SHAPING UP WITH ANGULAR.JS
Shaping Up with Angular

Level 1: Getting Started
What you need to know

**Must know**
- HTML & CSS
- JavaScript

**Nice to know**
- Automated Testing
- BDD - Behavior Driven Development
- TDD - Test Driven Development
- etc

**Not so important**
- jQuery
- Ruby on Rails
- Python, PHP, etc
- Databases
If you’re using JavaScript to create a dynamic website, Angular is a good choice.

• Angular helps you organize your JavaScript
• Angular helps create responsive (as in fast) websites.
• Angular plays well with jQuery
• Angular is easy to test
Traditional Page-Refresh

Make your way down a Path and build specific skills, or wander through All Courses.

Ruby
Master your Ruby skills and increase your Rails street cred by learning to build dynamic, sustainable applications for the web.

Topics covered:
- Ruby Basics
- Ruby Only
- Ruby on Rails
- Starting Rails
- Advanced Ruby
- Testing

JavaScript
Spend some time with this powerful scripting language and learn to build lightweight applications with enhanced user interfaces.

Topics covered:
- JavaScript
- jQuery
- Backbone.js
- Node.js
- CoffeeScript
- Ember.js
Web Server

URL Request to server

Response with Webpage & Assets

Web Browser

Browser loads up entire webpage.

User clicks on link, new Request

Response with Webpage & Assets

Browser loads up entire webpage.
A “responsive” website using Angular
Web Server

URL Request to server

Response with Webpage & Assets

Web Browser

User clicks on link, new Request

Response with JSON Data

Browser loads up entire webpage.

HTML
JavaScript

DATA

Data is loaded into existing page.
What is Angular JS?

A client-side JavaScript Framework for adding interactivity to HTML.

How do we tell our HTML when to trigger our JavaScript?

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

```javascript
function Store()
{
  alert('Welcome, Gregg!');
}
```

index.html

app.js
Directives

A Directive is a marker on a HTML tag that tells Angular to run or reference some JavaScript code.

```
<!DOCTYPE html>
<html>
  <body ng-controller="StoreController">
    ...
  </body>
</html>
```

```
function StoreController(){
  alert('Welcome, Gregg!');
}
```
Flatlander Crafted Gems
– an Angular store –

Gem #1: Zircon $1,100.00

Description

In is our most coveted and sought after gem.
Downloading the libraries

Download AngularJS http://angularjs.org/
We’ll need angular.min.js

Download Twitter Bootstrap http://getbootstrap.com/
We’ll need bootstrap.min.css
Getting Started

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="bootstrap.min.css" />
  </head>
  <body>
    <script type="text/javascript" src="angular.min.js"></script>
  </body>
</html>
```
Modules

- Where we write pieces of our Angular application.
- Makes our code more maintainable, testable, and readable.
- Where we define dependencies for our app.

Modules can use other Modules...
Creating Our First Module

```javascript
var app = angular.module('store', []);
```

AngularJS

Application Name

Dependencies

Other libraries we might need. We have none... for now...
Including Our Module

```javascript
var app = angular.module('store', []);
```

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="bootstrap.min.css" />
</head>
<body>
    <script type="text/javascript" src="angular.min.js"></script>
    <script type="text/javascript" src="app.js"></script>
</body>
</html>
```
Including Our Module

```html
<!DOCTYPE html>
<html ng-app="store">
  <head>
    <link rel="stylesheet" type="text/css" href="bootstrap.min.css" />
  </head>
  <body>
    <script type="text/javascript" src="angular.min.js"></script>
    <script type="text/javascript" src="app.js"></script>
  </body>
</html>
```

Run this module when the document loads.

```javascript
var app = angular.module('store', []);
```

index.html

app.js

SHAPING UP WITH ANGULAR.JS
Expressions

Allow you to insert dynamic values into your HTML.

Numerical Operations

\[
\begin{align*}
<p>
I am {{4 + 6}}
</p>
\end{align*}
\]

evaluates to

\[
\begin{align*}
<p>
I am 10
</p>
\end{align*}
\]

String Operations

\[
\begin{align*}
<p>
{{"hello" + " you"}}
</p>
\end{align*}
\]

evaluates to

\[
\begin{align*}
<p>
hello you
</p>
\end{align*}
\]

+ More Operations:
http://docs.angularjs.org/guide/expression
Including Our Module

```javascript
var app = angular.module('store', []);
```

```html
<!DOCTYPE html>
<html ng-app="store">
  <head>
    <link rel="stylesheet" type="text/css" href="bootstrap.min.css" />
  </head>
  <body>
    <script type="text/javascript" src="angular.min.js"></script>
    <script type="text/javascript" src="app.js"></script>
    <p>{{"hello" + " you"}}</p>
  </body>
</html>
```
Challenges
Working With Data

...just a simple object we want to print to the page.

```javascript
var gem = {
    name: 'Dodecahedron',
    price: 2.95,
    description: '....',
}
```

Dodecahedron

Some gems have hidden qualities beyond their luster, beyond their shine... Dodeca is one of those gems.

$2.95
Controllers

Controllers are where we define our app’s behavior by defining functions and values.

