Controllers Lab

At this point you have some good code written but it may not be organized in the best way. We should implement the best practices we learned in class.

Creating the project structure

A large project will have multiple modules and a shared module. Might as well prepare ourselves.

1. Review the project directory structure. Make sure you have a *main* folder and a *shared* folder under "app".

Splitting the module and controllers

2. You've currently got a file called "mainModule.js" with your module creation and at least two controllers. Split that into as many other files as you'll need. Here's a suggestion:
   - mainModule.js
   - mainProductController.js
   - aboutUsController.js
3. Make sure mainModule only contains the definition/creation of the module. It has no dependencies at this point.

   mainProductController.js should have nothing but the controller in there. Same with aboutUsController.js.

4. You'll need to add references to these new files in your HTML files. Go ahead and do that.
5. Run and test everything. If it still works, you can proceed.

As you work with these files throughout these labs, make sure you're putting the right things in the right files. Just remember the Single Responsibility Principle--each file should contain only one thing (one controller, one module, etc).

Encapsulating the functions

If we're not careful we could end up leaking objects onto the global namespace and get ourselves in trouble as the project grows. Let's make 100% sure that can't happen with IIFEs.

6. Edit each of your module/controller files and wrap each function in an IIFE.
7. Make sure all variables are declared with the *var* keyword, otherwise our IIFE won't do us any good.
8. Once again, run and test. Make sure we haven't broken anything.

Naming the functions

This step helps with clarity of code and also in debugging the call stack. Currently we're passing anonymous functions into the module.controller() function as the second argument. Let's give them names.

9. Pull each function out of the controller() parameter list and give each a name like so:

```
function controllerName() { /* All your logic goes here */ }
```

10. Then put the controllerName as the second argument in your controller function parameter list.
11. Run and test again. Still working? Good!

Protecting from minification renaming

As we grow we're going to want to start minifying our code but if we do, our controller dependencies are going to fail. We'll fix that with explicit dependency declarations.

12. In both of the controllers, add an array as the 2nd argument between the two existing ones. Sort of like this:

```
angular.module('moduleName')
.controller('controllerName', ['$scope', 'otherDep', ctrlFuncName]);
```

13. Of course if you don't have a second dependency, you won't have 'otherDep' in the example above. So you may not need that second argument.
14. You know the drill by now. Run and test.

Adding a dependency

Just to get practice, we're going to add a dependency. You won't really need $http, but let's see how we would implement that dependency.

15. Open mainProductController.js. Add $http as an argument in the function declaration.
16. Now you'll have to match that in the explicit dependency array. Look at the array and add $http there as well. Make sure it is a literal string by putting it in quotes. And double-check that it is in the right place.
17. Run and test. Make sure everything still works after your refactor.