BAN401
"Applied Programming and Data Analysis for Business"

MID-TERM ASSIGNMENT

Submission deadline: **30 September 2018**

IMPORTANT:
Assignment should be submitted by **the group of 3-4 people.**
Note: you had to submit your group-list by September 06, 2018 (first assignment deadline) based on the organizational setup of the course.

REPORT PREPARATION:

The assignment should be submitted as one compressed file (ZIP or RAR format).
Name of the compressed file should be in the following format:
Group_N
where N is the number of your group

The compressed file ***Group_N*** should include the following files:

a) One report file (only PDF format):

   a. Python codes for problems 1-12:
      Copy your code and past it into you report (remember about indentations in Python)
   b. Detailed explanation of your solutions for problems 1-12
      1. *Describe the idea of your designed solution*
      2. *How your code is solving the problem*
   c. For each problem: Screenshots of the results from the PyCharm

b) 12 code files (.py format) - for problems 1-12, respectively
   Names of the **.py** files should include problem numbers in the following format:
   problem_1.py, problem_2.py, . . . , problem_12.py
   Required: comment in your code explaining what you are doing. For example:

```
s = [5, 35, 26, 44, 4, 1] # list s is created
```

- All codes should be implemented in Python 3 (i.e., not in Python 2)
- The report should be typed (not handwritten).
- The report should be written in English.
- Submission should be done only via CANVAS within the deadline.

YOUR SUBMISSION WILL BE CHECKED FOR PLAGIARISM!

PROBLEM 1

Create a list:

```
new_list = [11, 2, 4, 5, 6, 7, 8, 9, 10]
```

Write a code that prints any number from new_list that satisfies two conditions:

1) The number is odd;

2) Its index in the list is odd.

For example, the code should print 5 as it’s an odd number and its index is also odd. But it shouldn't print 11 (an odd number) as its index is 0.

PROBLEM 2

In this problem, you should make a code that asks the user for his/her annual income and then prints the income tax and after-tax income. For simplicity, assume that everyone has the same tax rates as follows:

Annual Income	Tax Rate (in percentage)
≤ 55,000	0
> 55,000 and ≤ 90,000	10
> 90,000 and ≤ 190,000	20
> 190,000 and ≤ 390,000	30
> 390,000	40

NOTES:

- Use input () – method to ask the user for his/her annual income
- We assume a progressive tax system[1] in which the tax rate increases as an individual's income increases.

Example output is as follows:

```
Enter your annual income in numbers: 190000
Your tax is: 23500.0
Your after-tax income is: 166500.0
```

[1] https://en.wikipedia.org/wiki/Progressive_tax

PROBLEM 3

Create a function using **for loop**(s) and conditional statements that prints an X using #s. The function takes a natural number *n* (i.e. 1, 2, 3, 4, etc.) as the only parameter and the resulting X should be of size *n***n*.

Example output is as follows:

`drawx(5)` should print:

```
#   #
 # #
  #
 # #
#   #
```

`drawx(10)` should print:

```
#         #
 #       #
  #     #
   #   #
    ##
    ##
   #   #
  #     #
 #       #
#         #
```

PROBLEM 4

Given a list of observations (see `list` below), we want to check if the observations are conforming to a normal distribution or not. We do this by (1) calculating the mean and standard deviation of the provided data, and then (2) checking if the observations fall within three standard deviations of the mean. Steps (1) and (2) are as follows:

Step (1):

- Write code that first calculates the mean: $\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$

  Note: $\sum_{i=1}^{N}(x_i) = x_1 + x_2 + \ldots + x_N$

- Write code that calculates the standard deviation: $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$

Step (2):

After calculating the mean and standard deviation, write a code that checks if the observations are within three standard deviations of the mean. For example, for element x_i in the given list, we check if (μ - 3*σ) ≤ x_i ≤ (μ + 3*σ). Remember, in order to compute the square root using sqrt() function, you must import it by adding `from math import sqrt` to the top of your script.

