پروژه دست گرمی

بازی جنگ ساده

قبل از شروع به پروژه مرحله OOP 2، بیایید با هم نحوه استفاده از OOP برای یک برنامه قدرتمند و پیچیده مانند یک بازی را مرور کنیم. ما از OOP پایتون برای شبیه‌سازی نسخه ساده‌ای از بازی جنگ استفاده خواهیم کرد. دو بازیکن هر کدام با نیمی از deck شروع می‌کنند، سپس هر کدام یک کارت برمی‌دارند، کارت‌ها را با یکدیگر مقایسه می‌کنند و بازیکنی که کارت با بالاترین ارزش را دارد، هر دو کارت را برای خود برمی‌دارد. در صورت برابری ارزش کارت‌ها، یک تساوی اتفاق می‌افتد.

کلاس کارت تکی

ایجاد یک کلاس کارت با استفاده از متغیرهای خارجی

در اینجا ما از برخی متغیرهای خارجی استفاده خواهیم کرد که بدون توجه به شرایط موجود تغییر نمی‌کنند، مانند یک دک کارت. بدون توجه به اینکه در چه دوره‌ای، مسابقه‌ای یا بازی‌ای هستیم، همچنان به یک دک کارت نیاز خواهیم داشت.

In [1]:

```
# We'll use this later
import random
```

In [2]:

```
suits = ('Hearts', 'Diamonds', 'Spades', 'Clubs')
ranks = ('Two', 'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight', 'Nine', 'Ten', 'Jac
values = {'Two': 2, 'Three':3, 'Four':4, 'Five':5, 'Six':6, 'Seven':7, 'Eight':8,
            'Nine':9, 'Ten':10, 'Jack':11, 'Queen':12, 'King':13, 'Ace':14}

```

In [3]:

```
class Card:

    def __init__(self, suit, rank):
        self.suit = suit
        self.rank = rank
        self.value = values[rank]

    def __str__(self):
        return self.rank + ' of ' + self.suit
```

ساخت یک نمونه از کارت ها

In [ ]:

```
suits[0]
```

In [ ]:

```
ranks[0]
```

In [ ]:

```
two_hearts = Card(suits[0],ranks[0])
```

In [ ]:

```
two_hearts
```

In [ ]:

```
print(two_hearts)
```

In [ ]:

```
two_hearts.rank
```

In [ ]:

```
two_hearts.value
```

In [ ]:

```
values[two_hearts.rank]
```

کلاس دسته کارت‌ها

استفاده از یک کلاس درون یک کلاس دیگر

تا به حال ما فقط یک کارت تکی ایجاد کرده‌ایم، اما چگونه می‌توانیم یک دسته کامل از کارت‌ها ایجاد کنیم؟ بیایید با استفاده از یک کلاس که از کلاس کارت استفاده می‌کند، این کار را بررسی کنیم.

یک deck، از چندین کارت تشکیل می‌شود. این به این معناست که در __init__ کلاس deck، به واقع از کلاس کارت استفاده خواهیم کرد.

In [4]:

```
class Deck:

    def __init__(self):
        # Note this only happens once upon creation of a new Deck
        self.all_cards = []
        for suit in suits:
            for rank in ranks:
                # This assumes the Card class has already been defined!
                self.all_cards.append(Card(suit,rank))

    def shuffle(self):
        # Note this doesn't return anything
        random.shuffle(self.all_cards)

    def deal_one(self):
        # Note we remove one card from the list of all_cards
        return self.all_cards.pop()
```

ایجاد یک deck

In [ ]:

```
mydeck = Deck()
```

In [ ]:

```
len(mydeck.all_cards)
```

In [ ]:

```
mydeck.all_cards[0]
```

In [ ]:

```
print(mydeck.all_cards[0])
```

In [ ]:

```
mydeck.shuffle()
```

In [ ]:

```
print(mydeck.all_cards[0])
```

In [ ]:

```
my_card = mydeck.deal_one()
```

In [ ]:

```
print(my_card)
```

کلاس بازیکن

بیایید یک کلاس بازیکن ایجاد کنیم. یک بازیکن باید قادر باشد نمونه‌هایی از کارت‌ها را در اختیار داشته باشد و همچنین بتواند آن‌ها را از دست خود حذف یا اضافه کند. ما می‌خواهیم که کلاس بازیکن به اندازه کافی انعطاف پذیر باشد تا بتواند یک کارت یا چند کارت را به آن اضافه کند، بنابراین از یک بررسی ساده استفاده می‌کنیم تا همه این کارها در یک متد یکسان قرار گیرد.

