Departamento de Informática
Universidad Carlos III de Madrid

Bachelor's Degree in Data Science and Engineering

Programming
Final Exam

January 2019

General indications

- You have **3 hours**.
- You are not permitted to leave the room during the exam unless you have handed it in.
- The exams cannot be written in pencil.
- All answer sheets should be numbered and with your name on them.
- Built-in functions not seen during the course are forbidden; their use will be penalized.
- Notes, slides, books, etc. are allowed; electronic devices are not allowed.

Exercise 1 (5p)

The aim of this exercise is to perform a simulation to verify if a Python random list generator produces sequences that are really random or follow a certain pattern. In order to do that, the program generates several lists of random integers between 0 and 19 (both included). For each created list, the program logs the position of the lowest number in the list. In this way, it can verify if these positions are distributed uniformly as will be explained later.
Create a Python program that:

- Asks the user the size of the list and the maximum number of times that the lowest number can be located at the same position in the list.
- Performs simulations, randomly generating the list of numbers, until (1) a total of 100 simulations are performed (a new list is generated for each simulation), or (2) the lowest number for such list is located at any position of the list the maximum number of times specified by the user.
- For each simulation, it shows the generated list with the lowest number enclosed in `<< >>`.
- Once the simulations are done, it shows the number of times that the lowest value is located at each position of the list.
- Finally, it checks the quality of the generator as follows: if the frequency of the lowest number at any position of the list is at least the double of the expected frequency, which is the number of iterations divided by the length of the list, the program will print `PROBLEMATIC SEQUENCE`.

The code must contain functions for those parts of your code that include repeated tasks.
An example of the expected execution is:

```
Length of the list: 6
Maximum number of repetitions: 3
13 19 12 17 14 <<11>>
16 <<3>> 14 3 4 18
15 9 9 12 <<0>> 14
9 6 7 16 12 <<2>>
17 12 8 13 <<3>> 17
13 19 16 11 14 <<0>>
Frequency of the minimum number:
Position 0: 0
Position 1: 1
Position 2: 0
Position 3: 0
```

```
Position 4: 2
Position 5: 3
PROBLEMATIC SEQUENCE
```

Solution

```
import random

def generateRandomList(size):
    l = []
    while size > 0 :
        l.append(random.randrange(0,20))
        size -= 1
    return l

def findPositionMinNumber(l):
    p = 0
    lowest = 20
    for i in range(len(l)):
        if l[i] < lowest:
            lowest = l[i]
            p = i
    return p

def printList(l, pos):
    s = ''
    for i in range(len(l)):
        if i != pos :
            s += str(l[i]) + ' '
        else:
            s += '<<' + str(l[i]) + '>> '
    return s

def checkFrequencyPosMinNumber(l):
    maxFrequency = 0
    for i in range(len(l)):
        if maxFrequency < l[i]:
            maxFrequency = l[i]
    return maxFrequency

###

size = int(input('Length of the list: '))
pos_min = int(input('Maximum number of repetitions: '))
num_sim = 0
max_min = 0
# list to keep track of the number of times the lowest number is in a position
counterList = []
for _ in range(size):
    counterList.append(0)

while num_sim < 100 and max_min < pos_min:

    # generates the random list
    randomList = generateRandomList(size)
    # find the position in the list of the lowest number
    pos = findPositionMinNumber(randomList)
```

```
    # print the list with the lowest number within << >>
    print(printList(randomList, pos))
    # updates the counter for such position
    counterList[pos] += 1
    # computes the frequency of the minimum number
    max_min = checkFrequencyPosMinNumber(counterList)
    num_sim += 1

print('Frequency of the minimum number: ')
for f in range(len(counterList)):
    print('Position %i: %i'%(f,counterList[f]))

if (2*(num_sim/size)) <= max_min:
    print('PROBLEMATIC SEQUENCE')
```

Exercise 2 (5p)

A company logs the amount of money of its clients' purchases. To reward its best clients, the company decides to send a special offer to those that have spent more than 500 Euros in three consecutive purchases. The company has 100 clients. Each client has an id and can perform 10 purchases at most. The company stores the information in a dictionary of clients. To do this exercise, it is not allowed the use of properties, setters, or any method directly changing or receiving the values of any attribute.

- Define the needed classes and their constructors taking into account that all attributes must be private. Implement their constructors.
- Implement a method called `computeOffer()`, with no parameters and no return statement, which determines for each client if he/she must receive an offer. Specify the class you consider most appropriate to place this method. Implement auxiliary methods if needed.
- In a main program, create 100 clients. Each client will have 10 random purchases between 1 Euro and 200 Euros. Print on the screen the id of those clients that will received an offer. Implement auxiliary methods if needed.

An example of the expected execution is:

```
Send offer to client 0
Send offer to client 9
Send offer to client 13
Send offer to client 20
Send offer to client 27
```

Solution

```
class Client:

    def __init__(self, idClient, purchases):
        self.__id = idClient
        self.__purchases = purchases
        self.__offer = False

    def computeOffer(self):
        total = 0
        i = 0
        while i < (len(self.__purchases) - 3) and total <= 500 :
            total = self.__purchases[i] + self.__purchases[i + 1] + self.__purchases[i + 2]
```

```
            i += 1
        self.__offer = total > 500

    def printOffer(self):
        if self.__offer :
            print('Send offer to client ' + str(self.__id))

import Client

class Company:

    def __init__(self):
        self.__client = {}

    def addClient(self, idClient, client):
        self.__client[idClient] = client

    def computeOffer(self):
        for c in self.__client.values():
            c.computeOffer()
            c.printOffer()

### main

import random

company = Company()

for i in range(100):
    purchase = []
    for j in range(10):
        purchase.append(random.randrange(1,201))
    client = Client.Client(i, purchase)
    company.addClient(i,client)

company.computeOffer()
```