Python - Clase

În Python, cuvântul "obiect" nu se referă neapărat la instanțierea unei clase. Clasele în sine sunt obiecte, iar, în sens mai larg, în Python toate tipurile de date sunt obiecte. Există tipuri de date care nu sunt clase: numerele întregi, listele, fișierele.

O clasă este creată folosind cuvântul cheie *class*. Atributele şi metodele clasei sunt listate într-un bloc indentat.

```
class nume_clasa:
    [instructiune 1]
    [instructiune 2]
    [instructiune 3]
```

Clasele şi metodele au o diferenţă specifică faţă de funcţiile obişnuite - ele trebuie să aibă un prefix suplimentar care trebuie adăugat la începutul listei de parametri, dar nu trebuie să-i daţi o valoare când apelaţi metoda, Python o va furniza. Această variabilă specială se referă la obiectul însuşi (engl. *self*) şi prin convenţie este numită *self*. Când ne referim la membrii clasei, vom folosi *self.membru*, într-un mod asemănător cu folosirea "*this*" din Java.

- Metoda specială **__init__()** este apelată la instanțierea clasei (crearea unui obiect de tipul clasei) și poate fi asemănată cu un constructor.
- Metoda specială **__del__()** este apelată când nu mai sunt referințe la acel obiect (mecanism de *garbage collector*) și poate fi asemănată cu un destructor.
- Instanțierea se face prin apelarea obiectului clasă, posibil cu argumente.

Exemplu: definirea clasei Complex

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

z = Complex(3.0, -4.5)
```

Valorile furnizate între paranteze sunt parametri transmiși funcției **__init__**. Aceasta va fi executată automat, inițializând proprietățile *real* și *imag* ale obiectului cu valorile 3.0 și respectiv -4.5. Instanța clasei este reținută în variabila z.

Variabile de clasă, variabile de instanță

Partea de date, atributele clasei, nu sunt altceva decât variabile obişnuite care sunt legate de spațiile de nume ale claselor şi obiectelor. Asta înseamnă că aceste nume sunt valabile numai în contextul claselor şi obiectelor respective. Din acest motiv acestea sunt numite *spaţii de nume*.

Există două feluri de *câmpuri* - variabile de clasă şi variabile de obiect/instanţă, care sunt clasificate în funcţie de *proprietarul* variabilei.

- *Variabilele de clasă* sunt partajate - ele pot fi accesate de toate instanţele acelei clase. Există doar un exemplar al variabilei de clasă şi când o instanţă îi modifică valoarea, această modificare este văzută imediat de celelalte instanţe.
- *Variabilele de instanţă* sunt proprietatea fiecărei instanţe a clasei. În acest caz, fiecare obiect are propriul exemplar al acelui câmp adică ele nu sunt relaţionate în niciun fel cu câmpurile având acelaşi nume în alte instanţe.

Exemplu: Clasa Student implementată mai jos. Atributul **numar_studenti** aparţine clasei **Student** şi este deci o variabilă de clasă. Variabilele **nume**, **prenume** şi **medie** aparţin obiectului, sunt deci variabile de obiect/instanță. Referirea la variabila de clasă **numar_studenti** se face cu notaţia **Student.numar_studenti** şi nu cu **self.numar_studenti**. Referirea la variabila de instanță nume se face cu notația **self.nume** în metodele acelui obiect. O variabilă de obiect cu același nume ca o variabilă de clasă, va ascunde variabila de clasă față de metodele clasei! Metoda **nr_studenti** este în fapt o metodă a clasei şi nu a instanţei. Asta înseamnă că trebuie să o definim cu declaraţia **static method**.

```
class Student:
    numar_studenti = 0

    def __init__(self, nume, prenume, medie):
        self.nume = nume
        self.prenume = prenume
        self.medie = medie
```

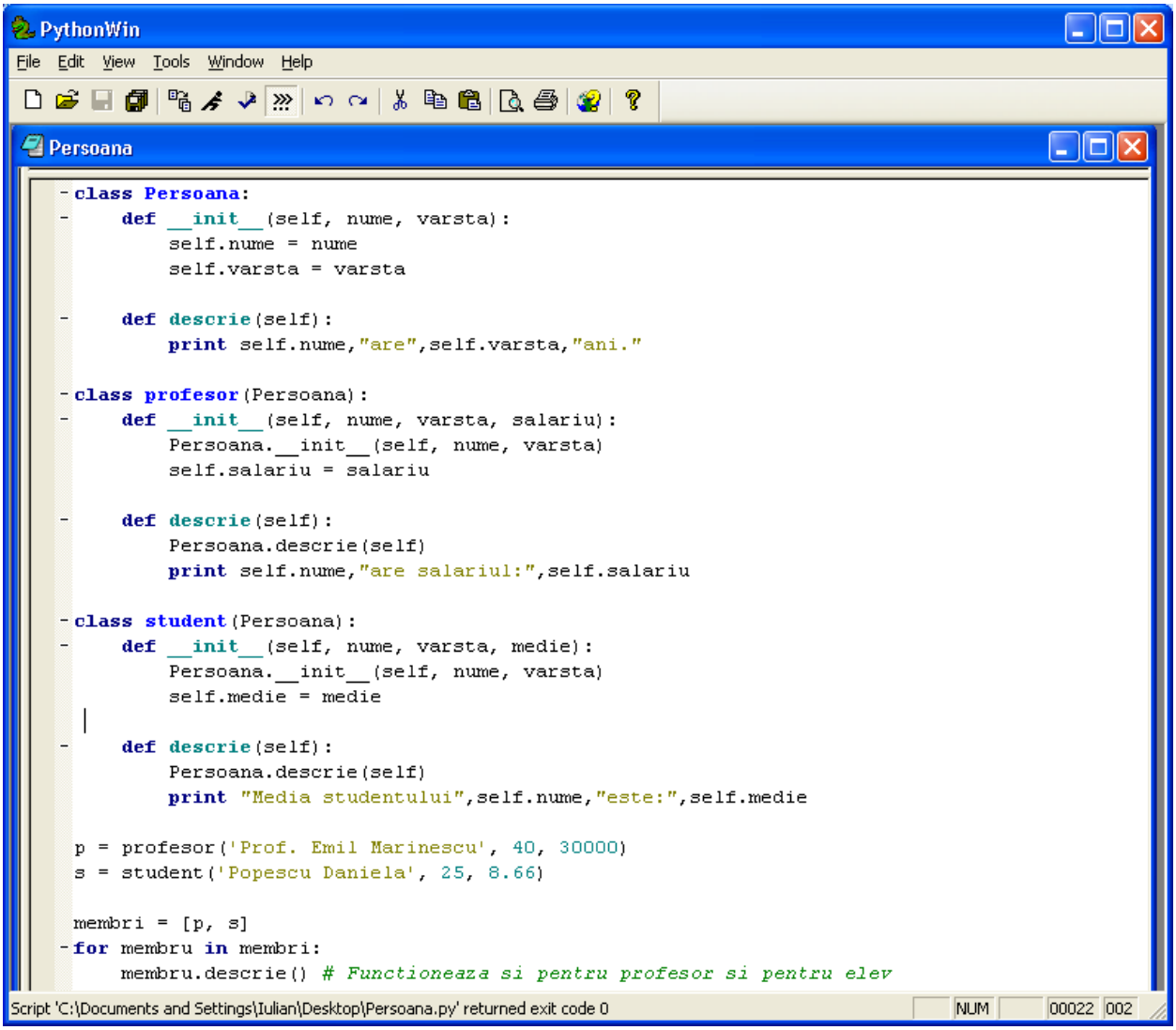
```
        print("Initializare studenti: ",self.nume,self.prenume)
        Student.numar_studenti += 1