Wrapping your Javascript in a closure is a good habit!

```javascript
var app = angular.module('store', []);

app.controller('StoreController', function(){
    
});

var gem = {
    name: 'Dodecahedron',
    price: 2.95,
    description: '...
};
```
Now how do we print out this data inside our webpage?

```javascript
(function(){
    var app = angular.module('store', [ ]); 

    app.controller('StoreController', function(){
        this.product = gem;
    });

    var gem = {
        name: 'Dodecahedron',
        price: 2.95,
        description: '....',
    }

    })();
```
Our Current HTML

<!DOCTYPE html>
<html ng-app="store">
<head>
  <link rel="stylesheet" type="text/css" href="bootstrap.min.css" />
</head>
<body>
  <div>
    <h1>Product Name</h1>
    <h2>$Product Price</h2>
    <p>Product Description</p>
  </div>
  <script type="text/javascript" src="angular.min.js"></script>
  <script type="text/javascript" src="app.js"></script>
</body>
</html>

Let's load our data into this part of the page.
Attaching the Controller

```html
<body>
  <div>
    <h1>Product Name</h1>
    <h2>$Product Price</h2>
    <p>Product Description</p>
  </div>

  <script type="text/javascript" src="angular.min.js"></script>
  <script type="text/javascript" src="app.js"></script>
</body>
```

```javascript
(function(){
  var app = angular.module('store', [ ]);

  app.controller('StoreController', function(){
    this.product = gem;
  });

  ...
})();
```
Attaching the Controller

### Directive

```html
<div ng-controller="StoreController as store">
  <h1>Product Name</h1>
  <h2>Product Price</h2>
  <p>Product Description</p>
</div>
```

### Controller name

```javascript
(function(){
  var app = angular.module('store', [ ]);

  app.controller('StoreController', function(){
    this.product = gem;
  });
...}
()());
```

### Alias

```javascript
<script type="text/javascript" src="angular.min.js"></script>
<script type="text/javascript" src="app.js"></script>
```
Displaying Our First Product

```html
<body>
  <div ng-controller="StoreController as store">
    <h1>{{store.product.name}}</h1>
    <h2>${{store.product.price}}</h2>
    <p>{{store.product.description}}</p>
  </div>
</body>
```

```javascript
(function(){
  var app = angular.module('store', [ ]);

  app.controller('StoreController', function(){
    this.product = gem;
  });

  ...}

 })();
```

Dodecahedron

Some gems have hidden qualities beyond their luster, beyond their shine... Dodeca is one of those gems. $2.95
Understanding Scope

The scope of the Controller is only inside here...

Would never print a value!
Challenges
Adding A Button

```html
<body ng-controller="StoreController as store">
  <div>
    <h1>{{store.product.name}}</h1>
    <h2>$ {{store.product.price}}</h2>
    <p>{{store.product.description}}</p>
  </div>
  <script type="text/javascript" src="angular.min.js"></script>
  <script type="text/javascript" src="app.js"></script>
</body>

index.html

```
Adding A Button

Directives to the rescue!

How can we only show this button...

...when this is true?

```html
<body ng-controller="StoreController as store">
  <div>
    <h1>{{store.product.name}}</h1>
    <h2>$ {{store.product.price}}</h2>
    <p>{{store.product.description}}</p>
    <button>Add to Cart</button>
  </div>
</body>
```

```javascript
var gem = {
  name: 'Dodecahedron',
  price: 2.95,
  description: '. . .',
  canPurchase: false
}
```
NgShow Directive

```html
<body ng-controller="StoreController as store">
  <div>
    <h1>{{store.product.name}}</h1>
    <h2>$ {{store.product.price}}</h2>
    <p>{{store.product.description}}</p>
    <button ng-show="store.product.canPurchase">Add to Cart</button>
  </div>
<script type="text/javascript" src="angular.min.js"></script>
<script type="text/javascript" src="app.js"></script>
</body>
```

`var gem = {
  name: 'Dodecahedron',
  price: 2.95,
  description: '...',
  canPurchase: false
};`

Will only show the element if the value of the Expression is true.
<script data-require="angular.js@1.2.x" src="http://code.angularjs.org/1.2.15/angular.js"></script>
<script src="app.js"></script>

<body ng-controller="StoreController as store">

<div class="list-group">
  <div class="list-group-item">
    <h1>{{store.product.name}}</h1>
    <h2>${{store.product.price}}</h2>
    <p>{{store.product.description}}</p>
    <button ng-show="store.product.canPurchase">Add to Cart</button>
  </div>
</div>
</body>
</html>
NgHide Directive

If the product is sold out we want to hide it.

```javascript
var gem = {
    name: 'Dodecahedron',
    price: 2.95,
    description: '...',
    canPurchase: true,
    soldOut: true
};
```
NgHide Directive

If the product is sold out we want to hide it.

```javascript
var gem = {
    name: 'Dodecahedron',
    price: 2.95,
    description: '. . .',
    canPurchase: true,
    soldOut: true,
}
```

This is awkward and a good example to use ng-hide!
NgHide Directive

If the product is sold out we want to hide it.
app.controller('StoreController', function(){
  this.product = gem;
});

var gem = {
  name: "Dodecahedron",
  price: 2.95,
  description: ". . .",
  canPurchase: true,
}
Multiple Products

```javascript
var gems = [
    {
        name: "Dodecahedron",
        price: 2.95,
        description: "...",
        canPurchase: true,
    },
    {
        name: "Pentagonal Gem",
        price: 5.95,
        description: "...",
        canPurchase: false,
    }...
];
```

Now we have an array...

How might we display all these products in our template?