The input data to use is as follows:

```
list = [53.21, 50.73, 75.34, 45.18, 51.95, 47.47,
52.85, 49.93, 49.66, 46.09, 50.16, 47.21, 46.02,
47.87, 49.40, 51.47, 22.80, 46.80, 49.66, 53.24]
```

The output should look similar to this (the numbers are only examples):

```
Mean: 49.35, standard deviation: 8.64
Observation value 53.21 is within the control limits
Observation value 50.73 is within the control limits
Observation value 75.34 is above the upper control limit
Observation value 45.18 is within the control limits
Observation value 51.95 is within the control limits
Observation value 47.47 is within the control limits
...
```

This is a generic example of process control, where one checks if a process is under statistical control by calculating and using control limits. If interested, a short description of control limits can be found here: https://en.wikipedia.org/wiki/Control_limits.

PROBLEM 5

In this problem, you are to make a guessing game. Here are the instructions:

- Import the randint () - method from the *random* library to generate a random integer above zero. This integer should be the correct guess in the game. You decide the highest possible correct guess.
- Print a string to inform the player about the game.
- Use a **while-loop** for the guessing game that takes an integer from the user as input. The game is to guess the randomly generated integer.
- In the case of a wrong guess, the game should continue – asking for a new guess. The player is to be informed if the guess is higher or lower than the answer using a **print**-statement.
- In the case of a correct guess, or if the player enters zero, the game ends. The player should be informed about the actions and the answer, either when the player quits or guesses correctly.

For information on how to use randint (), see:
https://docs.python.org/3/library/random.html#random.randint

The output should look similar to this:

```
Please enter a number between 1 and 1000 to play the game, or enter 0 to quit:
7
Your guess is too low. Please guess a higher number
9000
Your guess is too high. Please guess a  lower number
```

PROBLEM 6

Make a nested **for-loop** to print the times tables from one to ten. The tables should be separated by a string of stars "*". Write the output to a *text* file (i.e., file extension: .txt) using **with** statement and **open** function. Make sure the rows in the time tables and the stars are aligned properly. The last two tables in the text file should look similar to the following:

```
******************
 1 times  9 is   9
 2 times  9 is  18
 3 times  9 is  27
 4 times  9 is  36
 5 times  9 is  45
 6 times  9 is  54
 7 times  9 is  63
 8 times  9 is  72
 9 times  9 is  81
10 times  9 is  90
******************
 1 times 10 is  10
 2 times 10 is  20
 3 times 10 is  30
 4 times 10 is  40
 5 times 10 is  50
 6 times 10 is  60
 7 times 10 is  70
 8 times 10 is  80
 9 times 10 is  90
10 times 10 is 100
******************
```

Note:

In this problem, you should get to know how to use **open**-function.

For information on how to use **with**-statement and **open**-function, see:

1) https://docs.python.org/3.7/library/functions.html#open

2) Video tutorial:

https://www.coursera.org/lecture/python-for-applied-data-science/reading-files-with-open-78ZHl

3) https://www.pythonforbeginners.com/files/with-statement-in-python

PROBLEM 7

Imagine that you are provided with a data set containing two lists of monthly revenues of product A and B of company X over a period. Your task is to write a function named `maxRevenues` that takes these two lists of numbers as parameters and then compute the total monthly revenues for both products A and B. The function then identifies the month(s) in which the total revenue is highest and returns that results. After that, apply the function to the two following lists:

```
revenueProductA = [700, 10000, 200, 33333, 4000, 666, 777, 150203] # from month 1 to 8

revenueProductB = [3333, 38586, 190293, 38383,70999, 13868, 113969, 40290] # from month 1 to 8
```

Your code should display results similar to the following output when function `maxRevenues(revenueProductA, revenueProductB)` is called:

```
$190,493.00 is the highest revenue over months
That was the revenue for the following months: 3  and  8
```

PROBLEM 8

Write a function that asks the user to write a sentence. The function then returns a dictionary in which the keys are different letters and the keys' values are the number of times they appear in the sentence. Irrelevant letters (that do not appear) and spaces are not printed/counted.

Example output is as follows:

```
Please write any sentence that you like:
I am here
{'I': 1, 'a': 1, 'm': 1, 'h': 1, 'e': 2, 'r': 1}

Please write any sentence that you like:
It is a car
{'I': 1, 't': 1, 'i': 1, 's': 1, 'a': 2, 'c': 1, 'r': 1}
```

PROBLEM 9

Write a code that prompts the user for a positive integer number. The user may enter a positive, zero, or negative value. The program should print out the total sum and sum of the squares of all numbers from 1 up to the given input only if the input entered by the user is a positive integer. Otherwise (e.g., the user enters zero or negative number), the program should give a warning message (i.e., "Could you please enter a positive number once again!!") and **it should redirect to the input mode again without exiting the program**.

