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></scri
pt>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
```

```
ui/2.4.1/semantic.min.css" />
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></s
cript>
</head>
<body>
    <div class="ui container">
        <br/><br/>
        <h1>AngularJS Tutorials</h1>
        <h3>Modules and Controllers</h3>
        <div ng-App="myModule">
        </div>
        <script>
            var myApp=angular.module("myModule",[]);
        </script>
    </div>
</body>
</html>
```

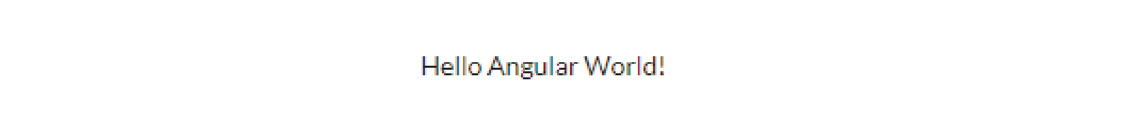

AngularJS Tutorials

Modules and Controllers

In the above example, the angular.module() method creates an application module, where the first parameter is a module name which is same as specified by ng-app directive. The second parameter is an array of other dependent modules []. In the above example we are passing an empty array because there is no dependency.
Note: The angular.module() method returns specified module object if no dependency is specified. Therefore, specify an empty array even if the current module is not dependent on another module.
Now, you can add other modules in the myApp module.
The following example demonstrates creating controller module in myApp module.

Program 3.7
File: program3-7.html

```
<!DOCTYPE html>
<html >
<head>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></
    script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.min.css" />
    <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"
    ></script>
</head>
<body ng-app="myApp">
    <div class="ui container">
        <br><br>
        <div ng-controller="myController">
            {{message}}
        </div>
        <script>
            var myApp = angular.module("myApp", []);
            myApp.controller("myController", function ($scope) {
                $scope.message = "Hello Angular World!";
            });
        </script>
    </div>
</body>
</html>
```



```
Hello Angular World!
```

In the above example, we have created a controller named "myController" using myApp.controller() method. Here, myApp is an object of a module, and controller() method creates a controller inside "myApp" module. So, "myController" will not become a global function.

Modules in Separate Files

In the controller example shown above, we created application module and controller in the same HTML file. However, we can create separate JavaScript files for each module as shown below.

Program 3.8

File: program3-8.html

```
<!DOCTYPE html>
<html >
<head>
    <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="myApp">
    <div ng-controller="myController">
        {{message}}
    </div>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
     src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"><
     /script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
     ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
     href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
     ui/2.4.1/semantic.min.css" />
    <script
     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js
     "></script>
    <script src="program8-app.js" ></script>
    <script src="program8-myController.js" ></script>
</body>
</html>
```

File: program8-myApp.js

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

File: program8-myController.js

```
myApp.controller("myController", function ($scope) {
    $scope.message = "Hello Angular World!";
});
```

```
Hello Angular World!
```

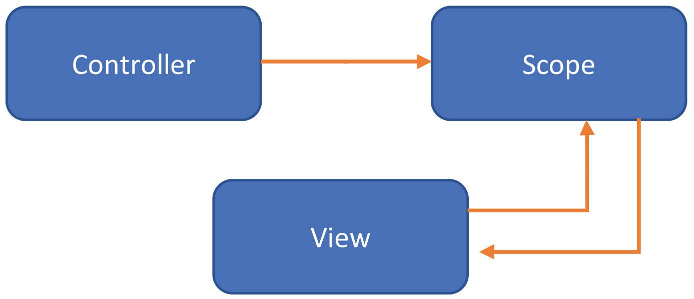
What is Controller in AngularJs?

When the user submits the data in the browser View, it is sent to the controller for processing. After the controllers process all the relevant data it sends back to the view of the browser for displaying it to the end user. Thus, controller will have the core business logic.

In other words to cary out the reuired processing controller will use data model and then pass it back to the view for displaying it to to the end user.

We will explore -

- What Controller does from Angular' s perspective?
- How to build a basic Controller?
- How to define Methods in Controller?
- Using ng-controller in External Files?
- What Controller does from Angular's perspective?



Following is a simple definition of working of Angular JS Controller.

* 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. More over 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.

Program 3.9

File: program3-9.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
     src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"><
     /script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
     ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
     href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
     ui/2.4.1/semantic.min.css" />
    <script
     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js
     "></script>
    <script src="program3-9.js"></script>
</head>
<body ng-App="myModule">
    <div class="ui container">
        <br/><br/>
        <h1>AngularJS</h1>
        <h3>Controller in AngularJS</h3>
        <div ng-controller="myController">
            {{ fullName() }}
        </div>
    </div>
</body>
</html>
```

File: program3-9.js

```
var app=angular.module("myModule",[]);
app.controller('myController', function($scope){
    $scope.firstName = "Niraj";
    $scope.lastName = "Bhagchandani";
    $scope.fullName = function(){
        return $scope.firstName + " " +$scope.lastName;
    }
});
```

- Data in this way pass 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!

But, in here the below example says something different wherein we can create a simple scope by defining the various properties such as name and degree and watch out the basic construction of the program as describe below.

Program 3.10

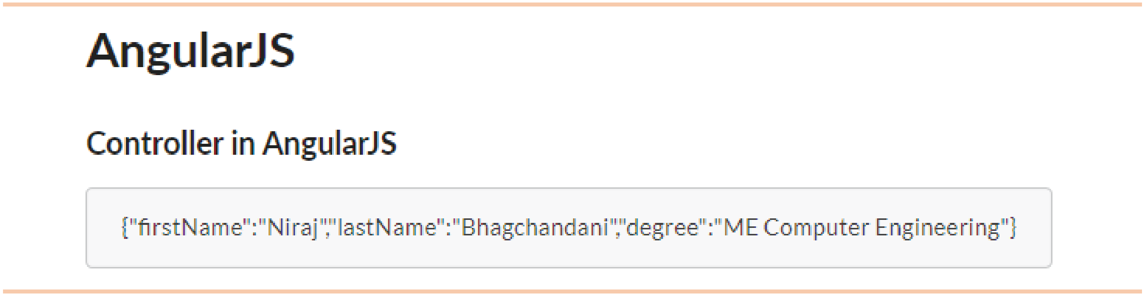
File: program3-10.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></
    script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.min.css" />
    <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"
    ></script>
    <script src="program3-10.js"></script>
</head>
<body ng-App="myModule">
    <div class="ui container">
        <br/><br/>
        <h1>AngularJS</h1>
        <h3>Controller in AngularJS</h3>
        <div ng-controller="myController">
            <div class="ui compact message">
```

