AngularJS
BSc-ECS-III/Semester-VI

Unit 1: Introduction to JavaScript [8]

Including scripts on a page, adding statements or expressions, comments, functions, parameters and return values, primitive types, JavaScript operators, Equality Vs. Identity, pre, post-increment, Reading and modifying objects properties, adding methods to objects, Control flow statements, working with arrays, Error handling mechanisms using try/catch/finally, throwing our own exceptions.

Unit 2: Basics of Angular JS [9]

Introduction to Angular JS, Features of Angular JS, MVC Architecture, Setting up the Environment, First Simple Application, Working with Directives- Directive lifecycle, Using Angular JS built-in directives, Core Directives, Conditional Directives, Style Directives, Mouse and Keyboard Events Directives, Matching directives, creating a custom directive. Expressions and Data Biding- Number and String Expressions, Object Binding and Expressions, Working with Arrays, Forgiving Behavior, Understanding Data binding.

Unit 3: Controllers [6]

Understanding Controllers, Programming Controllers, and scope object Adding Behavior to a Scope Object Passing Parameters to the Methods, Array as members in Controller Scope, Nested Controllers and Scope Inheritance, Multiple Controllers and their scopes.

Unit 4: Filters and Modules [6]

Filters: Built-in filters, Uppercase, and Lowercase Filters, Currency and Number Formatting Filters, Order By Filter, Filter Filter, Creating custom filters.
AngularJS Modules: Introduction to Angular JS Module, Module Loading and Dependencies, Recommended Setup of Application, Creation vs Retrieval.

Unit 5: Forms [10]

Angular JS Forms: Working with Angular Forms, Model binding, Understanding Data Binding, Binding controls to data, Form controller, Validating Angular Forms, Form events, Updating models with a twist, $error object, Scope- What is scope, Scope lifecycle, Two-way data binding, Scope inheritance, Scope and controllers, Scope and directives, $apply and $watch, Rootscope, Scope broadcasting, Scope events.

Unit 6: Services and Ajax in Angular JS [5]

Understanding Services, Developing Creating Services, Using a Service, Injecting Dependencies in a Service. $http Service, $q Service, Ajax Implementation using $http and $q Service

Reference Books:

- Beginning AngularJS- By Andrew Grant- 2014
- Professional AngularJS by Diego Netto and Valeri Karpov-Wrox press
- Learning AngularJS by Brad Dayley- Addison-Wesley Professiona
- AngularJS by Brad Green and Shyam Seshadri- O'Reilly

Introduction to JavaScript

Our **JavaScript Tutorial** is designed for beginners and professionals both. JavaScript is used to create client-side dynamic pages.

JavaScript is *an object-based scripting language* which is lightweight and cross-platform.

JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

What is JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.

Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

History of JavaScript

In 1993, Mosaic, the first popular web browser, came into existence. In the year 1994, Netscape was founded by Marc Andreessen. He realized that the web needed to become more dynamic. Thus, a 'glue language' was believed to be provided to HTML to make web designing easy for designers and part-time programmers. Consequently, in 1995, the company recruited Brendan Eich intending to implement and embed Scheme programming language to the browser. But, before Brendan could start, the company merged with Sun Microsystems for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms. Now, two languages were there: Java and the scripting language. Further, Netscape decided to give a similar name to the scripting language as Java's. It led to 'Javascript'. Finally, in May 1995, Marc Andreessen coined the first code of Javascript named 'Mocha'. Later, the marketing team replaced the name with 'LiveScript'. But, due to trademark reasons and certain other reasons, in December 1995, the language was finally renamed to 'JavaScript'. From then, JavaScript came into existence.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

JavaScript Example

```
<script>
document.write("Hello JavaScript by JavaScript");
</script>
```

JavaScript Example

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.
Let's create the first JavaScript example.

```
<script type="text/javascript">
document.write("JavaScript is a simple language for javatpoint learners");
</script>
```

The **script** tag specifies that we are using JavaScript.
The **text/javascript** is the content type that provides information to the browser about the data.
The **document.write()** function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

3 Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external javaScript)

1) JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

```
<script type="text/javascript">
 alert("Hello Javatpoint");
</script>
```

2) JavaScript Example : code between the head tag

Let's see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.
In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name as given below.
To call function, you need to work on event. Here we are using onclick event to call msg() function.

```
<html>
<head>
<script type="text/javascript">
function msg(){
 alert("Hello Javatpoint");
}
</script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

External JavaScript file

We can create external JavaScript file and embed it in many html page.
It provides **code re usability** because single JavaScript file can be used in several html pages.
An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.
Let's create an external JavaScript file that prints Hello Javatpoint in a alert dialog box.

message.js

```
function msg(){
 alert("Hello Javatpoint");
}
```

Let's include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

```
<html>
<head>
<script type="text/javascript" src="message.js"></script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

Advantages of External JavaScript

There will be following benefits if a user creates an external javascript:

1. It helps in the reusability of code in more than one HTML file.
2. It allows easy code readability.
3. It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
4. It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
5. The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

There are the following disadvantages of external files:

1. The stealer may download the coder's code using the url of the js file.
2. If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
3. The web browser needs to make an additional http request to get the js code.
4. A tiny to a large change in the js code may cause unexpected results in all its dependent files.
5. We need to check each file that depends on the commonly created external javascript file.
6. If it is a few lines of code, then better to implement the internal javascript code.

JavaScript Comment

1. JavaScript comments
2. Advantage of javaScript comments
3. Single-line and Multi-line comments

The **JavaScript comments** are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.
The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

1. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.
2. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

Types of JavaScript Comments

There are two types of comments in JavaScript.

1. Single-line Comment
2. Multi-line Comment

JavaScript Single line Comment

It is represented by double forward slashes (//). It can be used before and after the statement.
Let's see the example of single-line comment i.e. added before the statement.

```
<script>
// It is single line comment
document.write("hello javascript");
</script>
```

Let's see the example of single-line comment i.e. added after the statement.

```
<script>
var a=10;
var b=20;
var c=a+b;//It adds values of a and b variable
document.write(c);//prints sum of 10 and 20
</script>
```

JavaScript Multi line Comment

It can be used to add single as well as multi line comments. So, it is more convenient.
It is represented by forward slash with asterisk then asterisk with forward slash. For example:

1. /* your code here */

It can be used before, after and middle of the statement.

```
<script>
/* It is multi line comment.
It will not be displayed */
document.write("example of javascript multiline comment");
</script>
```

JavaScript Variable

1. JavaScript variable
2. JavaScript Local variable
3. JavaScript Global variable

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.
There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

```
var x = 10;
var _value="sonoo";
```

Incorrect JavaScript variables

```
var  123=30;
var *aa=320;
```

Example of JavaScript variable

Let's see a simple example of JavaScript variable.

```
<script>
var x = 10;
var y = 20;
var z=x+y;
document.write(z);
</script>
```

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>
function abc(){
var x=10;//local variable
}
</script>
```

Or,

```
<script>
If(10<13){
var y=20;//JavaScript local variable
}
</script>
```

JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

```
<script>
var data=200;//gloabal variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
</script>
```

JavaScript Global Variable

A **JavaScript global variable** is declared outside the function or declared with window object. It can be accessed from any function.

Let's see the simple example of global variable in JavaScript.

```
<script>
var value=50;//global variable
function a(){
alert(value);
```

```
}
function b(){
alert(value);
}
</script>
```

Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object**. For example:

1. window.value=90;

Now it can be declared inside any function and can be accessed from any function. For example:

```
function m(){
window.value=100;//declaring global variable by window object
}
function n(){
alert(window.value);//accessing global variable from other function
}
```

Internals of global variable in JavaScript

When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also. For example:

```
var value=50;
function a(){
alert(window.value);//accessing global variable
}
```

Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a=40;//holding number
var b="Rahul";//holding string
```

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. For example:

```
var sum=10+20;
```

Here, + is the arithmetic operator and = is the assignment operator.
There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Special Operators

The following operators are known as JavaScript special operators.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

JavaScript If-else

The **JavaScript if-else statement** is used *to execute the code whether condition is true or false*. There are three forms of if statement in JavaScript.

1. If Statement
2. If else statement
3. if else if statement

JavaScript If statement

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

```
if(expression){
//content to be evaluated
}
```

Flowchart of JavaScript If statement

Let's see the simple example of if statement in javascript.

```
<script>
var a=20;
if(a>10){
document.write("value of a is greater than 10");
}
</script>
```

JavaScript If...else Statement

It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

```
if(expression){
//content to be evaluated if condition is true
}
else{
//content to be evaluated if condition is false
}
```

Flowchart of JavaScript If...else statement

Let's see the example of if-else statement in JavaScript to find out the even or odd number.

```
<script>
var a=20;
if(a%2==0){
document.write("a is even number");
}
else{
document.write("a is odd number");
}
</script>
```

JavaScript If...else if statement

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

```
if(expression1){
//content to be evaluated if expression1 is true
}
else if(expression2){
//content to be evaluated if expression2 is true
}
else if(expression3){
//content to be evaluated if expression3 is true
}
else{
//content to be evaluated if no expression is true
}
```

Let's see the simple example of if else if statement in javascript.

```
<script>
var a=20;
if(a==10){
document.write("a is equal to 10");
}
else if(a==15){
document.write("a is equal to 15");
}
else if(a==20){
document.write("a is equal to 20");
}
else{
document.write("a is not equal to 10, 15 or 20");
}
</script>
```

JavaScript Switch

The **JavaScript switch statement** is used *to execute one code from multiple expressions*. It is just like else if statement that we have learned in previous page. But it is convenient than *if..else..if* because it can be used with numbers, characters etc.

The signature of JavaScript switch statement is given below.

```
switch(expression){
case value1:
 code to be executed;
 break;
case value2:
 code to be executed;
 break;
.....
default:
 code to be executed if above values are not matched;
}
```

Let's see the simple example of switch statement in javascript.

```
<script>
var grade='B';
var result;
switch(grade){
case 'A':
result="A Grade";
break;
case 'B':
result="B Grade";
break;
case 'C':
result="C Grade";
break;
default:
result="No Grade";
}
document.write(result);
</script>
```

The switch statement is fall-through i.e. all the cases will be evaluated if you don't use break statement.

