Just two problems on this last problem set! Don't forget to fill out the metadata at the top of each problem please!

Get Assignment link: https://classroom.github.com/a/7DPqpmTK

1. (8 points) When originally learning to work with classes, I found it useful to work with objects which were very tactile and which I could easily envision defining an "object" for. To that end, in this problem you will be working with playing cards. I have already provided code to you which defines a `Card` class, which stores both a rank and a suit. Internally, both of these values are stored as numbers, but you will notice when you print a card to the screen that those numbers are converted to characters that better describe a typical playing card. Internally, a rank of 1 corresponds to an Ace, 2–10 corresponds to the number cards, 11 corresponds to a Jack, 12 to a Queen, and 13 to a King. Suits range from 0 to 3 and correspond to club, diamond, heart, and spade, respectively. Getter functions are provided for both rank and suit that will return to you their numeric representation. Make sure you understand what is happening in this class before continuing.

   Your task in this problem is to define a new class to represent an entire *deck* of playing cards. Your standard deck of playing cards has 52 cards, with Ace through King in each of the 4 suits. You will be importing and using the `Card` class I already defined for you to represent the individual cards comprising your `Deck`. The primary (and only) attribute of your `Deck` will be a list of all the cards in the deck, and then you will have several methods to support the types of actions you would commonly want to make with a deck of cards. In particular, your `Deck` class should include:

   - A constructor which takes no extra arguments and which initializes a deck of the standard 52 playing cards whenever a new deck object is created. This can be done by appending the necessary 52 card objects to your deck list attribute. Loops are your friend here.
   - A `shuffle` method with will shuffle or randomize the order of the cards currently within your deck. (The random class has a shuffle method that should make this pretty straightforward.)
   - A `draw` method which takes one argument $n$ and draws the "top" $n$ cards from your deck, returning a list of those cards. Here we are defining the "top" of the deck to be the beginning of the list of cards. The cards which were drawn should also be removed from the deck so that subsequent calls to `draw` will draw new cards.
   - A getter method called `get_deck` which returns a *copy* of the current list of cards in the deck.
   - An `__str__` or `__repr__` method which prints out a nice string representation of the cards still in the deck. Note that you may need to loop through and call `str` directly on the individual cards to have them display correctly as strings. Using the `join` string method may also be useful.

Creating and using some of these methods of your `Deck` class may result in output similar to below:

```
>>> D = Deck()
>>> print(D)
[A♣, 2♣, 3♣, 4♣, 5♣, 6♣, 7♣, 8♣, 9♣, T♣, J♣, Q♣, K♣, A♦, 2♦, 3♦, 4♦, 5♦, 6♦,
 7♦, 8♦, 9♦, T♦, J♦, Q♦, K♦, A♠, 2♠, 3♠, 4♠, 5♠, 6♠, 7♠, 8♠, 9♠, T♠, J♠, Q♠,
 K♠, A♥, 2♥, 3♥, 4♥, 5♥, 6♥, 7♥, 8♥, 9♥, T♥, J♥, Q♥, K♥]
>>> D.shuffle()
>>> print(D)
[K♦, 9♥, A♠, A♦, 7♥, Q♦, 2♦, 3♠, 2♠, J♥, K♠, Q♥, 7♣, J♣, 2♥, 8♥, 4♠, K♣, 7♦,
 8♠, Q♣, 3♥, 4♣, T♣, 8♦, T♠, 6♦, A♥, 5♥, A♣, 4♦, J♠, 5♠, 9♦, 9♣, 9♠, 8♣, 2♣,
 6♣, 7♠, 3♦, 4♥, K♥, 3♣, J♦, 5♣, 6♠, 5♦, 6♥, Q♠, T♥, T♦]
>>> print(D.draw(3))
[K♦, 9♥, A♠]
>>> print(D)
[A♦, 7♥, Q♦, 2♦, 3♠, 2♠, J♥, K♠, Q♥, 7♣, J♣, 2♥, 8♥, 4♠, K♣, 7♦, 8♠, Q♣, 3♥,
 4♣, T♣, 8♦, T♠, 6♦, A♥, 5♥, A♣, 4♦, J♠, 5♠, 9♦, 9♣, 9♠, 8♣, 2♣, 6♣, 7♠, 3♦,
 4♥, K♥, 3♣, J♦, 5♣, 6♠, 5♦, 6♥, Q♠, T♥, T♦]
>>> print(len(D.get_deck()))
49
```

Your `Prob1.py` file should **also** include a test program that uses `assert` statements to check that your methods all work correctly. You can look at the code for `Card.py` for a model.

2. (8 points) Before the collapse of the trading company FTX in November, a large number of small investors invested in cryptocurrencies like Bitcoin because of the rapidly rising market. Those investments lost most of their value as the sector reeled from the FTX bankruptcy. Your job in this problem is to write a Python program that lets users determine how much they lost (or in rare cases gained) from their transactions.

   You have access to a large data file called `CrypocurrencyTradingData.txt` that is formatted identically to the below snippet. The actual file has many more entries for other dates and currencies than are shown below.

   CryptocurrencyTradingData.txt

```
1-Dec-22
Bitcoin: 16967.13
Binance: 1276.27
DogeCoin: 0.10160
1-Sep-22
Bitcoin: 20127.14
Binance: 1586.18
DogeCoin: 0.6237
1-Jun-22
Bitcoin: 29799.08
Binance: 1823.57
DogeCoin: 0.08106
1-Mar-22
Bitcoin: 44354.64
Binance: 2972.49
DogeCoin: 0.13390
```

   The file is grouped into sections by date, and each section has entries consisting of a currency name, a colon, and then a dollar value of the currency on that date. For example, if you had purchased one Bitcoin on 1-Mar-22, you would have paid $44,354.64 to get it. Had you then sold it on 1-Dec-22, you would have received only $16,967.13, for a loss of $27,387.51. Yikes! Conversely, if you had been foresighted enough to buy 1000 DogeCoins on 1-Jun-22, and then sold them on 1-Dec-22, you would have realized a tidy profit of $20.54, which is $1000 \times (0.10160 - 0.08106)$.

   Write a program that reads the data file into a suitable internal structure and then asks the user for the following inputs:

   - The name of the currency
   - The number of units purchased
   - The date at which the currency was bought, formatted exactly as it is in the file
   - The date at which the currency was sold

Your program should then print out the profit or loss on the transaction.

The below images show two sample runs of this program, where blue text indicates text that the user typed in when prompted:

```
〉 python Prob2.py

Currency: Bitcoin
Units purchased: 1
Purchase date: 1-Mar-22
Sell date: 1-Dec-22
Net loss = $27,387.51
```

```
〉 python Prob2.py

Currency: DogeCoin
Units purchased: 1000
Purchase date: 1-Jun-22
Sell date: 1-Dec-22
Net profit = $20.54
```

In solving this problem, you should keep the following points in mind:

- The primary focus of this problem is reading the data file into an internal code structure
- The unique key that identifies a particular value is a combination of currency name and the date, so it makes sense to have your data structure use this combination.
- You don't need to respond to any error conditions. In particular, you may assume that the data file is formatted correctly and that the strings that the user types (the currency names and dates) are formatted exactly as they are in the file.
- You should use the features of *f*-strings to form the number printed in the net loss/profit line so that it uses a comma to separate the number into three-digit groups and displays two digits after the decimal point, as shown in the example runs.