Lab 3

Python: Classes (card game)

[Compulsory]

Authors: Ragnar Nohre, moved to English Johannes Schmidt

This lab's goal is to exercise on classes and objects in python. The theoretical background is found in the *web-tutorial chapter 9*.

3.1 Card game

In this task you shall implement two classes, one card-class and one card-game-class. In practice you shall type all the skeleton code given in this lab and replace the `pass`-instructions with your own code.

3.1.1 Card

As commonly known, a card game has a *suit* (spades, hearts, diamonds, clubs) and a *rank* or *value* (two, three, ..., nine, ten, jack, queen, king, ace). Every card object stores its suit and value as *two integers* and *not* as strings[1]!
Below you see the skeleton code for the class `Card`. In your own implementation you shall replace the `pass`-statements with real code.

```
class Card:
    def __init__(self, suit, value):
        assert  1 <= suit <= 4 and 1 <= value <= 13
        self._suit = suit
```

[1] It would be even nicer to store the suit as an *enum*.

```
        self._value = value

    def getValue(self):
        pass

    def getSuit(self):
        pass

    def __str__(self):
        pass
```

Some explanation:

1. The `pass`-instructions don't do anything, but we need them to indicate an empty block (you can not write `{ }` in python).
2. The method `__init__` is the constructor that takes suit and value as parameters. With the assert-statement we ensure that these parameters have valid values, otherwise the program will abort with an error message.
3. The method `getValue()` shall *return* the card's value as an integer between 1 and 13.
4. `getSuit()` shall *return* the card's suit as an integer between 1 and 4.
5. The method `__str__()` shall return a string that describes the card. For instance *Queen of Hearts* or *Two of Clubs*. This method will be called automatically every time python needs to convert a card to a string.

Note that it would be wrong if any of the above functions printed anything to the screen (with print).

3.1.2 CardDeck

A card game has 52 cards at the beginning. We want to be able to shuffle the cards, remove cards, determine how many cards remain, and reset the game to 52 cards again.
Bellow you find the skeleton code for the class `CardDeck`. In your own implementation you shall replace the `pass`-statements with real code.

```
class CardDeck:
```

```
    def __init__(self):
        pass

    def shuffle(self):
        pass

    def getCard(self):
        pass

    def size(self):
        pass

    def reset(self):
        pass
```

Short explanation:

1. The constructor should create a list and populate it with 52 diffrent cards. The function `reset` should do the same, so it could be appropirate if the constructor calls `reset()`.
2. The method `shuffle()` shall shuffle the cards. The easiest way to achieve this is to iterate over all positions in the list and swap the content with a random position. It might also be that the `list`-class has a built-in method to shuffle.
3. The method `getCard()` works similar to pop. It shall reduce the list by one card and return that card.
4. The method `reset()` shall reset the game to the initial state, that is, all 52 cards are available again in order.

Note again that it would be wrong if any of the above functions printed anything to the screen (with print).

3.1.3 Test code

Write also test code that creates a card game, shuffles it, prints all cards to the screen. The following test code must work.

```
deck = CardDeck()
deck.shuffle()
while deck.size()>0:
     card = deck.getCard()
     print("Card {} has value {}".format(card, card.getValue()))
```

3.2 Examination

Submit your python file on Canvas. It must include all test code and skeleton code. Present your solution to your lab assistant.