## 1.2 Creating a Store Module

200 PTS

**The Flatlanders need a store** to sell their gems and more! They need it really quick, Angular will do the trick!

They have figured out how to manipulate time and space, allowing them to create three-dimensional gems. The buying and selling of their gems has become a popular Flatlander practice and they believe the next step is taking their wonderful wares to the fourth dimension (the web).

*Can you help them reach their online peddling goals?*

Help Me

### Task 1/3

Create a Module named `gemStore` so we can get started on this marketing journey.

app.js | index.html

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

### Task 2/3

Attach the `gemStore` module to our HTML page with a Directive.

app.js | index.html

```
<!DOCTYPE html>
<html ng-app="gemStore">
  <head>
    <title>Gem Store</title>
```

### Task 3/3

In `index.html`, create a simple Expression to display a friendly "Hello, Angular!" message.

```
  <body>
    <h1>{{"Hello, Angular!"}}</h1>
  </body>
</html>
```

## 1.3 Index HTML Setup

## Working With Data

## 1.4 Our First Controller

200 PTS

In order to add some behavior to our application, we need a controller. Add a controller named `StoreController` to our `gemStore` application.

Help Me

### Task 1/5

Add a controller named `StoreController` to our `gemStore` application.

```
(function(){
  var gem = { name: 'Aaurite', price: 2.95 };
      app = angular.module('gemStore', []);
  app.controller('StoreController',function(){

  });
})();
```

### Task 2/5

Attach the `StoreController` to the `<body>` tag. Be sure to alias it as `store`.

app.js | index.html

```
<!DOCTYPE html>
<html ng-app='gemStore'>
  <head>...</head>
  <body ng-controller="StoreController as store">
    <div class='product row'>
      <h3>
```

### Task 3/5

In `app.js`, we've added a `gem` object to represent one of the products in our `gemStore`. Assign it to the `product` property of `StoreController` so we can use them in the page.

```
      app = angular.module('gemStore', []);
app.controller('StoreController',function(){
  this.product = gem;
});
})();
```

### Task 4/5

Display the `name` of the product inside the `<h3>` tag.

```
<body ng-controller="StoreController as
  <div class="product row">
    <h3>
      {{store.product.name}}
      <em class="pull-right"></em>
    </h3>
```

### Task 5/5

Display the `price` of the product inside the `<em>` tag.

app.js | index.html

```
<!DOCTYPE html>
<html ng-app='gemStore'>
  <head>...</head>
  <body ng-controller="StoreController as store">
    <div class='product row'>
      <h3>
        {{store.product.name}}
        <em class='pull-right'>${{store.product.price}}</em>
      </h3>
    </div>
  </body>
</html>
```

## 1.6 Not For Sale

200 PTS

We've added two new properties to our `product` that we can use on the HTML side. The first of these two is `canPurchase`, which is a `boolean` indicating if the product can be purchased. The second is `soldOut` which, as you can imagine, is a `boolean` indicating if the product is sold out.

Use these two new properties in our HTML page to solve the following objectives.

Help Me

### Task 1/2

Use a directive to ensure that we can only see the "Add to Cart" button if the `canPurchase` property is true.

```
    </h3>
    <button ng-show =" store.product.canPurchase">Add to Cart</button
  </div>
</body>
```

### Task 2/2

Our first `gem` is so popular that we've run out of stock already! Well, Flatlander gems are pretty rare, so it shouldn't be a big surprise. Luckily there is a `soldOut` property to our `gem`. When a gem is `soldOut`, hide the `.product` element.

```
<head>...</head>
<body class='container' ng-controller='StoreController as store'>
  <div ng-hide="store.product.soldOut" class="product row">
    <h3>
```

## 1.7 Look, More Gems!

200 PTS

Looks like the Flatlanders have discovered *more* `gems` for us to sell in the `gemStore`. That's a relief! Follow the objectives below to add these new gems to the store.

Help Me

### Task 1/2

In the `app.js` file we changed things up a little with a new `gems` array. Assign gems to a `products` property inside `StoreController`.

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

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

  var gems = [
    { name: 'Aaurite', price: 2.95 },
```

### Task 2/2

You know how to display all the `products`, don't you? Use the correct directive to display all the products in `product row` divs.

```
<!DOCTYPE html>
<html ng-app='gemStore'>
  <head>...</head>
  <body class='container' ng-controller='StoreController as store'>
    <div class="product row" ng-repeat = "product in store.products">
      <h3>
        {{product.name}}
        <em class='pull-right'>${{product.price}}</em>
      </h3>
    </div>
  </body>
</html>
```