Maybe a Directive?
Working with An Array

```html
<body ng-controller="StoreController as store">
    <div>
        <h1>{{store.products[0].name}}</h1>
        <h2>${{store.products[0].price}}</h2>
        <p>{{store.products[0].description}}</p>
        <button ng-show="store.products[0].canPurchase">Add to Cart</button>
    </div>
</body>
```
Working with An Array

```html
<body ng-controller="StoreController as store">
  <div>
    <h1>{{store.products[0].name}}</h1>
    <h2>${{store.products[0].price}}</h2>
    <p>{{store.products[0].description}}</p>
    <button ng-show="store.products[0].canPurchase">Add to Cart</button>
  </div>
...</body>
```

Displaying the first product is easy enough...
Working with An Array

```html
<body ng-controller="StoreController as store">
  <div>
    <h1>{{store.products[0].name}}</h1>
    <h2>$ {{store.products[0].price}}</h2>
    <p>{{store.products[0].description}}</p>
    <button ng-show="store.products[0].canPurchase">Add to Cart</button>
  </div>
  <div>
    <h1>{{store.products[1].name}}</h1>
    <h2>$ {{store.products[1].price}}</h2>
    <p>{{store.products[1].description}}</p>
    <button ng-show="store.products[1].canPurchase">Add to Cart</button>
  </div>
  ... That works...
</body>
```

Why repeat yourself?
That works...
Why... You get it.

index.html
Working with An Array

```html
<body ng-controller="StoreController as store">
  <div ng-repeat="product in store.products">
    <h1>{{product.name}}</h1>
    <h2>${{product.price}}</h2>
    <p>{{product.description}}</p>
    <button ng-show="product.canPurchase">
      Add to Cart
    </button>
  </div>
  ...
</body>
```

Repeat this section for each product.
What We Have Learned So Far

- **Directives** – HTML annotations that trigger Javascript behaviors
- **Modules** – Where our application components live
- **Controllers** – Where we add application behavior
- **Expressions** – How values get displayed within the page
Challenges
SHAPING UP WITH ANGULAR.JS
Shaping Up with Angular.JS

Level 2: Filters, Directives, and Cleaner Code
Directives We Know & Love

ng-app – attach the Application Module to the page

```html
<html ng-app="store">
</html>
```

ng-controller – attach a Controller function to the page

```html
<body ng-controller="StoreController as store">
</body>
```

ng-show / ng-hide – display a section based on an Expression

```html
<h1 ng-show="name"> Hello, {{name}}! </h1>
```

ng-repeat – repeat a section for each item in an Array

```html
<li ng-repeat="product in store.products"> {{product.name}} </li>
```
Our Current Code

```html
<body ng-controller="StoreController as store">
  <ul class="list-group">
    <li class="list-group-item ng-repeat="product in store.products">
      <h3>{{product.name}}</h3>
      <em class="pull-right">${{product.price}}</em>
    </li>
  </ul>
</body>
```

There's a better way to print out prices.

Dodecahedron   $2
Pentagonal Gem  $5.95
Our First Filter

```html
<body ng-controller="StoreController as store">
  <ul class="list-group">
    <li class="list-group-item" ng-repeat="product in store.products">
      <h3>{{product.name}}</h3>
      <em class="pull-right">{{product.price | currency}}</em>
    </li>
  </ul>
</body>
```

Pipe - "send the output into"
Notice it gives the dollar sign (localized)
Specifies number of decimals
Formatting with Filters

*Our Recipe* {{ data* | filter:options* }}

date
{{'1388123412323' | date:'MM/dd/yyyy @ h:mma'}} 12/27/2013 @ 12:50AM

uppercase & lowercase
{{'octagon gem' | uppercase}} OCTAGON GEM

limitTo
{{'My Description' | limitTo:8}} My Descr

<li ng-repeat="product in store.products | limitTo:3">

orderBy
Will list products by descending price.
<li ng-repeat="product in store.products | orderBy:'-price'">
Without the - products would list in ascending order.
```javascript
var gems = [
  {
    name: 'Dodecahedron Gem',
    price: 2.95,
    description: '...',
    images: [
      {
        full: 'dodecahedron-01-full.jpg',
        thumb: 'dodecahedron-01-thumb.jpg'
      },
      {
        full: 'dodecahedron-02-full.jpg',
        ...,
      }
    ]
  },
  ...,
];
```

Adding an Image Array to our Product Array

Our New Array

Image Object

To display the first image in a product:

```
{{product.images[0].full}}
```
Using ng-src for Images

Using Angular Expressions inside a `src` attribute causes an error!

...the browser tries to load the image *before* the Expression evaluates.

```html
<body ng-controller="StoreController as store">
  <ul class="list-group">
    <li class="list-group-item" ng-repeat="product in store.products">
      <h3>{{product.name}}</h3>
      <em class="pull-right">{{product.price | currency}}</em>
      <img ng-src="{{product.images[0].full}}"/>
    </li>
  </ul>
</body>
```
Flatlander Crafted Gems
– an Angular store –

Pentagonal Gem $5.95

Dodecahedron $2.95
Challenges
How can I make my application more interactive?
A Simple Set of Tabs

Description

Some gems have hidden qualities beyond their luster, beyond their shine... Dodeca is one of those gems.
Introducing a new Directive!

Assigning a value to `tab`.

For now just print this value to the screen.
Introducing a new Directive!

Flatlander Crafted Gems
— an Angular store —

Pentagonal Gem $5.95

Dodecahedron $2.95
Whoa, it’s dynamic and stuff...

When `ng-click` changes the value of `tab`...

... the `{{tab}}` expression automatically gets updated!

