Homework 2

Ex1: Matrix class (50 pts)

The purpose of the exercise is to implement a class that represents a 2-dimesional matrix, fitted with a few of the matrix operations.

Implement the class in a module called Matrix.py. To test the class, you are provided with multiple use case scripts and console commands and their expected output. They will provide more insight into all the required validation.

Class specification:

The constructor:

- The constructor takes only one argument: the elements of the matrix, passed as a sequence of sequences (e.g. list of lists, list of tuples, tuple of tuples, tuple of lists, etc …).
    - If the len() function is not supported on the outer sequence and all the inner sequences of the argument, the constructor must raise an exception "data must be a sequence of sequences" (see console commands examples).
    - If the inner sequences of the argument don't all have the same length, the constructor must raise an exception "rows must have same number of elements" (see console commands examples).
    - The constructor must check that all the elements are numerical (ultimately floating point) values. If any element does not convert to a float, the constructor must raise an exception "elements must be numerical values (floating points)" (see console commands examples).
- The constructor must create two private attributes:
    - size: which is a pair (number of rows, number of columns)
    - data: which is a list-of-lists of all the elements provided as argument. Note that a (deep) copy of the argument is needed, so that any subsequent modification of the constructor argument in the user code does not affect the data of the Matrix object (see script6.py)

Output (printing)

- The class must expose a method print(), which when called upon an object, prints to the console the matrix data in row-column format. For simplicity, using a tab character to separate elements of a row is acceptable (see script1.py).
- Objects of the class must also be printed, in the same way described above, when passed as arguments to the print function (from the standard library) (see script1.py)

Operations

- The class must expose a method add(), which when called on Matrix object A, takes as argument a Matrix object B and returns a Matrix object C, result of the addition of A and B (see script1.py).
    - If the argument is not a Matrix object, the method raises an exception "other operand must be have Matrix type" (see script2).
    - If A and B don't have the same size, the method raises an exception "dimension mismatch" (see script3).
- The class must also overload the + operator to perform the same matrix addition (see script1). The operation would raise the same exceptions as above, in similar situations.
- The class must expose a method transpose(), which when called on Matrix object A, returns a Matrix object B corresponding to the transpose of A (see script1).
- The class must also expose a method matrix_multiply(), which when called on Matrix object A, takes as argument a Matrix object B and returns a Matrix object C, result of the matrix-matrix multiplication of A and B (see script1).
    - If the argument is not a Matrix object, the method raises an exception "other operand must be have Matrix type" (see script4).
    - If the number of columns of A is not equal to the number of rows of B, the method raises an exception "dimension mismatch" (see script5).

Use cases:

In the following are cases of exceptions due to wrong constructor argument. A long section of the error message is replaced by <…>; the last couple of lines, which are the most important part, are shown:

```
>>> import Matrix as mx
>>> A = mx.Matrix(1)
Traceback (most recent call last):
  <…>
    raise Exception("data must be a sequence of sequences")
Exception: data must be a sequence of sequences
>>> A = mx.Matrix([1, 2])
Traceback (most recent call last):
  <…>
    raise Exception("data must be a sequence of sequences")
Exception: data must be a sequence of sequences
>>> A = mx.Matrix([[1, 2], [3, 4, 5]])
Traceback (most recent call last):
  <…>
    raise Exception("rows must have same number of elements")
Exception: rows must have same number of elements
>>> A = mx.Matrix([[1, 'r'], [4, 5]])
Traceback (most recent call last):
  <…>
    raise Exception("elements must be numerical values (floating points)")
```

```
Exception: elements must be numerical values (floating points)
```

The following script shows how the Matrix object must be independent from the list-of-lists (or any other sequence) used as argument in its instantiation.

script6.py:

```
import Matrix as mx

L = [[1, 2], [3, 4]]
A = mx.Matrix(L)
print("L =", L)
print("A = Matrix(L) = ")
print(A)
L[0][0] = 10
print("Modifying L ...")
print("L =", L)
print("A = ")
print(A)
```

Expected output:

```
L = [[1, 2], [3, 4]]
A = Matrix(L) =
1     2
3     4

Modifying L ...
L = [[10, 2], [3, 4]]
A =
1     2
3     4
```

The following script shows proper use of all methods and operations.

script1.py:

```
import Matrix as mx

A = mx.Matrix([[1, 2, 3], [4, 5, 6]])
print("A =")
A.print()
B = mx.Matrix([(7, 8, 9), (10, 11, 12)])
print("B =")
print(B)
C = A.add(B)
print("C = A.add(B) = ")
print(C)
D = A + B
print("D = A + B =")
print(D)
E = B.transpose()
print("E = B.transpose() = ")
print(E)
F = A.matrix_multiply(E)
print("F = A.matrix_multiply(E) = ")
print(F)
```

