Lab 10

[Frivillig] Python: Prioritetskö

I denna laboration tränar vi på att använda pythons *prioritetskö* (något som kan vara bra att kunna om man exempelvis vill utföra en senare lab som behandlar huffmankodning). Laborationen berör också pythons motsvarighet till operatoröverlagring.

Laborationen är enkel i den bemärkelse att man i huvudsak skall knappa in och testköra lite exempelkod.

Laborationen är frivillig men ger labpoäng om den lämnas in före deadline: torsdag vecka 11, 2018-03-15, 23:55.

10.1 Uppgift 1

Från DALGO känner vi till vad som menas med en *prioritetskö* (och att den kan implementeras med hjälp av en s.k. min-heap). I pythonmodulen `queue` finns det en prioritetsköklass som vi skall bekanta oss med.

Betrakta nedanstående kod. Funktionen `test1()` skapar en prioritetskö (pq) och stoppar därefter in ett antal *tuples* i denna. För att se hur dessa ordnas i kön anropar vi `printAndPop(pq)` som skriver ut alla element i prioritetsordning. (Funktionen tömmer också kön på innehåll, detta eftersom det inte går att iterera över köns element på något annat sätt än att avlägsna dem från kön (jfr min-heap)).

Knappa in och provkör koden!

```
import queue
```

```
def printAndPop(pq):
    while pq.qsize()>0:
        print( pq.get() )


def test1():
    print("running test 1")

    pq = queue.PriorityQueue()

    pq.put( (4.0, 10) )
    pq.put( (2.0, 8) )
    pq.put( (5.0, 2)  )
    pq.put( (1.5, 8) )
    pq.put( (4.0, 8) )
    pq.put( (1.0, 8) )

    printAndPop( pq)

test1()
```

Notera att kön "sorteras" med avseénde på tuplernas 1:a element, och att *minsta talet* har högst prioritet! Notera också att två av tuplerna har samma värde på 1:a elementet (4.0), och att (4.0, 8) skrivs ut före (4.0, 10).

10.1.1 Introducera en annan slags tuppel

Modifiera funktionen `test1()` genom att lägga in följande rad före anropet till `printAndPop()`

```
pq.put( (3.0, (1,2)) )
```

kodraden stoppar alltså in en ny tuppel med en lite mer komplicerad struktur.

Provkör programmet och notera att den nya tuplen hamnar på rätt plats i utskriften.

10.1.2 Krash!

Modifiera återigen funktionen `test1()` genom att även lägga in följande kodrad mellan det föregående och anropet till `printAndPop()`

```
pq.put( (2.0, (1,2)) )
```

Provkör programmet! Om du inte ändrat några värden kommer programmet nu att *krasha! Förklara varför det krashar nu men inte i föregående experiment!* Googla om så behövs. Att redovisa!

10.2 Uppgift 2

Vi har sett hur man skapar och använder en prioritetskö, och att problem kan uppstå om man vill kunna sortera in olika typer av objekt i denna kö. Enklast löser man problemet genom att skapa en ny objekttyp (`class`). För att prioritetskön skall kunna sortera de nya objekten måste den nya klassen definiera en metod med namnet `__lt__` (där lt står för *less than*).
Knappa in nedanstående klassdefinition i din kod

```
class Node:
    def __init__(self, prio, data):
        self.prio = prio
        self.data = data

    def __lt__(self, other):
        return self.prio<other.prio
```

Modifiera därefter `test1()` så att den stoppar in `Node`-objekt istället för tupplar i kön (Exempelvis övergår anropet `pq.put( (3.0, (1,2) ) )` till `pq.put( Node(3.0, (1,2) ) )`.

Provkör programmet! Notera att utskriften av `Node`-objekten inte är särskilt meningsfull. Man kan rätta till detta på två olika sätt. Antingen ändrar man i `printAndPop()` eller så definierar man en metod som kan konvertera ett `Node`-objekt till en sträng. Det senare sättet är att föredra, definiera därför följande metod i din `Node` -klass:

```
  def __str__(self):
        return "({} {})".format(self.prio, self.data)
```

När du nu återigen provkör programmet bör det fungera finfint!

10.3 Redovisning

Redovisa genom att ladda upp följande till pingpong

1. Källkoden (py-filen)
2. En vanlig pingpongkommentar som besvarar den eller de frågor som är märkta med *att redovisa* i marginalen i detta pm.