Koç University
COMP 125: Programming with Python
Homework #3

Deadline: May 24th at 23:59pm
Submission through: Blackboard

Make sure you carefully read and understand all of the rules on this page.

This homework assignment contains 3 programming questions. It is an individual assignment. You are not allowed to collaborate with your classmates or receive help from unauthorized third-party resources. Cheating and/or plagiarism will be immediately reported to the Disciplinary Committee.

Download Hw3.zip from Blackboard and unzip the contents to a convenient location on your computer. Each py file in the zip is for a different question. Solve each question in its own file. **DO NOT CHANGE THE NAMES OF THE FILES.**

When you are finished, compress your Hw3 folder containing all of your answers. The result should be a SINGLE compressed file (file extension: .zip or .rar). Upload this compressed file to Blackboard. Incorrect submission formats (e.g., uploading 3 Python files rather than a single compressed zip/rar) may be penalized at the instructors' discretion.

Follow instructions, print messages, input-output formats closely. Your code may be partially graded by an autograder, which means any inconsistency will be automatically penalized.

You cannot use any built-in or third-party Python functions, modules or libraries that we did not cover in class.

After you submit your homework, download it from Blackboard to make sure it is not corrupted and it has the latest version of your code. You are only going to be graded based on your Blackboard submission. We will not accept homework via e-mail or other means.

It is YOUR duty to make sure you are submitting the correct files. We will not accept objections or excuses after the deadline such as: "I forgot to submit one file" or "I submitted the wrong files" or "I submitted empty files". These occur more often than you think. It is YOUR duty to ensure your submission is correct!

Late submission policy can be found on the course syllabus.

Code that does not run (e.g., syntax errors) or does not terminate (e.g., infinite loops) may receive 0. Also, your code should run without errors in the way that it is submitted. We should not have to open your file and uncomment/edit parts of your code to be able to run it.

Best of luck and happy coding!

Q1: Calculate Grade - 30 points

The file associated with this question is CalculateGrade.py.

Write a program to calculate a student's end-of-semester grade average in an imaginary course. In this course, the student's end-of-semester average is computed is as follows:

- There are **11** labs in the course. Lowest **3** lab grades are dropped. Then, the average lab grade is computed. This is worth **10%** of the end-of-semester average.
- There are **6** homework assignments in the course. Lowest **2** homework grades are dropped. Then, the student's homework average is computed. This is worth **20%** of the end-of-semester average.
- There are **4** midterms in the course. Lowest **1** midterm grade is dropped. Then, the student's midterm average is computed. This is worth **40%** of the end-of-semester average.
- There is a final exam in the course. This is worth **30%** of the end-of-semester average.

First, write the following function: drop_and_average(scores, n).

- scores is a list of numeric elements (each element is of type int or float)
    - e.g.: scores = [83.5, 82.0, 100, 100, 54, 83]
- n is an integer denoting how many grades should be dropped from scores
- This function will drop n lowest grades from the list scores, compute the average of the remaining, and **return** that average.

Then, write the following function: calculate_grade(labs, hws, mts, fin_exam).

- labs, hws, mts are three lists containing student's lab grades, homework grades, and midterm grades respectively.
- This function will **return** the student's end-of-semester average according to the rules and weights given above. **Hint:** In this function, you can call drop_and_average multiple times.

Inside calculate_grade, you should also check for the following erroneous inputs:

- labs should contain exactly 11 elements, hws should contain exactly 6 elements, mts should contain exactly 4 elements. If this rule is violated, then calculate_grade should return -1 (integer).
- All lab, homework, midterm, and final exam scores must be non-negative. If this rule is violated, then calculate_grade should return -2 (integer).
- If both rules are simultaneously violated, calculate_grade should return -1 (integer).

Q2: Pearson Correlation Coefficient - 30 points

The file associated with this question is PearsonCorr.py.

Given two lists X and Y, composed of N floating point numbers such as:

X = [0.1, -3.2, 4.9, …, 0.5]

Y = [1.2, -4.4, 0.1, …, 0.2]

Write a program to calculate the Pearson correlation coefficient between X and Y. We will achieve this in multiple steps.

Step 1: Write a function calc_mean(X) to calculate and return the mean of a given list X.

Example: If X = [-4.2, 3.6, 0.1, 12.3], then calc_mean(X) returns: 2.95

Step 2: Write a function calc_stdev(X) to calculate and return the standard deviation of a given list X. The standard deviation of X, denoted by s_X is given by the formula:

$$s_X = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(X[i]-\bar{X})^2}$$

where $\bar{X}$ denotes the mean of X, and *N* denotes the length of the lists X and Y.

