HW: Unit Testing

Homework Objective

- Write effective unit tests.
- Use tests to verify that new code works as specified.

Deliverables and Submission Process

For the main assignment, modify the **unittest_cards.py** to test cards.py and submit it to Gradescope.

Background

To complete this assignment, you will need to familiarize yourself with the Card class covered in the lecture and the discussion. You will also want to review the Lecture Notes on Unit Testing in Python.

Then you will include tests for the cases described below. There are a few notes though:

- You may create as many or a few unit test methods as you like.
- You may assume that other programmers will **NOT** invoke these functions with unacceptable inputs (e.g. no one will try to create a card with rank 0). You just need to ensure that the code works as intended.

Tests

Note: Each test case will be written in a different method. (You need to write 8 methods in total.) All test cases for the main(not extra credit) part are visible.

1. Test that if you create a card with rank 12, its `rank_name` will be "Queen"
2. Test that if you create a card instance with suit 1, its `suit_name` will be "Clubs"
3. Test that if you invoke the str method of a card instance that is created with `suit=3, rank=13`, it returns the string "King of Spades"
4. Test that if you create a deck instance, it will have 52 cards in its cards instance variable
5. Test that if you invoke the `deal_card` method on a deck, it will return a card instance.
6. Test that if you invoke the `deal_card` method on a deck, the deck has one fewer card in it afterward.
7. Test that if you invoke the `replace_card` method, the deck has one more card in it afterward. (Please note that you want to use the deal_card function first to remove a card from the deck and then add the same card back in)
8. Test that if you invoke the `replace_card` method with a card already in the deck, the deck size is not affected. (The function must silently ignore it if you try to add a card that's already in the deck)

Extra Credit 1 (5 points)

Tasks

In this part, you will implement the class **`Hand`** and create tests to verify that it works as specified. You might want to import Card and Deck from cards.py in your implementation. You must create a new testing class

called **`TestHand`** that subclasses **`unittest.TestCase`**, and implement at least 3 test functions. Note that, the name of the test methods should start with test_. (For the code, you can refer to the starter file **hand_starter.py**.)

1. Test that a hand is initialized properly.
2. Test that `add_card()` and `remove_card()` behave as specified (you can write one test for this, called `testAddAndRemove`.
3. Test that `draw()` works as specified. Be sure to test side effects as well.

Notice

- These tests are less specified than those in part 1, so you will have to think about writing good tests for each of these functions.
- You do not need to worry about testing for invalid inputs. For example, you can assume that Hand will be initialized with a valid list of valid Cards, and that `draw()` will be called with a valid non-empty Deck.
- **Hint:** there is a magic method defined in cards.py that is very useful.

What to turn in

Submit **hand.py** and **unittest_hand.py** on Gradescope. Note you do not need to submit cards.py.

Extra Credit 2 (5 points)

Tasks

Implement two new pieces of functionality:

1. Add a function **"`remove_pairs`" (takes no parameter)** to Hand that looks for **all** the pairs of cards in a hand and removes **all** of them. Note that if there are three of a kind, only two should be removed (it doesn't matter which two). If there are four cards of the same rank, remove all of them. If there are multiple pairs in different ranks, such as two Queen and two Ace, you need to remove all of them. Write a docstring and tests to verify that the function works as specified.

2. Create a **copy of cards.py** and rename it to **ec_cards.py(do not change anything if not needed)**. Copy Hand class implementation to **ec_cards.py** without the import card and deck, since these implementations are already in **ec_cards.py**.

   Add a function "**`deal(numHand, numCard)`**" to **Deck** class that takes **two parameters** representing the **number of hands(int)** and the **number of cards per hand(int)**. It should return **`numHand`** hand instances in a **list** and each hand should have **`numCard`** cards instances in its **cards attribute**. For example, if **`numHand`** is 4, return a list with four hands instance in it. If the number of cards per hand is set to -1, **all** of the cards in the deck should be dealt, even if this results in an uneven number of cards per hand. If there is an insufficient amount of cards left in deck to generate **`numHand`** hand with **`numCard`** cards in each hand, distribute all the cards like the previous situation.

Notice

- Write a docstring and tests to verify that the function works as specified.

- Each hand will **receive cards in turn**, rather than getting all the consecutive cards of the correct amount.

What to turn in

Submit **ec_cards.py** and **hand.py** to Gradescope.