Expressions define a 2-way Data Binding...

this means Expressions are re-evaluated when a property changes.
Let's add the tab content panels

How do we make the tabs trigger the panel to show?

```html
<div class="panel">
  <h4>Description</h4>
  <p>{{product.description}}</p>
</div>
<div class="panel">
  <h4>Specifications</h4>
  <blockquote>None yet</blockquote>
</div>
<div class="panel">
  <h4>Reviews</h4>
  <blockquote>None yet</blockquote>
</div>
```
Let's add the tab content panels

Now when a tab is selected it will show the appropriate panel!
But how do we set an initial value for a tab?
Setting the Initial Value

**ng-init** allows us to evaluate an expression in the current scope.

```html
<section ng-init="tab = 1">
  <ul class="nav nav-pills">
    <li><a href ng-click="tab = 1">Description</a></li>
    <li><a href ng-click="tab = 2">Specifications</a></li>
    <li><a href ng-click="tab = 3">Reviews</a></li>
  </ul>
</section>
```
Flatlander Crafted Gems
– an Angular store –

Pentagonal Gem  $5.95

Description
Origin of the Pentagonal Gem is unknown, hence its low value. It has a very high shine and 12 sides, making it a popular choice for collectors.
Setting the Active Class

We need to set the class to active if current tab is selected ...

```html
<section ng-init="tab = 1">
  <ul class="nav nav-pills">
    <li><a href ng-click="tab = 1">Description</a></li>
    <li><a href ng-click="tab = 2">Specifications</a></li>
    <li><a href ng-click="tab = 3">Reviews</a></li>
  </ul>
</section>
```
The ng-class directive

Expression to evaluate if true, set class to "active", otherwise nothing.

Name of the class to set.
Flatlander Crafted Gems
– an Angular store –

Pentagonal Gem $5.95

Description

Origin of the Pentagonal Gem is unknown, hence its low value. It has a very high shine and 12 sides,
Feels dirty, doesn’t it?

All our application’s logic is inside our HTML.

How might we pull this logic into a Controller?
Creating our Panel Controller

```html
<section ng-init="tab = 1" ng-controller="PanelController as panel">
  <ul class="nav nav-pills">
    <li ng-class="{ active: tab === 1 }">
      <a href ng-click="tab = 1">Description</a>
    </li>
    <li ng-class="{ active: tab === 2 }">
      <a href ng-click="tab = 2">Specifications</a>
    </li>
    <li ng-class="{ active: tab === 3 }">
      <a href ng-click="tab = 3">Reviews</a>
    </li>
  </ul>
  <div class="panel" ng-show="tab === 1">
    <h4>Description</h4>
    <p>{{product.description}}</p>
  </div>
  ...
</section>
```

```javascript
app.controller("PanelController", function() {
});
```
Moving our tab initializer

```html
<section ng-controller="PanelController as panel">
  <ul class="nav nav-pills">
    <li ng-class="{ active: tab === 1 }">
      <a href ng-click="tab = 1">Description</a>
    </li>
    <li ng-class="{ active: tab === 2 }">
      <a href ng-click="tab = 2">Specifications</a>
    </li>
    <li ng-class="{ active: tab === 3 }">
      <a href ng-click="tab = 3">Reviews</a>
    </li>
  </ul>
  <div class="panel" ng-show="tab === 1">
    <h4>Description</h4>
    <p>{{product.description}}</p>
  </div>
</section>

app.controller("PanelController", function(){
  this.tab = 1;
});
```
Creating our selectTab function

```html
<section ng-controller="PanelController as panel">
  <ul class="nav nav-pills">
    <li ng-class="{ active: tab === 1 }">
      <a href ng-click="panel.selectTab(1)">Description</a>
    </li>
    <li ng-class="{ active: tab === 2 }">
      <a href ng-click="panel.selectTab(2)">Specifications</a>
    </li>
    <li ng-class="{ active: tab === 3 }">
      <a href ng-click="panel.selectTab(3)">Reviews</a>
    </li>
  </ul>
  <div class="panel" ng-show="tab === 1">
    <h4>Description</h4>
    <p>{{product.description}}</p>
  </div>
  ...
</section>
```

```javascript
app.controller("PanelController", function(){
  this.tab = 1;

  this.selectTab = function(setTab) {
    this.tab = setTab;
  };
});
```
Creating our is_selected function

```html
<section ng-controller="PanelController as panel">
  <ul class="nav nav-pills">
    <li ng-class="{ active: panel.isSelected(1) }">
      <a href ng-click="panel.selectTab(1)">Description</a>
    </li>
    <li ng-class="{ active: panel.isSelected(2) }">
      <a href ng-click="panel.selectTab(2)">Specifications</a>
    </li>
    <li ng-class="{ active: panel.isSelected(3) }">
      <a href ng-click="panel.selectTab(3)">Reviews</a>
    </li>
  </ul>
  <div class="panel" ng-show="panel.isSelected(1)">
    <h4>Description</h4>
    <p>{{product.description}}</p>
  </div>
</section>
```
Origin of the Pentagonal Gem is unknown, hence its low value. It has a very high shine and 12 sides,
Challenges
SHAPING UP WITH ANGULAR.JS
Shaping Up with Angular

Level 3: Forms, Models, and Validations
How can I let my users add content?
Adding reviews to our products

```javascript
app.controller("StoreController", function(){
  this.products = [
    {
      name: 'Awesome Multi-touch Keyboard',
      price: 250.00,
      description: "...",
      images: [...],
      reviews: [
        {
          stars: 5,
          body: "I love this product!",
          author: "joe@thomas.com"
        },
        {
          stars: 1,
          body: "This product sucks",
          author: "tim@hater.com"
        }
      ]
    }
  ]
```

Looping Over Reviews in our Tab

```html
<li class="list-group-item" ng-repeat="product in store.products">
  ...
</li>

<div class="panel" ng-show="panel.isSelected(3)">
  <h4>Reviews</h4>
  <blockquote ng-repeat="review in product.reviews">
    <b>Stars: {{review.stars}}</b>
    {{review.body}}
    <cite>by: {{review.author}}</cite>
  </blockquote>
</div>
```

*index.html*
We’re displaying Reviews!

