Zeru-Zhou-project11(1)

April 8, 2022

1 Project11 – Zeru Zhou

TA Help: NA

Collaboration: NA

- Get help from dr. Ward's videos
- Get help from this link https://stackoverflow.com/questions/2361945/detecting-consecutive-integers-in-a-list

1.1 Question 1

[2]:
```python
from collections import Counter

class Player:
    def __init__(self, name, deck):
        self.name = name
        self.deck = deck
        self.hand = []

    def __str__(self):
        return(f"""
        {self.name}\n
        Top 5 cards: {self.deck[:5]}
        """)

    def draw(self):
        card = self.deck.cards.pop(0)
        self.hand.append(card)

    def has_set(self):
        summarizedhand = Counter(self.hand)
        for key, value in summarizedhand.items():
            if value >= 3:
                return True
        return False

    def get_sets(self):
        summarizedhand = Counter(self.hand)
        my_set = []
        for key, value in summarizedhand.items():
            if value >= 3:
                my_set.append([card for card in self.hand if card == key])
        return my_set
```

[3]:
```
class Card:
    _value_dict = {"2": 2, "3": 3, "4": 4, "5": 5, "6": 6, "7": 7, "8":8, "9":
↪9, "10": 10, "j": 11, "q": 12, "k": 13, "a": 1}
    def __init__(self, number, suit):
        if str(number).lower() not in [str(num) for num in range(2, 11)] +␣
↪list("jqka"):
            raise Exception("Number wasn't 2-10 or J, Q, K, or A.")
        else:
            self.number = str(number).lower()
        if suit.lower() not in ["clubs", "hearts", "diamonds", "spades"]:
            raise Exception("Suit wasn't one of: clubs, hearts, spades, or␣
↪diamonds.")
        else:
            self.suit = suit.lower()

    def __str__(self):
        return(f'{self.number} of {self.suit.lower()}')

    def __repr__(self):
        return(f'Card(str({self.number}), "{self.suit}")')

    def __eq__(self, other):
        if self.number == other.number:
            return True
        else:
            return False

    def __lt__(self, other):
        if self._value_dict[self.number] < self._value_dict[other.number]:
            return True
        else:
            return False

    def __gt__(self, other):
        if self._value_dict[self.number] > self._value_dict[other.number]:
            return True
        else:
            return False
    def __hash__(self):
        return hash(self.number)
```

[4]:
```
class Deck:
    brand = "Bicycle"
    _suits = ["clubs", "hearts", "diamonds", "spades"]
    _numbers = [str(num) for num in range(2, 11)] + list("jqka")

    def __init__(self):
        self.cards = [Card(number, suit) for suit in self._suits for number in␣
 ↪self._numbers]

    def __len__(self):
        return len(self.cards)

    def __getitem__(self, key):
        return self.cards[key]

    def __setitem__(self, key, value):
        self.cards[key] = value

    def __str__(self):
        return f"A {self.brand.lower()} deck."
```

[5]:
```
import random
deck = Deck()
player1 = Player("Eric", deck)
random.shuffle(deck)
for i in range(20):
    player1.draw()
sets = player1.get_sets()
sets
```

[5]:
```
[[Card(str(10), "diamonds"), Card(str(10), "clubs"), Card(str(10), "hearts")],
 [Card(str(7), "hearts"), Card(str(7), "clubs"), Card(str(7), "spades")]]
```

As above, get_sets method is added.

1.2 Question 2

[6]:
```
from collections import Counter

class Player:
    def __init__(self, name, deck):
        self.name = name
        self.deck = deck
        self.hand = []
    def __str__(self):
        return(f"""
        {self.name}\n
        Top 5 cards: {self.deck[:5]}
        """)

    def draw(self):
        card = self.deck.cards.pop(0)
        self.hand.append(card)

    def has_set(self):
        summarizedhand = Counter(self.hand)
        for key, value in summarizedhand.items():
            if value >= 3:
                return True
        return False

    def get_sets(self):
        summarizedhand = Counter(self.hand)
        my_set = []
        for key, value in summarizedhand.items():
            if value >= 3:
                my_set.append([card for card in self.hand if card == key])
        return my_set

    def hand_as_df(self):
        my_df = {'suit': [], 'numeric_value': [], 'card': []}
        for card in self.hand:
            my_df['suit'].append(card.suit)
            my_df['numeric_value'].append(card._value_dict[card.number])
            my_df['card'].append(card)
        return my_df
```

[7]:
```
import random
import pandas as pd
deck = Deck()
player1 = Player("Eric", deck)
random.shuffle(deck)
for i in range(20):
    player1.draw()

sets = pd.DataFrame(data = player1.hand_as_df())
sets
```