Expected output:

```
A =
1      2      3
4      5      6

B =
7      8      9
10     11     12

C = A.add(B) =
8      10     12
14     16     18

D = A + B =
8      10     12
14     16     18

E = B.transpose() =
7      10
8      11
9      12

F = A.matrix_multiply(E) =
50     68
122    167
```

The following scripts show the exceptions raised with wrong use of the add() and matrix_multiply() methods. The same kind of exceptions are raised for the wrong use of the + operator.

script2.py

```
import Matrix as mx

A = mx.Matrix([[1, 2, 3], [4, 5, 6]])
print("A =")
A.print()
B = [[7, 8, 9], [10, 11, 12]]
print("B =")
print(B)
C = A.add(B)
print("C = A.add(B) = ")
print(C)
```

Expected output:

```
A =
1      2      3
4      5      6

B =
[[7, 8, 9], [10, 11, 12]]
```

```
Traceback (most recent call last):
     <…>
    raise Exception("other operand must be have Matrix type")
Exception: other operand must be have Matrix type
```

script3.py

```
import Matrix as mx

A = mx.Matrix([[1, 2, 3], [4, 5, 6]])
print("A =")
A.print()
B = mx.Matrix([[7, 8], [10, 11]])
print("B =")
print(B)
C = A.add(B)
print("C = A.add(B) = ")
```

Expected output:

```
A =
1      2      3
4      5      6

B =
7      8
10     11

Traceback (most recent call last):
   <…>
    raise Exception("dimension mismatch")
Exception: dimension mismatch
```

script4.py

```
import Matrix as mx

A = mx.Matrix([[1, 2, 3], [4, 5, 6]])
print("A =")
A.print()
B = [[7, 8], [9, 10], [11, 12]]
print("B =")
print(B)
C = A.matrix_multiply(B)
print("C = A.matrix_multiply(B) = ")
print(C)
```

Expected output:

```
A =
1      2      3
4      5      6

B =
```

```
[[7, 8], [9, 10], [11, 12]]
Traceback (most recent call last):
  <…>
    raise Exception("other operand must be have Matrix type")
Exception: other operand must be have Matrix type
```

script5.py

```
import Matrix as mx

A = mx.Matrix([[1, 2, 3], [4, 5, 6]])
print("A =")
A.print()
B = mx.Matrix([[7, 8, 9], [10, 11, 12]])
print("B =")
print(B)
C = A.matrix_multiply(B)
print("C = A.matrix_multiply(B) = ")
print(C)
```

Expected output:

```
A =
1      2      3
4      5      6

B =
7      8      9
10     11     12

Traceback (most recent call last):
  <…>
    raise Exception("dimension mismatch")
Exception: dimension mismatch
```

Ex2: Athlete class and derived (25 pts)

You are given the code scaffold of the module Athlete.py and the file champions.csv. Complete the missing method in the class Athlete and properly derive from it the missing classes for the module to produce the following output. You are not allowed to modify the main function or the script.

```
Michael Phelps -28 medals- is competing in swimming
Usain Bolt -3 medals- is competing in running
Yohan Blake -3 medals- is competing in running
Lasha Talakhadze -2 medals- is competing in weightlifting
Oussama Mellouli -3 medals- is competing in swimming
Akbar Djuraev -1 medals- is competing in weightlifting
```

Ex3: Patients blood pressure analysis (25pts)

You are given a CSV file `cardio_data.csv` containing the health records of 54848 patients. The features are separated by a semicolon. For each patient, we know:

- Age in number of days (age)
- Height in centimeters (height)
- Weight in kilograms (weight)
- Systolic blood pressure (ap_hi)
- Diastolic blood pressure (ap_lo)

Use pandas to discover, clean and analyze this dataset:

- Compute age in years. Overwrite the original age column but keep the dtype to be integers
- Compute new attribute BMI, which is given by the weight in kilograms divided by the square of the height in meters. Round to first decimal place
- categorize patients based on their BMI into the following categories, save results into a new column:

  ```
  < 18.5: underweight
  18.5~24.9: healthy
  25~29.9: overweight
  >= 30: obese
  ```
- Compute the mean, min and max of these attributes: age, height, weight, bmi, ap_lo, ap_hi
- Compute the mean of ap_lo and ap_hi, grouped by the BMI categories
- Plot a histogram of ap_lo, ap_hi:
  - Use subplots
  - The figure size should be (10, 5)
  - Add a title and an x label for each subplot
  - Save results of plotting into a png file
- Save final data frame into a new file of type CSV. Use "\t" separator and make sure index=None

Group by result:

```
                  ap_lo       ap_hi
bmi category
healthy       82.844161  126.180457
obese         86.313712  134.803954
overweight    84.327004  129.877508
underweight   82.458204  124.455108
```

Plotting results: