THE EXPERT'S VOICE® IN WEB DEVELOPMENT

Beginning AngularJS

LEARN TO BUILD WEB APPLICATIONS USING ANGULARJS, ONE OF THE WEB'S MOST INNOVATIVE JAVASCRIPT FRAMEWORKS

Andrew Grant

Apress®

Code Examples from:

Beginning AngularJS

By: Andrew Grant

Listing_2-1_Hello-World.html

```
<!DOCTYPE html>
<html ng-app>
<head>
    <title>Listing 2-1</title>
    <script src="js/angular.min.js"></script>
</head>
    <body>
        <p>Hello {{'Wor' + 'ld'}}</p>
    </body>
</html>

```

Listing_2-2_Hello-World.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Listing 2-2</title>
    <script src="js/angular.min.js"></script>
</head>
    <body>
        <p ng-app>Hello {{'Wor' + 'ld'}}</p>
        <p>Hello {{'Wor' + 'ld'}}</p>
    </body>
</html>

<!--
All that we have done here is to move the ngApp directive out of the opening HTML element
and place it on the first paragraph element. We also added another paragraph element,
which is almost identical to the first. However this one is without an ngApp directive.
Save Listing 2- 2, and load it up in your browser.

Two interesting things happen:

1. The first interesting thing is that the expression binding in the first paragraph worked
just as it did before. Even though we relocated the ngApp directive, the expression binding
is still nested within its boundaries and, therefore, still under AngularJS control.

2. The second interesting thing is that the second paragraph uses an expression too. However,
this expression binding simply renders as is; it is not evaluated at all. AngularJS simply
isn't interested in it, because it is not contained within the boundaries of an ngApp directive.
In fact, AngularJS has no knowledge of this particular paragraph element or anything contained within it.
 -->

```

Listing_2-3_Live-Updates.html

```
<!DOCTYPE html> <html ng-app>
<head>
    <title>Listing 2-3</title>
    <script src="js/angular.min.js"></script>
</head>
<body>
    <label>City: </label><input ng-model="city" type="text" /></label>
    <p>You entered: {{city}}</p>
</body>
</html>

```

Listing_2-4_A-First-Look-at-ngShow.html

```
<!DOCTYPE html>
<html ng-app>
<head>
    <title>Listing 2-4</title>
    <script src="js/angular.min.js"></script>
</head>
    <body>
    <p ng-show="true">Paragraph 1, can you see me?</p>
    <p ng-show="false">Paragraph 2, can you see me?</p>
    <p ng-show="1 == 1">Paragraph 3, can you see me?</p>
    <p ng-show="1 == 2">Paragraph 4, can you see me?</p>
    </body>
</html>

<!--
Listing 2-4 shows four paragraph elements, each has been "augmented" by an AngularJS directive
that goes by the name of ngShow.

What does ngShow do?
 Much of the answer is in the name.
  The ngShow directive will show, or hide, the element on which it is declared,
  based on the expression provided to it.
Load up Listing 2-4 in your browser, and you will see that only the first and third paragraphs appear
(as confirmed in Figure 2-3). They only appear because, in both cases, their respective expressions
evaluate to the Boolean value of true. The second and fourth paragraphs, however, do not appear
because their respective ngShow expressions evaluate to the Boolean value of false.

* Tip If an ngShow expression evaluates to false, then a CSS class named .ng-hide is dynamically
added to the element, causing it to become hidden. So, the element still exists in the DOM,
but it is not displayed.
 -->

```

Listing_2-5_Demonstrating-ngClick.html

```
<!DOCTYPE html>
<html ng-app> <head>
<title>Listing 2-5</title>
<script src="js/angular.min.js"></script> </head>
<body>
    <button ng-click="count = count + 1" ng-init="count = 0">Increment</button> count: {{count}}
</body>
</html>
```

Listing_2-6_A-Quick-Look-at-AngularJS-Expressions.html

```
<!DOCTYPE html>
<html ng-app>
<head>
    <title>Listing 2-5</title>
    <script src="js/angular.min.js"></script>
</head>
<body>
    <h1>Expression Samples</h1>

    <!-- Basic arithmetic --> <p>6 + 4 = {{6 + 4}}</p>

    <!-- Using a JavaScript string method --> <p>{{"Beginning AngularJS".toUpperCase()}}</p>

    <!-- Searching for an occurence of 'D' --> <p>{{"ABCDEFG".indexOf('D')}}</p>

    <!-- Ternary operation --> <p>{{1==1 ? "Red" : "Blue"}}</p>
</body>
</html>
```

Listing_3-1_Singleton-Pattern.js

