Lab 6 – Exceptions and Files Handling

Lab Objectives

The aim of this worksheet is to introduce you with files and exceptions handling in Python by getting you to:

- Write robust code – dealing with exceptions;
- Make use of assertions for defensive programming;
- Read/write files and manipulate file data.

Sample solutions to this Lab can be found here.

Question 1

Download the file `text.txt` and save it into the directory in which you have all your Labs for the module CO1102. This could be for example `/CO1102/Labs`

1. Write a function that takes in a file name and print the contents of the file (line by line) all in upper case. This function does not need to return anything.
2. Test your program with the file `text.txt`.

Question 2

Consider the following code snippet:

```
import random as rd
def mysqrt(x, eps):
    """Inputs:two numbers x, eps such that x >= 0, eps>=0
    Output: a number res such that x-eps <= res*res <=x+eps """
    g = rd.uniform(0, x) #random guess
    gsquared = g*g
    while (gsquared<x-eps or gsquared>x+eps):
        g = (g+x/g)/2 #new g is average of g and x/g
        gsquared = g*g
    return g
```

1. Modify the given so as all the estimates of the value g for each iteration of the loop are stored into the file `outputs.txt`. You need to record also the iteration number. Use the Python command `format()` to format the numbers so that floats are printed with 3 decimal places. A sample output with `mysqrt(45, 1.e-8)` may look as follows:

```
At iteration 1, the estimated squared root is 12.691
At iteration 2, the estimated squared root is 8.118
At iteration 3, the estimated squared root is 6.831
At iteration 4, the estimated squared root is 6.709
At iteration 5, the estimated squared root is 6.708
At iteration 6, the estimated squared root is 6.708
```

2. Add an exception-handling construct to deal with the possibility of a `ZeroDivisionError` in the code.
3. Add assertions to the code in order to ensure that the inputs and outputs satisfy the constraints as given in the docstring of the function. Remember that constraints on the inputs/outputs as called pre-conditions/post-conditions respectively. Ensure your code is robust by handling exceptions that may arise.

Question 3

Download the csv file `data.csv` and save it into your local directory. For each function that you will write, use exceptions and assertions whenever appropriate.

1. Open the csv file and observe that it contains rows of numbers separated by comma. Please do not edit nor modify the given csv file.
2. Write the function `read_csv_data`, which takes as input a file name and returns a list of lists storing all the data in the given file.
3. Write the function `averages` that takes as input the output from the function `read_csv_data` and returns a list composed of the averages for each row of the data. If the row is empty, then you should return `nan` as the average for that row. For that purpose, you can import the package `numpy` and the nan value is represented by `numpy.nan`. Before setting the average to nan, ensure the row is empty by using an **assertion** into your code.
4. Write a function print_averages that takes as input the averages calculated in question 3 and displays on the screen the average for each data row. A sample run this function is as follows:

```
Row  0 average is 15.783
Row  1 average is 11.117
Row  2 average is 15.025
Row  3 average is 10.025
Row  4 average is 27.300
Row  5 average is   nan
Row  6 average is 4.600
```