CSCA08 Exercise 8

Due: November 24, 2013. 5:00pm

In our final exercise of the term, we're going to be practicing writing UnitTests. The structure of the tests isn't really as important as the testing plan, but we get to test both.

What to Test

We wanted to come up with a function for you to test that had a bit of everything: loops, selection, lists, strings, dictionaries, etc. But we had a bit of a conundrum. We wanted something complicated enough to be interesting, but knew you were busy with your assignment, and didn't want everyone to have to spend lots of time writing a new function in order to test it.

That's when it hit us. You're already writing a function that you'll probably want to test anyway... `cartesian_product`! It's got everything we need, you already have to write it, and now it can help you test your own code for A2. (A pretty smart move if I do say so myself).

How to Start

Before you start writing any code, you should think about **coverage testing**, and how we came up with a test plan in lecture. Figure out all of the parameters, and the important ranges they fall into. Then write one test for each possible combination of ranges. The goal here is to find one example test case for all possible regions of your testing space.

What to Do

In a file called `ex8.py`, you should write a UnitTest to thoroughly test `cartesian_product`. Your tests will actually be run on a version of the code that I have written. Therefore, you will be doing `black box` testing (you don't know if I implemented the function in the same way you will... in fact, you can bet that I probably won't). My cartesian product function will be in a file called `squeal.py` and placed into the same directory as your UnitTest.

What to Submit

Submit your `ex8.py` file to MarkUs as usual. Your UnitTest methods do not need any DocStrings, and unless you're doing something particularly unusual, you probably don't need any internal comments either. However, your method names and error messages should be descriptive enough to properly explain what each test case does and why it's useful. Remember that writing frivolous test cases is no better than missing useful ones.