```
var Logger = (function() {
    // private variable to hold the only
    // instance of Logger that will exist var loggerInstance;
    // Create the logger instance
    var createLogger = function() {
        var _logWarning = function(message) {
            // some complex work coud go here, but
            // let's just fake it
            return message.toUpperCase();
            };
        return {
            logWarning: _logWarning
        };
    };

    return {
    // Here is the crucial part. First we check
    // to see if an instance already exists. If
    // it does, we return it. If it does not, we
    // create it.
    getInstance: function() {
        if (!loggerInstance) {
            loggerInstance = createLogger();
        }
        return loggerInstance;
    }
}; })();
// Notice how we use getInstance() and we
// do not use direct object creation with the
// new keyword
var myLogger = Logger.getInstance();
myLogger.logWarning("Memory use nearing maximum!");

```

Listing_3-2_MVC-in-Action.js

```
// Listing 3-2. MVC in Action
function MyFirstCtrl($scope) {
    // populate the employees variable with some model data
    var employees = ['Christopher Grant', 'Monica Grant', 'Christopher Grant', 'Jennifer Grant'];
    // Now put this model data into the scope so it can be used in the view
    $scope.ourEmployees = employees;
}

```

Listing_3-3_A-Complete-MVC-Example.html

```
<!DOCTYPE html>
<html ng-app>
<head>
<script src="js/angular.min.js"></script>
<script>

function MyFirstCtrl($scope) {
    var employees = ['Catherine Grant', 'Monica Grant', 'Christopher Grant', 'Jennifer Grant'];
    $scope.ourEmployees = employees;
}

</script>
</head>
    <body ng-controller='MyFirstCtrl'>
        <h2>Number of Employees: {{ ourEmployees.length }}</h2>
    </body>
</html>

```

Listing_3-4_Displaying-the-Employee-Names.html

```
<!DOCTYPE html>
<html ng-app>
<head>
    <script src="js/angular.min.js"></script>
    <script>

    function MyFirstCtrl($scope) {
        var employees = ['Catherine Grant', 'Monica Grant', 'Christopher Grant', 'Jennifer Grant'];
        $scope.ourEmployees = employees;
    }

    </script>
</head>
<body ng-controller='MyFirstCtrl'>
    <h2>Number of Employees: {{ ourEmployees.length }}</h2>
    <p ng-repeat="employee in ourEmployees">{{employee}}</p>
</body>
</html>

```

Listing_4-1_Raw-Sample-Data.html

```
<script>

function MyFilterDemoCtrl($scope) { var someData = {
        firstName: 'JENNA',
        surname: 'GRANT',
        dateJoined: new Date(2010, 2, 23), consumption: 123.659855,
        plan: 'super-basic-plan'
    };
    $scope.data = someData;
}

</script>

```

Listing_4-2_Angular-Filter-Example.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Listing 4-2</title>
    <script src="js/angular.min.js"></script>
    <script>

    function MyFilterDemoCtrl($scope) {
        var someData = {
            firstName: 'JENNA',
            surname: 'GRANT',
            dateJoined: new Date(2010, 2, 23), consumption: 123.659855,
            plan: 'super-basic-plan'
        };
        $scope.data = someData;
    }

    </script>
</head>
<body ng-app ng-controller="MyFilterDemoCtrl">
    <p>
    <!-- Unfiltered data -->
    <strong>First Name</strong>: {{data.firstName}}<br/> <strong>Surname:</strong> {{data.surname}}
    </p>
    <p>
    <!-- Filtered data -->
    <strong>First Name</strong>: {{data.firstName | lowercase}}<br/><strong>Surname:</strong> {{data.surname | lowercase }}
    </p>
</body>
</html>

```

Listing_4-3_Achieving-Same-Result-Without-Filter.html

```
<script>
function MyFilterDemoCtrl($scope) {
    var someData = {
        firstName: 'JENNA',
        surname: 'GRANT',
        dateJoined: new Date(2010, 2, 23),
        consumption: 123.659855,
        plan: 'super-basic-plan'
    };
    // do the lowercaing here instead of using a filter
    someData.firstName = someData.firstName.toLowerCase();
    someData.surname = someData.surname.toLowerCase();
    $scope.data = someData;
}
</script>
```

