Java Script

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.

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.

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 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.

   Example

   ```
   <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>
```

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 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.

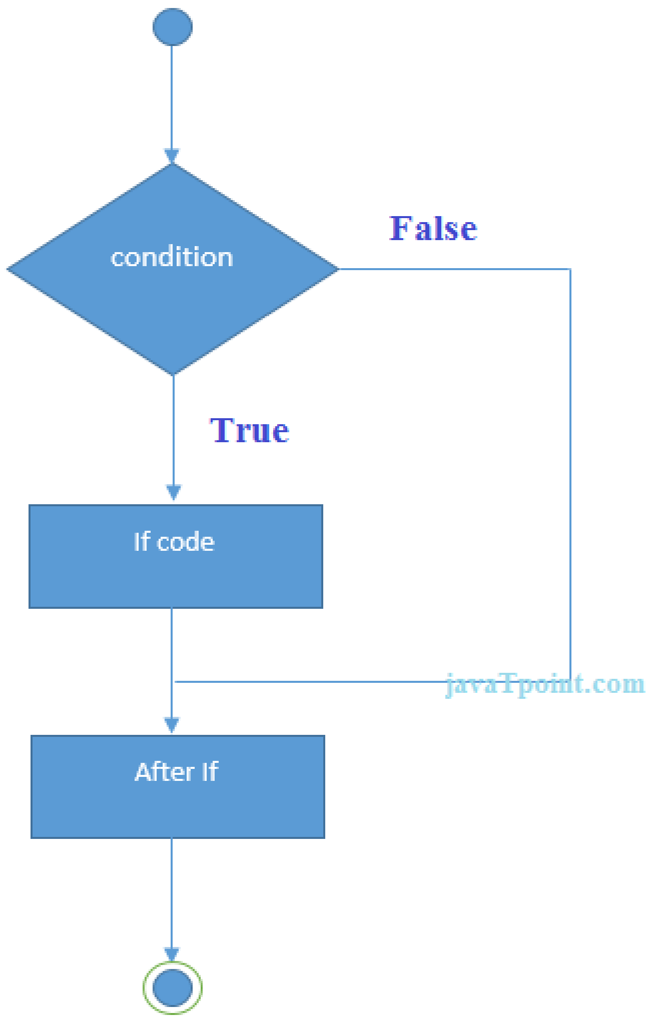
1. If Statement
2. If else statement
3. if else if statement

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

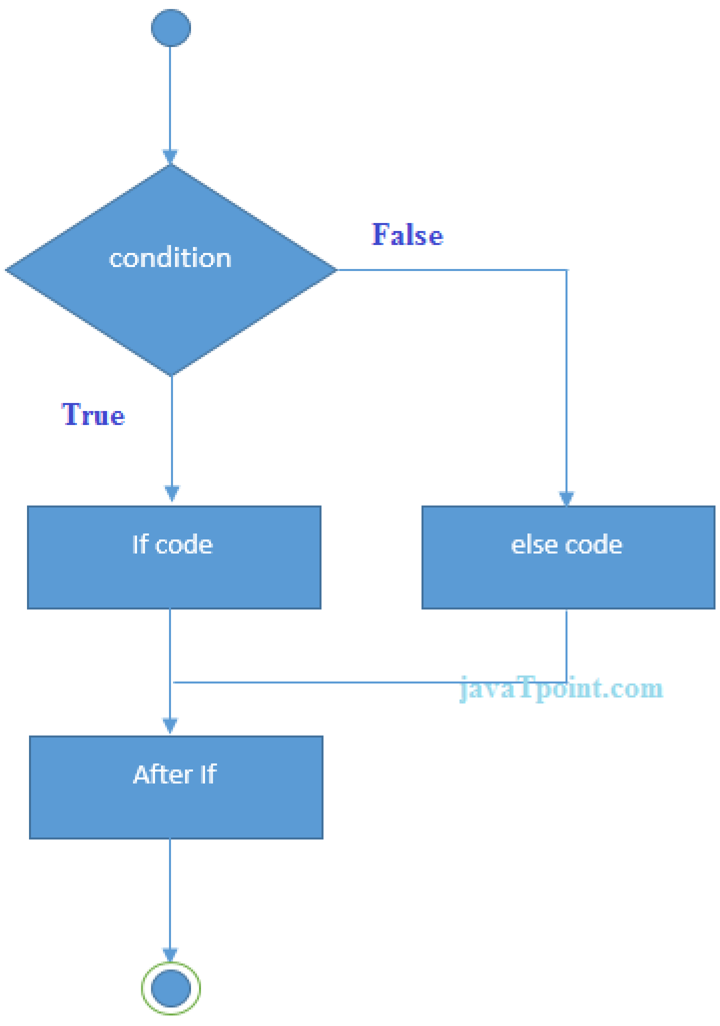
It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

```
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 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.

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

JavaScript Functions can have 0 or more arguments.

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