Let's understand the behaviour of switch statement in JavaScript.

```
<script>
var grade='B';
var result;
switch(grade){
case 'A':
result+=" A Grade";
case 'B':
result+=" B Grade";
case 'C':
```

```
result+=" C Grade";
default:
result+=" No Grade";
}
document.write(result);
</script>
```

JavaScript Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

1) JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

```
for (initialization; condition; increment)
{
   code to be executed
}
```

Let's see the simple example of for loop in javascript.

```
<script>
for (i=1; i<=5; i++)
{
document.write(i + "<br/>")
}
</script>
```

2) JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

```
while (condition)
{
   code to be executed
}
```

Let's see the simple example of while loop in javascript.

```
<script>
var i=11;
while (i<=15)
{
document.write(i + "<br/>");
i++;
}
</script>
```

3) JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

```
do{
   code to be executed
}while (condition);
```

Let's see the simple example of do while loop in javascript.

```
<script>
var i=21;
do{
document.write(i + "<br/>");
i++;
}while (i<=25);
</script>
```

JavaScript Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

JavaScript Function Syntax

The syntax of declaring function is given below.

```
1. function functionName([arg1, arg2, ...argN]){
2.  //code to be executed
3. }
```

JavaScript Functions can have 0 or more arguments.

JavaScript Function Example

Let's see the simple example of function in JavaScript that does not has arguments.

```
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="call function"/>
```

JavaScript Function Arguments

We can call function by passing arguments. Let's see the example of function that has one argument.

```
<script>
function getcube(number){
alert(number*number*number);
}
</script>
<form>
<input type="button" value="click" onclick="getcube(4)"/> </form>
```

Function with Return Value

We can call function that returns a value and use it in our program. Let's see the example of function that returns value.

```
<script>
function getInfo(){
return "hello javatpoint! How r u?";
}
</script>
<script>
document.write(getInfo());
</script>
```

JavaScript Function Object

In JavaScript, the purpose of **Function constructor** is to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

Syntax

1. new Function ([arg1[, arg2[, ...argn]],] functionBody)

Parameter

arg1, arg2, ... , argn - It represents the argument used by function.

functionBody - It represents the function definition.

JavaScript Function Methods

Let's see function methods with description.

Method	Description
apply()	It is used to call a function contains this value and a single array of arguments.
bind()	It is used to create a new function.
call()	It is used to call a function contains this value and an argument list.
toString()	It returns the result in a form of a string.

JavaScript Function Object Examples

Example 1

Let's see an example to display the sum of given numbers.

```
<script>
var add=new Function("num1","num2","return num1+num2");

document.writeln(add(2,5));

</script>
```

Example 2

Let's see an example to display the power of provided value.

```
<script>
var pow=new Function("num1","num2","return Math.pow(num1,num2)");
document.writeln(pow(2,3));
</script>
```

JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

1) JavaScript array literal

The syntax of creating array using array literal is given below:

1. var arrayname=[value1,value2.....valueN];

As you can see, values are contained inside [ ] and separated by , (comma).
Let's see the simple example of creating and using array in JavaScript.

```
<script>
var emp=["Sonoo","Vimal","Ratan"];
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br/>");
}
```

The .length property returns the length of an array.

2) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

1. var arrayname=new Array();

Here, **new keyword** is used to create instance of array.
Let's see the example of creating array directly.

```
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";

for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.
The example of creating object by array constructor is given below.

```
<script>
var emp=new Array("Jai","Vijay","Smith");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

JavaScript Array Methods

Let's see the list of JavaScript array methods with their description.

Methods	Description
concat()	It returns a new array object that contains two or more merged arrays.

copywithin()	It copies the part of the given array with its own elements and returns the modified array.
entries()	It creates an iterator object and a loop that iterates over each key/value pair.
every()	It determines whether all the elements of an array are satisfying the provided function conditions.
flat()	It creates a new array carrying sub-array elements concatenated recursively till the specified depth.
flatMap()	It maps all array elements via mapping function, then flattens the result into a new array.
fill()	It fills elements into an array with static values.
from()	It creates a new array carrying the exact copy of another array element.
filter()	It returns the new array containing the elements that pass the provided function conditions.
find()	It returns the value of the first element in the given array that satisfies the specified condition.
findIndex()	It returns the index value of the first element in the given array that satisfies the specified condition.
forEach()	It invokes the provided function once for each element of an array.
includes()	It checks whether the given array contains the specified element.
indexOf()	It searches the specified element in the given array and returns the index of the first match.
isArray()	It tests if the passed value ia an array.
join()	It joins the elements of an array as a string.
keys()	It creates an iterator object that contains only the keys of the array, then loops through these keys.
lastIndexOf()	It searches the specified element in the given array and returns the index of the last match.
map()	It calls the specified function for every array element and returns the new array
of()	It creates a new array from a variable number of arguments, holding any type of argument.
pop()	It removes and returns the last element of an array.
push()	It adds one or more elements to the end of an array.
reverse()	It reverses the elements of given array.
reduce(function, initial)	It executes a provided function for each value from left to right and reduces the array to a single value.
reduceRight()	It executes a provided function for each value from right to left and reduces the array to a single value.
some()	It determines if any element of the array passes the test of the implemented function.
shift()	It removes and returns the first element of an array.
slice()	It returns a new array containing the copy of the part of the given array.
sort()	It returns the element of the given array in a sorted order.
splice()	It add/remove elements to/from the given array.
toLocaleString()	It returns a string containing all the elements of a specified array.
toString()	It converts the elements of a specified array into string form, without affecting the original array.
unshift()	It adds one or more elements in the beginning of the given array.
values()	It creates a new iterator object carrying values for each index in the array.

Exception Handling in JavaScript

An exception signifies the presence of an abnormal condition which requires special operable techniques. In programming terms, an exception is the anomalous code that breaks the normal flow of the code. Such exceptions require specialized programming constructs for its execution.

What is Exception Handling

In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of the code/program. For handling the code, various handlers are used that process the exception and execute the code. **For example**, the Division of a non-zero value with zero will result into infinity always, and it is an exception. Thus, with the help of exception handling, it can be executed and handled.

In exception handling:

A throw statement is used to raise an exception. It means when an abnormal condition occurs, an exception is thrown using throw.

The thrown exception is handled by wrapping the code into the try…catch block. If an error is present, the catch block will execute, else only the try block statements will get executed.

Thus, in a programming language, there can be different types of errors which may disturb the proper execution of the program.

Types of Errors

While coding, there can be three types of errors in the code:

1. **Syntax Error:** When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.
2. **Runtime Error:** When an error occurs during the execution of the program, such an error is known as Runtime error. The codes which create runtime errors are known as Exceptions. Thus, exception handlers are used for handling runtime errors.
3. **Logical Error:** An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error.

Error Object

When a runtime error occurs, it creates and throws an Error object. Such an object can be used as a base for the user-defined exceptions too. An error object has two properties:

1. name: This is an object property that sets or returns an error name.
2. **message:** This property returns an error message in the string form.

Although Error is a generic constructor, there are following standard built-in error types or error constructors beside it:

1. **EvalError:** It creates an instance for the error that occurred in the eval(), which is a global function used for evaluating the js string code.
2. **InternalError:** It creates an instance when the js engine throws an internal error.
3. **RangeError:** It creates an instance for the error that occurs when a numeric variable or parameter is out of its valid range.
4. **ReferenceError:** It creates an instance for the error that occurs when an invalid reference is de-referenced.
5. **SyntaxError:** An instance is created for the syntax error that may occur while parsing the eval().
6. **TypeError:** When a variable is not a valid type, an instance is created for such an error.
7. **URIError:** An instance is created for the error that occurs when invalid parameters are passed in **encodeURI()** or **decodeURI()**.

Exception Handling Statements

There are following statements that handle if any exception occurs:

- throw statements
- try…catch statements
- try…catch…finally statements.

JavaScript try…catch

A try…catch is a commonly used statement in various programming languages. Basically, it is used to handle the error-prone part of the code. It initially tests the code for all possible errors it may contain, then it implements actions to tackle those errors (if occur). A good programming approach is to keep the complex code within the try…catch statements.

Let's discuss each block of statement individually:

try{} statement: Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the catch{} block for taking suitable actions and handle the error. Otherwise, it executes the code written within.

catch{} statement: This block handles the error of the code by executing the set of statements written within the block. This block contains either the user-defined exception handler or the built-in handler. This block executes only when any error-prone code needs to be handled in the try block. Otherwise, the catch block is skipped.

Note: catch {} statement executes only after the execution of the try {} statement. Also, one try block can contain one or more catch blocks.

Syntax:

```
try{
expression; } //code to be written.
catch(error){
expression; } // code for handling the error.
```

try…catch example

```
<html>
<head> Exception Handling</br></head>
<body>
<script>
try{
var a= ["34","32","5","31","24","44","67"]; //a is an array
document.write(a);    // displays elements of a
document.write(b); //b is undefined but still trying to fetch its value. Thus catch block will be invoked
}catch(e){
alert("There is error which shows "+e.message); //Handling error
}
</script>
</body>
</html>
```

Throw Statement

Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors. When throw statement is executed, the statements present after it will not execute. The control will directly pass to the catch block.

Syntax:

1. throw exception;

try…catch…throw syntax

```
try{
throw exception; // user can define their own exception
}
catch(error){
expression; }  // code for handling exception.
```

The exception can be a string, number, object, or boolean value.

throw example with try…catch

```
<html>
<head>Exception Handling</head>
<body>
<script>
try {
  throw new Error('This is the throw keyword'); //user-defined throw statement.
}
catch (e) {
 document.write(e.message); // This will generate an error message
}
</script>
</body>
</html>
```

With the help of throw statement, users can create their own errors.

try…catch…finally statements

Finally is an optional block of statements which is executed after the execution of try and catch statements. Finally block does not hold for the exception to be thrown. Any exception is thrown or not, finally block code, if present, will definitely execute. It does not care for the output too.

Syntax:

```
try{
expression;
}
catch(error){
expression;
}
finally{
expression; } //Executable code
```

try…catch…finally example

```
<html>
<head>Exception Handling</head>
<body>
<script>
try{
var a=2;
if(a==2)
document.write("ok");
}
catch(Error){
document.write("Error found"+e.message);
}
finally{
document.write("Value of a is 2 ");
}
</script>
</body>
```

```
</html>
```

Unit 2: Basics of Angular JS

Introduction to Angular JS:

Angular JS is an open-sourcee JavaScript framework by Google to build web applications. It is an excellent framework for building single phase applications and line of business applications. It is used in Single Page Application (SPA) projects. It extends HTML DOM with additional attributes and makes it more responsive to user actions. AngularJS is open source, completely free, and used by thousands of developers around the world.

AngularJS is an open-source web application framework. It was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google. Its latest version is 1.2.21.

- AngularJS is a efficient framework that can create Rich Internet Applications (RIA).
- AngularJS provides developers an options to write client side applications using JavaScript in a clean Model View Controller (MVC) way.
- Applications written in AngularJS are cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
- AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache license version 2.0.