Reviews

3 Stars I think this gem was just OK, could honestly use more shine, IMO.
—JimmyDean@sausage.com

4 Stars Any gem with 12 faces is for me!
—gemsRock@alyssaNicoll.com

Nothing new here, but how do we start to implement forms?
Writing out our Review Form

<h4>Reviews</h4>

<blockquote ng-repeat="review in product.reviews">
...
</blockquote>

<form name="reviewForm">
  <select>
    <option value="1">1 star</option>
    <option value="2">2 stars</option>
    ...
  </select>
  <textarea></textarea>
  <label>by:</label>
  <input type="email" />
  <input type="submit" value="Submit" />
</form>

Reviews
Submit a Review
Rate the Product
Write a short review of the product...

JimmyDean@sausage.com
Submit Review
With Live Preview

```html
<form name="reviewForm">
  <blockquote>
    <b>Stars: {{review.stars}}</b>
    {{review.body}}
    <cite>by: {{review.author}}</cite>
  </blockquote>
  <select>
    <option value="1">1 star</option>
    <option value="2">2 stars</option>
    ... 
  </select>
  <textarea></textarea>
  <label>by:</label>
  <input type="email" />
  <input type="submit" value="Submit" />
</form>
```

How do we bind this review object to the form?
Introducing ng-model

<form name="reviewForm">
  <blockquote>
    <b>Stars: {{review.stars}}</b>
    {{review.body}}
    <cite>by: {{review.author}}</cite>
  </blockquote>
  <select ng-model="review.stars">
    <option value="1">1 star</option>
    <option value="2">2 stars</option>
    . . .
  </select>
  <textarea ng-model="review.body"></textarea>
  <label>by:</label>
  <input ng-model="review.author" type="email" />
  <input type="submit" value="Submit" />
</form>
But how do we actually add the new review?
Two More Binding Examples

With a Checkbox

```html
<input ng-model="review.terms" type="checkbox" />
I agree to the terms
```

Sets value to true or false

With Radio Buttons

What color would you like?

```html
<input ng-model="review.color" type="radio" value="red" />
Red
<input ng-model="review.color" type="radio" value="blue" />
Blue
<input ng-model="review.color" type="radio" value="green" />
Green
```

Sets the proper value based on which is selected
Challenges
We need to Initialize the Review

We could do ng-init, but we're better off creating a controller.

```html
<form name="reviewForm">
  <blockquote>
    <b>Stars: {{review.stars}}</b>
    {{review.body}}
    <cite>by: {{review.author}}</cite>
  </blockquote>
  <select ng-model="review.stars">
    <option value="1">1 star</option>
    <option value="2">2 stars</option>
    ... ...
  </select>
  <textarea ng-model="review.body"></textarea>
  <label>by:</label>
  <input ng-model="review.author" type="email" />
  <input type="submit" value="Submit" />
</form>
```
Creating the Review Controller

Now we need to update all the Expressions to use this controller alias.

```javascript
app.controller("ReviewController", function(){
  this.review = {}; 
});
```

```html
<form name="reviewForm" ng-controller="ReviewController as reviewCtrl">
  <blockquote>
    <b>Stars: {{review.stars}}</b>
    {{review.body}}
    <cite>by: {{review.author}}</cite>
  </blockquote>
  <select ng-model="review.stars">
    <option value="1">1 star</option>
    <option value="2">2 stars</option>
    . . .
  </select>
  <textarea ng-model="review.body"></textarea>
</form>
```
Using the `reviewCtrl.review`

```javascript
app.controller("ReviewController", function(){
  this.review = {}; 
});
```

```html
<form name="reviewForm" ng-controller="ReviewController as reviewCtrl">
  <blockquote>
    <b>Stars: {{reviewCtrl.review.stars}}</b>
    {{reviewCtrl.review.body}}
    <cite>by: {{reviewCtrl.review.author}}</cite>
  </blockquote>

  <select ng-model="reviewCtrl.review.stars">
    <option value="1">1 star</option>
    <option value="2">2 stars</option>
    ... 
  </select>

  <textarea ng-model="reviewCtrl.review.body"></textarea>
</form>
```
Using ng-submit to make the Form Work

ng-submit allows us to call a function when the form is submitted.

We need to define this function.

```javascript
app.controller("ReviewController", function(){
  this.review = {}; 
});
```

```html
<form name="reviewForm" ng-controller="ReviewController as reviewCtrl"
  ng-submit="reviewCtrl.addReview(product)"
>
  <blockquote>
    <b>Stars: {{reviewCtrl.review.stars}}</b>
    {{reviewCtrl.review.body}}
    <cite>by: {{reviewCtrl.review.author}}</cite>
  </blockquote>
</form>
```
Using ng-submit to make the Form Work

```javascript
app.controller("ReviewController", function(){
  this.review = {};

  this.addReview = function(product) {
    product.reviews.push(this.review);
  }
});
```

```html
<form name="reviewForm" ng-controller="ReviewController as reviewCtrl"
  ng-submit="reviewCtrl.addReview(product)"
  
  <blockquote>
    <b>Stars: {{reviewCtrl.review.stars}}</b>
    {{reviewCtrl.review.body}}
    <cite>by: {{reviewCtrl.review.author}}</cite>
  </blockquote>

app.js

index.html
```
Now with Reviews!