```
                <p>{{ employee }}</p>
            </div>
        </div>
    </div>
</body>
</html>
```

File: program3-10.js

```
var app=angular.module("myModule",[]);
app.controller('myController', function($scope){
    var employee = {
        firstName: "Niraj",
        lastName: "Bhagchandani",
        degree: "ME Computer Engineering"
    };
    $scope.employee = employee;
});
```


AngularJS

Controller in AngularJS

```
{"firstName":"Niraj","lastName":"Bhagchandani","degree":"ME Computer Engineering"}
```

Here, we have taken the JSON value given to the variable. The JSON value includes firstName, lastName, university. These values are then passed on to the $scope variable and assigned globally so that angular can process that variable where ever is needed.

Thus, when we call {{ employee }} in the index.html it will print the entire value of the JSON.

What if we want to display the inner properties of the employee also, Okay then let us look the example for it too.

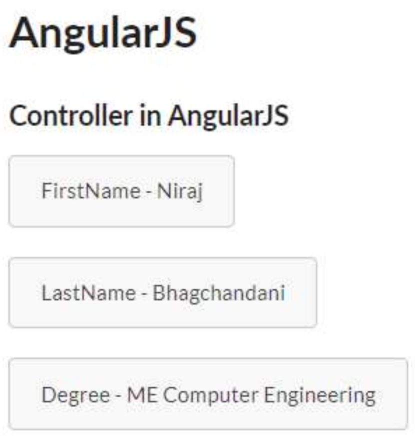
Program 3.11

File: program3-11.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></
    script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.min.css" />
    <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"
    ></script>
    <script src="program3-11.js"></script>
</head>
<body ng-App="myModule">
    <div class="ui container">
        <br/><br/>
        <h1>AngularJS</h1>
        <h3>Controller in AngularJS</h3>
        <div ng-controller="myController">
            <div class="row">
                <div class="ui compact message">
                    <p> FirstName - {{ employee.firstName }}</p>
                </div>
            </div><br>
            <div class="row">
                <div class="ui compact message">
                    <p> LastName - {{ employee.lastName }}</p>
                </div>
            </div><br>
            <div class="row">
                <div class="ui compact message">
                    <p> University - {{ employee.degree }}</p>
                </div>
            </div><br>
        </div>
    </div>
</body>
</html>
```

File: program3-11.js

```
var app=angular.module("myModule",[]);
app.controller('myController', function($scope){
    var employee = {
        firstName: "Niraj",
        lastName: "Bhagchandani",
        degree: "ME Computer Engineering"
    };
    $scope.employee = employee;
});
```



AngularJS

Controller in AngularJS

FirstName - Niraj

LastName - Bhagchandani

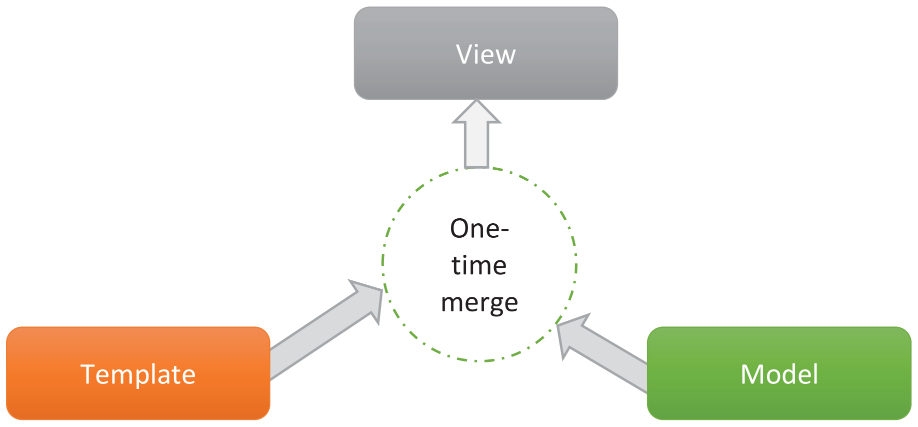
Degree - ME Computer Engineering

AngularJS Data Binding

The bridge which acts between the view and business logic of the application is known as data binding. This is the one of the very powerful and useful feature used for AngularJS web development.

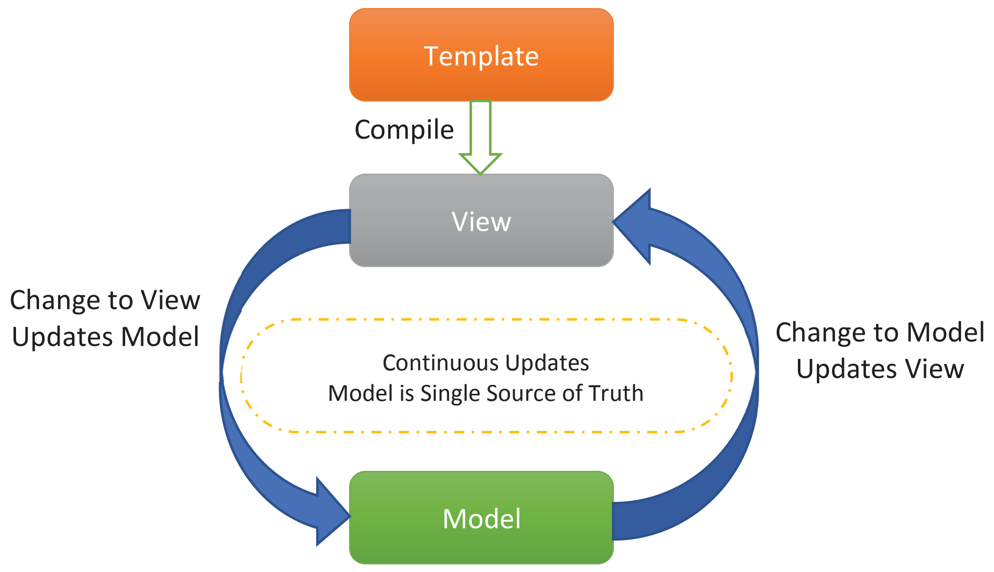
One-Way Data Binding

In one way data binding the data model actually approaches to HTML element i.e. views and displays the data with first order. But there is no provision to update the data model from the view instantly. Such type of application is known as classical template systems. These systems bind data in only one direction.