Advantages of AngularJS:

- Dependency Injection: Dependency Injection specifies a design pattern in which components are given their dependencies instead of hard coding them within the component.
- Two way data binding: AngularJS creates a two way data-binding between the select element and the orderProp model. orderProp is then used as the input for the orderBy filter.
- Testing: Angular JS is designed in a way that we can test right from the start. So, it is very easy to test any of its components through unit testing and end-to-end testing.
- Model View Controller: In Angular JS, it is very easy to develop application in a clean MVC way. You just have to split your application code into MVC components i.e. Model, View and the Controller.
- Directives, filters, modules, routes etc.

Disadvantages of AngularJS:

Though AngularJS comes with a lot of merits, here are some points of concern –

- Not Secure – Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
- Not degradable – If the user of your application disables JavaScript, then nothing would be visible, except the basic page.

Overall, AngularJS is a framework to build large scale, high-performance, and easyto-maintain web applications.

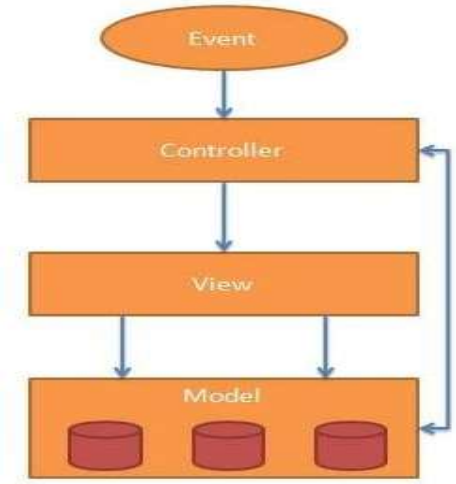
Hello World using AngularJS.

```
<html>
   <head>
      <title>AngularJS First Application</title>
   </head>

   <body>
      <h1>Sample Application</h1>

      <div ng-app = "">
         <p>Enter your Name: <input type = "text" ng-model = "name"></p>
         <p>Hello <span ng-bind = "name"></span>!</p>
      </div>

      <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
```

```
    </script>
```

AngularJS MVC Architecture

MVC stands for Model View Controller. It is a software design pattern for developing web applications. It is very popular because it isolates the application logic from the user interface layer and supports separation of concerns.



The MVC pattern is made up of the following three parts:

1. **Model:** It is responsible for managing application data. It responds to the requests from view and to the instructions from controller to update itself.
2. **View:** It is responsible for displaying all data or only a portion of data to the users. It also specifies the 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.
3. **Controller:** It is responsible to control the relation between models and views. It 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.

The AngularJS framework can be divided into three major parts –

- **ng-app** – This directive defines and links an AngularJS application to HTML.
- **ng-model** – This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind** – This directive binds the AngularJS application data to HTML tags.

Creating AngularJS Application

Step 1: Load framework

Being a pure JavaScript framework, it can be added using <Script> tag.

```
<script
   src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
```

Step 2: Define AngularJS application using ng-app directive

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

Step 3: Define a model name using ng-model directive

```
<p>Enter your Name: <input type = "text" ng-model = "name"></p>
```

Step 4: Bind the value of above model defined using ng-bind directive

```
<p>Hello <span ng-bind = "name"></span>!</p>
```

Executing AngularJS Application

Use the above-mentioned three steps in an HTML page.

testAngularJS.htm

```
<html>
  <head>
    <title>AngularJS First Application</title>
  </head>

  <body>
    <h1>Sample Application</h1>

    <div ng-app = "">
      <p>Enter your Name: <input type = "text" ng-model = "name"></p>
      <p>Hello <span ng-bind = "name"></span>!</p>
    </div>

    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
    </script>

  </body>
</html>
```

Output

Open the file *testAngularJS.htm* in a web browser. Enter your name and see the result.

Sample Application

Enter your Name:

Hello !

How AngularJS Integrates with HTML

- The ng-app directive indicates the start of AngularJS application.
- The ng-model directive creates a model variable named name, which can be used with the HTML page and within the div having ng-app directive.
- The ng-bind then uses the name model to be displayed in the HTML tag whenever user enters input in the text box.
- Closing tag indicates the end of AngularJS application.

AngularJS Directives

AngularJS facilitates you to extend HTML with new attributes. These attributes are called directives. There is a set of built-in directive in AngularJS which offers functionality to your applications. You can also define your own directives. Directives are special attributes starting with ng- prefix. Following are the most common directives:

- ng-app: This directive starts an AngularJS Application.
- ng-init: This directive initializes application data.
- ng-model: This directive defines the model that is variable to be used in AngularJS.
- ng-repeat: This directive repeats html elements for each item in a collection.

ng-app directive

ng-app directive defines the root element. It starts an AngularJS Application and automatically initializes or bootstraps the application when web page containing AngularJS Application is loaded. It is also used to load various AngularJS modules in AngularJS Application

See this example:

In following example, we've defined a default AngularJS application using ng-app attribute of a div element.

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

ng-init directive

ng-init directive initializes an AngularJS Application data. It defines the initial values for an AngularJS application.In following example, we'll initialize an array of countries. We're using JSON syntax to define array of countries.

```
<div ng-app = "" ng-init = "countries = [{locale:'en-IND',name:'India'}, {locale:'en-PAK',name:'Pakistan'}, {locale:'en-AUS',name:'Australia'}]"> ... </div>
```

ng-model directive:

ng-model directive defines the model/variable to be used in AngularJS Application.
In following example, we've defined a model named "name".

```
<div ng-app = "">
  ...
  <p>Enter your Name: <input type = "text" ng-model = "name"></p>
</div>
```

ng-repeat directive

ng-repeat directive repeats html elements for each item in a collection. In following example, we've iterated over array of countries.

```
<div ng-app = "">
  ...
  <p>List of Countries with locale:</p>

  <ol>
    <li ng-repeat = "country in countries">
      {{ 'Country: ' + country.name + ', Locale: ' + country.locale }}
    </li>
  </ol>
```

AngularJS directives Example

Let's take an example to use all the above discussed directives:

```
<!DOCTYPE html>
<html>
<head>
   <title>AngularJS Directives</title>
</head>
<body>
   <h1>Sample Application</h1>

      <div ng-app = "" ng-init = "countries = [{locale:'en-IND',name:'India'}, {locale:'en-PAK',name:'Pakistan'}, {locale:'en-AUS',name:'Australia'}]">
      <p>Enter your Name: <input type = "text" ng-model = "name"></p>
      <p>Hello <span ng-bind = "name"></span>!</p>
      <p>List of Countries with locale:</p>

      <ol>
```

```
        <li ng-repeat = "country in countries">
          {{ 'Country: ' + country.name + ', Locale: ' + country.locale }}
        </li>
      </ol>
    </div>
<script src = "http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
</body>
</html>
```

AngularJS Directives List

AnglarJS directives are used to add functionality to your application. You can also add your own directives for your applications.Following is a list of AngularJS directives:

Directive	Description
ng-app	It defines the root element of an application.
ng-bind	It binds the content of an html element to application data.
ng-bind-html	It binds the inner HTML of an HTML element to application data, and also removes dangerous code from the html string.
ng-bind-template	It specifies that the text content should be replaced with a template.
ng-blur	It specifies a behavior on blur events.
ng-change	It specifies an expression to evaluate when content is being changed by the user.
ng-checked	It specifies if an element is checked or not.
ng-class	It specifies css classes on html elements.
ng-class-even	It is same as ng-class, but will only take effect on even rows.
ng-class-odd	It is same as ng-class, but will only take effect on odd rows.
ng-click	It specifies an expression to evaluate when an element is being clicked.
ng-cloak	It prevents flickering when your application is being loaded.
ng-controller	It defines the controller object for an application.
ng-copy	It specifies a behavior on copy events.
ng-csp	It changes the content security policy.
ng-cut	It specifies a behavior on cut events.
ng-dblclick	It specifies a behavior on double-click events.
ng-focus	It specifies a behavior on focus events.
ng-hide	It hides or shows html elements.
ng-href	It specifies a URL for the <a> element.
ng-if	It removes the html element if a condition is false.
ng-include	It includes html in an application.
ng-init	It defines initial values for an application.
ng-jq	It specifies that the application must use a library, like jQuery.
ng-keydown	It specifies a behavior on keydown events.
ng-keypress	It specifies a behavior on keypress events.
ng-keyup	It specifies a behavior on keyup events.
ng-list	It converts text into a list (array).
ng-open	It specifies the open attribute of an element.

ng-options	It specifies <options> in a <select> list.
ng-paste	It specifies a behavior on paste events.
ng-pluralize	It specifies a message to display according to en-us localization rules.
ng-readonly	It specifies the readonly attribute of an element.
ng-required	It specifies the required attribute of an element.
ng-selected	It specifies the selected attribute of an element.
ng-show	It shows or hides html elements.
ng-src	It specifies the src attribute for the <img> element.
ng-srcset	It specifies the srcset attribute for the <img> element.
ng-style	It specifies the style attribute for an element.
ng-submit	It specifies expressions to run on onsubmit events.
ng-switch	It specifies a condition that will be used to show/hide child elements.
ng-transclude	It specifies a point to insert transcluded elements.
ng-value	It specifies the value of an input element.
ng-disabled	It specifies if an element is disabled or not.
ng-form	It specifies an html form to inherit controls from.
ng-model	It binds the value of html controls to application data.
ng-model-options	It specifies how updates in the model are done.
ng-mousedown	It specifies a behavior on mousedown events.
ng-mouseenter	It specifies a behavior on mouseenter events.
ng-mouseleave	It specifies a behavior on mouseleave events.
ng-mousemove	It specifies a behavior on mousemove events.
ng-mouseover	It specifies a behavior on mouseover events.
ng-mouseup	It specifies a behavior on mouseup events.
ng-non-bindable	It specifies that no data binding can happen in this element, or it's children.
ng-repeat	It defines a template for each data in a collection.

AngularJS ng-app Directive

The AngularJS ng-app specifies that it is the root element of the AngularJS application. All AngularJS application must contain a root element. You can only have one ng-app directive in your HTML document. If you have more than one ng-app directive; the first appeared directive will be used.