1 Star: My gem that I have for life.
— gemsRock@alyssaNicoll.com

2 Stars: Reviewing products is fun
—

Submit a Review

2

reviewing products is fun

funreviewer

Submit Review

Dodecahedron $2.95

Review gets added, but the form still has all previous values!
Resetting the Form on Submit

```javascript
app.controller("ReviewController", function(){
    this.review = {};

    this.addReview = function(product) {
        product.reviews.push(this.review);
        this.review = {};
    }
});
```

Clear out the review, so the form will reset.

```html
<form name="reviewForm" ng-controller="ReviewController as reviewCtrl"
    ng-submit="reviewCtrl.addReview(product)">
    <blockquote>
        <b>Stars: {{reviewCtrl.review.stars}}</b>
        {{reviewCtrl.review.body}}
        <cite>by: {{reviewCtrl.review.author}}</cite>
    </blockquote>
</form>
```
This Time the Form Resets

Submit a Review

5 Stars This gem is SO

This gem is SO

Submit Review

However, if we refresh, the reviews get reset!

We’re not saving the reviews anywhere yet...
What about validations?

Turns out Angular has some great client side validations we can use with our directives.
Our Old Form Code

```html
<form name="reviewForm" ng-controller="ReviewController as reviewCtrl"
  ng-submit="reviewCtrl.addReview(product)"
  ng-submit="reviewCtrl.addReview(product)">
  <select ng-model="reviewCtrl.review.stars">
    <option value="1">1 star</option>
    ...
  </select>
  
  <textarea name="body" ng-model="reviewCtrl.review.body"></textarea>
  
  <label>by:</label>
  <input name="author" ng-model="reviewCtrl.review.author" type="email" />

  <input type="submit" value="Submit" />
</form>
```
Now with validation

```
<form name="reviewForm" ng-controller="ReviewController as reviewCtrl"
  ng-submit="reviewCtrl.addReview(product)"
  novalidate>
  <select ng-model="reviewCtrl.review.stars" required>
    <option value="1">1 star</option>
    ...</select>
  
  <textarea name="body" ng-model="reviewCtrl.review.body" required></textarea>
  <label>by:</label>
  <input name="author" ng-model="reviewCtrl.review.author" type="email" required/>

  <div> reviewForm is {{reviewForm.$valid}} </div>
  <input type="submit" value="Submit" />
</form>
```
With validations

5 Stars
— alyssa@codeschool.com

Submit a Review

5

Write a short review of the product...

alyssa@codeschool.com

Submit Review

reviewForm is false

We don’t want the form to submit when it’s invalid.
Preventing the Submit

We only want this method to be called if reviewForm.$valid is true.
Preventing the Submit

If valid is false, then addReview is never called.
Doesn’t Submit an Invalid Form

How might we give a hint to the user why their form is invalid?
The Input Classes

Source before typing email

```html
<input name="author" ng-model="reviewCtrl.review.author" type="email" required />
```

Source after typing, with invalid email

```html
<input name="author" . . . class="ng-pristine ng-invalid">
```

Source after typing, with valid email

```html
<input name="author" . . . class="ng-dirty ng-valid">
```

So, let’s highlight the form field using classes after we start typing, ng-dirty showing if a field is valid or invalid.
The classes

```html
<input name="author" ng-model="reviewCtrl.review.author" type="email" required />
```

```css
.ng-invalid.ng-dirty { 
  border-color: #FA787E;
}

.ng-valid.ng-dirty { 
  border-color: #78FA89;
}
```

Red border for invalid

Green border for valid
Now with red and green borders!

4 Stars Any gem with 12 faces is for me!
—gemsRock@alyssaNicoll.com

Submit a Review

Rate the Product

Write a short review of the product...

jimmyDean@sausage.com

Submit Review

reviewForm is false

Dodecahedron

$2.95
HTML5-based type validations

Web forms usually have rules around valid input:

- Angular JS has built-in validations for common input types:

  ```html
  <input type="email" name="email">
  <input type="url" name="homepage">
  <input type="number" name="quantity">
  ```

  Can also define min & max with numbers

  ```html
  min=1 max=10
  ```
SHAPING UP WITH ANGULAR.JS
SHAPING UP WITH ANGULAR.JS
Shaping Up with Angular JS
Level 4: Creating a Directive with an Associated Controller
We’re going to have multiple pages that want to reuse this HTML snippet.

How do we eliminate this duplication?
Using ng-include for Templates

ng-include is expecting a variable with the name of the file to include. To pass the name directly as a string, use single quotes ('...').

```
<ul class="list-group">
  <li class="list-group-item" ng-repeat="product in store.products">
    <h3 ng-include='product-title.html'>name of file to include</h3>
    <section ng-controller="PanelController as panel">
      {{product.name}}
      <em class="pull-right">${{product.price}}</em>
    </section>
  </li>
</ul>
```

```
<h3 ng-include='product-title.html' class="ng-scope">
  <span class="ng-scope ng-binding">Awesome Multi-touch Keyboard</span>
  <em class="pull-right ng-scope ng-binding">$250.00</em>
</h3>
```
Flatlander Crafted Gems
– an Angular store –

Pentagonal Gem $5.95

Description

Origin of the Pentagonal Gem is unknown, hence its low value. It has a very high shine and 12 sides,
Web Server

URL Request to server

Response with Webpage & Assets

Fetches ng-included file

HTML Returned

Web Browser

HTML

JavaScript

Browser loads up Angular app.

HTML
Creating our First Custom Directive

Using ng-include...

```html
<h3 ng-include="'product-title.html'"></h3>
```

Custom Directive

```html
<product-title></product-title>
```

Our old code and our custom Directive will do the same thing... with some additional code.

