**ROBOTICS & ARTIFICIAL INTELLIGENCE DEPARTMENT**

# Total Marks: 04

**Obtained Marks:**

Programming for Artificial Intelligence

**Assignment # 02**

**Last date of Submission: 21st March 2024**

# Submitted To: Mr. Saad Irfan Khan

**Student Name: Syed Muhammad Mustafa**

# Reg. Number: 22108130

**ROBOTICS & ARTIFICIAL INTELLIGENCE DEPARTMENT**

***Instructions****: Copied or shown assignments will be marked zero. Late submissions are not entertained in any case.*

**CLO 2 – PLO A, B – C3**

**Question 01.** **(4 Marks)**

**Case Study Topic: Inheritance in Python**

The following are the possible poker hands, 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 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:
   * [Card.py](https://docs.google.com/document/d/e/2PACX-1vT2xR-nJvPXVUIwji-z0qVDFBDrazr697XAT2yzDjZdLwHkOZi1HSc9lT9qD5qOrtqDUBJfUAlqzST2/pub): A complete version of the **Card**, **Deck** , and **Hand** classes in this chapter.
   * [PokerHand.py](https://docs.google.com/document/d/e/2PACX-1vQ6VBYsX0G3ZBWbh7wLXSVI9Wu3FkT6aI4z1dy3PQ37lsUIuAxVkSjQp7db9SqjTLnyZkBRXk3fqPjf/pub): 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”.

## Code [Card.py]:

|  |
| --- |
| """  This module contains code from  Think Python: an Introduction to Software Design  Allen B. Downey  """  import random  class Card(object):      """represents a standard playing card."""  **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):          return '%s of %s' % (Card.**rank\_names**[self.**rank**],                               Card.**suit\_names**[self.**suit**])      def \_\_cmp\_\_(self, other):          t1 = self.**suit**, self.**rank**          t2 = other.suit, other.rank          return cmp(t1, t2)      def \_\_lt\_\_(self, other):          if self.**suit** < other.suit:              return True          elif self.**suit** > other.suit:              return False          else:              return self.**rank** < other.rank    class Deck(object):      """represents a deck of cards"""        def \_\_init\_\_(self):          self.**cards** = []          for suit in range(4):              for rank in range(1, 14):                  card = Card(suit, rank)                  self.**cards**.append(card)      def \_\_str\_\_(self):          res = []          for card in self.**cards**:              res.append(str(card))          return '\n'.join(res)      def add\_card(self, card):          """add a card to the deck"""          self.**cards**.append(card)      def pop\_card(self, i=-1):          """remove and return a card from the deck.          By default, pop the last card."""          return self.**cards**.pop(i)      def shuffle(self):          """shuffle the cards in this deck"""          random.**shuffle**(self.**cards**)      def sort(self):          """sort the cards in ascending order"""          self.**cards**.sort()      def move\_cards(self, hand, num):          """move the given number of cards from the deck into the Hand"""          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.**label** = label          self.**cards** = []    def find\_defining\_class(obj, meth\_name):      """find and return the class object that will provide      the definition of meth\_name (as a string) if it is      invoked on obj.      """      for ty in type(obj).mro():          if meth\_name in ty.\_\_dict\_\_:              return ty      return None    if \_\_name\_\_ == '\_\_main\_\_':      deck = Deck()      deck.shuffle()      hand = Hand()      print(find\_defining\_class(hand, 'shuffle'))      deck.move\_cards(hand, 5)      hand.sort()      print(hand) |

## Output:

|  |
| --- |
|  |

## Code:

|  |
| --- |
| """  This module contains code from  Think Python: an Introduction to Software Design  Allen B. Downey  """  from Card import \*  class PokerHand(Hand):        def suit\_hist(self):          """build a histogram of the suits that appear in the hand"""          self.**suits** = {}          for card in self.**cards**:              self.**suits**[card.suit] = self.**suits**.get(card.suit, 0) + 1      def has\_flush(self):          """return True if the hand has a flush, False otherwise"""          self.suit\_hist()          for val in self.**suits**.values():              if val >= 5:                  return True          return False        def has\_pairs(self):          self.**rank** = {}            for card in self.**cards**:              self.**rank**[card.rank] = self.**ranks**.get(card.rank, 0) + 1          for val in self.**ranks**.values():              if val == 2:                  return True          return False      def has\_two\_pairs(self):          pairs = 0          self.**ranks** = {}          for card in self.**cards**:              self.**ranks**[card.rank] = self.**ranks**.get(card.rank, 0) + 1          for val in self.**ranks**.values():              if val == 2:                  pairs += 1          return pairs == 2      def classify(self):          if self.has\_flush():              self.**label** = "Flush"          elif self.has\_two\_pairs():              self.**label** = "Two Pairs"          elif self.has\_pairs():              self.**label** = "Pairs"          else:              self.**label** = "None"  if \_\_name\_\_ == '\_\_main\_\_':      deck = Deck()      deck.shuffle()      for i in range(7):          hand = PokerHand()          deck.move\_cards(hand, 7)          hand.sort()          print(hand)          print(hand.has\_flush())          print() |

## 

## Output [True]:

|  |
| --- |
| You can see that since there are fair pairs of “Hearts” in this shuffled pair it returns “True”. Since in Poker 5 cards of the same suit is called a flush. Even though there are two spades it is still considered a flush/ |

## Output [False]:

|  |
| --- |
|  |

## Output [False]:

|  |
| --- |
|  |

**Note: Paste the complete code here along with output screenshots.**