Syntax:

```
<element ng-app="modulename">
...
  content inside the ng-app root element can contain AngularJS code
...
</element>
```

Parameter explanation:

modulename: It is an optional parameter. It specifies the name of a module that you want to add with the application.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
```

```
<div ng-app="">
<p>The calculated value is : {{ 5 + 5 * 8 / 2}}</p>
</div>
</body>
</html>
```

Note: ng-app is the simplest, easiest and most common way to bootstrap an application.

AngularJS ng-bind Directive

The AngularJS ng-bind directive replaces the content of an HTML element with the value of a given variable, or expression. If you change the value of the given variable or expression, AngularJS changes the content of the specified HTML element as well as.

It is an alternative to the interpolation directive.

Syntax:

As attribute:

1. `<element ng-bind="expression"></element>`

As CSS class:

1. `<element class="ng-bind: expression"></element>`

Parameter explanation:

expression: It specifies a variable, or an expression to evaluate.

Let's take an example to demonstrate ng-bind directive.

See this example:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>ng-bind example</title>
  <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
</head>
<body ng-app="bindExample">
  <script>
  angular.module('bindExample', [])
    .controller('ExampleController', ['$scope', function($scope) {
      $scope.name = 'JavaTpoint';
    }]);
</script>
<div ng-controller="ExampleController">
  <label>Enter name: <input type="text" ng-model="name"></label><br>
  Hello <span ng-bind="name"></span>!
</div>
</body>
</html>
```

AngularJS ng-bind-html Directive

The AngularJS ng-bind-html directive is used to bind content to an HTML element securely.
It evaluates the expressions and inserts the resulting HTML into the element in a secure way. By default, the resulting HTML content will be sanitized using the $sanitize service. You have to include ngSanitize in your module's dependencies to utilize this functionality. So "angular-sanitize.js" must be in your application.

Syntax:

```
<element ng-bind-html="expression"></element>
```

Parameter explanation:

expression: It specifies a variable or an expression to evaluate.

Let's take an example to demonstrate the ng-bind-html directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-sanitize.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
<p ng-bind-html="myText"></p>
</div>
<script>
var app = angular.module("myApp", ['ngSanitize']);
app.controller("myCtrl", function($scope) {
   $scope.myText = "Welcome to: <h1>JavaTpoint!</h1>";
});
</script>
<p><b>Note:</b> This example has "angular-sanitize.js",
which has functions for removing potentially dangerous tokens from the HTML.</p>
</body>
</html>
```

AngularJS ng-bind-template Directive

The AngularJS ng-bind-template directive specifies that the text content should be replaced with a template.
It replaces the content of an HTML element with the value of the given expressions.
Unlike ngBind, the ngBindTemplate can contain multiple {{ }} expressions. So, it is used when we want to bind more than one expression to your HTML element. It is required because some HTML elements like TITLE and OPTION cannot contain SPAN elements.

Syntax:

```
<element ng-bind-template="expression"></element>
```

Parameter explanation:

expression: It specifies one or more expressions to evaluate, each surrounded by {{ }}.

Let's take an example to demonstrate ng-bind-template directive.

See this example:

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

```
<body>
<div ng-app="myApp" ng-bind-template="{{firstName}} {{lastName}}" ng-controller="myCtrl">
</div>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
   $scope.firstName = "Sonoo";
   $scope.lastName = "Jaiswal";
});
</script>
</body>
</html>
```

AngularJS ng-change Directive

AngularJS ng-change directive specifies what to do when the user changes the value of an HTML element. The ng-change directive evaluates the expression immediately unlike the JavaScript onchange event which only triggers at the end of the change. It doesn't wait until all changes are made, or when the input field loses focus.

The ng-model directive must be presented to use ng-change directive.

Note: This ng-change directive is supported by the HTML tags like <input>, <select>, and <textarea>.

Syntax:

```
<element ng-change="expression"></element>
```

Parameter explanation:

expression: It specifies an expression that is executed when an element's value changes.

Let's take an example to demonstrate ng-change directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="myApp">
<div ng-controller="myCtrl">
 <p>Type something in the input field:</p>
 <input type="text" ng-change="myFunc()" ng-model="myValue" />
 <p>The input field has changed {{count}} times.</p>
</div>
<script>
 angular.module('myApp', [])
  .controller('myCtrl', ['$scope', function($scope) {
   $scope.count = 0;
   $scope.myFunc = function() {
    $scope.count++;
   };
  }]);
</script>
```

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

AngularJS ng-checked Directive

The AngularJS ng-checked directive is used to set a checked attribute on the element and add a checkbox or a radiobutton. You can set its value true or false. The checkbox, or radiobutton, will be checked if the expression inside the ng-checked attribute returns true.

Note: This ng-change directive is supported by the HTML tags like <input> elements of type checkbox or radio.

Syntax:

```
<input type="checkbox|radio" ng->
```

Parameter explanation:

expression: It specifies an expression that will set the element's checked attribute if it returns true.

Let's take an example to demonstrate ng-checked directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
<p>Courses offered:</p>
<input type="checkbox" ng-model="all"> Check all<br><br>
<input type="checkbox" ng-checked="all">Java<br>
<input type="checkbox" ng-checked="all">Oracle<br>
<input type="checkbox" ng-checked="all">C/C++<br>
<input type="checkbox" ng-checked="all">.NET<br>
<input type="checkbox" ng-checked="all">SQT<br>
<input type="checkbox" ng-checked="all">Hadoop<br>
<input type="checkbox" ng-checked="all">PHP
<p>Click "Check all" to select all courses.</p>
</body>
</html>
```

AngularJS ng-class Directive

The AngularJS ng-class directive facilitates you to dynamically set CSS classes on an HTML element by databinding an expression that represents all classes to be added. It may be a **String, an object or an array.**

In case of a string, it should contain one or more, space-separated class names. In case of an object, it should contain key-value pairs, where the key is the class name of the class you want to add, and the value is a Boolean value. In the case of an array, it can be a combination of both.

Note: The ng-class directive is supported by all HTML elements.

Syntax:

```
<element ng-class="expression"></element>
```

Parameter explanation:

expression: It specifies an expression that returns one or more class names.

Let's take an example to demonstrate ng-class directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<style>
```

```
.normal {
   color:white;
   background-color:grey;
   padding:20px;
   font-family:"Courier New";
}
.medium {
   color:white;
   background-color:brown;
   padding:30px;
   font-family:"Courier New";
}
.advance {
   background-color:red;
   padding:40px;
   font-family:Verdana;
}
</style>
<body ng-app="">
<p>Choose a class:</p>
<select ng-model="home">
<option value="normal">Normal</option>
<option value="medium">Medium</option>
<option value="advance">Advance</option>
</select>
<div ng-class="home">
 <h1>Welcome To JavaTpoint!</h1>
 <p>A solution of all technologies!</p>
</div>
</body>
</html>
```

AngularJS ng-click Directive

The AngularJS ng-click directive facilitates you to specify custom behavior when an element is clicked. So, it is responsible for the result what you get after clicking.

It is supported by all HTML elements.

Syntax:

```
<element ng-click="expression"></element>
```

Parameter explanation:

expression: It specifies an expression that is executed when an element is clicked.

Let's take an example to demonstrate the ng-click directive.

See this example:

Example1:

```
<!DOCTYPE html>
```

```
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
<p>Click the button:</p>
<button ng-click="count = count + 1" ng-init="count=0">OK</button>
<p>The button has been clicked <strong>{{count}} </strong>times.</p>
<p><strong>Note:</strong>This example counts a value every time you click on the button and increase the value of the variable.</p>
</body>
</html>
```

ng-click directive example using function

Example2:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="myApp">
<div ng-controller="myCtrl">
 <p>Click the button to execute a function:</p>
 <button ng-click="myFunc()">OK</button>
 <p>The button has been clicked <strong>{{count}}</strong> times.</p>
</div>
<script>
angular.module('myApp', [])
 .controller('myCtrl', ['$scope', function($scope) {
  $scope.count = 0;
  $scope.myFunc = function() {
   $scope.count++;
  };
 }]);
</script>
</body>
</html>
```

AngularJS ng-controller Directive

The AngularJS ng-controller directive adds a controller class to the view (your application). It is the key aspect which specifies the principles behind the Model-View-Controller design pattern.

It facilitates you to write code and make functions and variables, which will be parts of an object, available inside the current HTML element. This object is called scope.

This is supported by all HTML elements.

Syntax:

```
<element ng-controller="expression"></element>
```

Parameter explanation:

expression: It specifies the name of the controller.

Let's take an example to demonstrate ng-controller directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
Full Name: {{firstName + " " + lastName}}
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.firstName = "Aryan";
   $scope.lastName = "Jaiswal";
});
</script>
<p>In this example you can see how to define a controller, and how to use variables made for the scope.</p>
</body>
</html>
```

AngularJS ng-copy Directive

The AngularJS ng-copy directive specifies the custom behavior of AngularJS when an HTML element is being copied. It doesn't override the element's original on copy event; both the ng-copy expression and the original on copy event will be executed.

It is supported by all HTML elements.

Syntax:

```
<element ng-copy="expression"></element>
```

Parameter explanation:

expression: It specifies an expression that is executed when the text of an element is being copied.

Let's take an example to demonstrate the ng-copy directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
<input ng-copy="count = count + 1" ng-init="count=0" value="Copy this text" />
<p>The text has been copied <strong> {{count}} </strong>times.</p>
<p>This example shows that every time you copy the content in the input field, it will increase the value of the vari
able "count".</p>
</body>
</html>
```

Example2:

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>ng-copy example</title>
 <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
</head>
<body ng-app="">
 <input ng-copy="copied=true" ng-init="copied=false; value='copy me'" ng-model="value">
copied: {{copied}}
<p><strong>Note:</strong> By default value of this example is false. When you copy the text within the input field
, it will be true.</p>
</body>
</html>
```

AngularJS ng-cut Directive

The AnglarJS ng-cut directive specifies the custom behavior of AnglarJS when the text of the input field is being cut. It doesn?t override the element's original oncut event; both the ng-cut expression and the original oncut event will be executed.

It is supported by HTML tags like < a >, < input >, < select >, < textarea >, and the window object.

Syntax:

```
<element ng-cut="expression"></element>
```

Parameter explanation:

expression: It specifies an expression that is executed when the text of an element is being cut.

Let's take an example to demonstrate ng-ct directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
<input ng-cut="count = count + 1" ng-init="count=0" value="Cut this text" />
<p>The text has been cut<strong> {{count}} </strong>times.</p>
<p>This example shows that every time you cut the content in the input field, it will increase the value of the varia
ble "count".</p>
</body>
</html>
```

AngularJS ng-focus Directive

The AngularJS ng-focus directive specifies the custom behavior on focus event. It tells AngularJS what to do when an HTML element gets focus. It doesn't override the element's original onfocus event, both will be executed.

It is supported by HTML elements like < a >, < input >, < select >, < textarea >, and the window object.

Syntax:

```
<element ng-focus="expression"></element>
```

Parameter explanation:

expression: It specifies an expression that is executed when an element gets focus.

Let's take an example to demonstrate ng-focus directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
<input ng-focus="count = count + 1" ng-init="count=0" />
<h1>{{count}}</h1>
```

```
<p>In this example, everytime you make a focus on the specified box, it increases the value of the variable "count"
</p>
</body>
</html>
```

AngularJS ng-hide Directive

The AngularJS ng-hide directive is used to hide the HTML element if the expression is set to true.
The element is shown if you remove the ng-hide CSS class and hidden, if you add the ng-hide CSS class onto the element. The ng-hide CSS class is predefined in AngularJS and sets the element's display to none.

```
<element ng-hide="expression"></element>
```

As a CSS class:

```
<element class="ng-hide"></element>
```

Parameter explanation:

expression: It specifies an expression that will hide the element if the expression returns true.