```
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>
```

Test it Now

Output:

JavaScript Objects

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

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

1) JavaScript Object by object literal

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

```
object={property1:value1,property2:value2.....propertyN:valueN}
```

Let’s see the simple example of creating object in JavaScript.

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

2) By creating instance of Object

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Here, **new keyword** is used to create object.

Let’s see the example of creating object directly.

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

Test it Now

Output of the above example

```
103 Vimal Jaiswal 30000
```

Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;

this.changeSalary=changeSalary;
function changeSalary(otherSalary){
this.salary=otherSalary;
}
}
e=new emp(103,"Sonoo Jaiswal",30000);
document.write(e.id+" "+e.name+" "+e.salary);
e.changeSalary(45000);
document.write("<br>"+e.id+" "+e.name+" "+e.salary);
</script>
```

Test it Now

Output of the above example

```
103                    Sonoo                    Jaiswal                    30000
103 Sonoo Jaiswal 45000
```

JavaScript Object Methods

S.No	Methods	Description
1	Object.assign()	This method is used to copy enumerable and o properties from a source object to a target object
2	Object.create()	This method is used to create a new object with specified prototype object and properties.

3	Object.defineProperty()	This method is used to describe some behavic attributes of the property.
4	Object.defineProperties()	This method is used to create or configure multi object properties.
5	Object.entries()	This method returns an array with arrays of the k value pairs.
6	Object.freeze()	This method prevents existing properties from be removed.
7	Object.getOwnPropertyDescriptor()	This method returns a property descriptor for specified property of the specified object.
8	Object.getOwnPropertyDescriptors()	This method returns all own property descriptors c given object.
9	Object.getOwnPropertyNames()	This method returns an array of all propert (enumerable or not) found.
10	Object.getOwnPropertySymbols()	This method returns an array of all own symbol properties.
11	Object.getPrototypeOf()	This method returns the prototype of the specif object.
12	Object.is()	This method determines whether two values are same value.
13	Object.isExtensible()	This method determines if an object is extensible
14	Object.isFrozen()	This method determines if an object was frozen.
15	Object.isSealed()	This method determines if an object is sealed.
16	Object.keys()	This method returns an array of a given object's o property names.
17	Object.preventExtensions()	This method is used to prevent any extensions of object.
18	Object.seal()	This method prevents new properties from be added and marks all existing properties as no configurable.

19	Object.setPrototypeOf()	This method sets the prototype of a specified object another object.
20	Object.values()	This method returns an array of values.

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:

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

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/>");
}
</script>
```

The .length property returns the length of an array.

2) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

```
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 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 provic

	function conditions.
flat()	It creates a new array carrying sub-array elements concatenated recursively the specified depth.
flatMap()	It maps all array elements via mapping function, then flattens the result int 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 provic function conditions.
find()	It returns the value of the first element in the given array that satisfies specified condition.
findIndex()	It returns the index value of the first element in the given array that satis 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 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 loo through these keys.
lastIndexOf()	It searches the specified element in the given array and returns the index of last match.
map()	It calls the specified function for every array element and returns the new arr
of()	It creates a new array from a variable number of arguments, holding any ty 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 redu the array to a single value.
reduceRight()	It executes a provided function for each value from right to left and redu the array to a single value.
some()	It determines if any element of the array passes the test of the implemen 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 affect 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.

JavaScript Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser.
When javascript

code is included in HTML

, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads
resize	onresize	When the visitor resizes the window of the browser

Let's discuss some examples over events and their handlers.

