02_classes

July 31, 2019

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

```
In [ ]: NAME = "PETROVICI STEFAN"
        COLLABORATORS = "?"
```

---

1 1. Circle Class (2 points)

The goal is to create a class that represents a simple circle.

A Circle can be defined by either specifying the radius or the diameter, and the user can query the circle for either its radius or diameter.

Other abilities of a Circle instance:

- Compute the circle's area
- Print the circle and get something nice
- Be able to add two circles together
- Be able to compare two circles to see which is bigger
- Be able to compare to see if there are equal
- (follows from above) be able to put them in a list and sort them

You will also use:

- properties
- a classmethod
- a define a bunch of "special methods"

1.1 Step 1:

create class called `Circle` -- it's signature should look like::

c = Circle(the_radius)

The radius is a required parameter (can't have a circle without one!)

the resulting circle should have a attribute for the radius::

c.radius

So you can do:

```
>> c = Circle(4)
>> print c.radius
4
```

1.2 Step 2:

Add a "diameter" property, so the user can get the diameter of the circle:

```
>> c = Circle(4)
>> print c.diameter
8
```

1.3 Step 3:

Set up the diameter property so that the user can set the diameter of the circle:

```
>> c = Circle(4)
>> c.diameter = 2
>> print c.diameter
2
>> print c.radius
1
```

NOTE that the radius has changed! Look at `@property` decorator

1.4 Step 4:

Add an `area` property so the user can get the area of the circle:

```
>> c = Circle(2)
>> print c.area
12.566370
```

(`pi` can be found in the math module)
The user should not be able to set the area:

```
>> c = Circle(2)
>> c.area = 42
AttributeError
```

1.5 Step 5:

Add an "alternate constructor" that lets the user create a Circle directly with the diameter:

```
>> c = Circle.from_diameter(8)
>> print c.diameter
8
>> print c.radius
4
```

NOTE: Look at `@classmethod`

1.6 Step 6:

Add **str** and **repr** methods to your Circle class.

Now you can print it:

```
In [2]: c = Circle(4)

In [3]: print c
Circle with radius: 4.000000

In [4]: repr(c)
Out[4]: 'Circle(4)'

In [5]: d = eval(repr(c))

In [6]: d
Out[6]: Circle(4)
```

1.7 Step 7:

Add some of the numeric protocol to your Circle:

You should be able to add two circles:

```
In [7]: c1 = Circle(2)

In [8]: c2 = Circle(4)

In [9]: c1 + c2
Out[9]: Circle(6)
```

and multiply one times a number:

```
In [16]: c2 * 3
Out[16]: Circle(12)
```

(what happens with `3 * c2` ? -- can you fix that?)

1.8 Step 8:

add the ability to compare two circles:

```
In [10]: c1 > c2
Out[10]: False

In [11]: c1 < c2
Out[11]: True

In [12]: c1 == c2
Out[12]: False
```

```
In [13]: c3 = Circle(4)

In [14]: c2 == c3
Out[14]: True
```

