PROBLEMS – Lesson 8: Object Oriented Programming

Problems in yellow → Recommended problems. The rest of the problems are of intermediate difficulty. If you have any difficulty moving from one recommended problem to the next one, do some of the problems that appear in between for more progressive learning.

Problem 1

Make a program that creates a new data type (class) to represent a number as a fraction. At the data level we must be able to represent the numerator and the denominator.

The following functionalities must be implemented:

1. Addition and subtraction between fractions
2. Print, using the / symbol to represent the class instances (e.g. 3/2)
3. Convert a fraction to float, with a function named *frac2float()*)
4. Invert a fraction, with a function named *inv()*

Problem 2

A bank has three types of bank accounts:

- Current Account: an account that allows you to direct debit a payroll, associate cards, make transfers, make deposits and refunds or direct debit receipts, among the most common operations. Current accounts do not offer any kind of return.
- Remunerated Account: offers a small return on savings deposited in this account. Return is paid monthly. In this type of account, you cannot debit receipts or payroll, or associate a card to make payments.
- Fixed Account: offers a higher return on savings deposited but, for a period of time, no funds can be withdrawn. Return is paid monthly.

Define a class for each account type. All classes have an account holder (owner) and an amount of money. In the case of a remunerated and fixed account, they have an interest. In the case of a fixed account, you also have a period of time (months).

The return for the Remunerated Account is calculated using the formula of the compound capital:

$$Return = Capital_{initial} * [\,(interest_{between\ 0-1})^{time} - 1]$$

In the case of the Fixed Account, the return is calculated based on the simple capital formula:

$$Return = Capital_{initial} * (interest_{between\ 0-1} * time)$$

We want to calculate the amount that we will have in each of the three types of accounts.

Problem 3

Declare a new class, Book, with the fields: title, publisher, pages and price. Then make a program where two variables of the new data type are declared and the user enters the data from his two favorite books (a first one and then the second one) to fill in the two instances that have been created. Finally, print the contents of the two instances to check that the data has been correctly saved. The format of the outgoing message must be:

```
Favorite books:
<title 1> <editorial 1> <pages 1> <price 1>
<title 2> <editorial 2> <pages 2> <price 2>
```

Problem 4

Declare a new class, `Team`, as a class to store information about football league teams (team name, city of origin, points in the league, team budget in millions of euros). The Attributes will be: `name`, `city`, `points` and `budget`. Implement the following methods:

- Method for displaying the teams' data in the following format, when calling `print`:

```
Team: <name> <city> <points> <budget>
```

- Write all the methods to compare the points between teams:

`__lt__()` for `a < b`.	`__ge__()` for `a >= b`.
`__gt__()` for `a > b`.	`__ne__()` for `a != b`.
`__le__()` for `a <= b`.	`__eq__()` for `a == b`.

Problem 5

Define the `Point` class to represent a point in a 2D plane. The methods to implement are:

1. `distance_origin`. Distance of a point to the origin (0,0)
2. `distance`. Distance between two points
3. `midpoint`. Midpoint between two points
4. `slope`. Slope between two points
5. When printing an instance of the `Point` class, the output is `(x,y)`

Problem 6

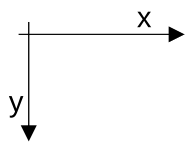
Define the `Rectangle` class to represent a rectangle in a 2D plane. The attributes will be:

1. a point, named `origin`, of the `Point` type (class defined in the previous problem).
2. the width of the rectangle, `width`.
3. the height of the rectangle, `height`.

The methods to implement are:

1. `area`. It will return the area of the rectangle.
2. `perimeter`. It will return the perimeter of the rectangle.
3. `is_square`. It will return `True` in case it is a square, `False` otherwise.
4. `zoom`. It will receive a real value as a parameter. It will return a `Rectangle` where the input parameter has been applied as a multiplicative factor.
5. `move`. It will receive as a parameter two real values. It will return a `Rectangle` where a shift to the origin point has been applied.
6. `get_vertex`. It will return a list of points (`Point`) with the 4 vertices that form the rectangle. The order in which it will be saved is: First the origin, and then, the rest of the vertices in clockwise direction.

   NOTE: in computer science the coordinate axes are:



7. `contains`. It will receive a point as a parameter. It will return `True` if the point is contained in the Rectangle, `False` otherwise.
8. `overlap`. It will receive a rectangle as a parameter. It will return `False` if there is no intersection between the two rectangles. Otherwise, it will return the Rectangle intersection.