Two-Way Data Binding

Two-Way Data binding actually helps in automatic synchronization of data between the data and view components. These will let the user treat the model as single-source-of-truth in the application. To project all the models at all the times, the data transfer between the model, view and vica-versa makes it possible due to two-way data binding.


Template
Compile
View
Change to View Updates Model
Change to Model Updates View
Continuous Updates
Model is Single Source of Truth
Model

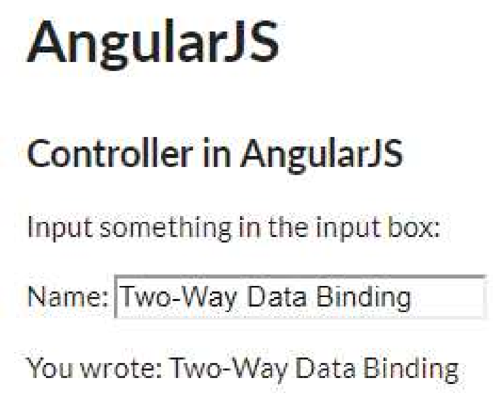
In the above example, the {{ firstName }} expression is an AngularJS data binding expression. Data binding in AngularJS binds AngularJS expressions with AngularJS data.
{{ firstName }} is bound with ng-model="firstName".

Program 3.12

File: program3-12.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></
    script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.min.css" />
    <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"
    ></script>
</head>
<body>
    <div class="ui container">
        <br/><br/>
        <h1>AngularJS</h1>
        <h3>Controller in AngularJS</h3>
        <div ng-app="" ng-init="firstName='Two-Way Data Binding'">
            <p>Input something in the input box:</p>
            <p>Name: <input type="text" ng-model="firstName"></p>
            <p>You wrote: {{ firstName }}</p>
        </div>
    </div>
</body>
</html>
```

Figure: Before Click


AngularJS

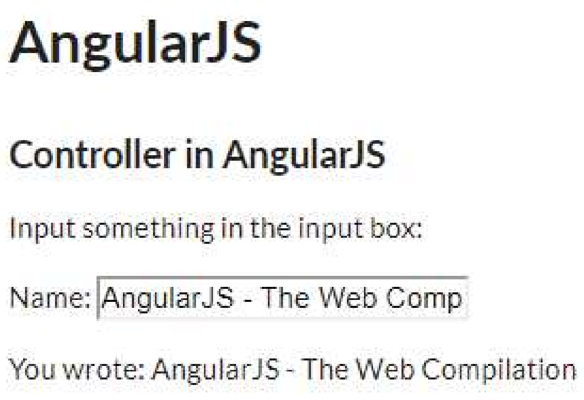
Controller in AngularJS

Input something in the input box:

Name: Two-Way Data Binding

You wrote: Two-Way Data Binding

Figure: After Click


AngularJS

Controller in AngularJS

Input something in the input box:

Name: AngularJS - The Web Comp

You wrote: AngularJS - The Web Compilation

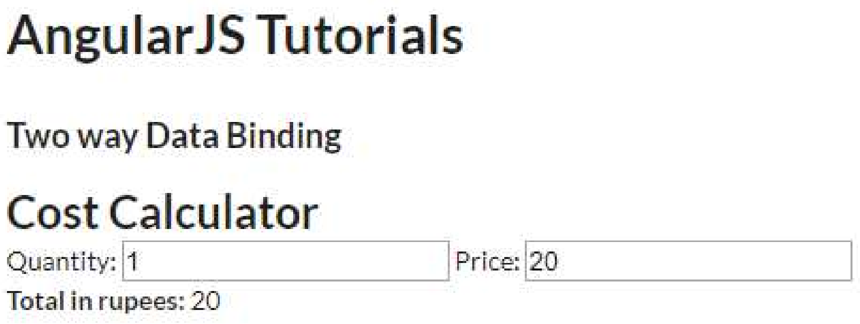
Let's take another example where two text fields are bound together with two ng-model directives:

Program 3.13

File: program3-13.html

```
<!DOCTYPE html>
<html>
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></
    script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.min.css" />
    <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"
    ></script>
<body>

    <div class="ui container">
        <br/><br/>
        <h1 class="ui header">AngularJS Tutorials</h1>
        <h3 class="ui header">Two way Data Binding</h3>
        <div data-ng-app="" data-ng-init="quantity=1;price=20">
            <div class="row">
                <h2>Cost Calculator</h2>
            </div>
            <div class="row">
                Quantity: <input type="number" data-ng-model="quantity">
                Price: <input type="number" data-ng-model="price">
            </div>
            <div class="row">
                <p><b>Total in rupees:</b> {{quantity * price}}</p>
            </div>
        </div>
    </div>
</body>
</html>
```


AngularJS Tutorials

Two way Data Binding

Cost Calculator

Quantity: 1 Price: 20

Total in rupees: 20

AngularJS ng-repeat Directive with Example

Displaying repeated information

Sometimes we may be required to display a list of items in the view, so the question is that how can we display a list of items defined in our controller onto our view page. Angular provides a directive called "ng-repeat" which can be used to display repeating values defined in our controller. Let's look at an example of how we can achieve this.

Program 3.14

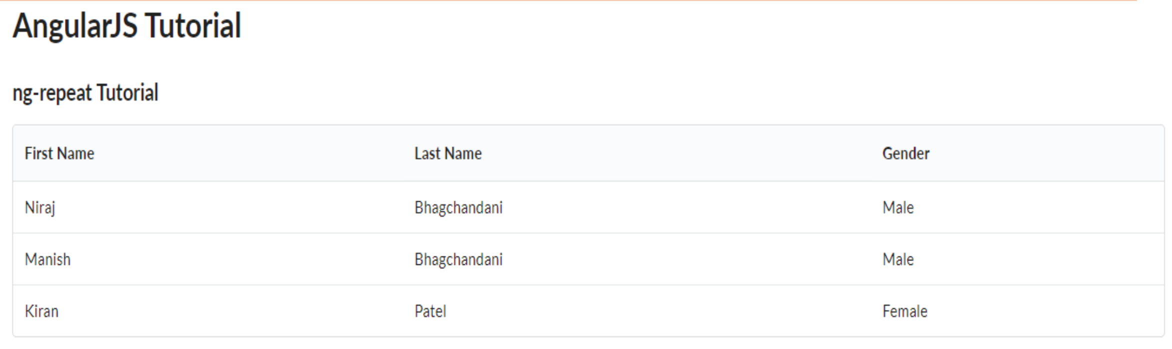
File: program3-14.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
     src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"><
     /script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
     ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
     href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
     ui/2.4.1/semantic.min.css" />
    <script
     src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js
     "></script>
    <script src="program3-14.js"></script>
```

```
</head>
<body ng-app="myModule">
    <div class="ui container">
        <br/><br/>
        <h1>AngularJS Tutorial</h1>
        <h3>ng-repeat Tutorial</h3>
        <div ng-controller="myController">
            <table class="ui very classic table">
                <thead>
                    <th>First Name</th>
                    <th>Last Name</th>
                    <th>Gender</th>
                </thead>
                <tbody>
                <tr ng-repeat = "employee in employees">
                    <td>{{employee.fName}}</td>
                    <td>{{employee.lName}}</td>
                    <td>{{employee.gender}}</td>
                </tr>
                </tbody>
            </table>
        </div>
    </div>
</body>
</html>
```

