Extra exercises

Programming

Remarks: there are many ways to solve these problems; some of them are better solved using OOP, while other ones are more suitable for the use of functions. In both cases, there are many ways to split the problem into different methods or functions. The solutions that will be provided are only a way to perform such decomposition; you are encouraged to try your own one.

Exercise 1. Create a program that allows the user to work with lists of `float` numbers. The program will offer the user the following options:

1) Fill randomly a list where the user selects the number of elements and the upper and lower bound.
2) Fill randomly a list where the number of elements and the upper and lower bound are in the range [1,1000).
3) Manually enter a list: the user will select the number of elements and he/she will enter their values.
4) Shrink the list: the program will create a new list with half the size of the original list and that will be the result of adding the elements of the original list by pairs (the first element of the resulting list will be the result of adding the first and second elements of the original list, the second element will be the sum of the third and the fourth, and so on). Consider the case where the original list has an odd number of elements.
5) Invert the list (without using the reverse method): once the list has been introduced, the program will return a list where the elements are placed in reverse order (the first will be the last and so on)
6) Print a list: the program will print the list, truncating the numbers to their integer values.

Example of execution:

```
How do you want to fill the list?
1) Partially random
2) Totally random
3) Manually
4
Please, enter 1, 2 or 3!
How do you want to fill the list?
1) Partially random
2) Totally random
3) Manually
1
Enter the number of elements
10
Enter the upper bound
20
Enter the lower bound
8
The generated list is
11 19 11 19 8 8 13 12 11 8
Enter the option
A) Shrink the list
B) Invert the list
C) Quit
B
8 11 12 13 8 8 19 11 19 11
Enter the option
```

```
A) Shrink the list
B) Invert the list
C) Quit
A
19 25 16 30 30
Enter the option
A) Shrink the list
B) Invert the list
C) Quit
F
Enter a right option!
Enter the option
A) Shrink the list
B) Invert the list
C) Quit
C
Thanks
```

Exercise 2. Time and date web pages are quite common in Internet. They provide tools to perform different calculations with dates and times like knowing the next day after a given one or calculating the number of seconds between two hours, the number of days between two dates, etc. The goal of this exercise is to create a program able to perform such operations. Namely the user will be able to carry out the following date-related operations:

- Selecting the date format to be used:
    i. dd/mm/yyyy
    ii. mm/dd/yyyy
    iii. yyyy/mm/dd
    iv. Month day, year (as in June 14th, 2016)
- Checking that a date is correct (taking into account leap years).
- Converting a date from one of the previous formats to any other. Note: In the next points the user must be able to enter the date in the format he/has previously chosen.
- Calculating the next day after a given one, taking into account leap years.
- Checking if a date is previous to other date.
- Calculating the number of years between two given dates.
- Calculating the number of days between two given dates.
- Printing all the days between two given ones.
- Checking if a given date is between two given ones.
- Sorting a list of dates in ascending or descending order (as chosen by the user)
- Creating a Person class with name and birthday with the following methods:
    i. A method to calculate the age of a person given his/her birthday and the current date that will be received as parameter.
    ii. A method that receives another Person object and returns 1 if this Person is older than the other one, 0 if they are the same age and -1 if this person is younger than the received one.

- In a main program, ask the user to fill a list of persons, sort them from youngest to oldest, and also print the difference in years between them as for example: "Pepe: 2 years old, he/she has the same age as Juana: 2 years old, he/she is 1 year(s) younger than Mario: 3 years old". It must work for any number of people.

Exercise 3. The goal of this exercise is to create a program to simulate a Weather Station able to measure both temperature and atmospheric pressure. To do that, the following definitions are useful:

- A timestamp is represented by hours, minutes and seconds. The program must ensure that the values for each of them make sense.
- A measure is made of a timestamp plus the quantity measured (for example 32 degrees at 12:23:48). The temperature will be measured in Celsius degrees, while the pressure will be expressed in millibars. In our case both the timestamp and the value of the measure will be randomly generated when the measure is created (choose appropriate maximum and minimum values for the values). Create `str` methods to return a String like "`1013 mb at 15:12:12`" or "`32 degrees at 08:21:13`" depending on the type of the measure.
- A Weather Station has an id, a place where it is located (which is represented by its name), two lists containing all the measures of temperature and pressure taken during the last 24 hours, the number of measures taken per hour and the mean temperature and pressure.
- Weather Stations are able to take measures; whether the measure is temperature or pressure will be received as parameter. To create a measure, a new one will be randomly generated and placed in the list of measures of temperature or pressure, which will be automatically sorted so the most recent one (in terms of its timestamp) is the first one. Once the list of measures of that type is full (all the possible measures in 24 hours are taken), no other measure will be stored into the list and new measures will be lost.
- A measure can be read in the following ways:
    i. By receiving a timestamp: the closest measure to the timestamp is returned and deleted from the list of measures.
    ii. By receiving a position on the list. As in the previous case the measure is extracted from the list and the remaining ones rearranged.
    iii. By reading the entire list. In this case, the list is emptied.
- In the main program the user will be asked about the name and id of the weather station, and the number of measures taken per hour. The user will be able to read a measure of any type (using any of the previous ways), to generate new measures (specifying the number of measures to create and their type), or to read the mean values.