Listing_4-4_Rounding-Up-Values-with-the-Number-Filter.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Listing 4-4</title>
    <script src="js/angular.min.js"></script>
    <script>
    function MyFilterDemoCtrl($scope) {
            var someData = {
            firstName: 'JENNA',
            surname: 'GRANT',
            dateJoined: new Date(2010, 2, 23), consumption: 123.659855,
            plan: 'super-basic-plan'
            };
        $scope.data = someData;
    }
    </script>
</head>
    <!-- I add "ng-app" to the body tag to get this example to work. -->
    <body ng-app ng-controller="MyFilterDemoCtrl">
        <p>
        Consumption: {{ data.consumption }}<br/>
        Consumption: {{ data.consumption | number }}
        </p>
    </body>
</html>
```

Listing_4-5_The-Date-Filter-in-Action.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Listing 4-5</title>
    <script src="js/angular.min.js"></script>
    <script>

    function MyFilterDemoCtrl($scope) {
        var someData = {
            firstName: 'JENNA', surname: 'GRANT',
            dateJoined: new Date(2010, 2, 23), consumption: 123.659855,
            plan: 'super-basic-plan'
        };
        $scope.data = someData;
    }

    </script>
</head>
<body ng-app ng-controller="MyFilterDemoCtrl">
    <p>medium:<strong> {{ data.dateJoined | date:'medium'}} </strong></p>
    <p>mediumDate:<strong> {{ data.dateJoined | date:'mediumDate'}} </strong></p>
    <p>shortDate:<strong> {{ data.dateJoined | date:'shortDate'}} </strong></p>
    <p>This customer joined in the month of {{ data.dateJoined | date:'MMMM'}} on a {{ data.dateJoined | date:'EEEE'}} at {{
    data.dateJoined | date:'ha'}}</p>
</body>
</html>

```

Listing_4-6_Adding-Historical-Data-to-the-Data-Source.html

```
<script>

function MyFilterDemoCtrl($scope) {
        var someData = {
        firstName: 'JENNA',
        surname: 'GRANT',
        dateJoined: new Date(2010, 2, 23), consumption: 123.659855,
        plan: 'super-basic-plan',
        // Last 12 months of data usage
        monthlyUsageHistory: [
            123.659855,
            89.645222,
            97.235644,
            129.555555,
            188.699855,
            65.652545,
            123.659855,
            89.645222,
            97.235644,
            129.555555,
            188.699855,
            65.652545]
        };
    $scope.data = someData;
}

</script>

```

Listing_4-6_and_Listing_4-7_combined.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Listing 4-6 and 4-7</title>
    <script src="js/angular.min.js"></script>

<!-- Listing 4-6 -->
<script>

function MyFilterDemoCtrl($scope) {
        var someData = {
        firstName: 'JENNA',
        surname: 'GRANT',
        dateJoined: new Date(2010, 2, 23), consumption: 123.659855,
        plan: 'super-basic-plan',
        // Last 12 months of data usage
        monthlyUsageHistory: [
            123.659855,
            89.645222,
            97.235644,
            129.555555,
            188.699855,
            65.652545,
            123.659855,
            89.645222,
            97.235644,
            129.555555,
            188.699855,
            65.652545]
        };
    $scope.data = someData;
}

</script>
</head>
<!-- Listing 4-7 -->
<body ng-app ng-controller="MyFilterDemoCtrl">
    <h2>Gigabytes used over the last 3 months</h2>
    <ul>
        <li ng-repeat="gigabytes in data.monthlyUsageHistory | limitTo:5"> {{ gigabytes | number:2 }}</li>
    <!--
    Of course, if you wanted to show only the first three items, you could do so by using limitTo:3.
    What if you wanted to show only the last three items? Specifying a negative value can do this.
    If you replace the li element in Listing 4-7 so that it uses the following code snippet,
    you should see results like those shown in Figure 4-6.

    <li ng-repeat="gb in data.monthlyUsageHistory | limitTo:-3">
     -->
    </ul>
</body>
</html>

```

Listing_4-7_Displaying-a-Subset-of-the-monthlyUsageHistory-Data.html

```
<body ng-app ng-controller="MyFilterDemoCtrl">
    <h2>Gigabytes used over the last 3 months</h2>
    <ul>
        <li ng-repeat="gigabytes in data.monthlyUsageHistory | limitTo:5"> {{ gigabytes | number:2 }}</li>
    </ul>
</body>
```

Listing_4-8_myAppModule.js

```
// So far, we have used ngApp in its simplest form, as an attribute without any value. However,
// you can specify an AngularJS default module, by providing a value. The following code snippet
// shows ngApp with a value of 'myAppModule', which is the name of the module we have just created.

// <html ng-app="myAppModule">

// With the ngApp directive in place, we can save our module, myAppModule.js, into the js directory.
// Then we can create a new page, index.html, which will make use of this module.
// The next two code listings (Listings 4-8 and Listing 4-9) will pull all of this together.