File: program3-14.js

```
var myApp = angular.module("myModule",[]);
myApp.controller("myController", function($scope){
        var employees = [
            {fName: "Niraj", lName: "Bhagchandani", gender: "Male"},
            {fName: "Manish", lName: "Bhagchandani", gender: "Male"},
            {fName: "Kiran", lName: "Patel", gender: "Female"},
        ];
        $scope.employees = employees;
    });
```


AngularJS Tutorial

ng-repeat Tutorial

First Name	Last Name	Gender
Niraj	Bhagchandani	Male
Manish	Bhagchandani	Male
Kiran	Patel	Female

Code Explanation:

1. In the controller, we first define our array of list items in JSON which we want to define in the view. Over here we have defined an array called "TopicNames" which contains

three items. Each item consists of a name-value pair.

2. The array of fName, lName and university is then added to a member variable called "employees" and attached to our scope object.
3. We are using the HTML tags of <tr>(table row) and <td>(table data) to display the list of items in our array. We then use the ng-repeat directive for going through each and every item in our array. The word "employee" is a variable which is used to point to each item in the array employee.fName etc..
4. In this, we will display the value of each array item.

If the code is executed successfully, the above Output will be shown when you run your code in the browser. You will see all items of the array displayed.

~ “Life begins at the end of the comfort zone” – Neale Donald Walsch ~

Chapter 6 - ng Elements in AngularJS

Showing HTML Elements using ng-show

The ng-show is a special kind of directive which can be used to show/hide the given HTML elements based on their expression provided in its attribute. Well, to deep dive more into the concept, in reality the ng-show or ng-hide actually adds or removes the CSS class onto that particular element. Don't you believe me? Never mind let me show you an example.

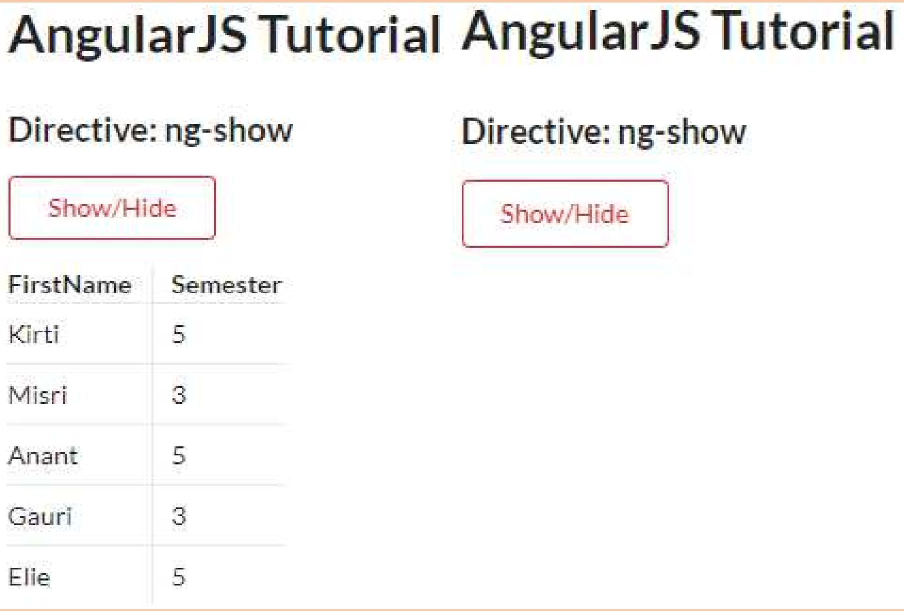
Program 6.1

File: program6-1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS Tutorial</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></
    script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.min.css" />
    <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"
    ></script>
    <script src="program6-1.js"></script>
</head>
<body ng-app="myModule">
    <div class="ui container">
        <h1>AngularJS Tutorial</h1>
        <h3>Directive: ng-show</h3>
        <div ng-controller="myController">
            <div ng-click="showHide()" class="ui animated red basic button"
             tabindex="0">
                <div class="visible content" >Show/Hide</div>
                    <div class="hidden content">
                        <i class="right arrow icon"></i>
                    </div>
            </div>
            <table ng-show="isVisible" class="ui very basic collapsing celled
            compact table">
                <th>FirstName</th>
                <th>Semester</th>
                <tr ng-repeat="d in data">
                    <td>{{d.fName}}</td>
                    <td>{{d.semester}}</td>
                </tr>
            </table>
        </div>
    </div>
</body>
</html>
```

File: program6-1.js