Let's take an example to demonstrate ng-hide directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
Hide HTML: <input type="checkbox" ng-model="myVar">
<div ng-hide="myVar">
<h1>Welcome to JavaTpoint!</h1>
<p>A  solution of all technologoes.</p>
</div>
</body>
</html>
```

AngularJS ng-href Directive

The AngularJS ng-href directive provides a replacement for the original href attribute of an <a> element. You can use the ng-href directive instead of href if you have AngularJS code inside the href value.
The ng-href directive ensures that the link is not broken even if the user clicks the link beforethe code execution. It is supported by <a> element.

Syntax:

1. `<a ng-href="string"></a>`

Parameter explanation:

string: It specifies a string value, or an expression resulting in a string.

Let's take an example to demonstrate the ng-href directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
<div ng-init="myVar = 'http://www.javatpoint.com/java-tutorial'">
<h1>Java Tutorials</h1>
<p>Go to <a ng-href="{{myVar}}">{{myVar}}</a> to learn java.</p>
</div>
```

```
<p>You can use here original href attribute, but in AngularJS, the ng-href attribute is safer.</p>
</body>
</html>
```

AngularJS ng-if Directive

The AngularJS ng-if directive is used to remove the HTML elements if the expression is set to false. If the if element is set to true, a copy of the element is added in the DOM.
ngIf is different from ngShow and ngHide which show and hide the elements while ngIf completely removes and recreates the element in the DOM rather than changing its visibility.
It is supported by all HTML elements.

Syntax:

1. `<element ng-if="expression"></element>`

Parameter explanation:

expression: It specifies an expression that completely removes the element if it returns false. If it returns true, it inserts an element in the DOM instead.
Let's take an example to demonstrate ng-if directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
Keep HTML: <input type="checkbox" ng-model="myVar" ng-init="myVar = true">
<div ng-if="myVar">
<h1>Welcome to JavaTpoint!</h1>
<p>A solution of all technologies..</p>
<hr>
</div>
<p>If you uncheck the checkbox then DIV element will be removed.</p>
<p>If you check the checkbox then DIV element will return.</p>
</body>
</html>
```

AngularJS ng-include Directive

The AngularJS ng-include directive is used to fetch, compile and include an external HTML fragment. These are added as childnodes of the specified element. The ng-include attribute's value can also be an expression, returning a filename.
By default, the included file must be located on the same domain as the document.
It is supported by all HTML elements.

Syntax:

`<element ng-include="filename" onload="expression" autoscroll="expression" ></element>`

It can also be used as an element:

`<ng-include src="filename" onload="expression" autoscroll="expression" ></ng-include>`

Parameter explanation:

filename: It specifies a filename, written with apostrophes, or an expression which returns a filename.
onload: It is optional. It specifies an expression to evaluate when the included file is loaded.
autoscroll: It is also optional. It specifies whether or not the included section should be able to scroll into a specific view.
Let's take an example to demonstrate the ng-include directive.

See this example:

```
<!DOCTYPE html>
```

```
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
<div ng-include="'include.htm'"></div>
</body>
</html>
```

AngularJS ng-init Directive

The AngularJS ng-init directive facilitates you to evaluate the given expression in the current scope. Ng-controller is preferred over ng-init because the ng-init directive can add some unnecessary logic into the scope.

It is supported by all HTML elements.

Syntax:

1. `<element ng-init="expression" ></element>`

Parameter explanation:

expression: It specifies an expression to evaluate.

Let's take an example to demonstrate ng-init directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="" ng-init="myText='Hello JavaTpoint!'">
<h1>{{myText}}</h1>
<p>The ng-init directive has created an AngularJS variable, which can be used in the application.</p>
</div>
</body>
</html>
```

AngularJS ng-list Directive

The ng-list directive is used to convert a string into an array of string, using a comma as the default separator. You can also convert an array of string and display the input field as a string by putting the ng-list directive on the input field.

The value of the ng-list attribute defines the separator.

Syntax:

1. `<element ng-list="separator"></element>`

Parameter explanation:

separator: It is an optional directive. It defines the separator. Its default value is a comma ","

Let's take an example to demonstrate the usage of ng-list directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="">
<p>Write some names in the input field, use a comma to separate them:</p>
<input ng-model="customers" ng-list/>
<p>This example will convert your input into an array, one item for each name:</p>
<pre>{{customers}}</pre>
```

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

AngularJS ng-switch Directive

The AngularJS ng-switch directive facilitates you to hide/show HTML elements according to an expression. Child elements with the ng-switch-when directive will be displayed if it gets a match, otherwise the element, and its children will be removed.

If you want to define a default section you can use ng-switch-default directive.

Syntax:

```
<element ng-switch="expression">
  <element ng-switch-when="value"></element>
  <element ng-switch-when="value"></element>
  <element ng-switch-when="value"></element>
  <element ng-switch-default></element>
</element>
```

Parameter explanation:

expression: It specifies an expression that will remove elements with no match, and display elements with a match.

Let's take an example to demonstrate the usage of ng-swith directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
Choose your favorite topic:
<select ng-model="myVar">
  <option value="animals">Zoology
  <option value="tuts">Tutorials
  <option value="cars">Cars
  <option value="bikes">Bikes
</select>
<hr>
<div ng-switch="myVar">
  <div ng-switch-when="animals">
    <h1>Zoology</h1>
    <p>Welcome to a world of zoology.</p>
  </div>
  <div ng-switch-when="tuts">
    <h1>Tutorials</h1>
    <p>Learn from examples.</p>
  </div>
  <div ng-switch-when="cars">
    <h1>Cars</h1>
    <p>Read about cars.</p>
  </div>
  <div ng-switch-when="bikes">
```

```
    <h1>Cars</h1>
    <p>Read about bikes.</p>
  </div>
  <div ng-switch-default>
    <h1>Switch</h1>
    <p>Select topic from the dropdown, to switch the content of this DIV.</p>
  </div>
</div>
<hr>
<p>The ng-switch directive hides and shows HTML sections depending on a certain value.</p>
</body>
</html>
```

AngularJS ng-value Directive

The AngularJS ng-value directive is used to set the value attribute of an input element, or a select element. It is mainly used on <radio> and <option> elements to set the bound values when these elements are selected. It is supported by <input> and <select> elements.

Syntax:

1. `<input ng-value="expression"></input>`

Parameter explanation:

expression: It specifies an expression that will set the element's value attribute.

Let's take an example to demonstrate ng-value directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
<input ng-value="myVar">
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.myVar = "Welcome to JavaTpoint.";
});
</script>
<p>JavaTpoint is the best tutorial website and a solution of all technology. </p>
</body>
</html>
```

AngularJS - Expressions

Expressions are used to bind application data to HTML. Expressions are written inside double curly braces such as in {{ expression}}. Expressions behave similar to ngbind directives. AngularJS expressions are pure JavaScript expressions and output the data where they are used.

Using numbers

```
<p>Expense on Books : {{cost * quantity}} Rs</p>
```

Using Strings
Hello {{student.firstname + " " + student.lastname}}!
Using Object
Roll No: {{student.rollno}}
Using Array
Marks(Math): {{marks[3]}}
Example
The following example shows the use of all the above-mentioned expressions −
testAngularJS.htm

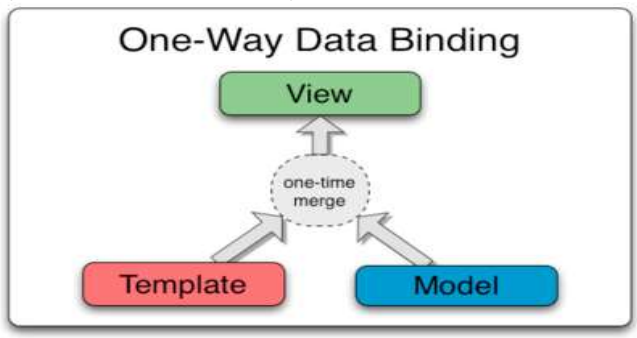
```
<html>
  <head>
    <title>AngularJS Expressions</title>
  </head>

  <body>
    <h1>Sample Application</h1>

    <div ng-app = "" ng-init = "quantity = 1;cost = 30;
      student = {firstname:'Mahesh',lastname:'Parashar',rollno:101};
      marks = [80,90,75,73,60]">
      <p>Hello {{student.firstname + " " + student.lastname}}!</p>
      <p>Expense on Books : {{cost * quantity}} Rs</p>
      <p>Roll No: {{student.rollno}}</p>
      <p>Marks(Math): {{marks[3]}}</p>
    </div>

    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
    </script>

  </body>
</html>
```

AngularJS Data Binding

Data binding is a very useful and powerful feature used in software development technologies. It acts as a bridge between the view and business logic of the application.
AngularJS follows Two-Way data binding model.

One-Way Data Binding

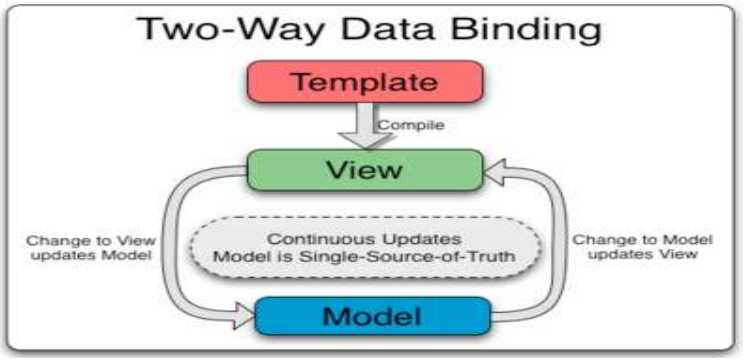
The one-way data binding is an approach where a value is taken from the data model and inserted into an HTML element. There is no way to update model from view. It is used in classical template systems. These systems bind data in only one direction.



Two-Way Data Binding

Data-binding in Angular apps is the automatic synchronization of data between the model and view components.

Data binding lets you treat the model as the single-source-of-truth in your application. The view is a projection of the model at all times. If the model is changed, the view reflects the change and vice versa.




