Chap18_TimeCardDeckHand_Theory&Exercises_TOCHECK

May 25, 2020

1 Chapter 18. Inheritance

1.1 Classes

class Card(object):

"""Represents a standard playing card"""

class Deck(object):

"""Represents a standard deck of cards"""

class Hand(Deck):

"""Represents a hand of playing cards."""

In [4]:
```
class Card(object):
    """Represents a standard playing card"""
    def __init__(self,suit=0,rank=2):
        self.suit=suit
        self.rank=rank

    suit_names=['Clubs','Diamonds','Hearts','Spades']
    rank_names=[None,'Ace','2','3','4','5','6','7','8','9','10','Jack','Queen','King']
    def __str__(self):
        return '%s of %s' % (Card.rank_names[self.rank],
                             Card.suit_names[self.suit])
    def __cmp__(self, other):
        if self.suit>other.suit: return 1
        if self.suit<other.suit: return -1

        if self.rank>other.rank:return 1
        if self.rank<other.rank: return -1

        return 0
    #def __cmp__(self, other):
    #   t1=self.suit,self.rank
    #   t2=other.suit,other.rank
```

```
    #    return cmp(t1,t2)"""
import random
class Deck(object):
    """Represents a standard 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 pop_card(self):
        return self.cards.pop()
    def add_card(self,card):
        self.cards.append(card)
    def shuffle(self):
        random.shuffle(self.cards)

class Hand(Deck):
    """Represents a hand of playing cards."""
    def __init__(self,label=''):
        self.cards=[]
        self.label=label
hand=Hand('new hand')
print hand.cards
print hand.label

deck=Deck()
print "Pachetul este:"
print deck
print "\nSterge ultima carte: \n"
deck.pop_card()
print deck
print"\nAdaugam ultima carte: \n "
deck.add_card(Card(2,11))
print deck
print"\nPachetul amestecat: \n"
deck.shuffle()
print deck
card1=Card(2,11)
print card1
card2=Card(1,12)
print card2
print card1==card2
```

```
        deck=Deck()
        card=deck.pop_card()
        hand.add_card(card)
        print hand
```

```
[]
new hand
Pachetul este:
Ace of Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
Jack of Clubs
Queen of Clubs
King of Clubs
Ace of Diamonds
2 of Diamonds
3 of Diamonds
4 of Diamonds
5 of Diamonds
6 of Diamonds
7 of Diamonds
8 of Diamonds
9 of Diamonds
10 of Diamonds
Jack of Diamonds
Queen of Diamonds
King of Diamonds
Ace of Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
```

```
Ace of Spades
2 of Spades
3 of Spades
4 of Spades
5 of Spades
6 of Spades
7 of Spades
8 of Spades
9 of Spades
10 of Spades
Jack of Spades
Queen of Spades
King of Spades

Sterge ultima carte:

Ace of Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
Jack of Clubs
Queen of Clubs
King of Clubs
Ace of Diamonds
2 of Diamonds
3 of Diamonds
4 of Diamonds
5 of Diamonds
6 of Diamonds
7 of Diamonds
8 of Diamonds
9 of Diamonds
10 of Diamonds
Jack of Diamonds
Queen of Diamonds
King of Diamonds
Ace of Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
```

```
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
Ace of Spades
2 of Spades
3 of Spades
4 of Spades
5 of Spades
6 of Spades
7 of Spades
8 of Spades
9 of Spades
10 of Spades
Jack of Spades
Queen of Spades

Adaugam ultima carte:

Ace of Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
Jack of Clubs
Queen of Clubs
King of Clubs
Ace of Diamonds
2 of Diamonds
3 of Diamonds
4 of Diamonds
5 of Diamonds
6 of Diamonds
7 of Diamonds
8 of Diamonds
9 of Diamonds
10 of Diamonds
Jack of Diamonds
Queen of Diamonds
King of Diamonds
```

```
Ace of Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
Ace of Spades
2 of Spades
3 of Spades
4 of Spades
5 of Spades
6 of Spades
7 of Spades
8 of Spades
9 of Spades
10 of Spades
Jack of Spades
Queen of Spades
Jack of Hearts

Pachetul amestecat:

6 of Spades
5 of Diamonds
9 of Clubs
7 of Diamonds
10 of Hearts
5 of Spades
Queen of Spades
4 of Diamonds
10 of Spades
5 of Clubs
6 of Clubs
Queen of Hearts
10 of Clubs
King of Clubs
Ace of Spades
8 of Hearts
Queen of Diamonds
9 of Hearts
4 of Hearts
```

```
7 of Hearts
2 of Spades
8 of Spades
9 of Diamonds
3 of Spades
4 of Clubs
3 of Clubs
2 of Diamonds
2 of Hearts
Queen of Clubs
6 of Hearts
Jack of Clubs
6 of Diamonds
2 of Clubs
4 of Spades
8 of Clubs
9 of Spades
3 of Diamonds
King of Diamonds
Ace of Clubs
7 of Spades
King of Hearts
Ace of Diamonds
10 of Diamonds
Jack of Diamonds
Jack of Spades
Jack of Hearts
5 of Hearts
Ace of Hearts
3 of Hearts
Jack of Hearts
7 of Clubs
8 of Diamonds
Jack of Hearts
Queen of Diamonds
False
King of Spades
```

1.2 Exercise 18.1

Write a __cmp__ method for Time object.