Click Event

```
<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
   <!--
   function clickevent()
   {
       document.write("This is JavaTpoint");
   }
   //-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>
```

MouseOver Event

```html
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
    <!--
    function mouseoverevent()
    {
        alert("This is JavaTpoint");
    }
    //-->
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>
```

Focus Event

```html
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>
<!--
    function focusevent()
    {
        document.getElementById("input1").style.background=" aqua";
    }
//-->
</script>
</body>
</html>
```

Keydown Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()"/>
<script>
<!--
    function keydownevent()
    {
        document.getElementById("input1");
        alert("Pressed a key");
    }
//-->
</script>
</body>
</html>
```

Load event

```
<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page successfully loaded');">
<script>
<!--
document.write("The page is loaded successfully");
//-->
</script>
</body>
</html>
```

JavaScript Form Validation

1. JavaScript form validation
2. Example of JavaScript validation
3. JavaScript email validation

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

JavaScript Form Validation Example

In this example, we are going to validate the name and password. The name can't be empty and password can't be less than 6 characters long.

Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

```
<script>
function validateform(){
var name=document.myform.name.value;
var password=document.myform.password.value;

if (name==null || name==""){
  alert("Name can't be blank");
  return false;
}else if(password.length<6){
  alert("Password must be at least 6 characters long.");
  return false;
  }
}
</script>
<body>
<form name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/>
<input type="submit" value="register">
</form>
```

JavaScript Retype Password Validation

```
<script type="text/javascript">
function matchpass(){
var firstpassword=document.f1.password.value;
var secondpassword=document.f1.password2.value;

if(firstpassword==secondpassword){
return true;
}
else{
alert("password must be same!");
return false;
}
}
</script>

<form name="f1" action="register.jsp" onsubmit="return matchpass()">
```

```
Password:<input type="password" name="password" /><br/>
Re-enter Password:<input type="password" name="password2"/><br/>
<input type="submit">
</form>
```

JavaScript Number Validation

Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

```
<script>
function validate(){
var num=document.myform.num.value;
if (isNaN(num)){
  document.getElementById("numloc").innerHTML="Enter Numeric value only";
  return false;
}else{
  return true;
  }
}
</script>
<form name="myform" onsubmit="return validate()" >
Number: <input type="text" name="num"><span id="numloc"></span><br/>
<input type="submit" value="submit">
</form>
```

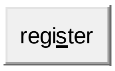
JavaScript validation with image

Let’s see an interactive JavaScript form validation example that displays correct and incorrect image if input is correct or incorrect.

```
<script>
function validate(){
var name=document.f1.name.value;
var password=document.f1.password.value;
var status=false;

if(name.length<1){
document.getElementById("nameloc").innerHTML=
" <img src='unchecked.gif'/> Please enter your name";
status=false;
}else{
document.getElementById("nameloc").innerHTML=" <img src='checked.gif'/>";
status=true;
}
if(password.length<6){
document.getElementById("passwordloc").innerHTML=
" <img src='unchecked.gif'/> Password must be at least 6 char long";
status=false;
}else{
document.getElementById("passwordloc").innerHTML=" <img src='checked.gif'/>";
}
return status;
}
</script>

<form name="f1" action="#" onsubmit="return validate()">
<table>
<tr><td>Enter Name:</td><td><input type="text" name="name"/>
<span id="nameloc"></span></td></tr>
<tr><td>Enter Password:</td><td><input type="password" name="password"/>

<span id="passwordloc"></span></td></tr>
<tr><td colspan="2"><input type="submit" value="register"/></td></tr>
</table>
</form>
```

Output:

```
Enter Name:         [          ]
Enter Password:     [          ]
```



JavaScript email validation

We can validate the email by the help of JavaScript.

There are many criteria that need to be follow to validate the email id such as:

- email id must contain the @ and . character
- There must be at least one character before and after the @.
- There must be at least two characters after . (dot).

Let's see the simple example to validate the email field.

