CSCI 235, Programming Languages, Python, Exercise 3

Deadline: 23.11.2018, at 21.00

Goal of this exercise is that you understand how to build classes in Python. The same principles that apply in C^{++}, also apply in Python. A class can have invariants, which must be established by the constructors, and maintained by all class methods. Unfortunately, Python does not have of private fields, so the user of the class needs self control.

If you have a feeling of déjà vu while working on this exercise, that is possible, because it is based on C^{++} exercise number 2.
Download the files **vector.py** and **matrix.py** from Moodle. Your task is to create a class **Rational** in a file **rational.py** that can be used with **Matrix** and **Vector**.

1. First write a function `gcd(n1,n2)` in a new file **rational.py**. This function **must be based** on the Euclidean algorithm.

   During the C^{++} exercise, a lot of problems came from not carefully tested **gcd** functions. Test carefully! Untested code is unwritten code! The **gcd** function should work well with negative arguments, and when one of the arguments is zero.

   If both arguments are zero, you can use:

   ```
   raise ArithmeticError( "gcd(0,0) does not exist" )
   ```

2. Now you can add

   ```
   from numbers import *

   class Rational( Number ) :
   def __init__( self, num, denum = 1 ) :
      self. num = num
      self. denum = denum

      self. normalize( )
   ```

   In order for this code to work, you have to write `normalize(self)` first. This function must establish the class invariants. These are:

- `num` and `denum` have no common factors except for 1 and -1.
- `denum` is not negative.

Use `//`, when dividing out common factors. If you use `/`, the result will be **float**.

3. Write `__repr__( self )`. Rationals that are integers (whose `denum` equals 1) should be printed as integers:

```
>>> Rational(1,7)
1 / 7
>>> Rational(-1,-7)
1 / 7
>>> Rational(-7,-1)
7
```

4. Complete the following methods of `class Rational`:

```
def __neg__( self ) :
def __add__( self, other ) :
def __sub__( self, other ) :
   # For both methods holds: It must be possible to
   # add/subtract a number to a rational. Use
   # isinstance( , ) for this purpose.

def __radd_( self, other ) :
def __rsub_( self, other ) :
   # So that it will be possible to add/subtract a rational
   # to a number.
```

5. Also, write the following methods:

```
def __mul__( self, other ) :
def __truediv__( self, other ) :
def __rmul__( self, other ) :
def __rtruediv__( self, other ) :
```

   As with the addition operators, the operators must work with numbers that are not rationals.

6. Now it should be possible to do the same tests as we did with C^{++} task 2. Actually, it will be nicer because Python uses unbounded precision integers.

   Createa file **yourname.py** starting with

```
from matrix import *
from vector import *
from rational import *
```

In this file, define a function `def tests( ):` that does the following:

- Compute (and print)

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{3} \\ -\frac{2}{7} & \frac{2}{8} \end{pmatrix} \times \begin{pmatrix} -\frac{1}{3} & \frac{2}{7} \\ \frac{2}{5} & -\frac{1}{7} \end{pmatrix}.$$

  Use `@` for the matrix product.

- Compute the inverse of

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{3} \\ -\frac{2}{7} & \frac{2}{8} \end{pmatrix}$$

- Verify (by subtracting the matrices) that matrix multiplication is associative,

$$(m_1 \times m_2) \times m_3 = m_1 \times (m_2 \times m_3),$$

  matrix multiplication with addition is distributive,

  $m_1 \times (m_2 + m_3) = m_1 \times m_2 + m_1 \times m_3$ and $(m_1 + m_2) \times m_3 = m_1 \times m_3 + m_2 \times m_3$,

  matrix multiplication corresponds to composition of application,

$$m_1(m_2(v)) = (m_1 \times m_2)(v),$$

  and determinant commutes over multiplication:

$$\det(m_1).\det(m_2) = \det(m_1 \times m_2).$$

- Finally, verify that inverse of matrix is indeed inverse:

$$m \times \text{inv}(m) = I \text{ and } \text{inv}(m) \times m = I.$$