CSCA08 Exercise 10

Due: December 3, 2017. 5:00pm
Pre-Run: December 1, 2017. 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 something for you to test that would be 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/method in order to test it.

That's when it hit us. You're already writing code that you'll probably want to test anyway... so now you can practice UnitTests, and 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.

We will only be testing `set_marker`, `set_by_marker`, `get_by_marker`, `set_by_pos` and `get_by_pos` in the `Female` class. So you shouldn't assume that any other classes/methods will be available to you, but you should test these methods as thoroughly as possible.

What to Do

In a file called `ex10.py`, you should write a UnitTest to thoroughly test the methods. Your tests will actually be run on a version of the code that we have written. Therefore, you will be doing `black box` testing (you don't know if we implemented the classes in the same way you will... in fact, you can bet that we probably won't). Our code will be in a file called `a2.py` which will be placed into the same directory as your UnitTest. We've provided you with some starter code, just to make sure you can access everything correctly, and even a sample test case to help you on your way.

What to Submit

Submit your `ex10.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.