```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="" ng-init="firstName='Ajeet'">
<p>Input something in the input box:</p>
<p>Name: <input type="text" ng-model="firstName"></p>
<p>You wrote: {{ firstName }}</p>
</div>
</body>
</html>
```

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".

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

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div data-ng-app="" data-ng-init="quantity=1;price=20">
<h2>Cost Calculator</h2>
Quantity: <input type="number" ng-model="quantity">
Price: <input type="number" ng-model="price">
<p><b>Total in rupees:</b> {{quantity * price}}</p>
</div>
</body>
</html>
```

AngularJS Controllers

AngularJS controllers are used to control the flow of data of AngularJS application. A controller is defined using ng-controller directive. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application/module that controller is to control. AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is defined using *ng-controller* directive. A controller is a JavaScript object that contains attributes/properties, and functions. Each controller accepts $scope as a parameter, which refers to the application/module that the controller needs to handle.

AngularJS Controller Example

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

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

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

</div>

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

Note:

- Here, the AngularJS application runs inside the is defined by ng-app="myApp".
- The AngularJS directive is ng-controller="myCtrl" attribute.
- The myCtrl function is a JavaScript function.
- AngularJS will invoke the controller with a $scope object.
- In AngularJS, $scope is the application object (the owner of application variables and functions).
- The controller creates two properties (variables) in the scope (firstName and lastName).
- The ng-model directives bind the input fields to the controller properties (firstName and lastName).

AngularJS controller example with methods (variables as functions)

```
<!DOCTYPE html>
```

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

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

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

</div>
<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
    $scope.firstName = "Aryan";
    $scope.lastName = "Khanna";
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    };
});
</script>
</body>
</html>
```

AngularJS Controller in external files

In larger applications, generally the controllers are stored in external files.
Create an external file named "personController.js" to store controller.
Here, "personController.js" is:

```
angular.module('myApp', []).controller('personCtrl', function($scope) {
    $scope.firstName = "Aryan",
    $scope.lastName = "Khanna",
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    }
});
```

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="personCtrl">
First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}
</div>
```

```
<script src="personController.js"></script>
</body>
</html>
```

AngularJS Scope

The scope is **the binding part between the HTML (view) and the JavaScript (controller)**. The scope is an object with the available properties and methods. The scope is available for both the view and the controller.

Adding Behavior to a Scope Object

In order to react to events or execute computation in the view we must provide behavior to the scope. We add behavior to the scope by attaching methods to the `$scope` object. These methods are then available to be called from the template/view.

The following example uses a Controller to add a method, which doubles a number, to the scope:

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


myApp.controller('DoubleController', ['$scope', function($scope) {
  $scope.double = function(value) { return value * 2; };
}]);
```

Once the Controller has been attached to the DOM, the `double` method can be invoked in an AngularJS expression in the template:

```
<div ng-controller="DoubleController">
  Two times <input ng-model="num"> equals {{ double(num) }}
</div>
```

Scope Inheritance Example(Nested)

It is common to attach Controllers at different levels of the DOM hierarchy. Since the ng-controller directive creates a new child scope, we get a hierarchy of scopes that inherit from each other. The `$scope` that each Controller receives will have access to properties and methods defined by Controllers higher up the hierarchy. See Understanding Scopes for more information about scope inheritance.

index.htmlapp.cssapp.js

```
<div class="spicy">
  <div ng-controller="MainController">
    <p>Good {{timeOfDay}}, {{name}}!</p>

    <div ng-controller="ChildController">
      <p>Good {{timeOfDay}}, {{name}}!</p>

      <div ng-controller="GrandChildController">
        <p>Good {{timeOfDay}}, {{name}}!</p>
      </div>
    </div>
  </div>
</div>
```

Notice how we nested three ng-controller directives in our template. This will result in four scopes being created for our view:

- The root scope
- The MainController scope, which contains timeOfDay and name properties
- The ChildController scope, which inherits the timeOfDay property but overrides (shadows) the name property from the previous scope
- The GrandChildController scope, which overrides (shadows) both the timeOfDay property defined in MainController and the name property defined in ChildController

Inheritance works with methods in the same way as it does with properties. So in our previous examples, all of the properties could be replaced with methods that return string values.

Angularjs multiple scope:

In case of multiple controllers AngularJS framework creates and pass a different $scope object to each controller so that data and methods of one controller not be accessed in another controller.

Multiple Controller in AngularJS

In angularjs we have learnt how to create controller now we are going to add multiple controllers in single module.

We can add multiple controllers to the single module. So I am going to create new controller with the name of trainer in which I want to show the trainer details.

We will use following code to create multiple controller

```
angmodule.controller("trainer", function($scope){
    var trainerdetail=[
                    { name:'isha', age:'27' },
                    { name:'avinash', age:'32' },
                    { name:'priya', age:'34' }
                    ];
    $scope.trainerdata=trainerdetail;
});
```

Now call this controller in your html in the following manner

```
<div ng-controller="trainer">
    <table border="1">
        <tr>
           <td>Trainer Name</td>
           <td>Trainer Age</td>
        </tr>
        <tr ng-repeat="x in trainerdata">
            <td>{{ x.name }}</td>
            <td>{{ x.age }}</td>
        </tr>
    </table>
</div>
```

Unit 4: Filters and Modules

AngularJS Filters

In AngularJS, filters are used to format data. Following is a list of filters used for transforming data.

Filter	Description
Currency	It formats a number to a currency format.
Date	It formats a date to a specified format.
Filter	It select a subset of items from an array.
Json	It formats an object to a Json string.
Limit	It is used to limit an array/string, into a specified number of elements/characters.
Lowercase	It formats a string to lower case.
Number	It formats a number to a string.

Orderby	It orders an array by an expression.
Uppercase	It formats a string to upper case.

How to add filters to expressions

You can add filters to expressions by using the pipe character |, followed by a filter.
In this example, the uppercase filter format strings to upper case:
See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="personCtrl">
<p>The name is {{ firstName | uppercase }}</p>
</div>
<script>
angular.module('myApp', []).controller('personCtrl', function($scope) {
    $scope.firstName = "Sonoo",
    $scope.lastName = "Jaiswal"
});
</script>
</body>
</html>
```

Let's apply the lowercase filter into the same example:
See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="personCtrl">
<p>The name is {{ firstName | lowercase }}</p>
</div>
<script>
angular.module('myApp', []).controller('personCtrl', function($scope) {
    $scope.firstName = "Sonoo",
    $scope.lastName = "Jaiswal"
});
</script>
</body>
</html>
```

How to add filters to directives

Filters can be added to directives, like ng-repeat, by using the pipe character |, followed by a filter.
Let's take an example:
In this example, orderBy filter is used to sort an array.
See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="namesCtrl">
<p>Looping with objects:</p>
<ul>
 <li ng-repeat="x in names | orderBy:'country'">
   {{ x.name + ', ' + x.country }}
 </li>
```

```
</ul>
</div>
<script>
angular.module('myApp', []).controller('namesCtrl', function($scope) {
  $scope.names = [
    {name:'Ramesh',country:'India'},
    {name:'Alex',country:'USA'},
    {name:'Pooja',country:'India'},
    {name:'Mahesh',country:'India'},
    {name:'Iqbal',country:'Pakistan'},
    {name:'Ramanujam',country:'India'},
    {name:'Osama',country:'Iraq'},
    {name:'Johnson',country:'UK'},
    {name:'Karl',country:'Russia'}
    ];
});
</script>
</body>
</html>
```

The filter Filter

The filter Filter can only be used on arrays because it selects a subset of an array. It returns an array containing only the matching items.

Let's take an example:

This example will return the names that contain the letter "o".

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="namesCtrl">
<ul>
 <li ng-repeat="x in names | filter : 'o'">
  {{ x }}
 </li>
</ul>
</div>
<script>
angular.module('myApp', []).controller('namesCtrl', function($scope) {
  $scope.names = [
'Ramesh',
'Pooja',
'Mahesh',
'Ramanujam',
'Osama',
'Iqbal',
'Karl',
'Johnson',
'Alex'
  ];
});
</script>
<p>This example displays only the names containing the letter "o".</p>
</body>
</html>
```

Filter an array based on user input

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

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="namesCtrl">
<p>Type a letter in the input field:</p>
<p><input type="text" ng-model="test"></p>
<ul>
  <li ng-repeat="x in names | filter:test">
    {{ x }}
  </li>
</ul>
</div>
<script>
angular.module('myApp', []).controller('namesCtrl', function($scope) {
    $scope.names = [
        'Ramesh',
'Pooja',
'Mahesh',
'Ramanujam',
'Osama',
'Iqbal',
'Karl',
'Johnson',
'Alex'
    ];
});
</script>
<p>The list will only contain the names matching the filter.</p>
</body>
</html>
```

Sort an array based on user input

You can sort an array according to the table columns.

See this example:

```
<!DOCTYPE html>

<html>

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

<body>

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

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

<table border="1" width="100%">

<tr>
```

```
<th ng-click="orderByMe('name')">Name</th>
<th ng-click="orderByMe('country')">Country</th>
</tr>
<tr ng-repeat="x in names | orderBy:myOrderBy">
<td>{{x.name}}</td>
<td>{{x.country}}</td>
</tr>
</table>
</div>
<script>
angular.module('myApp', []).controller('namesCtrl', function($scope) {
    $scope.names = [
     {name:'Ramesh',country:'India'},
     {name:'Alex',country:'USA'},
     {name:'Pooja',country:'India'},
     {name:'Mahesh',country:'India'},
     {name:'Iqbal',country:'Pakistan'},
     {name:'Ramanujam',country:'India'},
     {name:'Osama',country:'Iraq'},
     {name:'Johnson',country:'UK'},
     {name:'Karl',country:'Russia'}
     ];
  $scope.orderByMe = function(x) {
     $scope.myOrderBy = x;
  }
});
</script>
</body>
</html>
```

AngularJS Custom Filters

You can create your own filters by register a new filter factory function with your module.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<p>Click the table headers to change the sorting order:</p>
<div ng-app="myApp" ng-controller="namesCtrl">
<table border="1" width="100%">
<tr>
<th ng-click="orderByMe('name')">Name</th>
<th ng-click="orderByMe('country')">Country</th>
```

```
</tr>
<tr ng-repeat="x in names | orderBy:myOrderBy">
<td>{{x.name}}</td>
<td>{{x.country}}</td>
</tr>
</table>
</div>
<script>
angular.module('myApp', []).controller('namesCtrl', function($scope) {
    $scope.names = [
     {name:'Ramesh',country:'India'},
     {name:'Alex',country:'USA'},
     {name:'Pooja',country:'India'},
     {name:'Mahesh',country:'India'},
     {name:'Iqbal',country:'Pakistan'},
     {name:'Ramanujam',country:'India'},
     {name:'Osama',country:'Iraq'},
     {name:'Johnson',country:'UK'},
     {name:'Karl',country:'Russia'}
     ];

  $scope.orderByMe = function(x) {
    $scope.myOrderBy = x;
  }
});
</script>
</body>
</html>
```

OrderBy Filter

To order subjects by marks, we use orderBy marks.

