CSE 231 **Spring 2014**

Programming Project #9

Assignment Overview

This assignment focuses on the design, implementation and testing of a Python program which uses classes to solve the problem described below. Note: you are using a class we provide; you are not designing a class.

It is worth 50 points (5% of course grade) and must be completed no later than 11:59 PM on Monday, April 7.

Assignment Deliverable

The deliverable for this assignment is the following file:

proj09.py – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via the **handin system** before the project deadline.

Assignment Background

The goal of this project is to gain practice with use of classes and creating functions. You will design and implement a Python program which plays simplified Texas Hold'em Poker.

The program should deal two cards to two players (one card to each player, then a second card to each player), and then five community cards which players share to make their hands. A poker hand is the best five cards from the community cards plus the player's cards (i.e., best 5 out of 7 cards total). The goal of this assignment is to find the category of each player's hand and determine the winner.

The rules of this game are relatively simple and you can find information about the game and about the poker hands in the links below. Keep in mind that you will only find the category of the hands and all nine cards can be dealt at once (a real poker game deals cards in stages to allow for betting, but we aren't betting).

http://en.wikipedia.org/wiki/Texas_holdem
http://en.wikipedia.org/wiki/Poker_hands

The categories in order from lowest to highest are: High card, 1 pair, 2 pair, 3 of a kind, straight, flush, full house, 4 of a kind, straight flush.

You will not find a player's hand's value, but just the *category* of the hand.

It is a tie if both players have hands in the same category. That is, if both of them have straights, it is considered a tie no matter who has the higher straight.

Our game is simplified in two ways:

1. We only compare categories to determine the winner, not the value of the hands. For example, in a real poker game the winner of hands with exactly one pair would be determined by which pair had the highest card rank. That is, a pair of 10s wins over a pair of 3s. In our game, two hands with exactly one pair is considered a tie with no consideration given to the card ranks.

2. The Card class ranks an Ace as the lowest card in a suit; whereas traditional poker ranks an Ace as highest. For simplicity, we will keep Aces as the lowest ranked card because we are only comparing categories not values. In particular, the cards 10h, Jh, Qh, Kh, Ah are not considered to make a straight in our game (they would under normal poker rules).

Assignment Specifications

1. Your program will use the Card and Deck classes found in the file named `cards.py`. You may ***not*** modify the contents of that file: we will test your project using our copy of cards.py. We have included a `cardsDemo.py` program to illustrate how to use the Card and Deck classes.

2. Your program will be subdivided into meaningful functions. Use a function for each category (except that "high-card" doesn't need a function). See hints below.

3. Your program will display each player's dealt cards, the community cards, and announce the winner of the hand. Also, your program will display the winning combination (or either of the winning combinations, if a tie) as well as the category of the winning hand (see sample output below). For example, if the winning combination is "four-of-a-kind", those four cards will be displayed; the fifth card in the hand will not be displayed. The display will be appropriately labeled and formatted.

4. After each run, the program will prompt the user if they want to see another hand: "do you want to continue: y or n". The program should continue if user enters "y" or "Y"; otherwise, the program should halt. Also, if there are fewer than 9 cards left in the deck, the program should halt. Do not shuffle between hands; only shuffle at the beginning of the program.

5. Using dictionaries in this assignment is very useful as you can find how many cards have the same suit, or how many of them have the same rank. Most of my functions used a dictionary that had rank as the key, so I created a separate function to build such a dictionary from a hand.

Hints

1. There are 9 categories of hands. For each category (except high-card), I wrote a function that would take as an argument a list of 7 cards and return either a sub-list of cards that satisfied that category or an empty list. That design let me use the functions in Boolean expressions since an empty list evaluates to False whereas a non-empty list evaluates to True. Returning a list of cards also allowed me to use functions within functions, e.g., a straight flush must be a flush. Returning a list of cards also made testing easier because I could see not only that the function had found a combination of that type, but I also could easily check that it was correct. By the way, the function to find a *straight* was the hardest function to write.

2. Finding a winner is simple: start with the highest category and work your way down the categories in a cascade of "if" and "elif" statements. The result is a long, but repetitive structure. (You do not need to check all possible combinations of hands because you are only trying to find the highest category that a hand fits.)

3. Think strategically for testing. Custom build a set of hands for testing by creating particular cards, e.g. H1 = [5c, 2c, 5h, 4s, 3d, 2h, 5d] can be used to test for a *full house* (I had to create each card first, e.g. `c1 = cards.Card(5,1)` to create the 5c card in H1. It took many lines to create the testing hands, but once built they proved useful.) Test each category function separately using the custom hands you created for testing. For example, `result = full_house(H1)` should yield a result [5c, 5h, 5d, 2c, 2h].

4. I found dictionaries useful within functions. Sometimes using the rank as key worked best; other times using the suit as key was best.

5. In order for sort() and sorted() to work on any data type the less-than operation must be defined. That operation is not defined in the Cards class so neither sort() nor sorted() work on Cards. (We tried to make the Cards class as generic as possible to work with a wide variety of card games, and the ordering of cards considering both rank and suit varies across card games.)

Sample Output

```
---------------------------------------
Let's play poker!

Community cards: [9d, 3d, 6s, 2s, 2d]
Player 1: [4h, 5c]
Player 2: [9c, 5h]

Player 1 wins with a straight: [2s, 3d, 4h, 5c, 6s]

Do you wish to play another hand?(Y or N) y
---------------------------------------
Let's play poker!

Community cards: [Ks, 6d, Kc, 5s, 9h]
Player 1: [4c, 10s]
Player 2: [7s, Qc]

TIE with one pair: [Ks, Kc]

Do you wish to play another hand?(Y or N) y
---------------------------------------
Let's play poker!

Community cards: [Jc, 8h, Jd, Js, 3c]
Player 1: [3h, 4d]
Player 2: [As, 7c]

Player 1 wins with a full house: [Jc, Jd, Js, 3h, 3c]

Do you wish to play another hand?(Y or N) y
---------------------------------------
Let's play poker!
```

```
Community cards: [Qs, 4s, 8d, 2h, Qd]
Player 1: [7d, 6c]
Player 2: [2c, 8s]

Player 2 wins with two pairs: [8s, 8d, 2c, 2h]

Do you wish to play another hand?(Y or N) y
---------------------------------------
Let's play poker!

Community cards: [10h, 7h, 10d, Kh, Ah]
Player 1: [9s, Ad]
Player 2: [Ac, 3s]

TIE with two pairs: [10h, 10d, Ad, Ah]
Deck has too few cards so game is done.
```