Hint: you may use the tuple comparison, but you also might consider using integer subtraction.

```
In [1]: class Time(object):
            def __init__(self, hour=0, minute=0, second=0):
                self.hour=hour
```

```
            self.minute=minute
            self.second=second
        def __str__(self):
            return'%.2d:%.2d:%.2d'%(self.hour, self.minute, self.second)
        def time_to_int(self):
            return 3600*self.hour+60*self.minute+self.second
        def __cmp__(self, other):
            if self.time_to_int() > other.time_to_int() : return 1
            if self.time_to_int() < other.time_to_int() : return -1
    t1=Time(9,45,0)
    t2=Time(10,45,0)
    print t1
    print t2
    print t1<t2
```

```
09:45:00
10:45:00
True
```

1.3 Exercise 18.2

Write a Deck method named **sort** that uses the *list method* **sort** to sort the cards in a Deck. **sort** uses the previous __cmp__ method to determine sort order.

```
In [2]: import random
    class Card(object):
        """Represents a standard playing card."""
        def __init__(self, suit=0, rank=2):
            self.suit=suit
            self.rank=rank
        suit_names=['Clubs', 'Diamonds', 'Hearts', 'Spades']
        rank_names=[None, 'Ace', '2', '3','4','5', '6','7', '8', '9','10', 'Jack', 'Queen',
        def __str__(self):
            return'%s of %s'%(Card.rank_names[self.rank], Card.suit_names[self.suit])
        def __cmp__(self, other):
            #check the suits
            if self.suit > other.suit: return 1
            if self.suit < other.suit: return -1
            #suits are the same...check ranks
            if self.rank > other.rank: return 1
            if self.rank < other.rank: return -1
            #ranks are the same...it's a tie
            return 0
    class Deck(object):
        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 pop_card(self):
            return self.cards.pop()
        def add_card(self, card):
            self.cards.append(card)
        def shuffle(self):
            random.shuffle(self.cards)

        def sort(self):
            self.cards.sort()
    d=Deck()
    print d, '\n'
    d.shuffle()
    print '\n', d
    d.sort()
    print '\n', d
```

```
Ace of Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
Jack of Clubs
Queen of Clubs
King of Clubs
Ace of Diamonds
2 of Diamonds
3 of Diamonds
4 of Diamonds
5 of Diamonds
6 of Diamonds
7 of Diamonds
8 of Diamonds
```

```
9 of Diamonds
10 of Diamonds
Jack of Diamonds
Queen of Diamonds
King of Diamonds
Ace of Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
Ace of Spades
2 of Spades
3 of Spades
4 of Spades
5 of Spades
6 of Spades
7 of Spades
8 of Spades
9 of Spades
10 of Spades
Jack of Spades
Queen of Spades
King of Spades


Queen of Spades
5 of Hearts
Queen of Hearts
Queen of Clubs
4 of Spades
10 of Spades
King of Clubs
6 of Hearts
5 of Spades
4 of Diamonds
King of Diamonds
5 of Clubs
Jack of Clubs
3 of Hearts
10 of Hearts
```

```
9 of Diamonds
Queen of Diamonds
7 of Clubs
Jack of Diamonds
4 of Clubs
3 of Diamonds
8 of Hearts
Ace of Diamonds
4 of Hearts
8 of Spades
King of Hearts
King of Spades
10 of Diamonds
5 of Diamonds
7 of Diamonds
6 of Spades
9 of Spades
3 of Spades
6 of Diamonds
9 of Clubs
2 of Spades
Ace of Hearts
8 of Diamonds
2 of Clubs
Ace of Spades
6 of Clubs
2 of Hearts
10 of Clubs
9 of Hearts
Jack of Spades
8 of Clubs
Jack of Hearts
2 of Diamonds
7 of Hearts
Ace of Clubs
3 of Clubs
7 of Spades

Ace of Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
```

```
Jack of Clubs
Queen of Clubs
King of Clubs
Ace of Diamonds
2 of Diamonds
3 of Diamonds
4 of Diamonds
5 of Diamonds
6 of Diamonds
7 of Diamonds
8 of Diamonds
9 of Diamonds
10 of Diamonds
Jack of Diamonds
Queen of Diamonds
King of Diamonds
Ace of Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
Ace of Spades
2 of Spades
3 of Spades
4 of Spades
5 of Spades
6 of Spades
7 of Spades
8 of Spades
9 of Spades
10 of Spades
Jack of Spades
Queen of Spades
King of Spades
```

1.4 Exercise 18.3

Write a Deck method called **deal_hands** that takes two parameters, the number of hands and the number of cards per hand, and that creates new Hand objects, deals the appropriate number of

cards per hand, and returns a list of Hand objects.

```
In [ ]: # method
        def deal_hands(p1, p2):
            """definition"""
```

1.5 Exercise 18.4

Read **TurtleWorld.py**, **World.py** and **Gui.py** and draw a class diagram that shows the relationships among the classes defined there.

1.6 Exercise 18.5

. . . DATA ENCAPSULATION. . .

1.7 Exercise 18.6

Poker card game
http://thinkpython.com/code/PokerHandSoln.py

1.8 Exercise 18.7

Uses TurtleWorld form Chapter 4 to write code that makes Turtles to play tag (https://en.wikipedia.org/wiki/Tag_(game))
http://thinkpython.com/code/Tagger.py