```
<script>
function validateemail()
{
var x=document.myform.email.value;
var atposition=x.indexOf("@");
var dotposition=x.lastIndexOf(".");
if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
  alert("Please enter a valid e-mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);
  return false;
  }
}
</script>
<body>
<form name="myform"  method="post" action="#" onsubmit="return validateemail();">
Email: <input type="text" name="email"><br/>

<input type="submit" value="register">
</form>
```

Unit 4

Angular JS is an open source JavaScript framework by Google to build web applications. It can be freely used, changed and shared by anyone.

Following are the advantages of AngularJS over other JavaScript frameworks:

- **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.

AngularJS First Example

```
<!DOCTYPE html>
<html>
<head>
.
.
</head>
<body>
.
.
</body>
</html>
```

"Print Hello Word in AJS"

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.5/angular.min.js"></script>
</head>
<body ng-app="myapp">
<div ng-controller="HelloController" >
<h2>Hello {{helloTo.title}} !</h2>
</div>

<script>
angular.module("myapp", [])
    .controller("HelloController", function($scope) {
        $scope.helloTo = {};
        $scope.helloTo.title = "World, AngularJS";
    } );
</script>
</body>
</html>
```

View Part

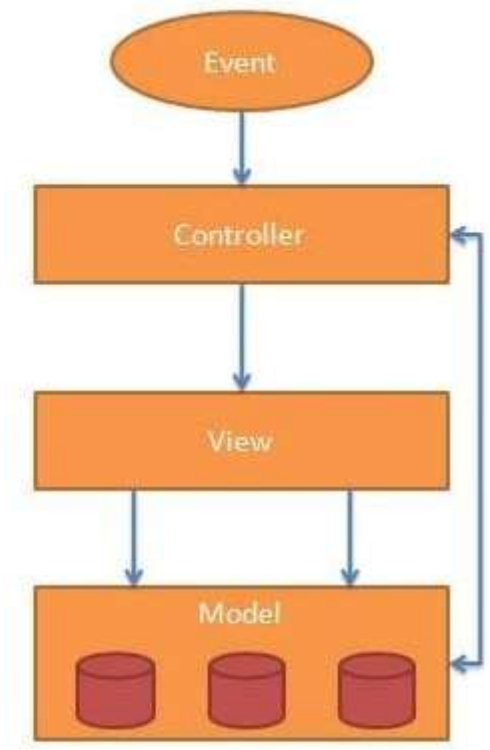
```
<div ng-controller="HelloController" >
<h2>Hello {{helloTo.title}} !</h2>
</div>
```

Controller Part

```
<script>
angular.module("myapp", [])
    .controller("HelloController", function($scope) {
        $scope.helloTo = {};
        $scope.helloTo.title = "World, AngularJS";
    });
</script>
```

AngularJS MVC Architecture

MVC stands for Model View Controller. It is a software design pattern for developing web applicati
is very popular because it isolates the application logic from the user interface layer and su
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
   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 th
   in a particular format triggered by the controller's decision to present the data. They are script
   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 use
   and performs interactions on the data model objects. The controller receives input, validates it, an
   performs business operations that modify the state of the data model.

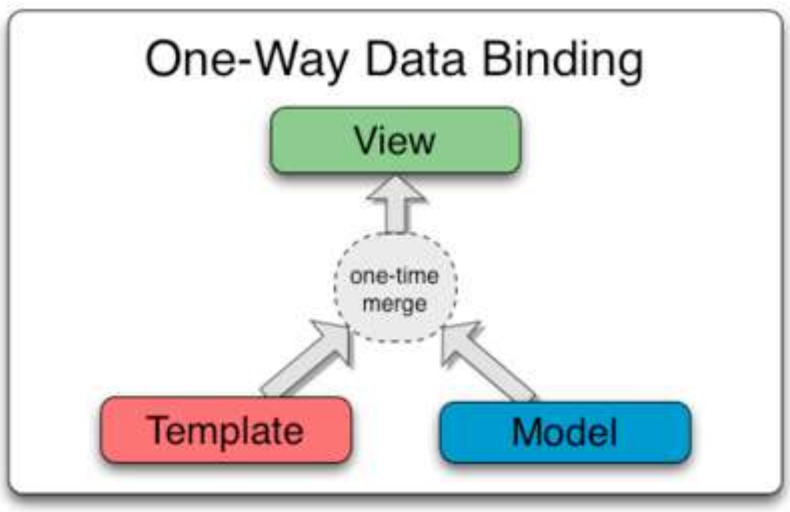
AngularJS Data Binding

Data binding is a very useful and powerful feature used in software development technologies. It a a bridge between the view and business logic of the application.

AngularJS follows Two-Way data binding model.