Once the comparing is done, you should be able to sort a list of circles:

```
In [18]: print circles
[Circle(6), Circle(7), Circle(8), Circle(4), Circle(0), Circle(2), Circle(3), Circle(5), Circle(

In [19]: circl
circle      circle.py   circle.pyc  circles

In [19]: circles.sort()

In [20]: print circles
[Circle(0), Circle(1), Circle(2), Circle(3), Circle(4), Circle(5), Circle(6), Circle(7), Circle(
```

```
In [126]: import math

          class Circle:

              def __init__(self, radius):
                  self.radius = radius
                  self._diameter = 2*radius
                  self._area = math.pi*radius*radius

              @property
              def diameter(self):
                  """I'm the diameter property."""
                  return self._diameter

              @diameter.setter
              def diameter(self, value):
                  self._diameter = value
                  self.radius = int(value/2)

              @diameter.deleter
              def diameter(self):
                  del self._diameter

              @property
              def area(self):
                  """I'm the area property."""
                  return self._area

              @area.setter
```

```
        def area(self, value):
            raise AttributeError

        @area.deleter
        def area(self):
            del self.area

        @classmethod
        def from_diameter(self,diam):
            return Circle(int(diam/2))

        def __str__(self):
            return str("Circle with radius ") + str(self.radius)

        def __repr__(self):
            return str("Circle({})".format(self.radius))

        def __add__(self, o):
            return Circle(self.radius + o.radius)

        def __mul__(self, n):
            return Circle(self.radius*int(n))

        def __rmul__(self, n):
            return Circle(self.radius*int(n))

        def __lt__(self, o):
            return self.radius < o.radius

        def __et__(self, o):
            return self.radius == o.radius

        def __gt__(self, o):
            return self.radius > o.radius
```

```
In [127]: c = Circle(4)
          print(c.radius)
```

```
4
```

```
In [128]: print(c.diameter)
```

```
8
```

```
In [129]: c = Circle(4)
          c.diameter = 2
```

```
         print(c.diameter)
         print(c.radius)

2
1


In [130]: del c

In [131]: c = Circle(2)
          print(c.area)

12.566370614359172


In [132]: c.area = 42


        ---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call last)

        <ipython-input-132-f90876d4d96b> in <module>()
    ----> 1 c.area = 42


        <ipython-input-126-c7054f2fffc5> in area(self, value)
         29     @area.setter
         30     def area(self, value):
    ---> 31         raise AttributeError
         32
         33     @area.deleter


        AttributeError:


In [133]: c = Circle.from_diameter(8)
          print(c.diameter)
          print(c.radius)

8
4


In [134]: print(c)
          repr(c)
          e = eval(repr(c))
          e
```

```
Circle with radius 4


Out[134]: Circle(4)

In [135]: c1 = Circle(2)
          c2 = Circle(4)

          c1 + c2

Out[135]: Circle(6)

In [136]: c2 * 3

Out[136]: Circle(12)

In [124]: 3 * c2

Out[124]: Circle(12)

In [137]: c1 > c2

Out[137]: False

In [138]: c1 < c2

Out[138]: True

In [139]: c1 == c2

Out[139]: False

In [140]: circles = [Circle(6), Circle(7), Circle(8), Circle(4), Circle(0), Circle(2), Circle(3)
          circles.sort()
          print(circles)

[Circle(0), Circle(1), Circle(2), Circle(3), Circle(4), Circle(5), Circle(6), Circle(7), Circle(
```

2 2. Sparse Array (2 points)

Oftentimes, at least in computation programming, we have large arrays of data that hold mostly zeros.

These are referred to as "sparse" as the information in them is widely scattered, or sparse.

Since they are mostly zeros, it can be memory and computationally efficient to store only the value that are non-zero.

But you want it to look like a regular array in user code.

In the real world, these are usually 2 dimensional arrays. But to keep it a bit simpler, we'll make a 1 dimensional sparse array in this class.

(feel free to make it 2d for an extra challenge!)

2.1 A Sparse array class

A spare array class should present to the user the same interface as a regular list.
Some ideas of how to do that:

- Internally, it can store the values in a dict, with the index as the keys. So that only the indexes with non-zero values will be stored.
- It should take a sequence of values as an initializer:

```
sa = SparseArray([1,2,0,0,0,0,3,0,0,4])
```

- you should be able to tell how long it is:

```
len(my_array)
```

This will give its "virtual" length -- with the zeros

- It should support getting and setting particular elements via indexing:

```
sa[5] = 12
sa[3] = 0 # the zero won't get stored!
val = sa[13] # it should get a zero if not set
```

- It should support deleting an element by index:

```
del sa[4]
```

- It should raise an `IndexError` if you try to access an index beyond the end.
- it should have an append() method
- Can you make it support slicing?
- How else can you make it like a list?

```
In [10]: my_array = SparseArray( (1,0,0,0,2,0,0,0,5) )
In [11]: my_array[4]
Out[11]: 2
In [12]: my_array[2]
Out[12]: 0
```

```
In [1]: class SparseArray:

            def __init__(self, *list):
                self.array = dict()
                self.length = len(*list)
                for idx, val in enumerate(*list):
                    if val != 0:
                        self.array[idx] = val

            def __len__(self):
```

```
        return self.length

    def __setitem__(self, key, value):
        if value == 0:
            raise ValueError
        self.array[key] = value

    def __getitem__(self, key):
        if key > max(k for k, v in self.array.items()):
            raise IndexError
        if self.array.get(key) == None:
            return 0
        return self.array[key]

    def __delitem__(self, key):
        if self.array.get(key) != None:
            del self.array[key]

    def to_array(self):
        newArray = [0]*self.length
        for i,v in self.array.items():
            newArray[i] = v
        return newArray

    def append(self, *list):
        newList = self.to_array()
        newList.extend(*list)
        self.__init__(newList)

    def slice(self, idx):
        newList = self.to_array()
        list1 = newList[:idx]
        list2 = newList[idx:]
        return SparseArray(list1), SparseArray(list2)


sa = SparseArray([1, 2, 0, 0, 0, 0, 3, 0, 0, 4])
print("Inner array:", end="")
print(sa.array)
print("Length:%d" % len(sa))

print("sa[5]=%d" % sa[5])
sa[5] = 12
print("put 12 in sa[5]=%d" % sa[5])

print("sa[1]=%d" % sa[1])
del sa[1]
print("del 1 sa[1]=%d" % sa[1])
```

```
        #print(sa[20]) -> Index error

        sa.append([6, 5])
        print("Append array:", end="")
        print(sa.array)
        print("Length:%d" % len(sa))

        l1, l2 = sa.slice(7)
        print("L1:", end="")
        print(l1.array)
        print("L2:", end="")
        print(l2.array)
```

```
Inner array:{0: 1, 1: 2, 6: 3, 9: 4}
Length:10
sa[5]=0
put 12 in sa[5]=12
sa[1]=2
del 1 sa[1]=0
Append array:{0: 1, 5: 12, 6: 3, 9: 4, 10: 6, 11: 5}
Length:12
L1:{0: 1, 5: 12, 6: 3}
L2:{2: 4, 3: 6, 4: 5}
```

