Python Review

CSCI 112, Lab 1

File names: Names of files, functions, and variables, when specified, must be EXACTLY as specified. This includes simple mistakes such as capitalization.

Individual work: All work must be your own. Do not share code with anyone other than the instructor and teaching assistants. This includes looking over shoulders at screens with the code open. You may discuss ideas, algorithms, approaches, *etc.* with other students but NEVER actual code. Do not use code written by anyone else, in the class or from the internet.

Documentation: Each file should begin with a docstring that includes your name, the class number and name, the lab number, and a short description of the lab, as well as documentation pertinent to that particular file.

Hand in: Write a single module (file) called `lab01.py`, with a solution for each of the following problems. Also write a unit test file called `lab01_test.py` that tests functionality for each of your solutions to each problem. Put the solution and unit test file into a single folder called `csci112lab01`<*yourname*> and zip this folder. Submit the zipped solutions to canvas before the due date.

The Problems:

1. Create a function to rotate a two-dimensional matrix of N * N integer elements num times, where if num is positive, the rotation is clockwise, and if not, counterclockwise.

   Examples

```
>>> rotate_transform([
  [2, 4],
  [0, 0]], 1)
[
  [0, 2],
  [0, 4]
]
>>> rotate_transform([
  [2, 4],
  [0, 0]], -1)
[
  [4, 0],
  [2, 0]
]
```

2. Create a function called `count_mines` that takes a list representation of a Minesweeper board, and returns another board where the value of each cell is the amount of its neighbouring mines.

   Examples: The input may look like this:

```
[
  [0, 1, 0, 0],
  [0, 0, 1, 0],
  [0, 1, 0, 1],
  [1, 1, 0, 0]
]
```

   The 0 represents an empty space . The 1 represents a mine.

   You will have to replace each mine with a 9 and each empty space with the number of adjacent mines, the output will look like this:

```
[
  [1, 9, 2, 1],
  [2, 3, 9, 2],
  [3, 9, 4, 9],
  [9, 9, 3, 1]
]
```

3. Create a function that takes in a nested list and an element and returns the frequency of that element by nested level.

   Examples:

```
>>> freq_count([1, 4, 4, [1, 1, [1, 2, 1, 1]]], 1)
[[0, 1], [1, 2], [2, 3]]
# The list has one 1 at level 0, 2 1's at level 1, and 3 1's at level 2.

>>> freq_count([1, 5, 5, [5, [1, 2, 1, 1], 5, 5], 5, [5]], 5)
[[0, 3], [1, 4], [2, 0]]

>>> freq_count([1, [2], 1, [[2]], 1, [[[2]]], 1, [[[[2]]]]], 2)
[[0, 0], [1, 1], [2, 1], [3, 1], [4, 1]]
```

4. Use and extend the textbook's implementation of a `Fraction` class to help you build a function to find Farey sequences. The Farey sequence of order n is the set of all fractions with a denominator between 1 and n, reduced and returned in ascending order. Given n, return the Farey sequence as a list, with each fraction being represented by a string in the form "numerator/denominator".

   Examples

```
>>> farey(1)
["0/1", "1/1"]

>>> farey(4)
["0/1", "1/4", "1/3", "1/2", "2/3", "3/4", "1/1"]

>>> farey(5)
["0/1", "1/5", "1/4", "1/3", "2/5", "1/2", "3/5", "2/3", "3/4", "4/5", "1/1"]
```

5. Write a function to determine the best Poker combination that is present in a hand of five cards. Every card is a string containing the card value (with the upper-case initial for face-cards and the lower-case initial for suits), as in the examples below:

   "Ah" → Ace of hearts
   "Ks" → King of spades
   "3d" → Three of diamonds
   "Qc" → Queen of clubs

   There are 10 different combinations. Here's the list, in decreasing order of importance:

Name	Description
Royal Flush	A, K, Q, J, 10, all with the same suit.
Straight Flush	Five cards in sequence, all with the same suit.
Four of a Kind	Four cards of the same rank.
Full House	Three of a Kind with a Pair.
Flush	Any five cards of the same suit, not in sequence.
Straight	Five cards in a sequence, but not of the same suit.
Three of a Kind	Three cards of the same rank.
Two Pair	Two different Pair.
Pair	Two cards of the same rank.
High Card	No other valid combination.

   For purposes of rank, Aces are considered both high and low, that is either 1 or 14, whichever gives the best hand.

Given a list hand containing five strings being the cards, implement a function that returns a string with the name of the highest combination obtained, accordingly to the table above.

Examples:

```
>>> poker_hand_ranking(["10h", "Jh", "Qh", "Ah", "Kh"])
"Royal Flush"

>>> poker_hand_ranking(["3h", "5h", "Qs", "9h", "Ad"])
"High Card"

>>> poker_hand_ranking(["10s", "10c", "8d", "10d", "10h"])
"Four of a Kind"
```

Implement a `Card` class, with a reasonable representation for cards, to help solve this problem. Justify your choice of representation and the memeber functions in the comments, with particular reference to how they help you with the Poker hand problem.

Optional: Check for legal hands, so that, for example, Yosemite Sam's "five aces" and Bugs Bunny's "six aces" would each raise an exception.