```
Subject:
<ul>
   <li ng-repeat = "subject in student.subjects | orderBy:'marks'">
      {{ subject.name + ', marks:' + subject.marks }}
   </li>
</ul>
```

AngularJS - Modules

AngularJS supports modular approach. Modules are used to separate logic such as services, controllers, application etc. from the code and maintain the code clean. We define modules in separate js files and name them as per the module.js file. In the following example, we are going to create two modules –

- **Application Module** – used to initialize an application with controller(s).
- **Controller Module** – used to define the controller.

Application Module

Here is a file named *mainApp.js* that contains the following code –

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

Here, we declare an application mainApp module using angular.module function and pass an empty array to it. This array generally contains dependent modules.

Controller Module

studentController.js

```
mainApp.controller("studentController", function($scope) {
  $scope.student = {
    firstName: "Mahesh",
    lastName: "Parashar",
    fees:500,

    subjects:[
      {name:'Physics',marks:70},
      {name:'Chemistry',marks:80},
      {name:'Math',marks:65},
      {name:'English',marks:75},
      {name:'Hindi',marks:67}
    ],
    fullName: function() {
      var studentObject;
      studentObject = $scope.student;
      return studentObject.firstName + " " + studentObject.lastName;
    }
  };
});
```

Here, we declare a controller studentController module using mainApp.controller function.

Use Modules

```
<div ng-app = "mainApp" ng-controller = "studentController">
  ...
  <script src = "mainApp.js"></script>
  <script src = "studentController.js"></script>

</div>
```

Here, we use application module using ng-app directive, and controller using ng-controller directive. We import the mainApp.js and studentController.js in the main HTML page.

Example

The following example shows use of all the above mentioned modules.

testAngularJS.htm

```
<html>
  <head>
    <title>Angular JS Modules</title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    <script src = "/angularjs/src/module/mainApp.js"></script>
    <script src = "/angularjs/src/module/studentController.js"></script>

    <style>
      table, th , td {
        border: 1px solid grey;
        border-collapse: collapse;
        padding: 5px;
      }
      table tr:nth-child(odd) {
        background-color: #f2f2f2;
      }
      table tr:nth-child(even) {
```

```
        background-color: #ffffff;
      }
    </style>
  </head>

  <body>
    <h2>AngularJS Sample Application</h2>
    <div ng-app = "mainApp" ng-controller = "studentController">

      <table border = "0">
        <tr>
          <td>Enter first name:</td>
          <td><input type = "text" ng-model = "student.firstName"></td>
        </tr>
        <tr>
          <td>Enter last name: </td>
          <td><input type = "text" ng-model = "student.lastName"></td>
        </tr>
        <tr>
          <td>Name: </td>
          <td>{{student.fullName()}}</td>
        </tr>
        <tr>
          <td>Subject:</td>

          <td>
            <table>
              <tr>
                <th>Name</th>
                <th>Marks</th>
              </tr>
              <tr ng-repeat = "subject in student.subjects">
                <td>{{ subject.name }}</td>
                <td>{{ subject.marks }}</td>
              </tr>
            </table>
          </td>
        </tr>
      </table>
    </div>

  </body>
</html>
```

mainApp.js

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

studentController.js

```
mainApp.controller("studentController", function($scope) {
  $scope.student = {
    firstName: "Mahesh",
    lastName: "Parashar",
    fees:500,

    subjects:[
      {name:'Physics',marks:70},
      {name:'Chemistry',marks:80},
```

```
      {name:'Math',marks:65},
      {name:'English',marks:75},
      {name:'Hindi',marks:67}
    ],
    fullName: function() {
      var studentObject;
      studentObject = $scope.student;
      return studentObject.firstName + " " + studentObject.lastName;
    }
  };
});
```

How to add controller to a module

If you want to add a controller to your application refer to the controller with the ng-controller directive.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
   $scope.firstName = "Ajeet";
   $scope.lastName = "Maurya";
});
</script>
</body>
</html>
```

How to add directive to a module

AnglarJS directives are used to add functionality to your application. You can also add your own directives for your applications.

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

<div ng-app="myApp" w3-test-directive></div>
<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
   return {
      template : "This is a directive constructor. "
   };
});
</script>
</body>
</html>
```

Modules and controllers in file

In AngularJS applications, you can put the module and the controllers in JavaScript files.

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

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<script src="myApp.js"></script>
<script src="myCtrl.js"></script>
</body>
</html>
```

Here "myApp.js" contains:

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

Here "myCtrl.js" contains:

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

This example can also be written as:

```
<!DOCTYPE html>
<html>
<body>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
    $scope.firstName = "Ajeet";
    $scope.lastName = "Maurya";
});
</script>
</body>
</html>
```

AngularJS Scopes

The Scope is an object that is specified as a binding part between the HTML (view) and the JavaScript (controller). It plays a role of joining controller with the views. It is available for both the view and the controller.

How to use Scope

To make a controller in AngularJS, you have to pass the $scope object as an argument.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
```

```
<div ng-app="myApp" ng-controller="myCtrl">
<h1>{{carname}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.carname = "Volvo";
});
</script>
<p>The property "carname" was made in the controller, and can be referred to in the view by using the {{ }} brack
ets.</p>
</body>
</html>
```

AngularJS Dependency Injection

AngularJS comes with a built-in dependency injection mechanism. It facilitates you to divide your application into multiple different types of components which can be injected into each other as dependencies.

Dependency Injection is a software design pattern that specifies how components get holds of their dependencies. In this pattern, components are given their dependencies instead of coding them within the component.

Modularizing your application makes it easier to reuse, configure and test the components in your application. Following are the core types of objects and components:

- value
- factory
- service
- provider
- constant

These objects and components can be injected into each other using AngularJS Dependency Injection.

Value

In AngularJS, value is a simple object. It can be a number, string or JavaScript object. It is used to pass values in factories, services or controllers during run and config phase.

```
//define a module
var myModule = angular.module("myModule", []);

//create a value object and pass it a data.
myModule.value("numberValue", 100);
myModule.value("stringValue", "abc");
myModule.value("objectValue", { val1 : 123, val2 : "abc"} );
```

Here, values are defined using the value() function on the module. The first parameter specifies the name of the value, and the second parameter is the value itself. Factories, services and controllers can now reference these values by their name.

Injecting a value

To inject a value into AngularJS controller function, add a parameter with the same when the value is defined.

```
var myModule = angular.module("myModule", []);
myModule.value("numberValue", 100);

myModule.controller("MyController", function($scope, numberValue) {
 console.log(numberValue);
});
```

Factory

Factory is a function that is used to return value. When a service or controller needs a value injected from the factory, it creates the value on demand. It normally uses a factory function to calculate and return the value. Let's take an example that defines a factory on a module, and a controller which gets the factory created value injected:

```
var myModule = angular.module("myModule", []);
myModule.factory("myFactory", function() {
   return "a value";
});
myModule.controller("MyController", function($scope, myFactory) {
console.log(myFactory);
});
```

Injecting values into factory

To inject a value into AngularJS controller function, add a parameter with the same when the value is defined.

```
var myModule = angular.module("myModule", []);
myModule.value("numberValue", 100);
myModule.controller("MyController", function($scope, numberValue) {
 console.log(numberValue);
});
```

Note: It is not the factory function that is injected, but the value produced by the factory function.

Service

In AngularJS, service is a JavaScript object which contains a set of functions to perform certain tasks. Services are created by using service() function on a module and then injected into controllers.

```
//define a module
var mainApp = angular.module("mainApp", []);
...
//create a service which defines a method square to return square of a number.
mainApp.service('CalcService', function(MathService){
  this.square = function(a) {
    return MathService.multiply(a,a);
  }
});
//inject the service "CalcService" into the controller
mainApp.controller('CalcController', function($scope, CalcService, defaultInput) {
  $scope.number = defaultInput;
  $scope.result = CalcService.square($scope.number);
   $scope.square = function() {
    $scope.result = CalcService.square($scope.number);
  }
});
```

Provider

In AngularJS, provider is used internally to create services, factory etc. during config phase (phase during which AngularJS bootstraps itself). It is the most flexible form of factory you can create. Provider is a special factory method with a get() function which is used to return the value/service/factory.

```
//define a module
var mainApp = angular.module("mainApp", []);
...
//create a service using provider which defines a method square to return square of a number.
mainApp.config(function($provide) {
  $provide.provider('MathService', function() {
    this.$get = function() {
      var factory = {};
      factory.multiply = function(a, b) {
        return a * b;
      }
      return factory;
    };
  });
});
```

Constants

You cannot inject values into the module.config() function. Instead constants are used to pass values at config phase.

```
mainApp.constant("configParam", "constant value");
```

Let's take an example to deploy all above mentioned directives.

```
<!DOCTYPE html>
<html>
  <head>
    <title>AngularJS Dependency Injection</title>
  </head>
  <body>
    <h2>AngularJS Sample Application</h2>

    <div ng-app = "mainApp" ng-controller = "CalcController">
      <p>Enter a number: <input type = "number" ng-model = "number" /></p>
      <button ng-click = "square()">X<sup>2</sup></button>
      <p>Result: {{result}}</p>
    </div>

    <script src = "http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

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

      mainApp.config(function($provide) {
        $provide.provider('MathService', function() {
          this.$get = function() {
            var factory = {};

            factory.multiply = function(a, b) {
              return a * b;
            }
            return factory;
          };
```

```
    });
  });
  mainApp.value("defaultInput", 10);
  mainApp.factory('MathService', function()
  {
    var factory = {};
    factory.multiply = function(a, b) {
      return a * b;
    }
    return factory;
  });
  mainApp.service('CalcService', function(MathService){
    this.square = function(a) {
      return MathService.multiply(a,a);
    }
  });
  mainApp.controller('CalcController', function($scope, CalcService, defaultInput) {
    $scope.number = defaultInput;
    $scope.result = CalcService.square($scope.number);
    $scope.square = function() {
      $scope.result = CalcService.square($scope.number);
    }
  });
  </script>
 </body>
</html>
```

AngularJS – Services

AngularJS supports the concept of Separation of Concerns using services architecture. Services are JavaScript functions, which are responsible to perform only specific tasks. This makes them individual entities which are maintainable and testable. The controllers and filters can call them on requirement basis. Services are normally injected using the dependency injection mechanism of AngularJS.

AngularJS provides many inbuilt services. For example, $http, $route, $window, $location, etc. Each service is responsible for a specific task such as the $http is used to make ajax call to get the server data, the $route is used to define the routing information, and so on. The inbuilt services are always prefixed with $ symbol. There are two ways to create a service −

- Factory
- Service

Using Factory Method

In this method, we first define a factory and then assign method to it.