```
var myApp = angular.module("myModule",[]);
myApp.controller("myController", function($scope){
    var student = [
        {fName:"Kirti", semester:5},
        {fName:"Misri", semester:3},
        {fName:"Anant", semester:5},
        {fName:"Gauri", semester:3},
        {fName:"Elie", semester:5},
    ];
    $scope.data = student;
    $scope.isVisible = true;
    $scope.showHide = function(){
        $scope.isVisible = !$scope.isVisible;
    }
});
```



FirstName	Semester
Kirti	5
Misri	3
Anant	5
Gauri	3
Elie	5

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 – myController.

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 true so that when the page is first displayed the div tag will be shown.

   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.

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.
5. When the Show/Hide button is again clicked the value gets reversed, i.e. If it is false it turns to true and vice versa.

The following output is shown when the code is executed successfully 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 shown for ngShow to showcase how the ngHide attribute can be used.

Program 6.2

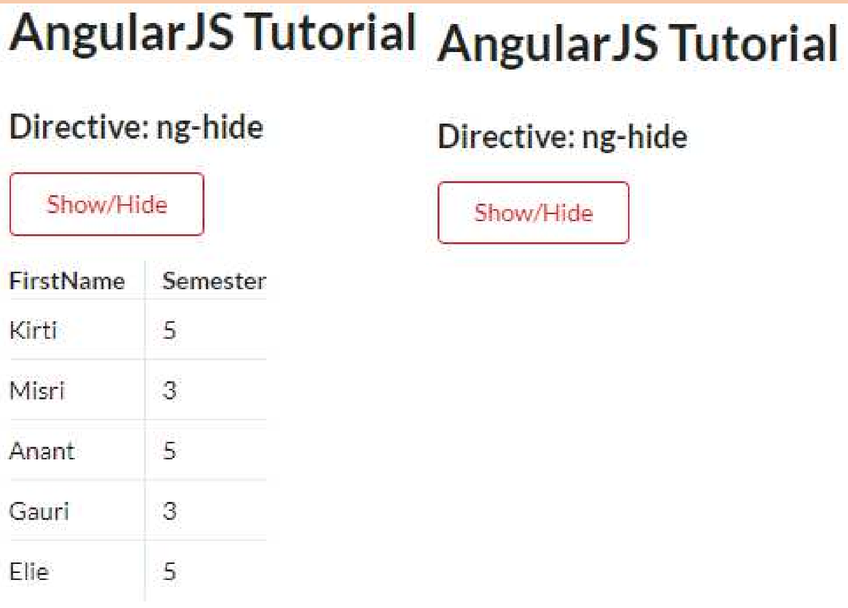
File: program6-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS Tutorial</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></
    script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.min.css" />
    <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"
    ></script>
    <script src="program6-2.js"></script>
</head>
<body ng-app="myModule">
```

```
    <div class="ui container">
        <h1>AngularJS Tutorial</h1>
        <h3>Directive: ng-hide</h3>
        <div ng-controller="myController">
            <div ng-click="showHide()" class="ui animated red basic button"
             tabindex="0">
                <div class="visible content" >Show/Hide</div>
                    <div class="hidden content">
                        <i class="right arrow icon"></i>
                    </div>
            </div>
            <table ng-hide="isVisible" class="ui very basic collapsing celled
            compact table">
                <th>FirstName</th>
                <th>Semester</th>
                <tr ng-repeat="d in data">
                    <td>{{d.fName}}</td>
                    <td>{{d.semester}}</td>
                </tr>
            </table>
        </div>
    </div>
</body>
</html>
```

File: program6-2.js

```
var myApp = angular.module("myModule",[]);
myApp.controller("myController", function($scope){
    var student = [
        {fName:"Kirti", semester:5},
        {fName:"Misri", semester:3},
        {fName:"Anant", semester:5},
        {fName:"Gauri", semester:3},
        {fName:"Elie", semester:5},
    ];
    $scope.data = student;
    $scope.isVisible = true;
    $scope.showHide = function(){
        $scope.isVisible = !$scope.isVisible;
    }
});
```



FirstName	Semester
Kirti	5
Misri	3
Anant	5
Gauri	3
Elie	5

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.

On executing the code successfully, you may find the following output in your browser.

The output gives,

- In in initial condition you may see that the div tag which has the text "AngularJs" is shown because the property value of false is sent to the ng-hide directive.
- When we click on the "Hide Angular" button the property value of true will sent to the ng-hide directive. Hence the below output will be shown, in which the word "Angular" will be hidden.

AngularJS | ng-src Directive

The **ng-src Directive** in AngularJS is used to specify the src attribute of an <img> element. It ensures that the wrong image is not produced until AngularJS has been evaluated. It is supported by <img> element.

Syntax:

```
<img ng-src="url"> </img>
```

Program 6.3

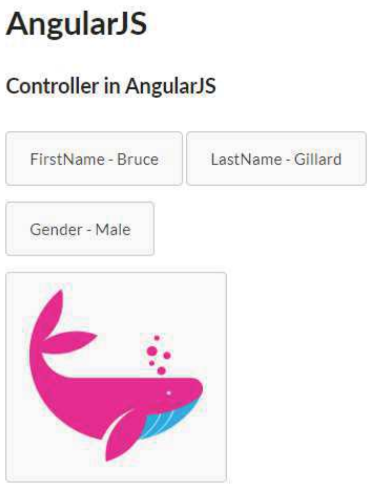
File: program6-3.html

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></scri
pt>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
ui/2.4.1/semantic.min.css" />
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></s
cript>
    <script src="program6-3.js"></script>
</head>
<body ng-App="myModule">
    <div class="ui container">
        <br/><br/>
        <h1>AngularJS</h1>
        <h3>Controller in AngularJS</h3>
        <div ng-controller="myController">
            <div class="row">
                <div class="ui compact message">
                    <p> FirstName - {{ employee.firstName }}</p>
                </div>
                <div class="ui compact message">
                    <p> LastName - {{ employee.lastName }}</p>
                </div>
            </div>
            <div class="row">
                <div class="ui compact message">
                    <p> Gender - {{ employee.gender }}</p>
                </div>
            </div>
            <div class="ui compact message">
                <p><img ng-src="{{ employee.logo }}" width=150
               height=150/></p>
            </div>
        </div>
    </div>
</body>
</html>
```

File: Program6-3.js

```
var app=angular.module("myModule",[]);
app.controller('myController', function($scope){
    var employee = {
        firstName: "Bruce",
        lastName: "Gillard",
        gender: "Male",
        logo: "logo.png"
    };
    $scope.employee = employee;
});
```


AngularJS

Controller in AngularJS

FirstName - Bruce

LastName - Gillard

Gender - Male

How to include HTML file in AngularJS

After learning a lot of good stuffs above, let us again get a good grasp over including the AngularJS HTML File which helps in distributed files work in a single place. To be more precise, the primary purpose of the ng-include directive is used to fetch, compile and include an external HTML fragment in the main application.

One more interesting fact is that HTML does not support other HTML pages to get embedded with other HTML pages. To achive this one can use any one of the following options -

- **Using Ajax** – Make a server call to get the corresponding HTML page and set it in the innerHTML of HTML control.
- **Using Server Side Includes** – JSP, PHP and other web side server technologies can include HTML pages within a dynamic page.

Using AngularJS, we can embed HTML pages within an HTML page using ng-include directive.

Note:

- The **ng-include** directive includes HTML from an external file.
- The included content will be included as child nodes of the specified element.
- The value of the **ng-include** attribute can also be an expression, returning a filename.
- By default, the included file must be located on the same domain as the document.

Interesting isn’t it? Let us watch the code on how this can be achieved?

Step 1) Let's write the below code in a file called myTable.html. This is the file which will be injected into 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.

Program 6.4

File: program6-4.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS Tutorial</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></
    script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.min.css" />
    <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"
    ></script>
    <script src="program6-4.js"></script>
</head>
<body ng-app="myModule">
    <div class="ui container">
        <h1>AngularJS Tutorial</h1>
        <h3>Directive: ng-include</h3>
        <div ng-controller="myController">
            <div ng-include="'myTable.html'"/>
        </div>
    </div>
</body>
</html>
```

