Assignment #4

Professor Ahmad Namini
Python and Applications to Business Analytics Fall 2018, Module 1

October 8, 2018

Exercise 1. Poker is a popular game throughout the world, whereby a player eventually (through common cards or just their cards) has five cards which is then ranked based on the probability that that 5-card hand rank can occur. Without the use of a wild card, the highest hands (corresponding to the lowest possible probability of occurring) are as follows:

Hand Rank	Name	Notes
1	Straight Flush	All cards are of the same suit and in an ordered sequence
2	Four of a Kind	Four of the same rank
3	Full House	Three of one rank and a pair of another rank
4	Flush	All cards of of the same suit
5	Straight	All cards form a ordered sequence
6	Three of a Kind	Three of one rank
7	Two Pair	Two pairs of the same rank
8	One Pair	One pair of the same
9	High Card	A hand of nothing

Card Suites are "Hearts", "Spades", "Diamonds", "Clubs" while card ranks are "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace".

1. Using the following code, which has classes for a card, a poker_hand, and a deck of cards, modify the code to write a computer program to determine the probability of each hand rank.

2. Using the following code, modify the code so that after the first two cards dealt, what is the probability of the poker hands rank conditioned on your first two cards.

```
import collections
import itertools
import random

SUIT_LIST = ("Hearts", "Spades", "Diamonds", "Clubs")
NUMERAL_LIST = ("2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace")

class card:
    def __init__(self, numeral, suit):
        self.numeral = numeral
        self.suit = suit
        self.card = self.numeral, self.suit
    def __repr__(self):
        return self.numeral + "-" + self.suit

class poker_hand():
    def __init__(self, card_list):
        self.card_list = card_list
    def __repr__(self):
        short_desc = "Nothing."
        numeral_dict = collections.defaultdict(int)
        suit_dict = collections.defaultdict(int)
        for my_card in self.card_list:
            numeral_dict[my_card.numeral] += 1
            suit_dict[my_card.suit] += 1
        # Pair
        if len(numeral_dict) == 4:
            short_desc = "One␣pair."
        # Two pair or 3-of-a-kind
        elif len(numeral_dict) == 3:
            if 3 in numeral_dict.values():
                short_desc ="Three-of-a-kind."
            else:
                short_desc ="Two␣pair."
        # Full house or 4-of-a-kind
        elif len(numeral_dict) == 2:
            if 2 in numeral_dict.values():
                short_desc ="Full␣house."
            else:
                short_desc ="Four-of-a-kind."
        else:
            # Flushes and straights
            straight, flush = False, False
            if len(suit_dict) == 1:
                flush = True
            min_numeral = min([NUMERAL_LIST.index(x) for x in numeral_dict.keys()])
            max_numeral = max([NUMERAL_LIST.index(x) for x in numeral_dict.keys()])
            if int(max_numeral) - int(min_numeral) == 4:
                straight = True
            # Ace can be low
            low_straight = set(("Ace", "2", "3", "4", "5"))
            if not set(numeral_dict.keys()).difference(low_straight):
                straight = True
            if straight and not flush:
                short_desc ="Straight."
            elif flush and not straight:
                short_desc ="Flush."
            elif flush and  straight:
                short_desc ="Straight␣flush."
        enumeration = "/".join([str(x) for x in self.card_list])
        return "{enumeration}␣({short_desc})".format(**locals())

class deck(set):
    def __init__(self):
        for numeral, suit in itertools.product(NUMERAL_LIST, SUIT_LIST):
            self.add(card(numeral, suit))
    def get_card(self):
        a_card = random.sample(self, 1)[0]
        self.remove(a_card)
        return a_card
    def get_hand(self, number_of_cards=5):
        if number_of_cards == 5:
            return poker_hand([self.get_card() for x in range(number_of_cards)])
        else:
            raise NotImplementedError

for i in range(10):
    print(deck().get_hand())
```