Lab 3

[Obligatorisk] Python: Klasser (Kortlek)

Syftet med denna laboration är att öva på klasser och objekt i programspråket python. Den teoretiska bakgrunden finns i *python tutorial, kapitel 9.*

3.1 Kortlek

Denna uppgift går ut på att implementera två klasser, en spelkortsklass och en kortleksklass. I praktiken skall du knappa in all skelettkod som finns återgiven i detta lab-pm, och byta ut `pass`-instruktionerna mot din egen kod.

3.1.1 Card

Ett spelkort har som bekant en s.k. *färg (eng. suite)* (hjärter, ruter, spader , klöver) och ett värde (två tre,. . . , nio,tio, knekt, dam, kung, ess). Varje spelkortsobjekt lagrar färg och värde som *två heltal* och *inte* som strängar[1]! Nedan ser vi skelettkoden för klassen `Card`. I din egen implementerang skall du (naturligtvis) byta ut `pass` mot riktig kod.

```
class Card:
    def __init__(self, suite, value):
        assert  1 <= suite <= 4 and 1 <= value <= 13
        self._suite = suite
        self._value = value
```

[1]Det hade varit snyggare att lagra färgen som en *enum*.

```
    def getValue(self):
        pass

    def getSuite(self):
        pass

    def __str__(self):
        pass
```

Kort förklaring:

1. `pass`-instruktionerna gör ingenting, men vi måste ha dem för att indikera ett tomt block (man kan ju inte skriva `{ }` i python).
2. Metoden `__init__` är konstruktören som tar suite och value som parametrar. Med assert-satsen får vi bekräftat att dessa parametrar har giltiga värden, annars kraschar programmet.
3. Metoden `getValue()` skall *returnera* kortets värde som ett heltal mellan 1 och 13.
4. `getSuite()` skall *returnera* spelkortets färg som ett heltal mellan 1 och 4.
5. Metoden `__str__()` skall returnera en sträng som beskriver spelkortet. Exempelvis *Hjärter Dam* eller *Klöver två*. Denna metod kommer att anropas automatiskt varje gång python behöver konvertera ett spelkort till en sträng.

Observera att det vore direkt felaktigt att låta någon av ovanstående funktioner skriva ut något på skärmen (med print).

3.1.2 CardDeck

En kortlek rymmer från början 52 olika kort. Man skall kunna blanda kortleken, ta kort från leken, undersöka hur många kort som finns kvar, samt återställa kortleken så att den återigen blir full.
Nedan ser vi skelettkod för klassen `CardDeck`

```
class CardDeck:
    def __init__(self):
```

```
        pass

    def shuffle(self):
        pass

    def getCard(self):
        pass

    def size(self):
        pass

    def reset(self):
        pass
```

Kort förklaring:

1. Kortlekens konstruktor bör skapa en lista och poppulera denna med 52 olika kort. Samma sak bör reset göra, så det kan eventuellt vara lämpligt att låta konstruktorn anropa `reset()`.
2. Metoden `shuffle()` skall blanda leken. Enklast gör man detta genom att iterera igenom alla positioner i listan och byta innehållet mot innehållet i en slumpmässig position. Det kan också hända att `list`-klassen har en inbyggd metod som blandar.
3. Metoden *getCard()* fungerar ungefär som pop. Den skall reducera listan med ett kort och också returnera detta kort.
4. Metoden `reset()` skall återställa kortleken så att den återigen får 52 kort i ordning.

Observera återigen att det vore direkt felaktigt att låta någon av ovanstående funktioner skriva ut något på skärmen (med print).

3.1.3 Testkod

Skriv också testkod som skapar en kortlek, blandar den, och skriver ut alla kort på skärmen. Nedanstående testkod måste fungera.

```
deck = CardDeck()
deck.shuffle()
while deck.size()>0:
     card = deck.getCard();
     print( "Kortet {} har värdet {}".format( card, card.getValue() )
```

3.2 Redovisning

Ladda upp din pythonfil till pingpong. Observera att all skelettkod inklusive testkod måste finnas med. Redovisa sedan muntligt under labtillfället.