Chapter18_Exercises

June 8, 2017

1 Chapter 18 Exercises

In [4]:
```
"""This module contains a code example related to

Think Python, 2nd Edition
by Allen Downey
http://thinkpython2.com

Copyright 2015 Allen Downey

License: http://creativecommons.org/licenses/by/4.0/
"""

from __future__ import print_function, division

import random


class Card:
    """Represents a standard playing card.

    Attributes:
      suit: integer 0-3
      rank: integer 1-13
    """

    suit_names = ["Clubs", "Diamonds", "Hearts", "Spades"]
    rank_names = [None, "Ace", "2", "3", "4", "5", "6", "7",
                  "8", "9", "10", "Jack", "Queen", "King"]

    def __init__(self, suit=0, rank=2):
        self.suit = suit
        self.rank = rank

    def __str__(self):
        """Returns a human-readable string representation."""
        return '%s of %s' % (Card.rank_names[self.rank],
```

```
                                    Card.suit_names[self.suit])

    def __eq__(self, other):
        """Checks whether self and other have the same rank and suit.

        returns: boolean
        """
        return self.suit == other.suit and self.rank == other.rank

    def __lt__(self, other):
        """Compares this card to other, first by suit, then rank.

        returns: boolean
        """
        t1 = self.suit, self.rank
        t2 = other.suit, other.rank
        return t1 < t2


class Deck:
    """Represents a deck of cards.

    Attributes:
      cards: list of Card objects.
    """

    def __init__(self):
        """Initializes the Deck with 52 cards.
        """
        self.cards = []
        for suit in range(4):
            for rank in range(1, 14):
                card = Card(suit, rank)
                self.cards.append(card)

    def __str__(self):
        """Returns a string representation of the deck.
        """
        res = []
        for card in self.cards:
            res.append(str(card))
        return '\n'.join(res)

    def add_card(self, card):
        """Adds a card to the deck.

        card: Card
        """
```

```
        self.cards.append(card)

    def remove_card(self, card):
        """Removes a card from the deck or raises exception if it is not th

        card: Card
        """
        self.cards.remove(card)

    def pop_card(self, i=-1):
        """Removes and returns a card from the deck.

        i: index of the card to pop; by default, pops the last card.
        """
        return self.cards.pop(i)

    def shuffle(self):
        """Shuffles the cards in this deck."""
        random.shuffle(self.cards)

    def sort(self):
        """Sorts the cards in ascending order."""
        self.cards.sort()

    def move_cards(self, hand, num):
        """Moves the given number of cards from the deck into the Hand.

        hand: destination Hand object
        num: integer number of cards to move
        """
        for i in range(num):
            hand.add_card(self.pop_card())


class Hand(Deck):
    """Represents a hand of playing cards."""

    def __init__(self, label=''):
        self.cards = []
        self.label = label


def find_defining_class(obj, method_name):
    """Finds and returns the class object that will provide
    the definition of method_name (as a string) if it is
    invoked on obj.

    obj: any python object
```

```
        method_name: string method name
        """
        for ty in type(obj).mro():
            if method_name in ty.__dict__:
                return ty
        return None
```

1.1 Exercise 18.2

Write a Deck method called deal_hands that takes two parameters, the number of hands and the number of cards per hand. It should create the appropriate number of Hand objects, deal the appropriate number of cards per hand, and return a list of Hands.

```
In [1]: def deal_hands(num_hands, cards_per_hand):
            list_hands = []
            for i in range(num_hands):
                hand = Hand()
                deck.move_cards(hand, cards_per_hand)
                list_hands.append(hand)
            return list_hands
```

1.2 Exercise 18.3

The following are the possible hands in poker, in increasing order of value and decreasing order of probability: pair: two cards with the same rank two pair: two pairs of cards with the same rank three of a kind: three cards with the same rank straight: five cards with ranks in sequence (aces can be high or low, so Ace-2-3-4-5 is a straight and so is 10-Jack-Queen-King-Ace, but Queen-King-Ace-2-3 is not.) flush: five cards with the same suit full house: three cards with one rank, two cards with another four of a kind: four cards with the same rank straight flush: five cards in sequence (as defined above) and with the same suit The goal of these exercises is to estimate the probability of drawing these various hands.

1. Download the following files from http: // thinkpython2. com/ code : Card.py : A complete version of the Card, Deck and Hand classes in this chapter. PokerHand.py : An incomplete implementation of a class that represents a poker hand, and some code that tests it.
2. If you run PokerHand.py, it deals seven 7-card poker hands and checks to see if any of them contains a flush. Read this code carefully before you go on.
3. Add methods to PokerHand.py named has_pair, has_twopair, etc. that return True or False according to whether or not the hand meets the relevant criteria. Your code should work correctly for "hands" that contain any number of cards (although 5 and 7 are the most common sizes).
4. Write a method named classify that figures out the highest-value classification for a hand and sets the label attribute accordingly. For example, a 7-card hand might contain a flush and a pair; it should be labeled "flush".
5. When you are convinced that your classification methods are working, the next step is to estimate the probabilities of the various hands. Write a function in PokerHand.py that shuffles a deck of cards, divides it into hands, classifies the hands, and counts the number of times various classifications appear.