Why write a Directive?
Why Directives?

Directives allow you to write HTML that expresses the behavior of your application.

```html
<aside class="col-sm-3">
  <book-cover></book-cover>
  <h4><book-rating></book-rating></h4>
</aside>

<div class="col-sm-9">
  <h3><book-title></book-title></h3>
  <book-authors></book-authors>
  <book-review-text></book-review-text>
  <book-genres></book-genres>
</div>
```

Can you tell what this does?
Writing Custom Directives

Template-expanding Directives are the simplest:

• define a custom tag or attribute that is expanded or replaced
• can include Controller logic, if needed

Directives can also be used for:

• Expressing complex UI
• Calling events and registering event handlers
• Reusing common components
How to Build Custom Directives

A configuration object defining how your directive will work
How to Build Custom Directives

```javascript
app.directive('productTitle', function(){
  return {
    restrict: 'E',
    templateUrl: 'product-title.html'
  };
});
```

Dash in HTML translates to ... camelCase in JavaScript

```html
<h3>{{product.name}}<em class="pull-right">$250.00</em></h3>
```
Origin of the Pentagonal Gem is unknown, hence its low value. It has a very high shine and 12 sides,
Attribute vs Element Directives

Element Directive

\[<\text{product-title}></\text{product-title}>\]

Notice we’re not using a self-closing tag... \[<\text{product-title}/>\]
...some browsers don’t like self-closing tags.

Attribute Directive

\[<\text{h3 product-title}></\text{h3}>\]

Use Element Directives for UI widgets and Attribute Directives for mixin behaviors... like a tooltip.
Defining an Attribute Directive

```html
<h3 product-title></h3>
```

```javascript
app.directive('productTitle', function(){
  return {
    restrict: 'A',
    templateUrl: 'product-title.html'
  };
});
```

```
<h3>
  {{product.name}}
  <em class="pull-right">$250.00</em>
</h3>
```

Though normally attributes would be for mixin behaviors ...
Directives allow you to write better HTML

When you think of a dynamic web application, do you think you’ll be able to understand the functionality just by looking at the HTML?

No, right?

When you're writing an Angular JS application, you should be able to understand the behavior and intent from just the HTML.

And you're likely using custom directives to write expressive HTML.
Shaping Up with Angular JS
Creating Our Own Directives
Reviewing our Directive

Template-Expanding Directives

An Attribute Directive

An Element Directive
What if we need a Controller?

```
<section ng-controller="PanelController as panels">
  <ul class="nav nav-pills"> ... </ul>
  <div class="panel ng-show="panels.isSelected(1)"> ... </div>
  <div class="panel ng-show="panels.isSelected(2)"> ... </div>
  <div class="panel ng-show="panels.isSelected(3)"> ... </div>
</section>
```
First, extract the template...

```html
<h3> <product-title> </h3>
<product-panels ng-controller="PanelController as panels">
  ...
</product-panels>
```

```html
<section>
  <ul class="nav nav-pills">
  ...
  </ul>
  <div class="panel" ng-show="panels.isSelected(1)">
    ...
  </div>
  <div class="panel" ng-show="panels.isSelected(2)">
    ...
  </div>
  <div class="panel" ng-show="panels.isSelected(3)">
    ...
  </div>
</section>
```
Now write the Directive ...

```
<product-panels ng-controller="PanelController as panels">
  ...
</product-panels>
```

```
app.directive('productPanels', function() {
  return {
    restrict: 'E',
    templateUrl: 'product-panels.html'
  };
});
```
What about the Controller?

First we need to move the functionality inside the directive
Moving the Controller Inside

```html
app.directive('productPanels', function() {  
  return {  
    restrict: 'E',  
    templateUrl: 'product-panels.html',  
    controller: function() {   
      ...
    }  
  }  
});
```

Next, move the alias inside

```html
<product-panels ng-controller="PanelController as panels" >
  ...
</product-panels>
```
Need to Specify the Alias

Now it works, using panels as our Controller Alias.
Description

Origin of the Pentagonal Gem is unknown, hence its low value. It has a very high shine and 12 sides,
Challenges
Shaping Up with Angular JS

Level 5: Dependencies and Services
Starting to get a bit cluttered?

Can we refactor these out?

(function(){
    var app = angular.module('store', []);

    app.controller('StoreController', function(){ ... });

    app.directive('productTitle', function(){ ... });
    app.directive('productGallery', function(){ ... });
    app.directive('productPanels', function(){ ... });

    ... }

})();

app.js
Extract into a new file ... products.js

```javascript
(function(){
  var app = angular.module('store', []);

  app.controller('StoreController', function(){ ... });

  app.directive('productTitle', function(){ ... });
  app.directive('productGallery', function(){ ... });
  app.directive('productPanels', function(){ ... });
})();
```
Make a new Module

```javascript
(function(){
    var app = angular.module('store', []);
    app.controller('StoreController', function(){
        ...
    })()
});
```

Define a new module just for Product stuff...

```javascript
(function(){
    var app = angular.module('store-products', []);
    app.directive('productTitle', function(){
        ...
    });
    app.directive('productGallery', function(){
        ...
    });
    app.directive('productPanels', function(){
        ...
    });
})()
```

Different closure means different app variable.
(function() {
    var app = angular.module('store', ['store-products']);
    app.controller('StoreController', function() {
        . . .
    });
})();

(app directive)

(app directive)

(app directive)

(app directive)

(add it to the dependencies)

store depends on store-products

Module Name

app.js

(products.js)