همه این موارد را در نظر خواهیم داشت هنگامی که متدهای کلاس بازیکن را ایجاد می‌کنیم.

کلاس بازیکن

In [5]:

```
class Player:

    def __init__(self,name):
        self.name = name
        # A new player has no cards
        self.all_cards = []

    def remove_one(self):
        # Note we remove one card from the list of all_cards
        # We state 0 to remove from the "top" of the deck
        # We'll imagine index -1 as the bottom of the deck
        return self.all_cards.pop(0)

    def add_cards(self,new_cards):
        if type(new_cards) == type([]):
            self.all_cards.extend(new_cards)
        else:
            self.all_cards.append(new_cards)


    def __str__(self):
        return f'Player {self.name} has {len(self.all_cards)} cards.'
```

In [ ]:

```
jose = Player("Jose")
```

In [ ]:

```
jose
```

In [ ]:

```
print(jose)
```

In [ ]:

```
two_hearts
```

In [ ]:

```
jose.add_cards(two_hearts)
```

In [ ]:

```
print(jose)
```

In [ ]:

```
jose.add_cards([two_hearts,two_hearts,two_hearts])
```

In [ ]:

```
print(jose)
```

In [ ]:

```
jose.remove_one()
```

In [ ]:

```
print(jose)
```

منطق بازی جنگ

In [6]:

```
player_one = Player("One")
```

In [7]:

```
player_two = Player("Two")
```

راه اندازی تنظیمات بازی

In [8]:

```
new_deck = Deck()
```

In [9]:

```
new_deck.shuffle()
```

تقسیم Deck میان دو بازیکن

In [ ]:

```
len(new_deck.all_cards)/2
```

In [10]:

```
for x in range(26):
    player_one.add_cards(new_deck.deal_one())
    player_two.add_cards(new_deck.deal_one())
```

In [ ]:

```
len(new_deck.all_cards)
```

In [ ]:

```
len(player_one.all_cards)
```

In [ ]:

```
len(player_two.all_cards)
```

اجرای بازی

In [27]:

```
import pdb
```

In [11]:

```
game_on = True
```

In [13]:

```
round_num = 0

while game_on:

    round_num += 1
    print(f"Round {round_num}")

    if len(player_one.all_cards) == 0:
        print("Player One out of cards! Game Over")
        print("Player Two Wins!")
        game_on = False
        break
    if len(player_two.all_cards) == 0:
        print("Player Two out of cards! Game Over")
        print("Player One Wins!")
        game_on = False
        break

    player_one_cards = []
    player_one_cards.append(player_one.remove_one())

    player_two_cards = []
    player_two_cards.append(player_two.remove_one())

    at_war = True

    while at_war:

        if player_one_cards[-1].value > player_two_cards[-1].value:
            player_one.add_cards(player_one_cards)
            player_one.add_cards(player_two_cards)

            at_war = False

        elif player_one_cards[-1].value < player_two_cards[-1].value:

            # Player Two gets the cards
            player_two.add_cards(player_one_cards)
            player_two.add_cards(player_two_cards)

            # No Longer at "war" , time for next round
            at_war = False

        else:

            print("WAR!")

            if len(player_one.all_cards) < 5:
                print("Player One unable to play war! Game Over at War")
                print("Player Two Wins! Player One Loses!")
                game_on = False
                break
            elif len(player_two.all_cards) < 5:
                print("Player Two unable to play war! Game Over at War")
                print("Player One Wins! Player One Loses!")
                game_on = False
                break
            else:
                for num in range(5):
```

```
                    player_one_cards.append(player_one.remove_one())
                    player_two_cards.append(player_two.remove_one())

```

```
Round 561
Round 562
Round 563
Round 564
Round 565
Round 566
Round 567
Round 568
Round 569
Round 570
Round 571
Round 572
Round 573
Round 574
Round 575
Round 576
WAR!
Player One unable to play war! Game Over at War
Player Two Wins! Player One Loses!
```

گرداوری منطق بازی در یک سلول

In [14]:

```
player_one = Player("One")
player_two = Player("Two")

new_deck = Deck()
new_deck.shuffle()

for x in range(26):
    player_one.add_cards(new_deck.deal_one())
    player_two.add_cards(new_deck.deal_one())

game_on = True
```