For example: if the user provides the positive number 3, then the program should print out the relevant sum (1+2+3=6) and the sum of squares ($1^2+2^2+3^2=14$).

The output should be like this:

```
Please enter the positive number:
-8
Could you please enter positive number once again!!
Please enter the positive number:
0
Could you please enter positive number once again!!
Please enter the positive number:
6
my number is 6
The total sum from 1 to 6 is :  21
the total sum of squares from 1 to 6 is: 91
```

PROBLEM 10

Make a list of multiple lists (at least 3 lists with each contains at least 3 numbers). Write a function that takes the above two-level list as a parameter, calculates the sum of all elements in each sub-list and returns the maximum among those sums.

For example, if applying the function to the following list:

```
Num = [[1, 2, 3], [2, 3, 4], [4, 5, 6]]
```

then the sums would be 6, 9 and 15 and the function should return 15.

Example output is as follows:

```
The list provided to the function is: [[2, 3, 4], [3, 4, 5], [5, 6, 8], [45, 67, 4]]
The maximum sum of the list is: 116
```

PROBLEM 11

For this problem, assume that all investments are risk free, and that cash flows are known. Write a function that returns the Compound Annual Growth Rate (CAGR) of an investment (see https://www.investopedia.com/terms/c/cagr.asp if you would like to see more details about the equation):

$$CAGR = \left(\frac{\text{Ending Value}}{\text{Beginning Value}}\right)^{\left(\frac{1}{\#\text{ of years}}\right)} - 1$$

Hints:

(1) Start Time, End Time, Start Value, and End value (specified in the table below) can be used as parameters in your function. (2) Number of years is a period between End Time and Start Time.

Next, create a nested dictionary (a dictionary where the value corresponding to each key (remember *key-value* format) is a dictionary). In the given dictionary, the "Company/investment name" should be at the top level, and the required variables (such as "Start Time", "End Time", "Start Value", and "End Value") and their corresponding values to compute CAGR should be at the bottom level.

Example of nested dictionary:

```
nested_dict = { 'dictA': {'key_1': 'value_1'}, 'dictB': {'key_2':
'value_2'}}
```

where:

`'dictA'` and `'dictB'` are at the top level;

`'key_1': 'value_1'` and `'key_2': 'value_2'` are at the bottom level.

Table - Example of investment opportunities that you can apply to your code:

Company/investment name	Start Time	End Time	Start Value	End Value
APL	0	1	1500	2000
TRVX	1	2	750	800
AKA	3	5	4500	5300
NAS	0	2	250	400
TEL	1	2	900	1300

Use the CAGR-function to add the CAGR value for each investment opportunity (i.e. for each "Company/investment name" in terms of the table above) in the dictionary. Do this with **for loop.** The CAGR value should be added to the dictionary under (nested under) the corresponding "Company/investment Name". In other words, it should be added at the bottom level of your nested dictionary.

Finally, print the "Company/investment name" of the investment with the highest CAGR.

PROBLEM 12

Write a tic-tac-toe game.

Draw the 3x3 game grid in a list of 9 elements, and store it in a variable named `game_board`. To complete this problem you need at least 2 functions:

1) A function that handles the insertion of pieces by changing the `game_board` list. The function should accept row index, column index and game piece shape as parameters.
2) A function that prints the `game_board` to the Pycharm console

Finally, use a **while loop** to run the game. While the game is running, let users place pieces in different shapes. No winning condition is required.

You console output (with interactive mode using input() function) should be similar to the following:

```
Z:\Codes\PyCharm\My_first_project\venv\Scripts\
['-', '-', '-']
['-', '-', '-']
['-', '-', '-']
Enter row index (type q to quit): 0
Enter col index: 0
Enter shape: X
['X', '-', '-']
['-', '-', '-']
['-', '-', '-']
Enter row index (type q to quit): 2
Enter col index: 1
Enter shape: O
['X', '-', '-']
['-', '-', '-']
['-', 'O', '-']
Enter row index (type q to quit): 2
Enter col index: 2
Enter shape: X
['X', '-', '-']
['-', '-', '-']
['-', 'O', 'X']
Enter row index (type q to quit): q

Process finished with exit code 0
```