One-Way Data Binding

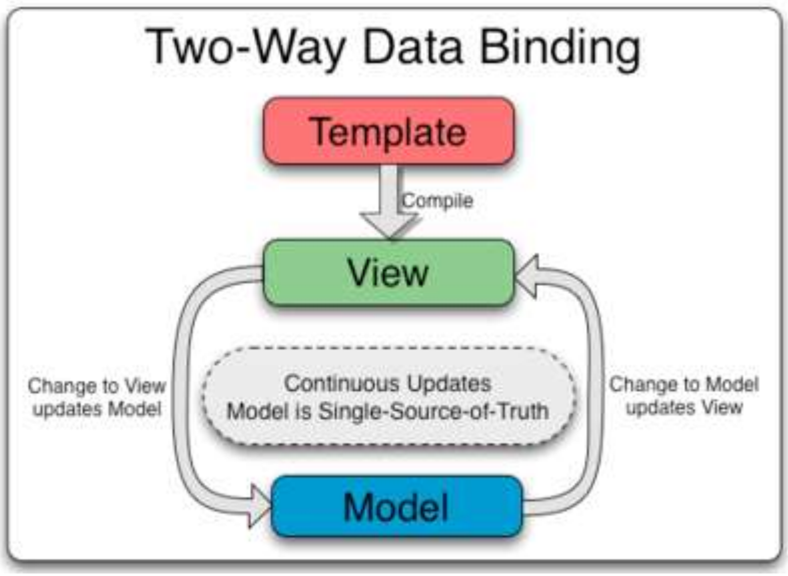
The one-way data binding is an approach where a value is taken from the data model and inserte
an HTML element. There is no way to update model from view. It is used in classical template sy
These systems bind data in only one direction.



Two-Way Data Binding

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

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


Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model updates View
Model

```
<!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>
```

Test it Now

In the above example, the {{ firstName }} expression is an AngularJS data binding expression 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>
```

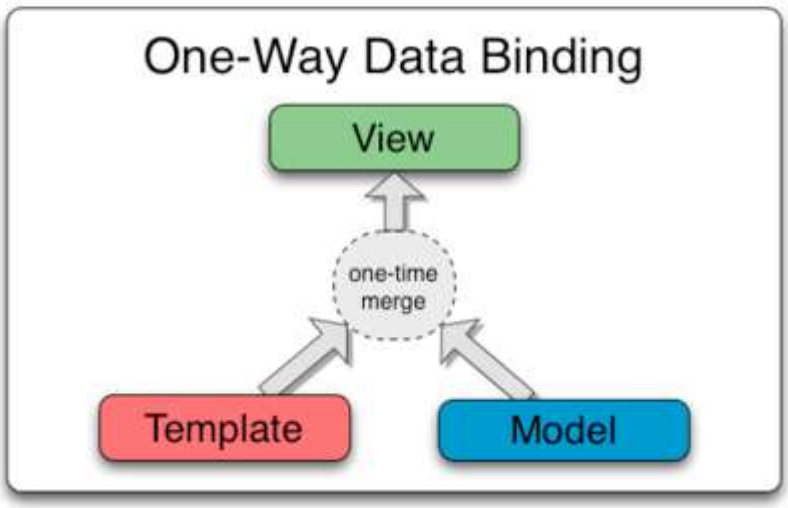
AngularJS Data Binding

Data binding is a very useful and powerful feature used in software development technologies. It a
a bridge between the view and business logic of the application.

AngularJS follows Two-Way data binding model.

One-Way Data Binding

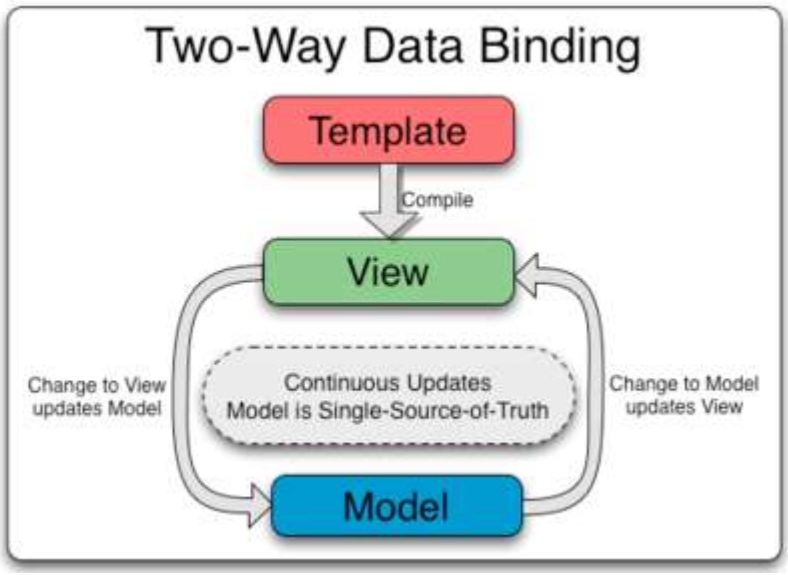
The one-way data binding is an approach where a value is taken from the data model and inserte
an HTML element. There is no way to update model from view. It is used in classical template sy
These systems bind data in only one direction.



Two-Way Data Binding

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

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


Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model updates View
Model

