**HW 4: Unit Testing & GitHub**

**Homework Objective:**

Demonstrate the ability to:

* Understanding basic commands of GitHub
* Write effective unit tests
* Use tests to verify that new code works as specified

**Part 1: GitHub practice (20 points)**

**GitHub practice: Create a repository and push a file**

If you haven’t setup Git and Unix environment on your laptop, see [Setting up Git](https://umich.instructure.com/courses/343048/discussion_topics/912349) (professor’s announcement on Canvas).

You will also want to review the [Lecture Notes on GitHub](https://paper.dropbox.com/doc/Lecture-9-Command-Line-Git-and-GitHub--At2NKWt~69V4HHc0Hkyhc7_IAg-K5GQ2i2oK1LxS4Ff54Sc8).

1. Create a directory hw4\_git\_practice on your laptop using command line.

* (Mac: Terminal, Windows: GitBash)

1. Move to the directory

* Hint: cd <dir name>

1. Create a file hw4\_git.py and save it in the directory you created. It must be a valid Python program, but other than that can be anything you want.
2. Edit (add some code, such as print("Hello World")) and save hw4\_git.py with your editor.
3. Create a local repository using command line.

* Hint: git init, git status

1. Tell git to track  the file.

* Hint: git add <file name>, git status

1. Commit the file.

* Hint: git commit -m "Adding hw4\_git", git status

1. Create a new **public** repository hw4-git-practice on GitHub (<https://github.com/new>)

* (Make sure to create a public repo, otherwise graders can’t access it. Also remember not to allow GitHub to add any other files such as .gitignore or README)

1. Link your local repository to GitHub

* Hint: git remote add origin + **whatever your link is (GitHub will tell you on this screen!)**

1. *push* your code to the GitHub repository

* Hint: git push -u origin master

1. Reload the GitHub page and confirm your **first** modification was *pushed* to GitHub
2. Modify your Python program (hw4\_git.py). It should still be a valid Python program but you can make any  modification you want (including just adding a comment). Do the necessary steps to ensure that the change you made is pushed to GitHub.
3. Reload the GitHub page and confirm your **second** modification was *push* to GitHub

* Hint: See your commit message, timestamp, # commits, and history

1. Record the GitHub URL to submit on Canvas

**Part 2: Unit Testing (80 points)**

**Deliverables and Submission Process:**

For the main assignment, modify the test file **hw4\_test.py** to test **hw4\_cards.py** and submit to Github.  The code must be executable! For the Extra Credit problems, submission details are included with the instructions.

**Background:**

In order 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 few unittest 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.

**Instructions:**

**Main assignment: Basic Unit Testing (80 points)**

***Preparation***

* Go to <https://classroom.github.com/a/hYazo3yK>
* Click “Accept this assignment”
* Create a clone of the assignment on your laptop
* Hint: cd, git clone XXXXX
* Check whether you have **hw4\_test.py** and **hw4\_cards.py** on your laptop.
* Start modifying **hw4\_test.py** to test the following eight points.

***UnitTesting***

Note: Each test case will be written in a different method. **(You need to write 8 methods in total.)**

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 cards in it afterwards.
7. Test that if you invoke the replace\_card method, the deck has one more card in it afterwards. (Please note that you want to use 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 that is 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)

***What to turn in:***

Add your name and uniqname to the head of **hw4\_test.py.** This is an example.

#########################################

##### Name: <write your name>       #####

##### Uniqname:<write your uniqname>#####

#########################################

push your modification on **hw4\_test.py** to GitHub.

Hint: Do not forget to add and commit before push.

After that, **submit the following two links to Canvas:**

* Part 1’s GitHub practice repository link
* Part 2’s GitHub repository link

Do not forget to answer the [form about your GitHub account](https://forms.gle/3hgsuFgJHwE5ir5p9).

**Extra Credit 1: Writing your own class/tests (2 points)**

In this part, you will implement the class Hand and create tests to verify that it works as specified. You will need to create a new testing class called TestHand that subclasses unittest.TestCase, and implement 3 test functions.

# create the Hand with an initial set of cards

class Hand:

    '''a hand for playing card

    Class Attributes

    ----------------

    None

    Instance Attributes

    -------------------

    init\_card: list

        a list of cards

    '''

    def \_\_init\_\_(self, init\_cards):

        pass

    def add\_card(self, card):

        '''add a card

        add a card to the hand

        silently fails if the card is already in the hand

        Parameters

        -------------------

        card: instance

            a card to add

        Returns

        -------

        None

        '''

        pass

    def remove\_card(self, card):

        '''remove a card from the hand

        Parameters

        -------------------

        card: instance

            a card to remove

        Returns

        -------

        the card, or None if the card was not in the Hand

        '''

        pass

    def draw(self, deck):

        '''draw a card

        draw a card from a deck and add it to the hand

        side effect: the deck will be depleted by one card

        Parameters

        -------------------

        deck: instance

            a deck from which to draw

        Returns

        -------

        None

        '''

        pass

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.