(function() {
    var app = angular.module('store-products');
    app.directive('productTitle', function() {
        . . .
    });
    app.directive('productGallery', function() {
        . . .
    });
    app.directive('productPanels', function() {
        . . .
    });
})(function() {
    var app = angular.module('store', ['store-products']);
    app.controller('StoreController', function() {
        . . .
    });
})();
We’ll also need to include the file

```html
<!DOCTYPE html>
<html ng-app="store">
  <head> . . . </head>
  <body ng-controller="StoreController as store">
    . . .
    <script src="angular.js"></script>
    <script src="app.js"></script>
    <script src="products.js"></script>
  </body>
</html>
```
Flatlander Crafted Gems
– an Angular store –

Gem #1: Zircon

$1,100.00

Description

Zircon is a rare and valuable gemstone, known for its sparkling appearance and durability. It is often used in jewelry and as a decorative stone. Zircon has a rich history, with ancient civilizations valuing its beauty and durability. This specific Zircon is of high quality, making it a perfect addition to any collection.

Specs

Reviews
How should I organize my application Modules?

Best to split Modules around functionality:

• `app.js` - top-level module attached via `ng-app`
• `products.js` - all the functionality for products and only products
Challenges
Does this feel strange?

What is all this data doing here?

Where can we put it?

Shouldn't we fetch this from an API?
How do we get that data?

```
(function(){
  var app = angular.module('store', [ 'store-products' ]);  

  app.controller('StoreController', function(){
    this.products = ???;  
    ...  
  });
})(

http://api.example.com/products.json

[
  { name: '...', price: 1.99, ... },
  { name: '...', price: 1.99, ... },
  { name: '...', price: 1.99, ... },
  ...
]  
```

How do we fetch our products from an API?
We need a Service!

Services give your Controller additional functionality, like ... 
- Fetching JSON data from a web service with $http
- Logging messages to the JavaScript console with $log
- Filtering an array with $filter

All built-in Services start with a $ sign ...
The $http Service is how we make an async request to a server ...

• By using $http as a function with an options object:

```javascript
$http({ method: 'GET', url: '/products.json' });
```

• Or using one of the shortcut methods:

```javascript
$http.get('/products.json', { apiKey: 'myApiKey' });
```

• Both return a Promise object with .success() and .error()

• If we tell $http to fetch JSON, the result will be automatically decoded into JavaScript objects and arrays

So how do we use it?
How does a Controller use a Service like $http?

Use this funky array syntax:

```javascript
app.controller('SomeController', [ '$http', function($http){
} ]);
```

Service name

Service name as an argument

Dependency Injection!

```javascript
app.controller('SomeController', [ '$http', '$log', function($http, $log){
} ]);
```

If you needed more than one
When Angular is Loaded Services are Registered

```
app.controller('SomeController', [ '$http', '$log', function($http, $log){
} ]);```

A Controller is Initialized

app.controller('SomeController', [ '$http', '$log', function($http, $log){
} ]);
Then When the Controller runs ...

```javascript
app.controller('SomeController', ['$http', '$log', function($http, $log){
}]);
```

Dependency Injection!

Here ya go!

Injectors:
- $http
- $log

Controller:
- SomeController
(function()
{
    var app = angular.module('store', [ 'store-products' ]); 

    app.controller('StoreController', function()
    {
        this.products = ???;
    });

})(());
Time for your injection!

(function(){
    var app = angular.module('store', [ 'store-products' ]); 

    app.controller('StoreController', [ '$http',function($http){
        this.products = ???;
    }]);
})();

StoreController needs the $http Service...

...so $http gets injected as an argument!

Now what?
Let’s use our Service!

```javascript
(function(){
    var app = angular.module('store', [ 'store-products' ]); 
    app.controller('StoreController', [ '$http', function($http){
        this.products = ???;

        $http.get('/products.json')
    }]);
})();
```
Our Service will Return Data

```javascript
(function(){
    var app = angular.module('store', [ 'store-products' ]);;

    app.controller('StoreController', [ '$http',function($http){
        this.products = ???;

        $http.get('/products.json').success(function(data){
            ??? = data;
        });
    }]);
})();
```

What do we assign `data` to, though...?

$http` returns a Promise, so `success()` gets the data...
Storing the Data for use in our Page

```
(function(){
    var app = angular.module('store', [ 'store-products' ]); 

    app.controller('StoreController', [ '$http', function($http) {
        var store = this;

        $http.get('/products.json').success(function(data){
            store.products = data;
        });
    }])
})();
```

We need to store what this is ... ... and now we have somewhere to put our data!

But the page might look funny until the data loads.
Initialize Products to be a Blank Array

(function(){
    var app = angular.module('store', [ 'store-products' ]); 

    app.controller('StoreController', [ '$http', function($http){
        var store = this;
        store.products = [];  

        $http.get('/products.json').success(function(data){
            store.products = data;
        });
    }]);
})();

We need to initialize products to an empty array, since the page will render before our data returns from our get request.
Flatlander Crafted Gems
– an Angular store –

Gem #1: Zircon

$1,100.00

Description

Zircon, in a cut to make it look like a flower.
Additional $http functionality

In addition to `get()` requests, $http can post(), put(), delete()...

```javascript
$http.post('/path/to/resource.json', { param: 'value' });
```

```javascript
$http.delete('/path/to/resource.json');
```

...or any other HTTP method by using a config object:

```javascript
$http({ method: 'OPTIONS', url: '/path/to/resource.json' });
```

```javascript
$http({ method: 'PATCH', url: '/path/to/resource.json' });
```

```javascript
$http({ method: 'TRACE', url: '/path/to/resource.json' });
```
Challenges
SHAPING UP WITH ANGULAR.JS