```
<!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>
```

Test it Now

In the above example, the {{ firstName }} expression is an AngularJS data binding expression
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>
```

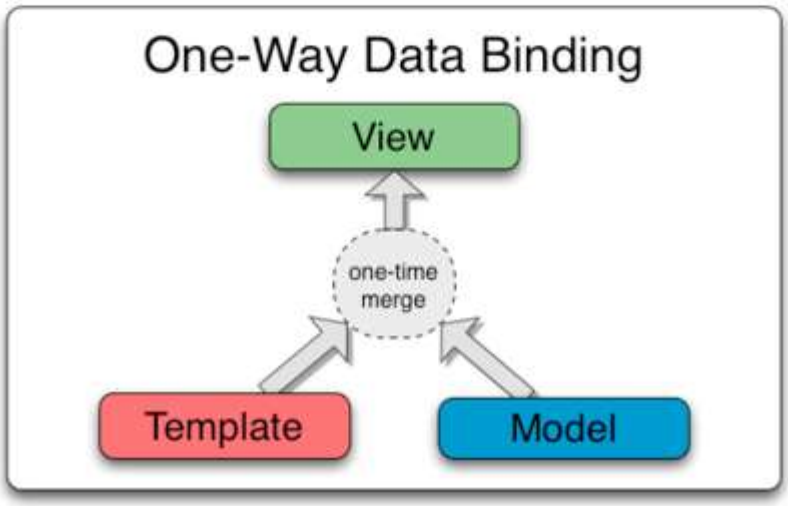
AngularJS Data Binding

Data binding is a very useful and powerful feature used in software development technologies. It a
a bridge between the view and business logic of the application.

AngularJS follows Two-Way data binding model.

One-Way Data Binding

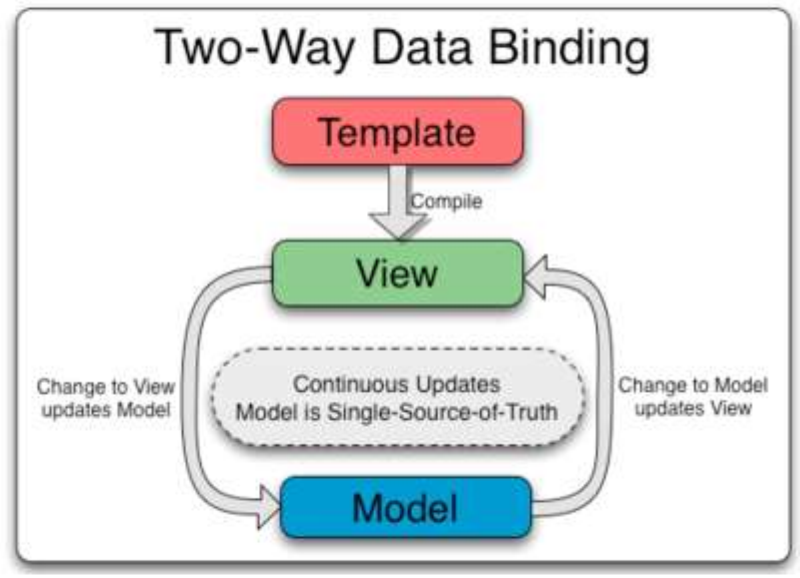
The one-way data binding is an approach where a value is taken from the data model and inserte
an HTML element. There is no way to update model from view. It is used in classical template sys
These systems bind data in only one direction.



