Python POO – continuare

Rezolvarea problemelor propuse data trecuta:

1. Implementați clasa Complex, împreună cu operaţiile de adunare, înmulţire şi calculul modulului unui număr complex.

```
class Complex:
    def __init__(self, real, imaginar):
        self.r = real
        self.i = imaginar

    def aflare_semn(self, im):
        if im < 0:
            im = im * -1
            semn = '-'
        else:
            semn = '+'
        return semn, im

    def inmultire(self, nr_complex):
        re = (self.r * nr_complex.r - self.i * nr_complex.i)
        im = (self.r * nr_complex.i + nr_complex.r * self.i)
        print('Inmultire: ' + str(re) + ' ' + self.aflare_semn(im)[0] + '
' + str(self.aflare_semn(im)[1]) + 'i')

    def adunare(self, nr_complex):
        re = (self.r + nr_complex.r)
        im = (self.i + nr_complex.i)
        print('Adunare : ' + str(re) + ' ' + self.aflare_semn(im)[0] + ' '
+ str(self.aflare_semn(im)[1]) + 'i')

    def scadere(self, nr_complex):
        re = (self.r - nr_complex.r)
        im = (self.i - nr_complex.i)
        print('Scadere : ' + str(re) + ' ' + self.aflare_semn(im)[0] + ' '
+ str(self.aflare_semn(im)[1]) + 'i')

z1 = Complex(-2, -5)
z2 = Complex(-3, -6)
z1.inmultire(z2)
z1.adunare(z2)
z1.scadere(z2)
```

2. Implementați o ierarhie de clase pentru următoarele figuri geometrice:
    a. Triunghi din care sunt derivate clasele triunghi isoscel și triunghi echilateral.
    b. Patrulater – din care sunt derivate clasele paralelogram, dreptunghi, romb si pătrat.
    c. Cerc.

   Clasele vor conține metode pentru calculul perimetrului și ariei figurilor geometrice.

```
from math import sqrt
from math import pi

#TRIUNGHIE//////////////////////////////////////////////////////////////////
//////////////////////////////////////////
class Triunghi:
    def __init__(self, cateta1, cateta2, ipotenuza):
        self.c1 = cateta1
        self.c2 = cateta2
        self.ip = ipotenuza

    def perimetru(self):
        return self.c1 + self.c2 + self.ip

    def arie(self):
        pass

class Isoscel(Triunghi):
    def __init__(self, cateta1, cateta2, ipotenuza):
        Triunghi.__init__(self, cateta1, cateta2, ipotenuza)

    def perimetru(self):
        return Triunghi.perimetru(self)

    def arie(self):
        self.med = sqrt(self.c1 ** 2 - (self.ip / 2) ** 2)
        return (self.ip * self.med) / 2

class Echilateral(Triunghi):
    def __init__(self, cateta1, cateta2, ipotenuza):
        Triunghi.__init__(self, cateta1, cateta2, ipotenuza)

    def perimetru(self):
        return Triunghi.perimetru(self)

    def arie(self):
        return (self.c1 ** 2 * sqrt(3)) / 4

#PATRULĂTRATURI/////////////////////////////////////////////////////////////
//////////////////////////////////////////
class Patrulater:
    def __init__(self, latura1, latura2):
        self.l1 = latura1
```

```
        self.l2 = latura2

    def perimetru(self):
        return 2 * (self.l1 + self.l2)

    def arie(self):
        pass

class Patrat(Patrulater):
    def __init__(self, latura1, latura2):
        Patrulater.__init__(self, latura1, latura2)

    def perimetru(self):
        return  Patrulater.perimetru(self)

    def arie(self):
        return self.l1 * self.l2

class Dreptunghi(Patrat):
    def __init__(self, latura1, latura2):
        Patrat.__init__(self, latura1, latura2)

    def perimetru(self):
        return Patrat.perimetru(self)

    def arie(self):
        return self.l1 * self.l2

class Paralelogram(Patrulater):
    def __init__(self, latura1, latura2, inaltime):
        Patrulater.__init__(self, latura1, latura2)
        self.ina = inaltime

    def perimetru(self):
        return Patrulater.perimetru(self)

    def arie(self):
        if self.l1 >= self.l2: return self.l1 * self.ina
        else: return self.l2 * self.ina

class Romb(Paralelogram):
    def __init__(self, latura1, latura2, inaltime):
        Paralelogram.__init__(self, latura1, latura2, inaltime)

    def perimetru(self):
        return Paralelogram.perimetru(self)

    def arie(self):
        return Paralelogram.arie(self)

#CERCUȚUL//////////////////////////////////////////////////////////////////
//////////////////////////////////////////
class Cerc:
    def __init__(self, raza):
```

```
        self.r = raza

    def perimetru(self):
        return 2 * pi * self.r

    def arie(self):
        return pi * self.r ** 2

tI = Isoscel(5,5,6)
tE = Echilateral(5,5,6)
p = Patrat(4,4)
d = Dreptunghi(4,6)
pp = Paralelogram(3,5, 2)
c = Cerc(5)
print('Perimetru T Isoscel: ' + str(tI.perimetru()))
print('Arie T Isoscel: ' + str(tE.perimetru()))
print('Perimetru T Echilateral: ' + str(tI.arie()))
print('Arie T Echilateral: ' + str(tE.arie()))
print('Perimetru Patrat: ' + str(p.perimetru()))
print('Arie Patrat: ' + str(p.arie()))
print('Perimetru Dreptunghi: ' + str(d.perimetru()))
print('Arie Dreptunghi: ' + str(d.arie()))
print('Perimetru Paralelogram: ' + str(pp.perimetru()))
print('Arie Paralelogram: ' + str(pp.arie()))
print('Perimetru Cerc: ' + str(c.perimetru()))
print('Arie Cerc: ' + str(c.arie()))
```