```
var mainApp = angular.module("mainApp", []);
mainApp.factory('MathService', function() {
  var factory = {};

  factory.multiply = function(a, b) {
    return a * b
  }

  return factory;
});
```

Using Service Method

In this method, we define a service and then assign method to it. We also inject an already available service to it.

```
mainApp.service('CalcService', function(MathService) {
  this.square = function(a) {
    return MathService.multiply(a,a);
  }
});
```

Example

The following example shows use of all the above mentioned directives −

testAngularJS.htm

```
<html>
  <head>
    <title>Angular JS Services</title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
    </script>
  </head>
  <body>
    <h2>AngularJS Sample Application</h2>

    <div ng-app = "mainApp" ng-controller = "CalcController">
      <p>Enter a number: <input type = "number" ng-model = "number" /></p>
      <button ng-click = "square()">X<sup>2</sup></button>
      <p>Result: {{result}}</p>
    </div>
    <script>
      var mainApp = angular.module("mainApp", []);
      mainApp.factory('MathService', function() {
        var factory = {};
        factory.multiply = function(a, b) {
          return a * b
        }
        return factory;
      });
      mainApp.service('CalcService', function(MathService) {
        this.square = function(a) {
          return MathService.multiply(a,a);
        }
      });
      mainApp.controller('CalcController', function($scope, CalcService) {
        $scope.square = function() {
          $scope.result = CalcService.square($scope.number);
        }
      });
    </script>

  </body>
</html>
```

AngularJS Forms

AngularJS facilitates you to create a form enriches with data binding and validation of input controls. Input controls are ways for a user to enter data. A form is a collection of controls for the purpose of grouping related controls together.

Following are the input controls used in AngularJS forms:

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

AngularJS provides multiple events that can be associated with the HTML controls. These events are associated with the different HTML input elements.

Following is a list of events supported in AngularJS:

- ng-click
- ng-dbl-click
- ng-mousedown
- ng-mouseup
- ng-mouseenter
- ng-mouseleave
- ng-mousemove
- ng-mouseover
- ng-keydown
- ng-keyup
- ng-keypress
- ng-change

Data Binding

ng-model directive is used to provide data binding.

Let's take an example where ng-model directive binds the input controller to the rest of your application

See this example:

```
<!DOCTYPE html>
<html lang="en">
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="formCtrl">
 <form>
  First Name: <input type="text" ng-model="firstname">
 </form>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
  $scope.firstname = "Ajeet";
});
</script>
</body>
```

```
</html>
```

You can also change the example in the following way:

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="">
 <form>
  First Name: <input type="text" ng-model="firstname">
 </form>
 <h2>You entered: {{firstname}}</h2>
</div>
<p>Change the name inside the input field, and you will see the name in the header changes according
ly.</p>
</body>
</html>
```

AngularJS Checkbox

A checkbox has a value true or false. The ng-model directive is used for a checkbox.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="">
 <form>
  Check to show this:
  <input type="checkbox" ng-model="myVar">
 </form>
 <h1 ng-show="myVar">Checked</h1>
</div>
<p>The ng-show attribute is set to true when the checkbox is checked.</p>
</body>
</html>
```

AngularJS Radio Buttons

ng-model directive is used to bind radio buttons in your applications.

Let's take an example to display some text, based on the value of the selected radio buttons. In this example, we are also using ng-switch directive to hide and show HTML sections depending on the value of the radio buttons.

See this example:

```
<!DOCTYPE html>
<html>
```

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
<form>
 Pick a topic:
 <input type="radio" ng-model="myVar" value="tuts">Tutorials
 <input type="radio" ng-model="myVar" value="fest">Festivals
 <input type="radio" ng-model="myVar" value="news">News
</form>
<div ng-switch="myVar">
 <div ng-switch-when="tuts">
   <h1>Tutorials</h1>
   <p>Welcome to the best tutorials over the net</p>
 </div>
 <div ng-switch-when="fest">
   <h1>Festivals</h1>
   <p>Most famous festivals</p>
 </div>
 <div ng-switch-when="news">
   <h1>News</h1>
   <p>Welcome to the news portal.</p>
 </div>
</div>
<p>The ng-switch directive hides and shows HTML sections depending on the value of the radio buttons.</p>
</body>
</html>
```

AngularJS Selectbox

ng-model directive is used to bind select boxes to your application.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body ng-app="">
<form>
 Select a topic:
 <select ng-model="myVar">
   <option value="">
   <option value="tuts">Tutorials
   <option value="fest">Festivals
   <option value="news">News
 </select>
</form>
<div ng-switch="myVar">
```

```
  <div ng-switch-when="tuts">
    <h1>Tutorials</h1>
    <p>Welcome to the best tutorials over the net.</p>
  </div>
  <div ng-switch-when="fest">
    <h1>Festivals</h1>
    <p>Most famous festivals.</p>
  </div>
  <div ng-switch-when="news">
    <h1>News</h1>
    <p>Welcome to the news portal.</p>
  </div>
</div>
<p>The ng-switch directive hides and shows HTML sections depending on the value of the radio buttons.</p>
</body>
</html>
```

AngularJS form example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular JS Forms</title>
    <script src = "http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

    <style>
      table, th , td {
        border: 1px solid grey;
        border-collapse: collapse;
        padding: 5px;
      }

      table tr:nth-child(odd) {
        background-color: lightpink;
      }

      table tr:nth-child(even) {
        background-color: lightyellow;
      }
    </style>

  </head>
  <body>

    <h2>AngularJS Sample Application</h2>
    <div ng-app = "mainApp" ng-controller = "studentController">

      <form name = "studentForm" novalidate>
        <table border = "0">
          <tr>
```

```
            <td>Enter first name:</td>
            <td><input name = "firstname" type = "text" ng-model = "firstName" required>
            <span style = "color:red" ng-show = "studentForm.firstname.$dirty && studentForm.firstname.$invalid">
                <span ng-show = "studentForm.firstname.$error.required">First Name is required.</span>
              </span>
            </td>
          </tr>
          <tr>
            <td>Enter last name: </td>
            <td><input name = "lastname"  type = "text" ng-model = "lastName" required>
            <span style = "color:red" ng-show = "studentForm.lastname.$dirty && studentForm.lastname.$invalid">
                <span ng-show = "studentForm.lastname.$error.required">Last Name is required.</span>
              </span>
            </td>
          </tr>
          <tr>
            <td>Email: </td><td><input name = "email" type = "email" ng-model = "email" length = "100" required>
              <span style = "color:red" ng-show = "studentForm.email.$dirty && studentForm.email.$invalid">
                <span ng-show = "studentForm.email.$error.required">Email is required.</span>
                <span ng-show = "studentForm.email.$error.email">Invalid email address.</span>
              </span>
            </td>
          </tr>
          <tr>
            <td>
              <button ng-click = "reset()">Reset</button>
            </td>
            <td>
              <button ng-disabled = "studentForm.firstname.$dirty &&
                studentForm.firstname.$invalid || studentForm.lastname.$dirty &&
                studentForm.lastname.$invalid || studentForm.email.$dirty &&
                studentForm.email.$invalid" ng-click="submit()">Submit</button>
            </td>
          </tr>

        </table>
      </form>
    </div>
    <script>
      var mainApp = angular.module("mainApp", []);

      mainApp.controller('studentController', function($scope) {
        $scope.reset = function(){
          $scope.firstName = "Sonoo";
          $scope.lastName = "Jaiswal";
          $scope.email = "sonoojaiswal@javatpoint.com";
        }
        $scope.reset();
      });
    </script>
  </body>
</html>
```

AngularJS AJAX

AngularJS provides a $http service for reading data and remote servers. It is used to retrieve the desired records from a server.

AngularJS requires data in JSON format. Once the data is ready, $http gets the data form server in the following manner:

```
function employeeController($scope,$http) {
r url = "data.txt";

 $http.get(url).success( function(response) {
   $scope.employees = response;
 });
```

Here the file "data.txt" is employee's records. $http service makes an AJAX call and sets response to its property employees. This model is used to draw tables in HTML.

AngularJS AJAX Example

testAngularJS.htm

```
<!DOCTYPE html>
<html>
<head>
 <title>Angular JS Includes</title>
   <style>
      table, th , td {
        border: 1px solid grey;
        border-collapse: collapse;
        padding: 5px;
      }

      table tr:nth-child(odd) {
        background-color: #f2f2f2;
      }

      table tr:nth-child(even) {
        background-color: #ffffff;
      }
   </style>
 </head>
 <body>
   <h2>AngularJS Sample Application</h2>
   <div ng-app = "" ng-controller = "employeeController">

     <table>
       <tr>
         <th>Name</th>
         <th>Age</th>
         <th>Salary</th>
       </tr>

       <tr ng-repeat = "employee in employees">
         <td>{{ employee.Name }}</td>
         <td>{{ employee.Age }}</td>
         <td>{{ employee.Salary }}</td>
       </tr>
     </table>
   </div>
```

```
        <script>
          function employeeController($scope,$http) {
            var url = "data.txt";

            $http.get(url).success( function(response) {
              $scope.employees = response;
            });
          }
        </script>
        <script src = "http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js"></script>
      </body>
    </html>
```

Here the file data.txt contains the employee's record.

"data.txt" (employee's data in JSON format)

```
 [
{
   "Name" : "Mahesh Sharma",
   "Age" : 25,
   "Salary" : "20000"
},
{
   "Name" : "Rohan Malik",
   "Age" : 20,
   "Salary" : "22000"
},
{
   "Name" : "Robert Petro",
   "Age" : 45,
   "Salary" : "67000"
},
{
   "Name" : "Jullia Roberts",
   "Age" : 21,
   "Salary" : "55000"
}
```

To execute the above example, you have to deploy testAngularJS.htm and data.txt file to a web server. Open the file testAngularJS.htm using the URL of your server in a web browser and see the result.

Output:

The result would look like this:

Name	Age	Salary
Mahesh Sharma	25	20000
Rohan Malik	20	22000
Robert Petro	45	67000
Jullia Roberts	21	55000

Table:

Name	Age	Salary
Mahesh Sharma	25	20000
Rohan Malik	20	22000
Robert Petro	45	67000
Jullia Roberts	21	55000

HTTP Service Methods

There are several shortcut methods of calling the $http service. In the above example, .get method of the $http service is used. Following are several other shortcut methods:

- .delete()
- .get()
- .head()
- .jsonp()
- .patch()
- .post()
- .put()

Properties

The response from the server is an object with these properties:

- .config the object used to generate the request.
- .data a string, or an object, carrying the response from the server.
- .headers a function to use to get header information.
- .status a number defining the HTTP status.
- .statusText a string defining the HTTP status.