In [15]:

```
round_num = 0
while game_on:

    round_num += 1
    print(f"Round {round_num}")

    # Check to see if a player is out of cards:
    if len(player_one.all_cards) == 0:
        print("Player One out of cards! Game Over")
        print("Player Two Wins!")
        game_on = False
        break

    if len(player_two.all_cards) == 0:
        print("Player Two out of cards! Game Over")
        print("Player One Wins!")
        game_on = False
        break

    # Otherwise, the game is still on!

    # Start a new round and reset current cards "on the table"
    player_one_cards = []
    player_one_cards.append(player_one.remove_one())

    player_two_cards = []
    player_two_cards.append(player_two.remove_one())

    at_war = True

    while at_war:


        if player_one_cards[-1].value > player_two_cards[-1].value:

            # Player One gets the cards
            player_one.add_cards(player_one_cards)
            player_one.add_cards(player_two_cards)


            # No Longer at "war" , time for next round
            at_war = False

        # Player Two Has higher Card
        elif player_one_cards[-1].value < player_two_cards[-1].value:

            # Player Two gets the cards
            player_two.add_cards(player_one_cards)
            player_two.add_cards(player_two_cards)

            # No Longer at "war" , time for next round
            at_war = False

        else:
            print('WAR!')
            # This occurs when the cards are equal.
            # We'll grab another card each and continue the current war.

            # First check to see if player has enough cards
```

```
60
61              # Check to see if a player is out of cards:
62              if len(player_one.all_cards) < 5:
63                  print("Player One unable to play war! Game Over at War")
64                  print("Player Two Wins! Player One Loses!")
65                  game_on = False
66                  break
67
68              elif len(player_two.all_cards) < 5:
69                  print("Player Two unable to play war! Game Over at War")
70                  print("Player One Wins! Player One Loses!")
71                  game_on = False
72                  break
73              # Otherwise, we're still at war, so we'll add the next cards
74              else:
75                  for num in range(5):
76                      player_one_cards.append(player_one.remove_one())
77                      player_two_cards.append(player_two.remove_one())
78
```

```
Round 1
Round 2
Round 3
Round 4
Round 5
Round 6
Round 7
Round 8
Round 9
Round 10
Round 11
Round 12
Round 13
Round 14
Round 15
Round 16
Round 17
Round 18
Round 19
WAR!
Round 20
Round 21
Round 22
Round 23
Round 24
Round 25
Round 26
Round 27
Round 28
Round 29
Round 30
Round 31
Round 32
WAR!
Round 33
Round 34
Round 35
Round 36
Round 37
Round 38
Round 39
Round 40
Round 41
Round 42
Round 43
Round 44
Round 45
Round 46
Round 47
Round 48
Round 49
Round 50
Round 51
Round 52
Round 53
Round 54
Round 55
Round 56
Round 57
Round 58
Round 59
```

```
Round 60
Round 61
Round 62
Round 63
Round 64
Round 65
Round 66
Round 67
Round 68
Round 69
WAR!
Round 70
Round 71
Round 72
Round 73
Round 74
WAR!
Player One unable to play war! Game Over at War
Player Two Wins! Player One Loses!
```

In [16]:

```
1 len(player_one.all_cards)
```

Out[16]:

```
3
```

In [17]:

```
1 len(player_two.all_cards)
```

Out[17]:

```
47
```

In [18]:

```
1 print(player_one_cards[-1])
```

```
Jack of Hearts
```

In [19]:

```
1 print(player_two_cards[-1])
```

```
Jack of Clubs
```

عالی!

لینک های که به درک بهتر شما می توانند کمک کنند:

- https://www.reddit.com/r/learnpython/comments/7ay83p/war_card_game/ (https://www.reddit.com/r/learnpython/comments/7ay83p/war_card_game/)
- https://codereview.stackexchange.com/questions/131174/war-card-game-using-classes (https://codereview.stackexchange.com/questions/131174/war-card-game-using-classes)

- https://gist.github.com/damianesteban/6896120 (https://gist.github.com/damianesteban/6896120)
- https://lethain.com/war-card-game-in-python/ (https://lethain.com/war-card-game-in-python/)
- https://hectorpefo.github.io/2017-09-13-Card-Wars/ (https://hectorpefo.github.io/2017-09-13-Card-Wars/)
- https://www.wimpyprogrammer.com/the-statistics-of-war-the-card-game (https://www.wimpyprogrammer.com/the-statistics-of-war-the-card-game)