Problem 7

Create a class called `Student` with the following attributes:

NIU, Name, Surname, Grade

Make a second class, `Subject`, with the following attributes:

CodeSubject, SubjectName and StudentList

Make a program that:

1. Requests the code and name of the subject and creates an instance. Initially, the list of students is empty.
2. Requests the information of the students enrolled in the subject.
   2.1. It will ask the NIU and it will check that it does not exist in the student list. To do this, create a method associated to the `Subject` class that returns `False` in case the student is not in the list. In case it exists, a message will be displayed:

   ```
   Error: Existing student
   ```

   2.2. In case it is not there, the rest of the information will be requested. It must control if there is an error in the note entry (`float`), raising an error of type `ValueError`, and avoiding the program to end. It will deal with the exception, with the message:

   ```
   Error: Incorrect input type
   ```

   and will ask for the grade again.

3. Once we have all the information, the instance will be created and added to the list of the subject. To do this, create a method associated with the `Subject` class.
4. Once this process has been completed, it will ask if the user wants to introduce more students. If so ('Y' or ‘y') the previous steps will be repeated.
5. At the end, it will print the list of students, in the following format (one student per line):

<NIU> <name> <surnames> <grade>

Problem 8

Define the `Film` class, with the attributes: title of the film, director's name, year of release and a list of the main actors.

Besides from the constructor we will implement the `__str__` operator to print the objects of type `Film` and the operator `__ge__` (greater or equal >=) to compare two films. One film is “greater” than another if it was released before.

Finally, we will define a `PushActor(a)` function to add actors to a certain movie, and `PopActor(a)` to delete one actor from the movie.

Problem 9

Declare a class to represent complex numbers. Implement the methods to add, subtract and multiply two complex numbers and calculate the conjugate of a complex number.

Also implement the method to print a complex number in the format: *(Re, Im i).*

Make a program that simulates a calculator of complex numbers that calls the methods of the class. The program should present a menu with all possible options and ask the user to choose an option. The selected operation must then be performed, and the result displayed on the screen. The users should be allowed to perform as many operations as they want, until they select the finish option. Implement each operation (addition, subtraction, multiplication, and conjugation) as a procedure that receives the info needed for the operation and to return the result of the operation.

Problem 10
Declare a new class, `Coordinate`, with the fields degrees, minutes and seconds that will be of integer type, and the direction (e.g. N,S) that will be a character.

Also declare a new class, `Location`, where the place fields will be a string, whereas the latitude and longitude fields will be of type `Coordinate`.

Make a program that asks the user for the data to create an instance of the `Location` class. But, prior to creation, the program must check that the coordinates are correct. A coordinate is correct if $0 \leq$ degrees ≤ 180, $0 \leq$ minutes ≤ 60 and $0 \leq$ seconds ≤ 60. In the case of Latitude, the letter is 'N' or 'S' and in the case of Longitude the letter is 'E' or 'O'.

If the data defining Latitude and Longitude is incorrect, the message "*Error: incorrect latitude*" or "*Error: incorrect longitude*" must be displayed on the screen, and then, the data is requested again. The process must be repeated until the input coordinate is correct.

Once all checks are correct, the `Location` class will print the instance on the screen in the following format: *"XXXX - latitude: GG degrees MM 'SS"L, longitude: GG degrees MM' SS"L"*

where XXXX will be the name of the location, GG, MM, SS will be the values of degrees, minutes and seconds of the coordinates and L will be the letter of each coordinate.

Problem 11
Define the `Point2D` class to represent a point in a 2D plane. Attributes are the `x` and `y` component. Implement the getters member functions: `get_x`, `get_y` and setters: `set_x`, `set_y`. Also implement the `distance` function and `__str__` to print with the format: (x, y).

Derive the `Point3D` class to represent a point in a 3D plane. The z component attribute must be added. Implement the member function *getters* : `get_z` and *setters* : `set_z`. Also redefine the `distance` function and `__str__` to print using the following format: (x, y, z) .

Problem 12
The university community is made up of: Students, PAS (Administration and services staff) and Faculty. The attributes that define each group are:

Student		PAS		Teachers	
- First name	str	- First name	str	- First name	str
- Surnames	str	- Surnames	str	- Surnames	str
- Home_address	str	- Home_address	str	- Home_address	str
- Career	str	- Faculty	str	- Faculty	str
- Year_Start	int	- Salary	int	- Salary	int
- Fee	int			- Subjects	list (str)

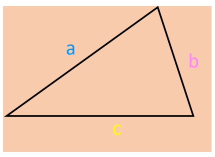
Create the necessary hierarchy to implement these classes.