Step 3: Write a function calc_cov(X,Y) to calculate and return the covariance of given lists X and Y. The covariance of X and Y, denoted by cov(X,Y), is given by the formula:

$$cov(X,Y) = \frac{1}{N-1}\sum_{i=1}^{N}(X[i]-\bar{X})(Y[i]-\bar{Y})$$

Step 4: Write a function calc_pearson(X,Y) to calculate and return the Pearson correlation coefficient between given lists X and Y, which can be computed by the formula:

$$r = \frac{cov(X,Y)}{s_X \cdot s_Y}$$

Hint: Functions in later steps should make use of functions from earlier steps.

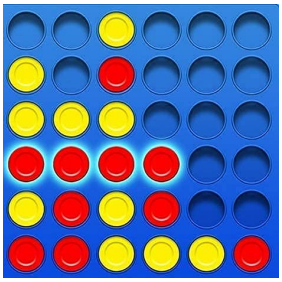
IMPORTANT: You are not allowed to import modules or libraries to solve this question. You have to implement each function from scratch by iterating through the lists.

FOR EXAMPLE:

Let X = [1.5, 3.2, 0.8] and Y = [1.9, 4.5, 6.0]. Then you will get the following results:

```
mean for X: 1.8333333333333333
mean for Y: 4.133333333333334
std for X: 1.2342339054382412
std for Y: 2.0744477176668816
covariance: -0.34166666666666656
pearson: -0.13344510392136738
```

Q3: Four in Line Game - 40 pts



Consider an NxN board for the Four in Line game, where N ≥ 3. You can simulate this board using a two-dimensional list (list of lists). The game has two players: 'r' for red and 'y' for yellow. Each position on the board contains either:

- the character 'r'
- the character 'y'
- or the character '.' (dot) to denote that position is currently empty

Write a function four_in_line(board) that takes as input an NxN board and determines whether the game has been won by any of the players at that moment.

Player 'r' has won the game if:

- A row has four consecutive 'r' values (horizontal)
- A column has four consecutive 'r' values (vertical)
- There are diagonally four consecutive 'r' values (can be any diagonal)

The same winning conditions apply to player 'y' (just swap 'r' by 'y' above).

If neither player has won, then the game is tied, or it is still ongoing. Your function four_in_line should return the character 'r' if red has won, the character 'y' if yellow has won, or the character '.' if the game has not yet been won by either player.

Important:

- Your function should work for any N ≥ 3. You can find N inside the four_in_line function, from the input parameter board.
- You do not have to check for erroneous inputs. You can assume the input board is always valid (i.e., only contains r/y/. and has size NxN).
- You can assume that on a given board, there cannot be a situation where players 'r' and 'y' simultaneously satisfy the winning conditions.

Open FourInLine.py, implement your code inside the function: four_in_line.

Examples:

```
board1 = [['.', '.', '.', '.', '.'],
          ['.', 'r', '.', 'y', '.'],
          ['.', 'r', '.', 'r', '.'],
          ['.', 'r', 'y', 'y', '.'],
          ['y', 'r', 'r', 'y', 'y']]
print(four_in_line(board1))
```

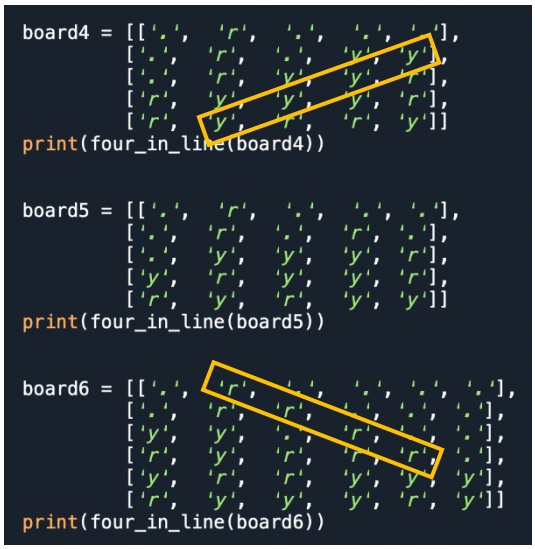
will return 'r'

```
board2 = [['.', '.', '.', '.', '.', '.'],
          ['.', '.', '.', '.', '.', '.'],
          ['y', '.', '.', 'r', '.', '.'],
          ['r', '.', 'r', 'r', '.', '.'],
          ['r', '.', 'r', 'y', 'y', 'y'],
          ['r', 'y', 'y', 'y', 'y', 'r']]
print(four_in_line(board2))
```

will return 'y'

```
board3 = [['r', '.', '.', '.'],
          ['y', 'r', '.', '.'],
          ['y', 'y', 'r', 'r'],
          ['y', 'y', 'r', 'r']]
print(four_in_line(board3))
```

will return 'r'



will return ‘y’

will return ‘.’

will return ‘r’