File: program6-4.js

```
var myApp = angular.module("myModule",[]);
myApp.controller("myController", function($scope){
    var student = [
        {fName:"Kirti", semester:5},
        {fName:"Misri", semester:3},
        {fName:"Anant", semester:5},
        {fName:"Gauri", semester:3},
        {fName:"Elie", semester:5},
    ];
    $scope.data = student;
});
```

File: myTable.html

```
<table ng-hide="isVisible" class="ui very basic collapsing celled compact
table">
```

AngularJS

```
    <th>FirstName</th>
    <th>Semester</th>
    <tr ng-repeat="d in data">
        <td>{{d.fName}}</td>
        <td>{{d.semester}}</td>
    </tr>
</table>
```

Step 2) let's write the below code in a file called index.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 'myTable.html'. The statement has been highlighted in bold in the below code. So the div tag ' **<div ng-include="myTable.html"></div>'** will be replaced by the entire code in the 'myTable.html' file.
2. In the controller, a "student" variable is created as part of the $scope object. This variable contains a list of key-value pairs.

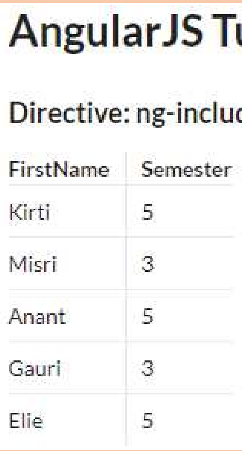
In our example, the key value pairs are

1. First Name – This denotes the name of the students in JSON Form.
2. Semester – This gives a semester of the students.

The student variable is also accessed in the 'myTable.html' file.

When you execute the above code, you will get the following output.

Output:



AngularJS Tutorial

Directive: ng-include

FirstName	Semester
Kirti	5
Misri	3
Anant	5
Gauri	3
Elie	5

Summary:

- By default, we know that HTML does not provide a direct way to include HTML content from other files like other programming languages.
- JavaScript along with jQuery is the best-preferred option for embedding HTML content from other files.
- Another way of including HTML content from other files is to use the <include> directive and the virtual parameter keyword. The virtual parameter keyword is used to denote the file which needs to be embedded. This is known as server-side includes.
- Angular also provides the facility to include files by using the ng-include directive. This directive can be used to inject code from external files directly into the main HTML file.

Okay, let me see give one more example for ng-include, here rather than defining directly in the HTML file, we would define the filename in $scope variable.

Program 6.5

File: program6-5.html

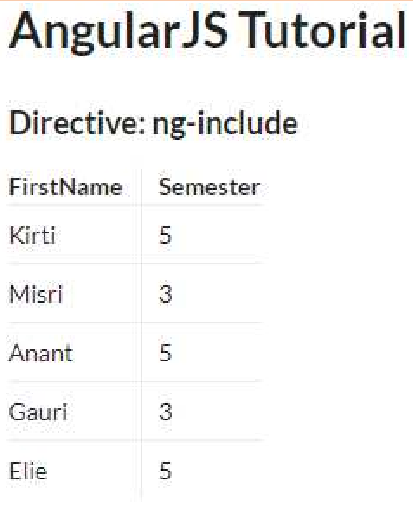
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>AngularJS Tutorial</title>
    <!-- Semantic UI / JQuery / AngularJS script added here-->
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></
    script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.js"></script>
    <link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-
    ui/2.4.1/semantic.min.css" />
    <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"
    ></script>
    <script src="program6-5.js"></script>
</head>
<body ng-app="myModule">
    <div class="ui container">
        <h1>AngularJS Tutorial</h1>
        <h3>Directive: ng-include</h3>
        <div ng-controller="myController">
            <div ng-include="myTable"/>
        </div>
    </div>
</body>
</html>
```

File: program6-5.js

```
var myApp = angular.module("myModule",[]);
myApp.controller("myController", function($scope){
    var student = [
        {fName:"Kirti", semester:5},
        {fName:"Misri", semester:3},
        {fName:"Anant", semester:5},
        {fName:"Gauri", semester:3},
        {fName:"Elie", semester:5},
    ];
    $scope.data = student;
    $scope.myTable = 'myTable.html';
});
```

File: myTable.html