6. Print a table of the classifications and their probabilities. Run your program with larger and larger numbers of hands until the output values converge to a reasonable degree of accuracy. Compare your results to the values at http: // en. wikipedia. org/ wiki/ Hand_ rankings .

In [70]:
```
"""This module contains a code example related to

Think Python, 2nd Edition
by Allen Downey
http://thinkpython2.com

Copyright 2015 Allen Downey

License: http://creativecommons.org/licenses/by/4.0/
"""

from __future__ import print_function, division

#from Card import Hand, Deck


class PokerHand(Hand):
    """Represents a poker hand."""

    def suit_hist(self):
        """Builds a histogram of the suits that appear in the hand.

        Stores the result in attribute suits.
        """
        self.suits = {}
        for card in self.cards:
            self.suits[card.suit] = self.suits.get(card.suit, 0) + 1

    def rank_hist(self):
        self.ranks = {}
        for card in self.cards:
            self.ranks[card.rank] = self.ranks.get(card.rank, 0) + 1

    def has_flush(self):
        """Returns True if the hand has a flush, False otherwise.

        Note that this works correctly for hands with more than 5 cards.
        """
        self.suit_hist()
        for val in self.suits.values():
            if val >= 5:
                return True
        return False
```

```
def has_pair(self):
    self.rank_hist()
    for val in self.ranks.values():
        if val == 2:
            return True
    return False

def has_two_pair(self):
    self.rank_hist()
    counter = 0
    for val in self.ranks.values():
        if val == 2:
            counter += 1
        if counter == 2:
            return True
    return False

def has_three_of_a_kind(self):
    self.rank_hist()
    for val in self.ranks.values():
        if val == 3:
            return True
    return False

def has_straight(self):
    self.rank_hist()
    ranks = self.ranks.copy()
    ranks[14] = ranks.get(1, 0)

    return self.in_a_row(ranks, 5)

def in_a_row(self, ranks, n=5):
    count = 0
    for i in range(1, 15):
        if ranks.get(i, 0):
            count += 1
            if count == n:
                return True
        else:
            count = 0
    return False

def has_straight_flush(self):
    self.rank_hist()
    s = set()
    for c in self.cards:
        s.add((c.rank, c.suit))
        if c.rank == 1:
```

```
                s.add((14, c.suit))

        for suit in range(4):
            count = 0
            for rank in range(1, 15):
                if (rank, suit) in s:
                    count += 1
                    if count == 5:
                        return True
                else:
                    count = 0
        return False

    def has_full_house(self):
        self.rank_hist()
        bool1 = False
        bool2 = False

        for val in self.ranks.values():
            if val == 3:
                bool1 = True
            if val == 2:
                bool2 = True
            if bool1 and bool2:
                return True

        return False

    def has_four_of_a_kind(self):
        self.rank_hist()
        for val in self.ranks.values():
            if val == 4:
                return True
        return False

    def classify(self):
        if self.has_pair():
            self.label = 'pair'
        if self.has_two_pair():
            self.label = 'two pair'
        if self.has_three_of_a_kind():
            self.label = 'three of a kind'
        if self.has_straight():
            self.label = 'straight'
        if self.has_flush():
            self.label = 'flush'
        if self.has_full_house():
            self.label = 'full house'
```

```
            if self.has_four_of_a_kind():
                self.label = 'four of a kind'
            if self.has_straight_flush():
                self.label = 'straight flush'

    def simulate(flag=False, flag2=False):
        deck = Deck()
        deck.shuffle()
        labels = {}

        for i in range(7):
            hand = PokerHand()
            deck.move_cards(hand, 7)
            hand.sort()
            hand.classify()
            labels[hand.label] = labels.get(hand.label, 0) + 1
            if flag:
                print(hand)
                print(hand.label)
                print()
        if flag2:
            print(labels)

        return labels

    if __name__ == '__main__':
        hist = {}
        num_simulations = 10000
        for i in range(num_simulations):
            t = simulate(False, False)

            for key, val in t.items():
                hist[key] = hist.get(key, 0) + val

        prob = []

        for key, val in hist.items():
            prob.append( ((val*100)/(num_simulations*7), key) )

        prob.sort(reverse=True)

        for item in prob:
            print('{:.2f}% probability of {}'.format(item[0], item[1]))
```

```
43.60% probability of pair
23.77% probability of two pair
17.49% probability of
```

```
4.79% probability of three of a kind
4.55% probability of straight
3.00% probability of flush
2.61% probability of full house
0.17% probability of four of a kind
0.03% probability of straight flush
```