// Listing 4-8. myAppModule.js

// create a new module called 'myAppModule' and save
// a reference to it in a variable called myAppModule
var myAppModule = angular.module('myAppModule', []);
// use the myAppModule variable to
// configure the module with a controller
    myAppModule.controller('MyFilterDemoCtrl', function ($scope) {
    // controller code would go here
    }
);
// use the myAppModule variable to
// configure the module with a filter
    myAppModule.filter('stripDashes', function() {
        return function(txt) {
    // filter code would go here
    };
});
```

Listing_4-9_Referring-to-the-Module-by-Name.js

```
// Listing 4-9. myAppModule.js

// Create a new module
angular.module('myAppModule', []);
// configure the module with a controller
angular.module('myAppModule').controller('MyFilterDemoCtrl', function ($scope) {
    // controller code would go here
});
// configure the module with a filter
angular.module('myAppModule').filter('stripDashes', function() {
    return function(txt) {
    // filter code would go here
    };
});

// This file does not use a variable to store a reference to the module. Instead,
// it uses the single argument version of the angular.module method to retrieve a
// reference to it.
// This single argument is the name we gave the module when we
// created it. It really doesn't make much difference which approach you use, and
// both are commonly used. I prefer the approach in Listing 4-8, where we store a
// reference, as there is less repetition of the module name, so fewer chances of
// typos creeping in. Sometimes, however, you might find you need to get a reference
// to a module, and the single argument version of the module method might be the
// only way to get it.

```

Listing_4-10_An-index.html-File-Set-Up-to-Use-myAppModule.html

```
<!DOCTYPE html >
<html ng-app="myAppModule">
<head lang="en">
    <meta charset="UTF-8">
    <title>Listing 4-10</title>
    <script src="js/angular.min.js"></script>
    <script src="js/myAppModule.js"></script>
</head>
<body ng-controller="MyFilterDemoCtrl">
</body>
</html>

```

Listing_4-11_A-Simple-Replace-Dashes-Function.html

```
<script>

function stripDashes(txt) {
    return txt.split('-').join(' ');
};

console.log(stripDashes("super-basic-plan"));
console.log(stripDashes("something-with-a-lot-more-dashes-plan"));
console.log(stripDashes("noDashesPlan"));

</script>
```

Listing_4-12_An-Angular-Filter-Implementation.js

```
myAppModule.filter('stripDashes', function () {
    // the function we are in returns
    // the function below
    return function(txt) {
        return textToFilter.split('-').join(' ');
    };
});

// Of particular note here is the fact that the filter function does not itself
// implement our logic; rather, it returns a function that implements it.
// This is why that second argument supplied to the filter method is called
// a "factory function"; its main purpose in life is to manufacture functions.
```

Listing_4-13_Trying-Out-the-stripDashes-Filter.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Filter Demo</title>
    <script src="js/angular.min.js"></script>
    <script src="js/myModules/myAppModule.js"></script>
</head>
<body ng-app="myAppModule" ng-controller="MyFilterDemoCtrl">
    <p>Plan type: {{ data.plan }}</p>
    <p>Plan type: {{ data.plan | stripDashes }}</p>
</body>
</html>
```

Listing_4-14_A-Basic-Title-Casing-Function.html

```
<script>
function toTitleCase(str) {
    return str.charAt(0).toUpperCase() + str.substr(1).toLowerCase();
}
console.log(toTitleCase("jennifer"));
console.log(toTitleCase("jENniFEr"));
console.log(toTitleCase("jENniFEr amanda Grant"));
</script>
```

Listing_4-15_A-Better-Title-Casing-Function.html

```
<script>
function toTitleCase(str) {
    return str.replace(/\w\S*/g, function(txt){return txt.charAt(0).toUpperCase() + txt.substr(1).toLowerCase();});
}
console.log(toTitleCase("jennifer"));
console.log(toTitleCase("jENniFEr"));
console.log(toTitleCase("jENniFEr amanda Grant"));
</script>
```

Listing_4-16_An-Angular-Filter-Implementation.js

```
myAppModule.filter("toTitleCase", function () {
    return function (str) {
        return str.replace(/\w\S*/g, function(txt){
            return txt.charAt(0).toUpperCase() + txt.substr(1).toLowerCase();
        });
    };
});
```

Listing_4-17_Using-the-toTitleCase-Filter.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Filter Demo</title>
    <script src="js/angular.min.js"></script>
    <script src="js/myModules/myAppModule.js"></script>
</head>
<body ng-app="myAppModule" ng-controller="MyFilterDemoCtrl">
    <!-- Display customer name in title case -->
    <p>First Name: {{data.firstName | toTitleCase}}</p>
    <p>Surname: {{data.surname | toTitleCase}}</p>
    <p>Full Name: {{ data.firstName + data.surname | toTitleCase}}</p>
</body>
</html>

```