Problem 13

A polygon in a plane can be defined from a list of vertices. A vertex is a 2D point:

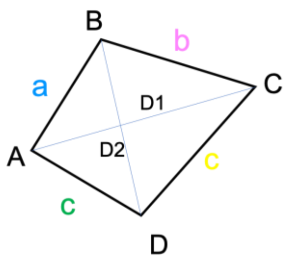
1. Define the `Point` class to represent a point in a 2D plane.
    a. Attributes must be the *x* and *y* component.
    b. Implement member functions:
        i. `distance` calculates the distance between two points.
        ii. The function `__str__` prints the object with the following format: *(x, y).*
2. Define the `Polygon` class to store a polygon through a list of points as attributes `Point` called `vertices`. The initialization function should ensure that the list is greater than 2. Perform the check using an `assert`. If it is not the case, then return the message: "*Vertex number cannot be lower than 3*".
    a. Implement the member functions:
        i. `add_point` to add a new vertex.
        ii. `perimeter` to calculate the perimeter of a polygon.
        iii. The function `__str__` to print the vertices of a polygon with the following format: < (x_0 , y_0) (x_1 , y_1) ... (x_{n-1} , y_{n-1})> .
3. Define a class called `Triangle` to store a triangle:
    a. The initialization function must ensure that the list has 3 vertices. Check it using an `assert`. If not the case, return the message, "Vertex number should be 3".
    b. Implement the member functions:
        i. `area` to calculate the area of a triangle:

$$A = \sqrt{S * (S-a)(S-b)(S-c)} \qquad S = \frac{(a+b+c)}{2}$$



        ii. `add_point` will `raise` an exception with the message "A triangle must have 3 vertices".
4. Define a class called `Quadrilateral` to store a square:
    a. The initialization function must ensure that the list has 4 vertices. Check that using an `assert`. If not the case, return the message, "Vertex number should be 4".
    b. Implement the member functions:
        i. `area` to calculate the area of a quadrilateral.

$$A = \frac{1}{4}\sqrt{(2 * D_1 * D_2)^2 - (b^2 + d^2 - a^2 - c^2)^2}$$



        ii. `add_point` must raise an Exception with the message "A quadrilateral must have 4 vertices".

Problem 14

There are forty-eight cards in a spanish deck of cards, each one belongs to one of the four suits and one of the twelve ranks. The suits are gold, clubs (*bastos*), swords and cups. The ranks are Ace, 2, 3, 4, 5, 6, 7, 8, 9, jack (*sota*), Horse, King.

Create a `Card` class to represent a playing card, with the obvious attributes: *rank* and *suit*. It's not so obvious what type of attributes they should be. One possibility is to use strings that contain words. One problem with this implementation is that it would not be easy to compare cards to see which one had a higher rank or suit. An alternative is to use integers to encode ranges and suits. In this context, we should code between numbers and suits, or between numbers and ranges. For example, this table shows the corresponding suits and integer codes, which will make it easier to compare cards:

Gold	3		Swords	1
Clubs (*bastos*)	2		Cups	0

The assignment of ranks is easier: each one of the numerical ranges is assigned to the corresponding integer, and for cards with figures, we assign:

As	0
Jack (*sota*)	9
Horse	10
King	11

Develop the member function `__str__` so that the message is: *<rank_value> of <suit_value>*.

Two more classes must be implemented: `Deck` and `Hand`. Both classes are a list of cards. In the case of the `Deck`, we have 52 cards. In the case of `Hand`, it depends on the type of game.

In the case of `Deck`, we need to implement:

- initialize all cards and add them to a list.
- print all cards, one card per line.
- `pop_card`: remove a card and return it.
- `add_card`: add a card to the deck.
- `mix`: to mix means to "disorder". Use *random.shuffle(list)*. Import the *random* module.

Derive the `Hand` class from the `Deck` class. `Hand` will inherit from `Deck` methods such as `pop_card` and `add_card` that are interesting for the functionalities that the `Hand` class needs. `Hand` also inherits `__init__` from `Deck`, but we do not want to create a list with 52 cards. For this reason, instead, the *init* method for `Hand` should initialize cards with an empty list.

Problem 15

Define the classes needed to store the information of a library's materials: Books and magazines.

- For a book, we need: *Title, Publisher, Author, Year of publication.*
- For a magazine, we need: *Title, Editorial, Number, Periodicity, Year of publication.*

Create the necessary hierarchy to be able to represent all publications.