Exercise 4. The goal of this exercise is to simulate a multi-poker game, a program that creates N random cards and evaluates if there are winning combinations among them. The rules are:

- Cards can take a value between 1 and 10, no suits are considered.
- The same number of cards is played in each round.
- Winning combinations are: a pair (two equal cards), a trio (three equal cards) and a straight (four consecutive cards). For example if we play (5,2,3,3,8,4) there are a pair (3,3) and a straight (2,3,4,5).
- If there is no winning combination the round is marked as "bad".

Create a program with functions to play the multi-poker game for a single player, as follows:

1. Ask the user the number of cards that will be played in each round.
2. Ask the number of "bad" rounds before the game ends.
3. For each round:
    a. Generate the N cards randomly.
    b. Print them.
    c. Print the number of pairs, trios and whether there is a straight or not.
    d. Check if the number of bad rounds has been reached and end the game if so.

Exercise 5. Create classes to simulate a banking system including the withdrawal of money in ATMs, taking into account the following:

- A bank account is open by a client on a certain bank, has an id number, a card number and holds a certain balance.
- A client has an id, an address, a name and a maximum of 5 bank accounts.
- An ATM belongs to a bank, has an address and holds a balance of the available money.

Create a program as follows:

1. Create classes to represent the former concepts.
2. Create init method and the needed properties for each class.
3. Create and place inside the most suitable class the `withdrawMoney` method that given a client and a quantity of money does the following:
    a. Checks if the client holds an account in the bank the ATM belongs to. If not, the system will warn the client and stop the transaction.
    b. Checks if the bank account balance is higher than the money to withdraw. Again the system will warn the client and stop the transaction if no enough money is available.
    c. Checks if the ATM has money enough to proceed with the transaction.
    d. Updates the balances of the user account and of the ATM.
4. Create a `fillATM` method that given an amount of money increases the ATM balance according to it.
5. Create a main program as follows:
    a. Create 3 ATMs, each one belonging to a different bank.
    b. Create 2 clients. The first one will have two accounts, each one in a different bank. The second one will have an account at the third bank.
    c. Simulate a client trying to withdraw money in a wrong bank, and then he/she doing it correctly.
    d. Print the final balance of the ATMs.

Exercise 6. The objective of this problem is to create a pursuit-evasion game following the object oriented programming approach and according to the following rules:

- The game takes place on an MxN board, which size must be selected by the user in the main method.
- There are two types of players, a hunter and some preys. The number of preys must also be selected by the user in the main program (it must be > than 0, if not 5 preys will be created)
- Both preys and hunter will start at different random positions inside the board. Also two preys cannot be placed at the same position.
- Preys move randomly in the board as follows:

  - In the initial state they will choose a random direction and move one square. Each turn they will have a 25% probability of turning right, a 25% of turning left, a 25% of keeping the previous direction and a 25% of staying put.
  - If while moving they reach the borders of the board they will "bounce back" and continue moving on the opposite direction. The same applies if when turning right or left they reach the borders of the board.
- The hunter pursues the preys as follows:
  - At the beginning of the game it calculates the distance to every prey. Distance is the number of squares it has to travel to reach the position of a prey. Diagonal movements are not allowed. Once all the distances are calculated, preys are sorted by their distance and the hunter moves one square each turn in the direction of the closest prey.
  - When the hunter reaches the current position of the prey (taking into account that the prey will also move each turn) the prey is caught and it disappears from the game. The hunter will calculate again the distances to the remaining preys and will sort them, pursuing the closest one.
- The game ends when all the preys have been caught or after 200 turns.
- Every turn the following tasks must be performed:
  - Each prey will be moved to its new position.
  - The hunter will move one position in the direction of the prey it is pursuing.
  - The hunter will check if the prey has been caught. If so the prey will be removed from the game and a new prey will be selected for the next turn. If no preys are alive, the game will end.
  - The current state of the board is printed as in the example, with squares represented by 'S', preys by an id number and hunter by 'H'.
- Create a class structure to implement the game: it needs to have at least classes to represent the preys, the hunter and the board. Define the fields that you consider relevant for all of them, as well as init method and properties as needed. Implement at least methods to: create the initial state of the board, calculate the distance to a prey, move a prey, move the hunter, sort the preys by distance, and check if the prey has been caught. Create also a `str` method that will be used to print the board. Place the methods inside the class you consider more appropriate.

Example of initial state

```
Enter the size of the board and the number of preys
5
5
3
2SSSS
SSSS0
SSSSS
SSHSS
SS1SS
```