```
<table ng-hide="isVisible" class="ui very basic collapsing celled compact
table">
    <th>FirstName</th>
    <th>Semester</th>
    <tr ng-repeat="d in data">
        <td>{{d.fName}}</td>
        <td>{{d.semester}}</td>
    </tr>
</table>
```



AngularJS Tutorial

Directive: ng-include

FirstName	Semester
Kirti	5
Misri	3
Anant	5
Gauri	3
Elie	5

Summary:

- AngularJS also allows you to use the $scope variable as mentioned in this example above, where in we pass the string to **$scope.myTable (program41.js)** in our case with the value **'myTable.html'**. This is then passed to the **<div ng-include="myTable">**

"A king needs a kingdom to be a King. Not a QUEEN. Remember That."

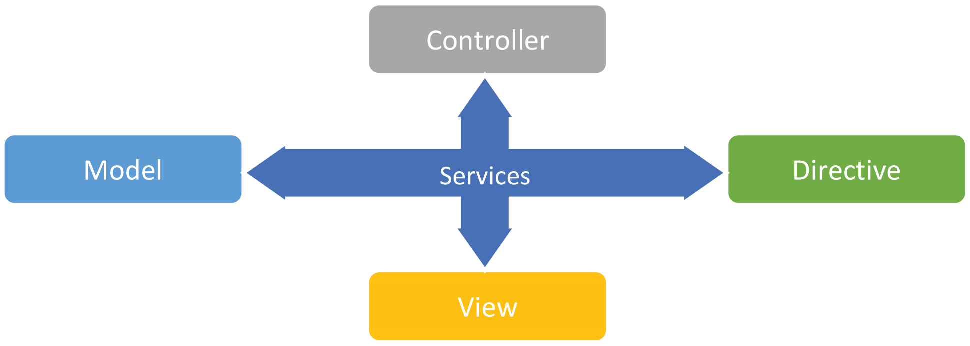
Chapter 7 - AngularJS Service

AngularJS services are JavaScript functions for specific tasks, which can be reused throughout the application.

AngularJS includes services for different purposes. For example, $http service can be used to send an AJAX request to the remote server. AngularJS also allows you to create custom service for your application.

Tips: AngularJS built-in services starts with $, same as other built-in objects.

Most AngularJS services interact with the controller, model or custom directives. However, some services interact with view (UI) also for UI specific tasks.



The following table lists all the built-in AngularJS services.

$anchorScroll	$exceptionHandler	$interval	$rootScope
$animate	$filter	$locale	$sceDelegate
$cacheFactory	$httpParamSerializer	$location	$sce
$templateCache	$httpParamSerializerJQLike	$log	$templateRequest
$compile	$http	$parse	$timeout
$controller	$httpBackend	$q	$window
$document	$interpolate	$rootElement	

The best part of the AngularJS services are they are lazy instantiated and are singleton. This in terms in simpler terms can be explained that whenever the services are needed or utilized, during that time itself the services get loaded and as and when the application needs the components, it instantiates the services as such. Also, all the component share the same instance of a service. Thus, making it easier for the browser to act fastly and flawlessly.

AngularJS $http.get Method

In AngularJS $http service is used to send or get data from remote http servers using browsers XMLHttpRequest object.

Bibliography

[1] "JavaScript Immediately-invoked Function Expressions (IIFE)," 20 May 2018. [Online]. Available: https://flaviocopes.com/javascript-iife/. [Accessed 10 July 2019].

[2] "AngularJS - MVC Architecture," TutorialsPoint, [Online]. Available: https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture. [Accessed 16 July 2019].

[3] "AngularJS Events: ng-click, ng-show, ng-hide [Example]," Guru99, [Online]. Available: https://www.guru99.com/angularjs-events.html. [Accessed 02 August 2019].

[4] "AngularJS | ng-src Directive," [Online]. Available: https://www.geeksforgeeks.org/angularjs-ng-src-directive/. [Accessed 22 07 2019].

[5] "ng-include in AngularJS: How to include HTML File [Example]," Guru99, [Online]. Available: https://www.guru99.com/ng-include-angularjs.html. [Accessed 02 August 2019].

[6] "AngularJS - Includes," TutorialsPoint, [Online]. Available: https://www.tutorialspoint.com/angularjs/angularjs_includes. [Accessed 02 August 2019].

[7] "AngularJS ng-include Directive," w3schools, [Online]. Available: https://www.w3schools.com/angular/ng_ng-include.asp. [Accessed 02 August 2019].

[8] "AngularJS Service," TutorialsTeacher, [Online]. Available: https://www.tutorialsteacher.com/angularjs/angularjs-service. [Accessed 04 August 2019].

[9] "AngularJS HTTP Get Method ($http.get) with Example," Tutlane, [Online]. Available: https://www.tutlane.com/tutorial/angularjs/angularjs-http-get-method-http-get-with-example. [Accessed 4 08 2019].

[10] "$log Service," TutorialsTeacher, [Online]. Available: https://www.tutorialsteacher.com/angularjs/angularjs-log-service. [Accessed 04 August 2019].

[11] kudvenkat, "Angular anchorscroll with database data," Youtube Video, 4 Feb 2016. [Online]. Available: https://www.youtube.com/watch?v=2OZMthAwldk. [Accessed 03 Sep 2019].

[12] T. CBT, "AngularJS: Developing Custom Services (factory vs service vs Provider)," Youtube.com, 29 December 2015. [Online]. Available: https://www.youtube.com/watch?v=oUXku28ex-M. [Accessed 18 August 2019].

[13] "AngularJS: Developing custom Services," Jaywant Topno , 20 Mar 2018. [Online]. Available: https://www.valuebound.com/resources/blog/angularjs-developing-custom-services. [Accessed 05 Aug 2019].

[14] kudvenkat, "Angular nested scopes and controller as syntax," Youtube, 16 March 2016. [Online]. Available: https://www.youtube.com/watch?v=eAMMvnolbZc&list=PL6n9fhu94yhWKHkcL7RJmmXyxkuFB3KSl&index=33. [Accessed 26 August 2019].

[15] "AngularJS - Scopes," TutorialsPoint, [Online]. Available: https://www.tutorialspoint.com/angularjs/angularjs_scopes.htm#. [Accessed 26 August 2019].

[16] "AngularJS scope inheritance:," Codes Java, [Online]. Available: https://codesjava.com/scope-inheritance-angularjs. [Accessed 26 August 2019].

[17] T. CBT, "AngularJS: Understanding Scope Inheritance," wwwyoutube.com, 17 December 2015. [Online]. Available: https://www.youtube.com/watch?v=V4nocbBMaFU&list=PLvZkOAgBYrsS_ugyamsNpCgLSmtIXZGiz. [Accessed 26 August 2019].

[18] S. Chauhan, "Understanding AngularJS $rootScope and $scope," DotnNetTricks, 10 Sep 2014. [Online]. Available: https://www.dotnettricks.com/learn/angularjs/understanding-angularjs-rootscope-and-scope. [Accessed 14 Sep 2019].

[19] "AngularJS - Directives," Tutorials Point, [Online]. Available: https://www.tutorialspoint.com/angularjs/angularjs_directives.htm. [Accessed 16 09 2019].

[20] "AngularJS - Custom Directives," TutorialsPoint, [Online]. Available: https://www.tutorialspoint.com/angularjs/angularjs_custom_directives.htm. [Accessed 16 09 2019].

[21] "AngularJS Directives," w3school, [Online]. Available: https://www.w3schools.com/angular/angular_directives.asp. [Accessed 16 09 2019].

[22] "AngularJS Custom Directive Template with Example," [Online]. Available: https://www.tutlane.com/tutorial/angularjs/angularjs-custom-directive-template-with-example. [Accessed 16 09 2019].

[23] J. Bennett, "AngularJS Routing Example – ngRoute, $routeProvider," [Online]. Available: https://www.journaldev.com/6225/angularjs-routing-example-ngroute-routeprovider. [Accessed 08 September 2019].

[24] Kudvenkat, "AngularJS route change events," Youtube, 23 Mar 2016. [Online]. Available: https://www.youtube.com/watch?v=tdIgsclQR6E. [Accessed 27 Sep 2019].

[25] C. o. code, "AngularJS Routing Using UI-Router," scotch.io, 10 March 2014. [Online]. Available: https://scotch.io/tutorials/angular-routing-using-ui-router. [Accessed 25 Sept 2019].

[26] I. Gelman, "UI-Router And Case-Sensitive URLs," https://500tech.com, 15 Feb 2015. [Online]. Available: https://500tech.com/blog/all/ui-router-and-case-sensitive-urls. [Accessed 10 Oct 2019].

[27] Chris, "3 Simple Tips for Using UI-Router," scotch.io, 11 Jul 2016. [Online]. Available: https://scotch.io/tutorials/3-simple-tips-for-using-ui-router. [Accessed 10 Oct 2019].

[28] "AngularJS - Dependency Injection," Tutorials Point, [Online]. Available: https://www.tutorialspoint.com/angularjs/angularjs_dependency_injection. [Accessed 14 July 2019].

[29] "AngularJS Data Binding," W3School, [Online]. Available: https://www.w3schools.com/angular/angular_databinding.asp. [Accessed 14 July 2019].

[30] "AngularJS Unit Testing: Karma Jasmine Tutorial," Guru99.com, [Online]. Available: https://www.guru99.com/angularjs-testing-unit.html. [Accessed 14 July 2019].

[31] C. Ribeiro, "A Practical Guide to Angular Directives," SitePoint, [Online]. Available: https://www.sitepoint.com/practical-guide-angular-directives/. [Accessed 17 July 2019].

[32] "AngularJS Filters," w3Schools, [Online]. Available: https://www.w3schools.com/angular/angular_filters.asp. [Accessed 17 July 2019].

[33] "AngularJS Modules," utorialsteacher, [Online]. Available: https://www.tutorialsteacher.com/angularjs/modules-in-angularjs. [Accessed 21 07 2019].

[34] "AngularJS Controller Tutorial with Example," Guru99, [Online]. Available: https://www.guru99.com/angularjs-controller.html. [Accessed 21 07 2019].

[35] "AngularJS Data Binding," JavaTPoint, [Online]. Available: https://www.javatpoint.com/angularjs-data-binding. [Accessed 22 07 2019].

[36] "Event Handling in AngularJS," TutorialRide, [Online]. Available: https://www.tutorialride.com/angularjs/event-handling-in-angularjs.htm. [Accessed 26 July 2019].

[37] N. Biswas, "Sorting in AngularJS," DotNetFunda, [Online]. Available: http://www.dotnetfunda.com/articles/show/3451/sorting-in-angularjs. [Accessed 26 07 2019].

[38] J. BENNETT, "AngularJS Filters Example Tutorial," JournalDev, [Online]. Available: https://www.journaldev.com/6020/angularjs-filters-example-tutorial. [Accessed 28 07 2019].

[39] A. Kukic, "Building Custom AngularJS Filters," Scoth.io, -2 February 2015. [Online]. Available: https://scotch.io/tutorials/building-custom-angularjs-filters. [Accessed 31 07 2019].

[40] "AngularJS Service," tutorialsteacher, [Online]. Available: https://www.tutorialsteacher.com/angularjs/angularjs-service. [Accessed 04 08 2019].

[41] T. CBT, "AngularJS: Developing Custom Services (factory vs service vs Provider)," 29 December 2015. [Online]. Available: https://www.youtube.com/watch?v=oUXku28ex-M. [Accessed 17 August 2019].

[42] "AngularJS: binding HTML code with ng-bind-html and $sce.trustAsHtml," https://benohead.com, 28 Sep 2019. [Online]. Available: https://benohead.com/angularjs-binding-html-code-with-ng-bind-html-and-sce-trustashtml/. [Accessed 21 Apr 2015].

[43] S. Venu, "Convert XML to JSON In Angular JS," https://dzone.com, 7 Nov j2015. [Online]. Available: https://dzone.com/articles/convert-xml-to-json-in-angular-js. [Accessed 29 Sep 2019].

[44] "ngBindHtml," AngularJS, [Online]. Available: https://docs.angularjs.org/api/ng/directive/ngBindHtml. [Accessed 29 Sep 2019].

[45] B. Morelli, "JavaScript: What the heck is a Callback?," codeburst.io, 12 June 2017. [Online]. Available: https://codeburst.io/javascript-what-the-heck-is-a-callback-aba4da2deced. [Accessed 10 July 2019].

[46] "JavaScript | Callbacks," GeeksForGeeks, [Online]. Available: https://www.geeksforgeeks.org/javascript-callbacks/. [Accessed 10 July 2019].

[47] A. Grant, Beginning AngularJS, 1 ed., Apress, 2014, pp. XII, 200.

[48] A. RCF, "GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/closure-in-javascript/. [Accessed 10 July 2019].

[49] N. S. Dutta, "How-to: call() , apply() and bind() in JavaScript," www.codementor.io, 05 June 2017. [Online]. Available: https://www.codementor.io/niladrisekhardutta/how-to-call-apply-and-bind-in-javascript-8i1jca6jp. [Accessed 10 July 2019].

Index

AngularJS
The Web Compilation

The book is intended to share the knowledge only.
The Book is for all the novice and intermediate programmers who want to explore Angular 1.x to its joyful and easy way for programming.The best part of the book is that it does not cover much of the details as theory but it focuses more on the AngularJS practical. The code is written as a whole and easy to understand.

The book intends to focus mainly on the following topic.
1. AngularJS 1.x
2. Basics of JavaScript
3. Semantic – UI
4. Basics of PHP for performing various operations of database.

I believe that programming language is a special art which can be best learned by practicing it. Thus, all the example code typed in it are purely helpful and quite understanding in nature. The book intends to try making you prepare for expanding,modifying and creating your own creation from the scratch. Slowly and steadily you will feel that you have become fluent in the language of the web. As, I have prepared this book by reading various sources.

I would be glad to hear your feedback, comments and questions from you all as the reader: bhagchandani.niraj@gmail.com

I hope you enjoy the journey towards the AngularJS programming. Happy Coding.

9 789353 963637

Niraj Bhagchandani