MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Kursledare: Anders Mörtberg
Examinator: Lars Arvestad

Tentamensskrivning i
DA2004 Programmeringsteknik för matematiker
7.5 hp
2021-03-12

- Del A har flervalsfrågor där minst ett svarsalternativ är korrekt. Om man svarar fel eller inte har exakt rätt antal alternativ får man 0 poäng på frågan.
- Man måste bli godkänd på del A (4 rätt på 8 frågor) för att få göra del B.
- Del B består av ett antal frågor med varierande poäng (totalt 12) vilka ska lösas genom att man skriver Python 3 kod.
- Svaren till del B lämnas in i en `.py`-fil namngiven `XXX-XXXX-XXX.py` där `XXX-XXXX-XXX` är koden som man får i Ladok när man registrerar sig på tentan. Man **måste** även ha med sin anonyma kod i en kommentar i toppen av filen. Man ska **inte** skriva sitt riktiga namn någonstans i filen!
- Var *noga* med att **namnge** *funktioner och klasser* rätt (på samma sätt som de är namngivna i uppgiften). Kommentera även koden så att det är tydligt var lösningen till varje uppgift är.
- Inga `import` får användas om de inte nämns eller finns med i uppgiften. Man får dock använda inbyggda funktioner som `len`, `range` och `map`.
- All kod avser **Python 3**, dvs *inte* t.ex. Python 2.7
- **Hjälpmedel:** Till del A får man ha ett A4 med så mycket information man vill. Du får skriva på båda sidorna. Del B är öppen bok då det är hemtenta och samma regler kring hjälpmedel gäller som för projekt och labbar.
- **Betygsgränser:** E: 10, D: 12, C: 14, B: 16, A: 18, av maximala 20.

Del A: flervalsfrågor (1p per fråga)

1. Vilka av följande ord har specifikt med felhantering att göra?

A. `with`
B. `raise`
C. `class`
D. `close`
E. `except`

2. Om `mylist = ['hej', 1, 5, True, 3, 9]` vad är `mylist[-3]`?

A. `None`
B. `5`
C. `3`
D. `True`
E. Inget av ovan då man inte kan indexera listor med negativa tal

3. Vilka av följande är reserverade ord i Python?

A. `class`
B. `instance`
C. `if`
D. `print`
E. `__init__`

4. Vilka av följande påståenden är sanna?

A. Listor är omuterbara
B. `(1,2,3)` är en lista med tre element
C. Strängar kan vara *nycklar* i en `dict`
D. Strängar kan vara *värden* i en `dict`
E. Det finns en typ `char` för bokstäver i Python

5. Vad blir resultatet av `list(filter(lambda s: s[0].lower()=='a', ['Hej', 'apan', 'heter', 'Anders']))`?

A. `[]`
B. `['Hej', 'apan', 'heter', 'anders']`
C. `['apan']`
D. `['apan', 'anders']`
E. `['apan', 'Anders']`

6. Givet funktionen till höger, vilka alternativ returnerar `True`?

A. `f(True)`
B. `f(y=False)`
C. `f(False,False)`
D. `f(True,False)`
E. `f()`

```
def f(x=True,y=True):
    return ((x and y) or (not x and not y))
```

7. Givet koden nedan, vilka av alternativen leder till särfall?

```
class A:
    x1 = 1

    def f1(self):
        return "f1 of A"

class B(A):
    def f2(self):
        x2 = 3
        return "f2 of B"

class C(B):
    def f3(self):
        return "f3 of C"

c = C()
```

A. `print(c.f1())`
B. `print(c.f2())`
C. `print(c.f3())`
D. `print(c.x1)`
E. `print(c.x2)`

8. Vad skrivs ut av koden till höger?

A. `[1, 2, 3]`
B. `[]`
C. `[1, 2, 3, 3, 2, 1]`
D. `[-1, -2, -3]`
E. `[3, 2, 1]`

```
def f(mylist):
    if mylist != []:
        x , t = mylist[-1] , mylist[:-1]
        return [x] + f(t)
    else:
        return mylist

print(f([1,2,3]))
```

Del B: kodfrågor (2p per fråga)

Obs: ni får använda `random` biblioteket på den här delen, så er fil får innehålla `import random`.

9. Implementera en klass `Dice` som representerar sexsidiga tärningar och utfall med dem. Klassen ska ha två metoder:

   - `roll_die()` : utfall från kast med en tärning.
   - `roll_dice(n)` : utfallet (dvs summan) från kast med `n` tärningar.

   Tester:

   ```
   [In] : d = Dice()
   [In] : print(d.roll_die())
   [Out]: 2
   [In] : print(d.roll_dice(2))
   [Out]: 7
   [In] : print(d.roll_dice(20))
   [Out]: 74
   ```

   Obs: då tärningskast är slumpmässiga kan ni få andra resultat från kasten.

10. Skriv en funktion `roll_two_dice(n_times)` som tar in ett positivt tal `n_times` (dvs med `n_times > 0`), och med hjälp av listomfattning returnerar en lista med utfallen av kast med två tärningar (från `Dice` klassen) `n_times` gånger.

    Obs: för poäng måste man använda `Dice` klassen och en listomfattning. Om `n_times` inte är ett positivt tal ska även ett lämpligt särfall lyftas av funktionen.

    Tester:

    ```
    [In] : print(roll_two_dice(10))
    [Out]: [9, 2, 7, 2, 9, 7, 8, 10, 3, 5]
    ```

    Obs: då tärningskast är slumpmässiga kan ni få andra resultat från kasten.