3 3. Deck of cards (2 points)

Create 3 classes emulating a deck of cards and a player interacting with it.

3.1 Step 1: Prepare our classes:

We will have three classes:

- A class Card, class Deck and class Player .

3.2 Step 2: Create our class Card:

The Card is going to take a suit and value self.
We're going to create the attributes suit, value and set them equal to whatever we pass in when we create a card.

We create another method to show the card. In the `show` method we want to print out the card, so we make a string that will print out the value and the suit.

3.3 Step 3: Create our class Deck:

Create a build method which takes in self, and create the 52 cards of 4 suits.
"suits" :["Spades", "Clubs", "Diamonds", "Hearts"]

3.4 Step 4: Create a Shuffle and Draw Card method:

Create a shuffle method using `random` and a `draw_card` method that removes a card from the deck and returns it.

3.5 Step 5: Create Player:

Lastly, we create a class Player with a name attribute set to name and a hand attribute set to an empty list. Next we create a draw method that takes in self and a deck in which we will pass in a deck. We append Deck.drawCard() to the self.hand and return self for the card drawn.

```
In [35]: from enum import Enum
         import random

         class Suits(Enum):
             Spades = 1
             Clubs = 2
             Diamonds = 3
             Hearts = 4

         class CardValues(Enum):
             Ace = 1
             Two = 2
             Three = 3
             Four = 4
             Five = 5
             Six = 6
             Seven = 7
             Eight = 8
             Nine = 9
             Ten = 10
             Jack = 11
             Queen = 12
             King = 13

         class Card:

             def __init__(self,s,v):
                 self.suit = s
                 self.value = v
             def show(self):
                 print(self.value.name + " of " + self.suit.name)

         #a = Card(Suits.Spades, CardValues.Ace)
         #a.show()

         class Deck:

             def build(self):
```

```
        self.cards = [Card(s,v) for s in Suits for v in CardValues]
    def shuffle(self):
        random.shuffle(self.cards)
    def draw_card(self):
        return self.cards.pop(-1)

d = Deck()
d.build()
d.shuffle()
#c = d.draw_card()
#for x in d.cards:
#     x.show()
#print("x=")
#c.show()

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

player = Player("Eu")
player.draw(d)
for x in player.hand:
    x.show()
```

```
Two of Hearts
```

4 4. Simulate a zoo (3 points)

You are a zoo keeper. Write a program in Python that simulates a simple zoo, using object oriented programming. You should be able to do the following:

- Create different cages in the zoo. At any time, you should be able to find out how many cages are in the zoo.
- Put different animals in the cages.
- Each animal should be of a particular species (e.g. ’Lion’), and have a name given to them by the zookeeper (e.g. ’Growler’).
- Find out which animals are in a particular cage.
- Some species of animals like eating other species of animals. For example, lions like eating wildebeest. If you put prey and predator in the same cage, then all the prey should be eaten by the predator. (The program should tell you which predator ate which prey by printing to the console.)

HINTS: Create different classes for each animal and a prey list containing each animal’s prey.

4.0.1 Create different cages

```
>>> cage_1 = Cage()
>>> london_zoo.cages = [cage_1, cage_7, cage_2, cage_5] # Populate Zoo with cages
Find out how many cages are in the zoo
>>> london_zoo.n_of_cages()
4
```

4.0.2 Place different animals (of particular species) in cages

```
>>> snake_3 = Snake()
>>> snake_5 = Snake()
>>> cage_5.add_animals(snake_3, snake_5)
>>> cage_5.animals_inside()
Animals in this cage: ['Unnamed Coyote', 'Flora Snake', 'Unnamed Snake', 'Unnamed Snake']
Be able to name the animals
>>> lion_2.name = 'Leo'
>>> snake_1.name = 'Flora'
```

