Expressions and Data Biding

- Number and String Expressions
- Object Binding and Expressions
- Working with Arrays
- Forgiving Behavior
- Understanding Data binding

Expressions are used to bind application data to HTML Template. They are pure JavaScript expressions.

Expressions are written inside double curly braces such as in **{{ expr }}.**

AngularJS Numbers:

```
<div ng-app ng-init="qty=1;cost=2">
   <b>Invoice:</b>
   <div>
      Quantity: <input type="number" min="0" ng-model="qty">
   </div>
   <div>
      Costs: <input type="number" min="0" ng-model="cost">
   </div>
   <div>
      <b>Total:</b> {{qty * cost }}
   </div>
</div>
```

AngularJS Strings:

These are like JavaScript strings:

```
<div ng-app="" ng-init="firstName='John';lastName='Doe'">
  <p>The name is {{ firstName + " " + lastName }}</p>
  <p>The name is <span ng-bind="firstName + ' ' + lastName"></span></p>
</div>
```

AngularJS Objects

```
<div ng-app="" ng-init="person={FirstName:'Sandeep',LastName:'Soni'}">
   <p>Hello {{ person.FirstName + " " + person.LastName}}</p>
   <p>Hello <span ng-bind="person.FirstName"></span> <span ng-bind="person.LastName"></span></p>
</div>
```

AngularJS Arrays

```
<div ng-app="" ng-init="numbers=[11,12,13,41,51]">
   <p>Hello {{ numbers[0] + numbers[1]}} </p>
</div>
```

```
<div ng-app="" ng-
init="names=[{FirstName:'Sandeep',LastName:'Soni'},{FirstName:'John',LastName:'Doe'},{FirstName:'Mohammed',
LastName:'Akbar'}]">
   <p>Hello {{ names[0].FirstName + " " + names[0].LastName}} </p>
   <p>Hello <span ng-bind="names[1].FirstName"></span> <span ng-bind="names[1].LastName"></span></p>
</div>
```

Forgiving

Expression evaluation is forgiving to undefined and null. In JavaScript, evaluating a.b throws an exception if a is not an object.

In AngularJS, undefined variables in expression would not cause an exception and it shows nothing. This behavior is because the variable may not be defined now but might be getting initialized based on some ajax response from server.

- In Angular, the **$interpolate** service is responsible for working with binding expression.
- The service returns a function you can invoke against a scope object to produce an **interpolated string**.
- The function is known as an **interpolation function**.Since this service is **forgiving** you do not see any errors,However, we can decorate the $interpolate service and wrap the interpolation functions it produces.
- The **wrapping function** will log every execution that produces a string and warn about every interpolation that produces a false looking empty string.

Wrapped up interpolation function

```
var app = angular.module('myApp',[]);
   app.config(function ($provide) {
      $provide.decorator("$interpolate", function ($delegate) {
         var interpolateWrap = function () {
            var interpolationFn = $delegate.apply(this, arguments);
            if (interpolationFn) {
               return interpolationFnWrap(interpolationFn, arguments);
            }
```

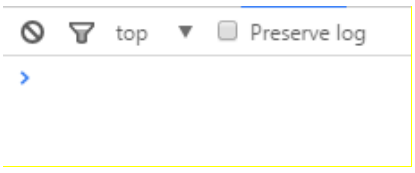
```
        };
        var interpolationFnWrap = function (interpolationFn, interpolationArgs) {
          return function () {
            var result = interpolationFn.apply(this, arguments);
            var log = result ? console.log : console.warn;
            log.call(console, "interpolation of  " + interpolationArgs[0].trim(),
                        ":", result.trim());
            return result;
          };
        };
        angular.extend(interpolateWrap, $delegate);
        return interpolateWrap;
      });
  });
```

Ex: **Without using interpolate service**

```
<body ng-app="myApp" ng-controller="myCtrl">
  <p>{{message}}</p>
  <p>{{name}}</p>
</body>

var app = angular.module('myApp', []);
  app.controller('myCtrl', function ($scope, $interpolate) {
 $scope.message = "hello";
  });
```

o/p: The following code does not throw an exception or give any indication of an error even though name is not define in scope in the binding expression.



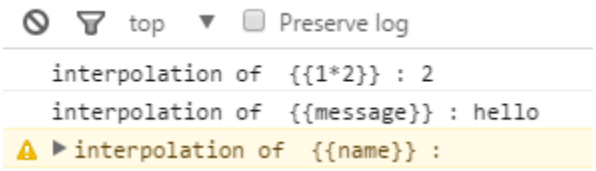
Ex: **With configuring interpolate funtion**

```
app.controller('myCtrl', function ($scope, $interpolate) {

    //can check the interpolation using $interpolate function
    var fn = $interpolate("{{1*2}}");
    fn({ message: "Hello!" });

    //can check interpolate using scope
    $scope.message = "hello";
  });
app.config(function ($provide) {
    $provide.decorator("$interpolate", function ($delegate) {
    //Copy and paste the above interpolate function here....
 });
});
```

o/p: Now if you observe in console window it shows message for every Html element and throws warning it scope object is not existed. Here in controller we are not initializing **$scope.name,** so it throws a warning.



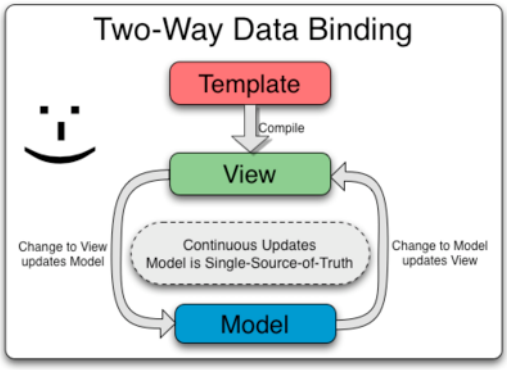
No Control Flow Statements

Apart from the ternary operator (`a ? b : c`), you cannot write a control flow statement in an expression. The reason behind this is core to the Angular philosophy that application logic should be in controllers, not the views. If you need a real conditional, loop, or to throw from a view expression, delegate to a JavaScript method instead.

Data Binding in Angular Templates

Form elements and input elements can be binded to model using ng-model attribute.

Chances to the element reflects in model and vice-versa.



Angular templates work differently. First the template (which is the uncompiled HTML along with any additional markup or directives) is compiled on the browser. The compilation step produces a live view. Any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view.

```
<div ng-app="">
   <p>
      First Name: <input type="text" ng-model="fname">
      Last Name: <input type="text" ng-model="lname">
   </p>
   Hello {{fname + " " + lname}}
</div>
```