11. Skriv en funktion `result_of_dice_rolls(n)` som tar in ett positivt tal `n` och presenterar resultatet av n kast med två tärningar (från klassen `Dice`) i en tabell. Raderna ska visa ett utfall och hur stor andel, i procent med en decimal, som utfallet utgör.

    Tips: använd `roll_two_dice` från uppgiften ovan.

    Exempel:

    ```
    2 2.4%
    3 6.1%
    4 7.9%
    5 11.6%
    6 13.5%
    7 16.7%
    8 14.0%
    9 10.3%
    10 8.9%
    11 6.5%
    12 2.1%
    ```

    Obs: då resultatet är slumpmässigt kan ni få en tabell med andra procent för varje utfall.

12. Givet en dict `d` från strängar till tal, skriv en funktion `print_right_aligned(d)` som skriver ut en tabell med nyckel-värde par på varje rad. Både nycklarna och värdena ska vara högerjusterade.

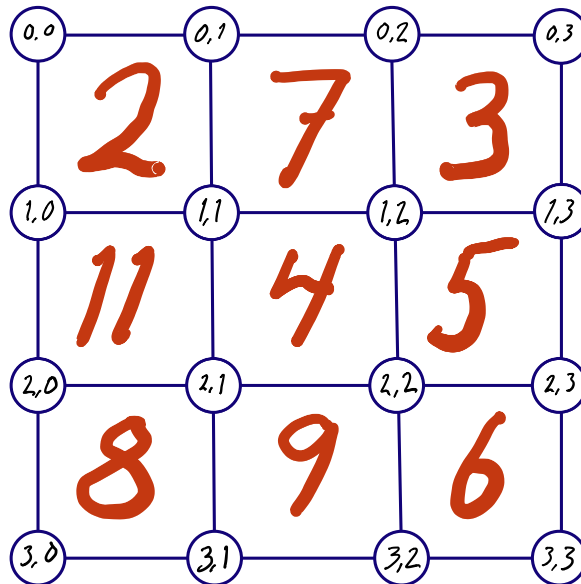
    Till exempel `{ 'hej': 12 , 'hopp': 10101 , 'a': 1 }` ska skrivas ut som:

    ```
     hej    12
    hopp 10101
       a     1
    ```

13. Vi definierar ett brädspel över $k \times k$ rutor, $k > 1$. Varje ruta har en poäng given i nästlade listor (k listor av längd k). Man lägger brickor på rutornas *hörn* och får poäng efter summan på poängen för de 1—4 rutor som en bricka ligger på. Hörnen har koordinater indexerade från 0 till k, se i Figur 1!

    Skriv en funktion `best_corner(board)` som returnerar bästa hörnet på bräde *board* att lägga första brickan på, samt den poäng man får på det hörnet. Om det finns flera hörn med samma poäng som spelar det inte någon roll vilket av dem man returnerar.

    Ledning: Det räcker att begränsa sig till de inre hörnen på brädet, som täcker fyra rutor.



Figur 1: Bräde till fråga 13 motsvarande `[[2, 7, 3], [11, 4, 5], [8, 9, 6]]`. Koordinaterna visas med svarta siffror i cirklar på hörnen och rutornas poäng är givna med röda siffror. Bästa placeringen av bricka är på koordinat `(2, 1)` eftersom den ger poäng $11 + 4 + 8 + 9 = 32$.

Tester:

```
[In] : board0 = [[0, 0], [0, 1]]
[In] : corner, score = best_corner(board0)
[In] : print(corner, score)
[Out]: (1, 1) 1
[In] : board1 = [[10, 1, 0], [1, 1, 0], [0, 0, 0]]
[In] : corner, score = best_corner(board1)
[In] : print(corner, score)
[Out]: (1, 1) 13
[In] : board2 = [[0, 0, 0], [0, 1, 10], [0, 1, 1]]
[In] : corner, score = best_corner(board2)
[In] : print(corner, score)
[Out]: (2, 2) 13
[In] : print(best_corner([[2, 7, 3], [11, 4, 5], [8, 9, 6]]))
[Out]: ((2,1), 32)
```

14. Implementera en klass `Board` för bräden till spelet i uppgift 13. Klassen ska ha en konstruktor med parametrar för både bräde och en lista över vilka positioner som redan valts, eftersom den informationen ska lagras i objekten. Lägg även till en metod `best_valid_corner` som returnerar det bästa hörnet, men tar hänsyn till listan över redan valda positioner (dvs, om någon redan valt en position ska den inte returneras). Du kan anta att metoden inte anropas om alla hörn redan är upptagna.

    Ledning: Nu kan du inte begränsa dig till de inre hörnen, eftersom de kan vara upptagna.

Tester:

```
board0 = [[0,0],[0,1]]
board1 = [[10,1,0], [1, 1, 0],[0,0,0]]
board2 = [[0,0,0], [0,1, 10], [0, 1, 1]]
board3 = [[2,7,3],[11, 4, 5],[8, 9, 6]]

b0 = Board(board0, [])
b1 = Board(board1, [])
b2 = Board(board2, [])

assert b0.best_valid_corner() in [(1, 1), (1, 2), (2, 1), (2, 2)]
assert b1.best_valid_corner() == (1, 1)
assert b2.best_valid_corner() == (2, 2)

b0 = Board(board0, [(1,1)])
b1 = Board(board1, [(1,1)])
b2 = Board(board2, [(2,1), (2,2)])

assert b0.best_valid_corner() in [(1, 2), (2, 1), (2, 2)]
assert b1.best_valid_corner() in [(0, 1), (1, 0)]
assert b2.best_valid_corner() in [(1, 2), (2, 3)]
```