1. Test that a hand is initialized properly.

1. Test that add\_card( ) and remove\_card( ) behave as specified (you can write one test for this, called testAddAndRemove.

1. Test that draw( ) works as specified. Be sure to test side effects as well.

***What to turn in:***

Create a *new* file called **hw4\_cards\_ec1.py** and **hw4\_test\_ec1.py**, and push to GitHub. This needs to be named separately from the **hw4\_cards.py** you turn in for Parts 1, even though it will be based on it.

**Extra Credit 2 (2 points):**

Implement two new pieces of functionality:

1. Add a function “remove\_pairs” to Hand that looks for pairs of cards in a hand and removes them. Note that if there are three of a kind, only two should be removed (it doesn’t matter which two). Write a docstring and tests to verify that the function works as specified.
2. Add a function “deal” to Deck that takes two parameters representing the number of hands and the number of cards per hand and returns a list of Hands. If the number of cards per hand is set to -1, *all* of the cards should be dealt, even if this results in an uneven number of cards per hand. Write a docstring and tests to verify that the function works as specified.

***What to turn in:***

Create a *new* file called  **hw4\_cards\_ec2.py** and **hw4\_test\_ec2.py,** and push to GitHub. This needs to be named separately from the **hw4\_cards.py** you turn in for Parts 1, even though it will be based on it.

**Grading**

**Part 1: GitHub practice**

We are not providing sample output, so you are encouraged to exercise reasonable judgment in following the instructions above to meet the requirements listed here.

|  |  |  |  |
| --- | --- | --- | --- |
| Req | Description | Category | Point Value |
| 1 | GitHub repository URL is accessible (created as pubic). | Behavior | 5 |
| 2 | GitHub repository has a file hw4\_git.py (file name can be different). | Behavior | 10 |
| 3 | commit at least twice | Behavior | 5 |
|  | **Total** |  | 20 |

**Part 2: UnitTesting**

We are not providing sample output, so you are encouraged to exercise reasonable judgment in following the instructions above to meet the requirements listed here.

|  |  |  |  |
| --- | --- | --- | --- |
| Req | Description | Category | Point Value |
| 1 | Successfully used the GitHub repo that was created when you accepted the  invitation (cloned, modified, pushed) | Behavior | 4 |
| 2 | hw4\_test.py is updated (committed and pushed). | Behavior | 4 |
| 3 | hw4\_test.py runs without syntax error. | Behavior | 4 |
| 4 | Eight test methods are made. | Code | 4 |
| 5 | Test 1 creates an instance of Card. | Code | 4 |
| 6 | Test 1 correctly compare the values. | Code | 4 |
| 7 | Test 2 creates an instance of Card. | Code | 4 |
| 8 | Test 2 correctly compare the values. | Code | 4 |
| 9 | Test 3 creates an instance of Card. | Code | 4 |
| 10 | Test 3 correctly compare the values. | Code | 4 |
| 11 | Test 4 creates an instance of Deck. | Code | 4 |
| 12 | Test 4 correctly compare the values. | Code | 4 |
| 13 | Test 5 invokes deal\_card and receive an instance. | Code | 4 |
| 14 | Test 5 correctly compare the types. | Code | 4 |
| 15 | Test 6 invokes deal\_card. | Code | 4 |
| 16 | Test 6 correctly compare before and after. | Code | 4 |
| 17 | Test 7 invokes deal\_card. | Code | 4 |
| 18 | Test 7 correctly compare before and after. | Code | 4 |
| 19 | Test 8 obtains an existing card from Deck. | Code | 4 |
| 20 | Test 8 correctly compare before and after. | Code | 4 |
|  | **Total** |  | 80 |

**Extra Credit #1**

We are not providing sample output, so you are encouraged to exercise reasonable judgment in following the instructions above to meet the requirements listed here.

|  |  |  |  |
| --- | --- | --- | --- |
| Req | Description | Category | Point Value |
| 1 | Implement test cases that test add\_card, remove\_card, and draw | Code | 1 |
| 2 | add\_card, remove\_card, and draw work as specified | Behavior | 1 |
|  | **Total** |  | **2** |

**Extra Credit #2**

We are not providing sample output, so you are encouraged to exercise reasonable judgment in following the instructions above to meet the requirements listed here.

|  |  |  |  |
| --- | --- | --- | --- |
| Req | Description | Category | Point Value |
| 1 | Implement test cases to test remove\_pairs and deal | Code | 1 |
| 2 | remove\_pairs  and deal work as specified | Behavior | 1 |
|  | **Total** |  | **2** |