Exam for the course DAT171 Object oriented programming in Python

Time: 18th August 2015 8:30-18:00

Teacher: Mikael Öhman (mobile: 0736 837 674, office: 031 772 1301)

Permitted aids: Cay Horstmann: Python for everyone. Manuals and lecture notes are available on the computers.

Teacher will visit the rooms: Around 10:00 and 11:30

Formalities: In each file you hand in, you should write your examination code. Also enter the computer number (from the exam cover).

The documentation and lectures notes are available on `C:\__EXAM__`. Verify that these files are there immediately. Handing in the code should be done under the same folder: `C:\__EXAM__\Assignments\`, when you are finished. If you do not store the files here, they are not included in the exam. Save the files for each question in the appropriate folder, e.g. `C:\__EXAM__\Assignments\Question1\`. *Make sure you only hand in one solution for each question, otherwise you will recieve* ***zero*** *points.* Complex sections of your code should have a descriptive comment of what is achieved.

When you finish the exam you should log out and fill in the (empty) exam cover page like normal and hand this in at the end of the exam.

To run the Lectures notes in IPython notebook, you must copy the link from the start menu, and change the "Start in:" property in the shortcut to `C:\`.

Corrections: The results will be announced the lastest on 25th of August on the course homepage. The review will the same day 12:20-13:00 and on the 26th of August 12:20-13:00.

Grading: There is a total of 25 points which yields grades at the standard 40%, 60%, and 80% limits:

```
def grade(points):
    if points >= 20:
        return 'Grade 5'
    elif points >= 15:
        return 'Grade 4'
    elif points >= 10:
        return 'Grade 3'
    else:
        return 'Fail'
```

Question 1 (6p total)

The ultimate goal is to convert a color image file to a greyscale image file.

Part A (1p)

Colors are often defined by red green and blue color channels. One can compute the luminance (greyscale value) with:

$$luminance = 0.2126\ R + 0.7152\ G + 0.0722\ B$$

Write a function to converts integer red green and blue color values to integer greyscale. Example usage

```
>>> x = luminance(40, 200, 130) # Passing in R = 40, G = 200, B = 130
>>> x
161
```

Part B (5p)

The PPM image format (file.ppm) is a simple text based image format

```
P3
# The P3 means colors are in ASCII, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
103 217 103    43  95 181   219 219 219
244  83 106   255 255 255     0  83 106
```

The lines starting with `#` are commented (ignored). The first characters `P3` denote the file format, and these are followd by the size (width and height). The next number denotes the maximum value of the channels (in this case, 0 to 255). The rest is the values of the colors, pixel by pixel (in this case, R=103, G=217, B=103 for the first pixel)

Linebreaks are optional; any whitespace seperator works. This is also a valid PPM file:

```
P3
3 2
255
103 217 103 43 95 181 219 219 219 244 83 106 255 255
255 0 83 106
```

The PGM image format looks very similar, but has only 1 channel per pixel (the luminance / greyscale)

```
P2
# Greyscale version
3 2
255
185 90 219
119 255 67
```

Write a function that converts a PPM to PGM file. Use your function from the previous question. Your function should handle arbitrary sizes for the images. Try it out on "kitten.ppm".

Hint: Remember to close the files. Hint: You can view PGM/PPM files in PyCharm.

Question 2 (4p total)

You are post-processing some data from an analysis, which requires you to compute the largest eigenvalue from an analysis. The file "matrix.csv" contains the row, column, and values for a matrix. You must utilize NumPy and SciPy for this.

Write a python script that takes the file "matrix.csv" and computes (and prints) the largest eigenvalue *in scientific/exponent form.* The correct value for "matrix.csv" is $7.0000e+00$. There is a example file in the exam folder with a small matrix for you to test your code it, however, you code should work for any matrix size.

Hint: `loadtxt` *in NumPy can be used. Hint:* `scipy.sparse`*. Hint:* `scipy.sparse.linalg`*.*

Question 3 (7p total)

Write a base class `BankAccount`, and subclasses `SavingsAccount` and `SpendingAccount`. *You should implement (or make abstract) methods in the base class, and overload in the subclasses when necessary.*

Savings account: Accounts are not allowed to be overdrafted (no negative amount).

Spending account: Accounts are allowed to be overdrafted to -5000 SEK, with a 10% overdraft fee (up to 500 SEK).

The following methods should be supported:

- `available_amount()` returns the value of the account.
- `withdraw(amount)` withdrawing the given amount from the account.
- `deposit(amount)` deposit the given amount to the account.
- `apply_overdraft_fee()` which withdraws the overdraft when called.
- Implement support so that the accounts can be printed; for example:

```
>>> account = SavingsAccount(1500)
>>> print(account)
Account: 1500 SEK.
```

Write and make appropriate use of a exception `OverdraftException` where an order cannot be complied with. Have the exception contain all relevant information about the failed transaction.

Question 4 (8p total)

Note: I want you to do this exercise without using the preexisting stuff in SciPy or NumPy to test your object oriented programming skills

A sparse matrix is a matrix which contains mostly zeros. In those cases, storing only the positions (row, column) and the corresponding nonzero value is more efficient.

Write a class `SparseMatrix` that represents a matrix without storing the zeros.

0. (1p) The initialization should be `SparseMatrix(rows, columns, values, nr, nc)` (see example usage).
1. (2p) Implement the method `m.get(row, col)` which returns the matrix value at the given position (including zero values).
2. (2p) Implement support for creating a transposed copy of the matrix; `m_t = m.transpose()` *Hint: There is a really simple solution*
3. (3p) Implement support for scalar product with vector by overloading the `*` operator (both `__mul__` and `__rmul__`). Code it so that it skips all the zeros of the matrix (otherwise, it would be pointless to use a sparse matrix).

Example usage:

$$M = \begin{pmatrix} 3 & 0 & 0 & 0 & 2 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 \\ 0 & 2 & 0 & 0 & 4 \\ 0 & 0 & 0 & 9 & 0 \end{pmatrix}$$

$$M \cdot \begin{pmatrix} 1 \\ 0 \\ 3 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 9 \\ 12 \\ 21 \\ 12 \\ 18 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 3 & 2 & 3 \end{pmatrix} \cdot M = M^T \cdot \begin{pmatrix} 1 \\ 0 \\ 3 \\ 2 \\ 3 \end{pmatrix}$$

```
>>> m = SparseMatrix([0, 3, 1, 2, 4, 0, 3], [0, 1, 2, 2, 3, 4, 4],
              [3., 2., 4., 7., 9., 2., 4.], 5, 5)
>>> m * [1, 0, 3, 2, 3]
[9, 12, 21, 12, 18]
>>> [1, 0, 3, 2, 3] * m == m.transpose() * [1, 0, 3, 2, 3]
True
>>> m.get(0,0)
3.0
>>> m.get(1,5)
0.0
```

Hint: You don't need to be concerned with optimal performance, just that it works