Services Lab

There are some activities which we may need on multiple pages and even in multiple sections of the site. Whenever we want to share activities, we should probably make a service out of them and put them under a shared folder.

1. If you don't already have a shared directory, create it.
2. Add a sharedModule.js and create the *sharedModule* ... uh ... module.

Creating a category service

3. Create a new file under *shared* called categoryService.js. In it create a service that is part of the shared module.
4. The service should expose one function called getAllCategories()
5. getAllCategories() should return an array of category objects in JSON format. Feel free to copy those categories from your productListController.

Using and running a service

6. To use the service you must include it. Open productList.html and add a link to this new script as well as the shared module.
7. Open your productModule.js file and add sharedModule as a dependency.
8. You'll also need to include it in the controller. Open the productListController.js and add categoryService as a dependency. Remember, it will need to be passed into the function as a parameter. You'll probably going to want to include it in the dependency array as a string to protect from minification.
9. Still in productListController, change the hardcoded category array to call your new service. Something like this might do the trick:

```
$scope.categories = categoryService.getAllCategories();
```

10. Run and test. Make sure your list of categories still populates like before.

Now your controller is much cleaner and the list of categories is available to other modules and controllers. It only needs to be changed in one place ... the service.

The productService

Looks like we have some more low-hanging fruit. Our product list is also repeated. Let's re-do the above steps with the product lists.

11. You can re-use the sharedModule. But create a new service called productService. Add all the trappings that will make this a service. You should know what to do by now. Go for it.
12. Go ahead and create these methods in productService:
    - getAllProducts() - Returns an array of all products

- getFeaturedProducts() - Returns all products for now. We'll change this in later labs
- getProduct(productID) - Returns one product. You'll want to write it so that it searches through your array of products and only returns the one whose productID matches the one passed in.

13. Once these are all created, they can be used in productListController, in productSearchController, and on the main page, index.html.
14. Find everywhere you're hardcoding a list of products in controllers. Change those to call getAllProducts(). Except in the mainProductController. Make that one call getFeaturedProducts() instead.
15. Run and test. Make sure that your products appear as expected on all pages with products.

Once your product search page, product browse page, and main page are working properly you can be finished.

Bonus! Let's do one more

If you have time, there's another opportunity for improvement. Notice that in the shipToController, you're still using a hardcoded local customer object. Change it to use a shared customerService(customerID) that will return a customer object given a customerID.