Rezumatul notiunilor principale ale POO in Python

1. Clase si obiecte
2. Incapsularea
3. Mostenirea
4. Polimorfismul

1. Clase si obiecte

```
2. class Person:
3.     def __init__(self, name, age):
4.         self.name = name
5.         self.age = age
```

- class – Here is a class named Person.

- Constructors have the default name __init__. They are functions that are implicitly called when an object of the class is created.
- All instance methods including the constructor have their first parameter as self.
- self refers to instance that is being referenced while calling the method.
- name and age are the instance variables.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

if __name__ == "__main__":
    p = Person("ranjeeta", 23)
    print(p.name)
```

- p – is the name of the object that we are creating based on Person class
- Even though the class has three parameters (self, name, age), we'll still pass only name and age while creating an object, as we don't need to refer self here. It's implicit.
- Once an object is created, you can refer to the attributes of the object using a dot. For example, p.name refers to the name attribute of that particular object

2. Mostenirea in Python

As the name suggest, this concept is about inheriting properties from an existing entity. This increases reusability of code. Single, Multiple and Multi-level inheritances are few of the many types supported by Python.

```
class Person:
  def __init__(self):
    pass

# Single level inheritance
class Employee(Person):
  def __init__(self):
    pass

# Multi-level inheritance
class Manager(Employee):
  def __init__(self):
    pass
```

```
# Multiple Inheritance
class Enterprenaur(Person, Employee):
  def __init__(self):
    pass
```

3. Incapsularea

It is the concept of wrapping data such that the outer world has access only to exposed properties. Some properties can be hidden to reduce vulnerability. This is an implementation of data hiding. For example, you want buy a pair of trousers from an online site. The data that you want is its cost and availability. The number of items present and their location is information that you are not bothered about. Hence it is hidden.

In Python this is implemented by creating private, protected and public instance variables and methods.

Private properties have double underscore (__) in the start, while protected properties have single underscore (_). By default, all other variable and methods are public.

Private properties are accessible from within the class only and are not available for child class(if inherited). Protected properties are accessible from within the class but are available to child class as well. All these restrictions are removed for public properties.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def _protected_method(self):
        print("protected method")
    def __private_method(self):
        print("privated method")

if __name__ == "__main__":
    p = Person("mohan", 23)
    p._protected_method() # shows a warning
    p.__private_method()  # throws Attribute error saying no such method exists
```

4. Polimorfismul in Python

This is a concept where a function can take multiple forms depending on the number of arguments or type of arguments given to the function.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def show_salary(self):
        print("Salary is unknown")

class Employee(Person):
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.salary = salary
    def show_salary(self):
        print("Salary is", self.salary)
if __name__ == "__main__":
    p = Person("y", 23)
    x = Employee("x", 20, 100000)
    p.show_salary() # Salary is unknown
    x.show_salary() # Salary is 100000
```

In the above example, super keyword is used to call a method of parent class. Both classes have the method show_salary. Depending on the object type that makes a call to this function, the output varies.

Python has inbuilt-polymorphism functions as well. One of the simplest examples is the print function in python.