Two-Way Data Binding

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

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


Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model updates View
Model

```
<!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>
```

Test it Now

In the above example, the {{ firstName }} expression is an AngularJS data binding expression
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 Expressions

In AngularJS, expressions are used to bind application data to HTML. AngularJS resolves the expre
and return the result exactly where the expression is written.

Expressions are written inside double braces {{expression}}.They can also be written inside a directiv

ng-bind="expression".

AnularJS expressions are very similar to JavaScript expressions. They can contain literals, operator
variables. For example:

```
{{ 5 + 5 }} or {{ firstName + " " + lastName }}
```

AngularJS Expressions 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>A simple expression example: {{ 5 + 5 }}</p>
</div>
</body>
</html>
```

Test it Now

Note: If you remove the directive "ng-app", HTML will display the expression without solving it.

See this example:

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

```
<p>If you remove the directive "ng-app", HTML will display the expression without solving it.</p>
<div>
<p>A simple expression example:  {{ 5 + 5 }}</p>
</div>
</body>
</html>
```

Test it Now

You can also write expressions wherever you want, AngularJS will resolve the expression and retu
result.

Let's take an example to change the color of input box by changing its value.

See this example:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<p>Change the value of the input field:</p>
<div ng-app="" ng-init="myCol='pink'">
<input style="background-color:{{myCol}}" ng-model="myCol" value="{{myCol}}">
</div>
<p>AngularJS resolves the expression and returns the result.</p>
<p>The background color of the input box will be whatever you write in the input field.</p>
</body>
</html>
```

Test it Now

AngularJS Numbers

AngularJS numbers are similar to JavaScript numbers.

```
<!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="quantity=5;cost=5">
<p>Total in dollar: {{ quantity * cost }}</p>
</div>
</body>
</html>
```

Test it Now

We can use the same example by using ng-bind:

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="quantity=5;cost=5">
<p>Total in dollar: <span ng-bind="quantity * cost"></span></p>
</div>
```

```
</body>
</html>
```
Test it Now

AngularJS Strings

```
<!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='Sonoo';lastName='Jaiswal'">
<p>My full name is: {{ firstName + " " + lastName }}</p>
</div>
</body>
</html>
```
Test it Now

Same example with ng-bind:

```
<!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='Sonoo';lastName='Jaiswal'">
<p>My full name is: <span ng-bind="firstName + ' ' + lastName"></span></p>
</div>
</body>
</html>
```
Test it Now

AngularJS Objects

```
<!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="person={firstName:'Sonoo',lastName:'Jaiswal'}">
<p>My name is {{ person.firstName }}</p>
</div>
</body>
</html>
```
Test it Now

Same example with ng-bind:

```
<!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="person={firstName:'Sonoo',lastName:'Jaiswal'}">
<p>The name is <span ng-bind="person.firstName"></span></p>
</div>
</body>
</html>
```
Test it Now

AngularJS Arrays

```
<!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="points=[1,15,19,2,40]">
<p>The first result is {{ points[0] }}</p>
</div>
</body>
</html>
```
Test it Now

Same example with ng-bind:

```
<!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="points=[1,15,19,2,40]">
<p>The first result is <span ng-bind="points[0]"></span></p>
</div>
</body>
</html>
```
Test it Now

Difference between AngularJS Expressions and JavaSc expressions:

- AngularJS expressions can be written inside HTML, while JavaScript expressions cannot.
- AngularJS expressions support filters, while JavaScript expressions do not.
- AngularJS expressions do not support conditionals, loops, and exceptions, while JavaScript expressio

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. Yo 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 init
or bootstraps the application when web page containing AngularJS Application is loaded. It is also
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
element.

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

ng-init directive

ng-init directive initializes an AngularJS Application data. It defines the initial values for an Ang
application.

In following example, we'll initialize an array of countries. We're using JSON syntax to define ar
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, 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>
```
Test it Now

AngularJS Directives List

AnglarJS directives are used to add functionality to your application. You can also add you
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 rem 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.

How to add directives

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

AngularJS Controllers

AngularJS controllers are used to control the flow of data of AngularJS application. A controller is d
using ng-controller directive. A controller is a JavaScript object containing attributes/propertie
functions. Each controller accepts $scope as a parameter which refers to the application/modul
controller is to control.

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>
```
Test it Now

Note:

- Here, the AngularJS application runs inside the <div> 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 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>
```
Test it Now

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>
```