CS121 Tutorial 5
Intro to testing in Xcode!

Purple bubbles give you information you’ll need to know.

Yellow Bubbles tell you what to do.

Orange bubbles tell you what you’re not expected to understand yet. 😊
1. Create a new iOS single view app for the iPad.

2. Call it tutorial5.

3. Check ARC and unit tests.

Xcode will set up the unit testing framework for you. But in this tutorial you’ll learn how to add unit tests to an existing project that isn’t set up for the tests.
Add a new Model class to the object. (It should use NSObject as its base class.)
Define the interface as shown.

```c
#import <Foundation/Foundation.h>

@interface Model : NSObject

@end
```

```c
- (void) setValue: (int) value atIndex: (int) index;
- (int) getValueAtIndex: (int) index;
@end
```
Implement the class as shown. Then run to make sure there are no problems. Then stop the app.
1. Open the product tab.

2. Select test.

A build succeeded message will flash briefly. After that a test failed message will flash.
1. Click here to open the log navigator.

2. Select Test unitTests.

This message tells us one of our tests failed.

You might be thinking that you didn’t write any tests. Go to the next slide to see what happened.
1. Open the project navigator.

Xcode provided us with template code including a test that fails!

2. Open the unitTests source file.

Next we’ll write some real tests that should pass.

3. Delete this method.
1. Open the `unitTests` header file.

2. Import our model header file.

3. We’ll need a model object to test.

4. We’ll create one test.
1. Open the unitTests source file.

2. Create our Model object to test.

When we choose to Test, Xcode runs our setUp method, then all of the tests we’ve created, and finally the tearDown method. In this example, our setUp method creates the Model object that we will test.
Add the `testSetAndGet` method.

I want to set the value of the data fields to something other than 0. I also want each data field to be unique.

Note that this test does not check that `data[i]` is unaffected by a change to `data[i+1]`. Devise another test to address that issue. (You might want to change the name of our current test to be more descriptive and consistent with the new test.)
Run the new tests then open the log. The test should pass!
Next we'll create a test that checks whether invalid input is handled properly in the `get` method.

Add this test to our `unitTests` interface.
Implement our new test.
Run the new tests then open the log. Our new test failed!
1. Open our Model source file.

2. Add an assertion to catch an out of range index in the getValue method.
Run the tests again then open the log. Now our new test succeeded!
Update the tests as follows:
1. `setUp` should fail if `testModel` is not allocated (i.e. it gets a nil pointer). Check out the link below for `STAssert` options that will be useful.
2. Add another test to check that the `setValue` method
   a. Checks that the index is valid
   b. Checks that the value is non-negative
3. Fix our Model implementation as needed so that all tests pass.

Appendix B at the following link gives the `STAssert` macros that are available.