```
print("Hello there", end=" ")
print("I am Aanisha")
print(len([1, 2, 3, 4, 5]))
```

Exercitii propuse:

1. Sa se implementze o clasa pentru un sir de numere. Clasa sa contina metode pentru calcularea elementului maxim, minim, mediei si a medianei sirului.
2. Aplicatie: gestiune studenti (POO)
   - membri (variabile): nume, specializare, an de studiu, mai multe note (vector)
   - clasa va permite afisarea numelui, disciplinei pe fiecare student si afisarea unei note
   - clasa va permite modificarea numelui, diciplinei și modificarea unei note
   - pentru afisarea / modificarea mai multor note se va repeta apelul functiei
   - clasa va permite verificarea unui student, daca este restantier sau nu (daca are vreo nota sub 5)
   - afisarea listei de restantieri se va face aplicand metoda de verificare manual pe fiecare student initializat

Programul va face urmatoarele:

- initializeaza 3 studenti
- adauga cate trei note pentru fiecare student
- afiseaza datele studentilor
- se modifica specializarea pentru un student
- se modifica anul pentru un student
- se reafiseaza studentii
- se afiseaza lista de restantieri

Rezolvare problema 1

```
class Sir:
    def __init__(self, s):
        self.sir = s

    def calculare_maxim(self):
        max = self.sir[0]
        for i in range(1,len(self.sir)):
            if self.sir[i] > max: max = self.sir[i]
        return max

    def calculare_minim(self):
        min = self.sir[0]
        for i in range(1,len(self.sir)):
            if self.sir[i] < min: min = self.sir[i]
        return min

    def caluclarea_mediei(self):
        medie = 0
        for i in range(0,len(self.sir)):
            medie += self.sir[i]
        return medie/len(self.sir)

    def __bubble_sort(self):
        flag = 1
        i = 0
        aux = 0
        while flag == 1:
            flag = 0
            for i in range (0,len(self.sir) - 1):
                if self.sir[i] > self.sir[i+1]:
                    aux = self.sir[i]
                    self.sir[i] = self.sir[i+1]
                    self.sir[i+1] = aux
                    flag = 1
        return self.sir
```

```
    def calcularea_medianei(self):
        self.sir = self.__bubble_sort()
        if len(self.sir) % 2 == 0:
            mediana = (self.sir[len(self.sir) / 2 - 1] +
self.sir[len(self.sir) / 2 + 1]) / 2
        else:
            mediana = self.sir[int((len(self.sir) - 1) / 2)]
        return mediana

sir = [8,4,3,9,6,1,2]
array = Sir(sir)
print('Maxim: ' + str(array.calculare_maxim()))
print('Minim: ' + str(array.calculare_minim()))
print('Media: ' + str(array.caluclarea_mediei()))
print('Mediana: ' + str(array.calcularea_medianei()))
```

Rezolvare problema 2

```
class GestiuneStudenti:
    nume = None
    specializare = None
    anStudiu = None
    note = [None] * 10

    def __init__(self, nume, sp, an, note):
        self.nume = nume
        self.specializare = sp
        self.anStudiu = an
        self.note = note

    def afisare(self):
        print("Nume:", self.nume)
        print("Specializare:", self.specializare)
        print("An studiu:", self.anStudiu)
        print("Note:", self.note)

    def setNume(self, n):
        self.nume = n

    def setSpecializare(self, sp):
        self.specializare = sp

    def setAn(self, a):
        self.anStudiu = a
```

```
    def setNota(self, index, n):
        if index <= len(self.note):
            self.note[index] = n

    def verificareRestantier(self):
        for i in self.note:
            if i < 5:
                return self.nume


s = GestiuneStudenti("Andrei", "PABD", 1, [8, 10, 9])
s2 = GestiuneStudenti("Robert", "PABD", 2, [8, 9, 7])
s3 = GestiuneStudenti("Daniel", "INFO", 3, [10, 4, 9])
s.afisare()
s2.afisare()
s3.afisare()
s2.setSpecializare("EA")
s3.setAn(2)
s2.setNota(1, 4)
s.afisare()
s2.afisare()
s3.afisare()
restantieri = []
for i in [s, s2, s3]:
    r = i.verificareRestantier()
    if r:
        restantieri.append(r)
print("Restantieri:", restantieri)
```