Københavns Universitet
PoP Assignment 4 - Adressed Feedback

Victor Vangkilde Jørgensen - kft410

January 31, 2025

Contents

1 Feedback on Question 2

2B - Forklar hvorfor du har valgt specifikt disse 3 måder.

Jeg har uddybet med:
"Jeg har valgt disse 3 måder at repræsentere farver på, at det var dem jeg højst sandsynligt selv ville havde benyttet, hvis det var mig der havde lavet colour.
Jeg kan især godt lide RGB-listen, da det giver mulighed for at arbejde med mere præcise farver, og kanExtendColour ville kunne bruges til at sammenligne farver, der er meget tæt på hinanden."

2C - Du skal ikke definere en NeighbourRelation klasse, men en Country klasse. Dvs. klassen skal repræsentere et land, og land skal opbevare en liste over landets naboer.

Jeg har rettet min repræsentation til følgende:

```
1  class Country:
2      """
3      Represents a country with a name and a list of neighboring
           countries.
4      """
5      def __init__(self, countryName: str):
6          """
7          Starts with a Country with the given name and an empty list of
               neighbors.
8
9          Inputs:
10             str: countryName with the name of the country.
11         """
12         self.countryName = countryName
13         self.neighbours = []
14
15     def addNeighbour(self, neighbour):
16         """
17         Adds a neighbor Country to this country's list of neighbors.
18
19         Inputs:
20             Country: the Country object to be added as a neighbor.
21         """
22         if neighbour not in self.neighbours:
23             self.neighbours.append(neighbour)
24             neighbour.addNeighbour(self)
25
26     def __repr__(self) -> str:
27         """
28         Returns a string of the Country.
29
30         Outputs:
31             str: with "CountryName: [Neighbour1, Neighbour2, ...]".
32         """
33         return f"{self.countryName}: {[neighbour.countryName for
               neighbour in self.neighbours]}"
34
35 de = Country("de")
36 da = Country("da")
```

```
37 se = Country("se")
38 no = Country("no")
39
40 # Not sure if 'no' and 'da' should be neighbours
41 de.addNeighbour(da)
42 da.addNeighbour(se)
43 se.addNeighbour(no)
44
45 print(de)
46 print(da)
47 print(se)
48 print(no)
49 # Output:
50 de: ['da']
51 da: ['de', 'se']
52 se: ['da', 'no']
53 no: ['se']
```

2 Feedback on question 3

3A - Du mangler at forklare hvordan du har fulgt Kens metode.

Jeg har uddybet, hvordan jeg har fulgt Kens method i rapporten.

3B - Forklar mere specifikt hvordan du bruger paradigmet/paradigmerne.

Min besvarelse er nu som følgende:
"Funktionel programmering fylder en del, da det er det, jeg bruger til at læse bogstavrækkerne og sortere dem i forskellige lister.
Imperativ programmering bruges til at tælle mængden af "DIKU" i hvert af disse lister, men fylder ud over dette ikke meget.
Nu når jeg ser tilbage på opgaven, bør jeg nok havde brugt imperativ programmering i højere grad, da jeg fra forelæsningen havde hørt, at er bedst til små opgaver. Jeg havde dog troet, at funktionel programmering ville være bedst, da det skulle være bedst, når man har meget data, hvilket ikke var tilfældet i denne opgave i samme grad, som jeg havde troet."

3C - Du skal teste programmet lidt mere eksplicit og inkludere testene.

Jeg har forbedret min metode jeg tester på, ved at blandt andet lave en funktion, der raiser en exception hvis inputten laver en specification violation.
Derudover har jeg lavet nogle test-filer, som indeholde nogle af test case'ne.

3C - Tænk mere over hvordan de forskellige inputs kunne se ud.

Jeg har lavet nogle mere repræsentative test cases, der bliver partitioned af den lavede test-function. Disse cases tester nogle af de nævnte måder man kan 'læse' DIKU på, som nævnt i opgaven.

3C - Representative inputs are not defined.

Jeg henviser til svaret ovenover

3 Feedback on question 4

4A - Programmet skal beregne det totale antal point for alle kortene tilsammen.

Jeg misfortolkede opgaven. Dette er rettet nu.

4A - Forklar gerne hvordan du har fulgt Kens metode her.

Jeg har givet samme gennemgang af Kens method som tidligere.

4B - Card klassen i din kode dominerer ret meget, da næsten alle funktioner er lavet som metoder i klassen. Genovervej hvilket paradigme dominerer her.

Jeg var klar over, at OOP fyldte mest, men havde glemt, at skrive det, da jeg opfattede members'ne som en del af funktionel programmering.

4C
- Du skal teste programmet mere eksplicit.
- Forklar hvordan du har opdelt inputtet.
- Input partitions are not correctly defined.
- Representative inputs are not defined.

Jeg er ikke helt sikker på, hvad der menes med 'eksplicit' i denne sammenhæng, men jeg har forsøgt, at indkludere mine tests mere.
Jeg har forsøgt, at lave mine tests, så de indkluderer de features, der skal brues til at løse opgaven. Som der nu forklares i min rapport, testes der for 'Card.cardPoints', da dette er member'en, vi bruger til at beregne det samlede antal points for hvert kort.
Der er en ny funktion, der raiser en exception, hvis inputten, som bliver kørt i 'Card.cardPoints', er ugyldig.
Den nye måde der testes:

```
1  # Test function
2  def testFun(testCase):
3      """
4      Runs testcases for the member 'cardPoints' in the class Card.
5
6      Inputs:
7          list[str]: of a card.
8
9      Outputs:
10         int: point amount for the card, or exception if input invalid.
11     """
12     try:
13         return Card(testCase).cardPoints()
14     except:
15         return Exception
16
17 # Test cases
18 print("Test Cases:")
19 print(testFun("Card 1: 41 48 83 86 17 | 83 86 6 31 17 9 48 53")) # Test
       Case 1: Input from assignment. Expected Output: 8
```

```
20 print(testFun("Card 2: 7 8 9 | 7 8 9")) # Test Case 2: Header cut into
       numbers. Expected Output: 1
21 print(testFun("Card 100: 1 2 3 | 4 5 6")) # Test Case 3: Long header
       that is cut short. Expected Output: 0
22 print(testFun("Card 9:")) # Test Case 4: Empty card. Expected Output: 0
23 print(testFun("Card 123: 10 20 | 10 20")) # Test Case 5: Dubble numbers
        After removing 'Card'. Expected Output: 1
24 print(testFun("Card 4: | 1 2 3")) # Test Case 6: No winning numbers.
       Expected Output: 0
25 print(testFun("Card 5: 1 2 3 |")) # Test Case 7: No your numbers.
       Expected Output: 0
26 print(testFun("Card 6: 5 | 5 5 5")) # Test Case 8: Duplicate numbers.
       Expected Output: 4
27 print(testFun("Card 7: 1 2 3 | 3 2 1")) # Test Case 9: All numbers
       match but reversed. Expected Output: 4
28 print(testFun("Card 8: 100 200 | 100 200")) # Test Case 10: Dubbles.
       Expected Output: 1
29 print(testFun("Card 999: 41 48 | 83 86")) # Test Case 11: Overlapping.
       Expected Output: 0
30 # Output:
31 Test Cases:
32 8
33 2
34 0
35 <class 'Exception'>
36 2
37 <class 'Exception'>
38 0
39 <class 'Exception'>
40 2
41 16
42 0
```