[7]:
```
        suit  numeric_value          card
0      clubs              5    5 of clubs
```

```
1     spades            5    5 of spades
2      clubs           13     k of clubs
3     spades           11    j of spades
4      clubs            1     a of clubs
5      clubs            7     7 of clubs
6      clubs            9     9 of clubs
7      clubs           11     j of clubs
8      clubs            3     3 of clubs
9     hearts            2    2 of hearts
10     clubs            2     2 of clubs
11    hearts            6    6 of hearts
12  diamonds            7  7 of diamonds
13    spades            2    2 of spades
14    hearts            9    9 of hearts
15    hearts            8    8 of hearts
16     clubs           12     q of clubs
17     clubs            4     4 of clubs
18    hearts            4    4 of hearts
19    spades            4    4 of spades
```

Data frame is created.

1.3 Question 3

[29]:
```python
class Player:
    def __init__(self, name, deck):
        self.name = name
        self.deck = deck
        self.hand = []

    def __str__(self):
        return(f"""
        {self.name}\n
        Top 5 cards: {self.deck[:5]}
        """)

    def draw(self):
        card = self.deck.cards.pop(0)
        self.hand.append(card)

    def has_set(self):
        summarizedhand = Counter(self.hand)
        for key, value in summarizedhand.items():
            if value >= 3:
                return True
        return False
    def get_sets(self):
        summarizedhand = Counter(self.hand)
        my_set = []
        for key, value in summarizedhand.items():
            if value >= 3:
                my_set.append([card for card in self.hand if card == key])
        return my_set

    def hand_as_df(self):
        my_df = {'suit': [], 'numeric_value': [], 'card': []}
        for card in self.hand:
            my_df['suit'].append(card.suit)
            my_df['numeric_value'].append(card._value_dict[card.number])
            my_df['card'].append(card)
        return my_df

    def get_runs(self):
        outcome = []
        consecutive = []
        consecutive1 = []
        final=[]
        for idx, group in df.groupby("suit"):
            group = group.sort_values(by = ['numeric_value'])
            outcome.append(group['numeric_value'].tolist())
        from itertools import groupby
        from operator import itemgetter
        for i in outcome:
            for a, b in groupby(enumerate(i), lambda ix : ix[0]-ix[1]):
                consecutive.append(list(map(itemgetter(1), b)))
        for i in consecutive:
            if len(i) >= 3:
                consecutive1.append(i)
        for lists in consecutive1:
            for idx, group in df.groupby("suit"):
                if all(item in group['numeric_value'].tolist() for item in␣
↪lists):
                    for element in lists:
                        for i in group['numeric_value']:
                            if element==i:
                                final.append(group.
↪loc[group['numeric_value']==i,'card'])
        return final
```

```
[24]: import random
      import pandas as pd
      deck = Deck()
      player1 = Player("Alice", deck)
      random.shuffle(deck)
      for _ in range(20):
          player1.draw()

      df = player1.hand_as_df()
      df = pd.DataFrame(df)
      df
```

```
[24]:         suit  numeric_value           card
      0      clubs             13      k of clubs
      1     spades             12     q of spades
      2     spades             13     k of spades
      3      clubs              5      5 of clubs
      4   diamonds              6   6 of diamonds
      5      clubs             11      j of clubs
      6      clubs              9      9 of clubs
      7   diamonds              2   2 of diamonds
      8     spades              4     4 of spades
      9   diamonds             13   k of diamonds
      10     clubs             10     10 of clubs
      11    hearts              9     9 of hearts
      12  diamonds              5   5 of diamonds
      13     clubs              2      2 of clubs
      14  diamonds             11   j of diamonds
      15     clubs              6      6 of clubs
      16    hearts             12     q of hearts
      17     clubs              7      7 of clubs
      18    spades              1     a of spades
      19    hearts             11     j of hearts
```

```
[30]: import random

      deck = Deck()
      player1 = Player("Alice", deck)
      random.shuffle(deck)
      for _ in range(20):
          player1.draw()

      runs = player1.get_runs()
      runs
```

```
[30]: [3    5 of clubs
       Name: card, dtype: object,
       15    6 of clubs
       Name: card, dtype: object,
       17    7 of clubs
       Name: card, dtype: object,
       6    9 of clubs
       Name: card, dtype: object,
       10    10 of clubs
       Name: card, dtype: object,
       5    j of clubs
       Name: card, dtype: object]
```

As above, get_runs is created.

1.4 Pledge

By submitting this work I hereby pledge that this is my own, personal work. I've acknowledged in the designated place at the top of this file all sources that I used to complete said work, including but not limited to: online resources, books, and electronic communications. I've noted all collaboration with fellow students and/or TA's. I did not copy or plagiarize another's work.

> As a Boilermaker pursuing academic excellence, I pledge to be honest and true in all that I do. Accountable together – We are Purdue.