Object-Oriented Programming
50:198:113 (Fall 2021)

Homework:	**5**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**11/24/21**	**E-mail:**	`suneeta.ramaswami@rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://crab.rutgers.edu/~rsuneeta`
		Phone:	**(856)-225-6439**

Homework Assignment 5

The assignment is due by 11:59PM of the due date. The point value is indicated in square braces next to the problem. Each solution must be the student's own work. Assistance should only be sought or accepted from the course instructor or the TA. Any violation of this rule will be dealt with harshly.

In this assignment, you are asked to implement a collection of classes to simulate a Poker game; you will need to use iterators when implementing some of these classes. As usual, you are graded not only on the correctness of the code, but also on clarity and readability. I will deduct points for not following the guidelines for class design, poor indentation, poor choice of object names, and lack of documentation. All modules, classes and methods should have appropriate docstring documentation.

Prior to starting your work, download the homework document (`hw5.pdf`) and the file (`poker.py`).

Important notes:

- When implementing the module, it is important that you name all classes and methods exactly as described because I will assume you are doing so when testing your programs. If your program produces errors because the functions do not satisfy the stated prototype, points will be deducted.
- **Download the `poker.py` file from Canvas.** You will make modifications to this file to complete the implementation for this problem.

Problem 1 [75 points] Cards, Decks, and Poker. In this problem, you are asked to implement some classes to play a simple game of *Poker*. For a list of poker hands and their ranking, see `http://en.wikipedia.org/wiki/List_of_poker_hands`. Some details are given below as well.

Our Poker game will be rather simple: We randomly shuffle a deck of cards, deal the top 5 cards of the deck, and then inform the player of the best Poker rank in the dealt hand. To do this, we will create a `Card` class for a standard playing card, a `CardDeck` class for a deck of 52 cards, and a `PokerHand` class for a standard 5-card hand of Poker. Further details of the classes are provided below.

`Card` **class (10 points)** An instance of this class is a standard playing card, which has a suit and a face value. Note that the `Card` class has two *class attributes* called `allsuits` and `allfaces`. The former is the string `'CDHS'` with each character representing a *suit* of a card (`C`lubs, `D`iamonds, `H`earts, `S`pade). The latter is the string `'23456789TJQKA'` with each character representing a *face* value of a card (`2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `T`en, `J`ack, `Q`ueen, `K`ing, `A`ce).

The `Card` class has the following methods: `__init__`, `__str__`, and all the comparison operators `__eq__`, `__ne__`, `__lt__`, `__le__`, `__gt__`, and `__ge__`. We define comparison operators on cards so that we can sort them (for pretty presentation of a dealt hand of cards, for example). Two cards are compared as follows: If they have different suits, we order them as `C` < `D` < `H` < `S`. If they have the same suit, then they are ordered by face value as $2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < T < J < Q < K < A$.

- In the file `poker.py`, the `Card` class attributes and the constructor have been implemented for you. The constructor implementation is complete and does not need to be modified. You must complete the remaining methods of the `Card` class first, as you will use instances of this class to implement the `CardDeck` and `PokerHand` classes (described below).
- Note that the instance attributes for `Card` have intentionally not been made private. This means that you can find the suit and face of a `Card` object by referring to the `.suit` and `.face` instance attributes directly; you may find this useful.

`CardDeck` **class** (**29 points**) An instance of this class is a standard deck of cards consisting of 52 playing cards. Hence, each `CardDeck` instance is a collection of `Card` objects. You are asked to implement the following methods for the `CardDeck` class:

1. `__init__`: The constructor has no parameters. It should initialize the instance to the standard 52-card deck. In other words, the instance is a collection of `Card` objects, which you can store in a list.
2. `__len__`: This method overloads the built-in `len` function. It has no parameters (other than `self`). It should return the number of cards in the deck.
3. `__str__`: This method returns a string that shows all the cards in the deck.
4. `__iter__`: Return an iterator object for the deck. You should implement this method by returning a new card deck iterator object. See below.
5. `deal`: This method returns the top card of the deck. The deck will have one less card after this method is called on an instance.
6. `shuffle`: This method randomly shuffles the deck by implementing the following algorithm: Choose a card at random from all the cards and exchange it with the topmost card. The topmost card will not be touched after this step. Next, choose a random card from among the remaining cards (i.e. all but the topmost), and exchange it with the second card from the top. The top two cards will not be touched after this step. Repeat this procedure until finally you pick a random card from among the last two cards and replace it with the last but one card from the top.
   When implementing this function, keep in mind that the deck being shuffled may have fewer than 52 cards (because some cards from the deck may have been dealt out).
   Note: You will need the `random` module for this method. In particular, you may need to use the function `random.randint(a, b)`, which returns a random integer in the range [`a`, `b`], including both end points.

`CardDeckIterator` **class** An instance of this class is an iterator for a `CardDeck`. Implement the following methods for this class:

1. `__init__`: Implement the constructor with appropriate parameters. This will depend on the instance attributes for your CardDeck class.
2. `__next__`: Return the next card from the deck. If there are no further cards (i.e, the iterator has reached the end of the deck), raise the `StopIteration` exception.