4.0.3 List animals inside a cage

```
>>> cage_5.animals_inside()
# Does not return the Animal objects inside the cage, but a list of strings related to them as a
Animals inside this cage: ['Unnamed Coyote', 'Flora Snake', 'Unnamed Snake']
>>> cage_6.animals_inside()
This cage is empty.
Repopulating a cage
>>> del cage_5.animals[2]
>>> cage_5.animals_inside()
Animals inside this cage: ['Unnamed Coyote', 'Flora Snake']
>>> cage_5.add_animals(racoon_3)
```

4.0.4 Make preys get eaten by predators if in same cage

```
>>> cage_5.add_animals(coyote_3, lion_1)
Oops! Seems like you put predator and prey in the same cage.
Yves Coyote was eaten by Leo Lion.
Animals in this cage: ['Unnamed Falcon', 'Leo Lion']
```

5 5. Simulate a game of blackjack (5 points)

Extend your deck classes and simulate a card game of black-jack.

1. Create a deck of 52 cards
2. Shuffle the deck
3. Ask the Player for their bet
4. Make sure that the Player's bet does not exceed their available chips
5. Deal two cards to the Dealer and two cards to the Player

6. Show only one of the Dealer's cards, the other remains hidden
7. Show both of the Player's cards
8. Ask the Player if they wish to Hit, and take another card
9. If the Player's hand doesn't Bust (go over 21), ask if they'd like to Hit again.
10. If a Player Stands, play the Dealer's hand. The dealer will always Hit until the Dealer's value meets or exceeds 17
11. Determine the winner and adjust the Player's chips accordingly
12. Ask the Player if they'd like to play again

5.1 Step 1. Modify the card class

Remember that in blackjack Queen, Jack, and King are 10 and Ace is 1 or 11.

5.2 Step 2. Add a Hand class

In addition to holding Card objects dealt from the Deck, the Hand class may be used to calculate the value of those cards using the values dictionary defined above. It may also need to adjust for the value of Aces when appropriate.

5.3 Step 3: Create a Chips Class

To keep track of a Player's starting chips, bets, and ongoing winnings.

5.4 Step 4: Modify player class

Create the methods:

- hit -> add card to hand
- get_hand_val -> get value of hand
- is_busted -> check if player is bust
- lose -> subtract ammount from chips
- win -> add ammount to chips

Modify the player class to contain a hand and chips, if the player is a dealer the total ammount of chips is infinite. At each turn the human player should be asked if he would like to hit or stand, the dealer will always hit untill he is at >= 17.

5.5 Step 5: Create the game loop

Create the game loop and show the player options and print the dealers shown card. After the game loop the player should be asked if he would like to continue or reset the chips.

```
In [ ]: class BlackjackCardValues(Enum):
            Ace = 1
            Two = 2
            Three = 3
            Four = 4
            Five = 5
            Six = 6
```

```
    Seven = 7
    Eight = 8
    Nine = 9
    Ten = 10
    Jack = 10
    Queen = 10
    King = 10

class BlackjackDeck(Deck):
    def build(self):
        self.cards = [Card(s,v) for s in Suits for v in BlackjackCardValues]

class BlackjackPlayer(Player):
    def __init__(self, nam, monei):
        super(BlackjackPlayer, self).__init__(nam)
        self.money = monei

gambler = BlackjackPlayer("Salam", 99)
dealer = Player("Dealer")

while gambler.money > 0:

    bet = int(input(">>Enter your bet: "))
    if bet > gambler.money:
        print(">>Wrong amount")
        continue

    gambler.money -= bet

    deck = BlackjackDeck()
    deck.build()
    deck.shuffle()

    gambler.hand = []
    dealer.hand = []

    gambler.draw(deck)
    dealer.draw(deck)
    gambler.draw(deck)
    dealer.draw(deck)

    turn = 0
    while True:
        print(">>Your hand:")
        for c in gambler.hand:
            c.show()
        print(">>Dealer hand:")
        if turn == 0:
            dealer.hand[0].show()
            print("???")
        else:
            for c in dealer.hand:
                c.show()

        if turn == 0:
            pScore = 0
            for c in gambler.hand:
                pScore += c.value.value
            if pScore > 21:
                print(">>You lost")
                break

        if turn == 1:
            dScore = 0
            for c in dealer.hand:
                dScore += c.value.value
            if dScore > 21:
                print(">>Dealer lost")
                gambler.money += 2 * bet
                break
            elif dScore > pScore:
                print(">>You lost")
                break
            else:
                print(">>Dealer has to draw card")
                dealer.draw(deck)
                continue

        choice = input(">>Do you want to hit?(y/n)")
        if choice == "y":
            gambler.draw(deck)
        else:
            print(">>You choose to stand")
            turn = 1

#functioneaza game-loop-ul, doar ca nu este implementata functionalitatea Asului
```