`PokerHand` **class** (**36 points**) An instance of this class is a five-card poker hand. Implement the following methods for the `PokerHand` class. Note that two *class attributes* have been set up for you. These are `numcards` (for the number of cards in the poker hand), which has been set to `5`. The other is `handtypes`, which is a list containing the names of the different types of poker hands. Further details are provided below.

1. `__init__`: The constructor has a single parameter `deck`, which is a deck of cards (an instance of `CardDeck`). It initializes the poker hand by dealing the top five cards from the deck of cards.
   Hints: Create an instance attribute, called `cards` say, to store the five cards of the hand. In addition, in order to determine the type of poker hand that has been dealt (see the methods below), maintain two other instance attributes as follows: Maintain a dictionary called `suitcounts` that keeps track of the number of cards of each suit in the poker hand. Maintain another dictionary called `facecounts` that keeps track of the number of cards of each face value in the poker hand.
2. `__str__`: Returns a nice string representation of the poker hand. The representation should list the cards in the hand in sorted order (the ordering is given by the overloaded boolean operators for Cards). This is done easily by simply sorting the cards in the hand by using the built-in `sort` method.
3. `evaluate`: This method returns the highest ranking that can be assigned to the hand. Note that this will be one of the entries in the `handtypes` class attribute. Keep in mind that the lowest ranking, *"High Card"*, is given to a hand that does not meet the requirements of any of the other eight possible rankings. Implement this method by calling the eight methods, described below, to assess the highest ranking of the hand.
4. `isStraightFlush`: A *straight flush* is the highest ranking for a poker hand. This method returns `True` if the poker hand is a straight flush and `False` otherwise. This is a hand that contains five cards in sequence, all of the same suit. For example, the poker hand `D7, D8, D9, D10, DJ` is a straight flush. Note that this is simply a hand that meets the requirements of both a straight (`isStraight`) and a flush (`isFlush`).
5. `isFourOfAKind`: The next highest ranking for a poker hand is *four of a kind*. This method returns `True` if the poker hand is four of a kind and `False` otherwise. This is a hand that contains four cards with the same face value, and any other card (which obviously will not match the other four). For example, the poker hand `C9, D9, H5, H9, S9` is four of a kind (there are four cards with `9` as the face value).
6. `isFullHouse`: The next highest ranking for a poker hand is a *full house*. This method returns `True` if the poker hand is a full house and `False` otherwise. This is a hand that contains three cards with the same face value, and two other cards with another face value. For example, the poker hand `C5, CQ, DQ, S5, SQ` is a full house (there are three cards with `Q` as the face value, and two cards with `5` as the face value).
7. `isFlush`: The next highest ranking for a poker hand is a *flush*. This method returns `True` if the poker hand is a flush and `False` otherwise. This is a hand that has all

five cards of the same suit. For example, the poker hand `S2, S4, S5, S10, SK` is a flush.

8. `isStraight`: The next highest ranking is a *straight*. This method returns `True` if the poker hand is a straight and `False` otherwise. This is a hand in which the five cards are in a consecutive sequence by face value. For example, the poker hand `C8, C9, D10, HJ, HQ` is a straight.
9. `isThreeOfAKind`: The next highest ranking is a *three of a kind*. This method returns `True` if the poker hand is three of a kind and `False` otherwise. This is a hand in which there are exactly three cards with the same face value, and two other cards that have other face values (different from each other). For example, `C3, CQ, DQ, S5, SQ` is three of a kind (there are three cards with `Q` as the face value, one card with `3` as the face value, and one card with `5` as the face value).
10. `isTwoPair`: The next ranking is a *two pair*. This method returns `True` if the poker hand is a two pair and `False` otherwise. This is a hand with two cards with the same face values, plus two other cards of the same face values (different from the first), plus a fifth card that is different from both. For example, `C2, D2, D4, H3, H4` is a two pair.
11. `isOnePair`: The next ranking is a *one pair*. This method returns `True` if the poker hand is a one pair and `False` otherwise. This is a hand with exactly two cards of matching face values, plus three other cards that are different from this face value and from each other. For example, `C2, D9, DK, HQ, S9` is a one pair.

A sample output from a program run is shown in a file called `outputSample`, available on Canvas. Please note that the test code provided in `poker.py` is fairly basic. Feel free to provide more intesting test code *as long as it works correctly and is not interactive, i.e., does not require input from the grader (in order to optimize grading time).*

Submission Guidelines

Download the file `poker.py` from Canvas. *You must implement all the classes for this problem within this module.* You can test your solution by running `poker.py`. Insert your name at the top of the module.

Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Canvas as follows:

1. Go to the “Assignments” tab of the Canvas site for this course.
2. Click on “Programming Assignment #5” under Homework Assignments.
3. Download the homework document (`hw5.pdf`), the problem module `poker.py`, and the sample output `outputSample`.
4. Use this same link to upload your homework file (`poker.py`) when you are ready to submit.

You must submit your assignment at or before 11:59PM on November 24, 2021.