    def test_bursier(self):
        if self.medie>=9.50:
            print("Bursa de merit")
        elif 8.50<=self.medie<9.50:
            print("Bursa studiu")

    def nr_studenti():
        print("Exista",Student.numar_studenti,"instante.")
    nr_studenti = staticmethod(nr_studenti)

student1 = Student('Bucur','Tudor',10)
student1.test_bursier()
Student.nr_studenti()
student2 = Student('Enache','Stefan',9)
student2.test_bursier()
Student.nr_studenti()
```

Moștenirea

Programarea orientată pe obiecte le permite claselor sa moștenească stările și comportamentele comune din alte clase, ceea ce înseamnă crearea unei clase copil pe baza unei clase părinte. Clasa copil conține toate proprietățile și metodele clasei părinte, dar poate include și elemente noi. În cazul în care vreuna dintre noile proprietăți sau metode are aceeași denumire ca o proprietate sau metoda din clasa părinte, vor fi utilizate cele din clasa copil.

Numele clasei părinte trebuie trecut între paranteze după numele clasei copil. Aceasta înseamnă ca noua clasa va conține toate proprietățile și metodele clasei părinte. Mai trebuie însa redefinita funcția **__init__**.

Exemplu:



Metoda __init__ a clasei bază este apelată explicit folosind variabila *self* ca să putem iniţializa partea din obiect care provine din clasa de bază. Python nu apelează automat constructorul clasei de bază, trebuie să faceți asta în mod explicit. Apelurile către clasa de bază se fac prefixând numele clasei apelului metodelor şi punând variabila *self* împreună cu celelalte argumente.

Instanţele claselor **profesor** şi **student** se folosesc ca şi cum ar fi instanţe ale clasei **Persoana** atunci când se apelează metoda **descrie** a clasei Persoana.

Dacă în tuplul de moștenire este listată mai mult de o clasă, acest caz este de *moştenire multiplă*.

Referințe și liste de obiecte

Operatorul de atribuire funcționează diferit pentru obiecte. În cazul variabilelor scalare, dacă scriem variabila2 = variabila1 înseamnă că variabila 2 va prelua valoarea variabilei 1. În cazul obiectelor, dacă avem o atribuire instanta2 = instanta1, cele două variabile vor reprezenta referințe către același obiect. Cu alte cuvinte, dacă proprietățile obiectului instanta1 sunt modificate, aceasta modificare va fi vizibilă și în instanta2.

Excepții

Unele apeluri de funcții pot arunca excepții care trebuie tratate. În Python exista mecanismul *try-except*, asemănător celui *try-catch* din Java.

```
try:
      x = int(buffer)
except(ValueError):
      print "Date de intrare invalide"
```

Mecanismul funcționează în felul următor: se execută instrucțiunile din blocul *try*. Dacă apare o excepție tratata de un bloc *except*, execuția sare la instrucțiunile din blocul respectiv. După ce excepția este tratată, execuția continuă cu prima instrucțiune din blocul *try*. Dacă apare o excepție ce nu este tratată de niciun bloc *except*, aceasta este propagată ascendent în alte blocuri *try* și primește denumirea de excepție netratată (*unhandled exception*).

O excepție poate fi aruncată folosind instrucțiunea *raise*. Aceasta poate fi folosită și fără argumente în interiorul unui bloc *except* pentru a re-arunca excepția prinsă de blocul respectiv.

```
if (j>100):
      raise ValueError,j
```

O instrucțiune *try* poate avea mai multe clauze *except*. Ultima clauză *except* poate să nu aibă specificată nicio excepție de tratat, fiind astfel folosită pentru a trata toate excepțiile netratate de celelalte clauze. Instrucțiunile *try* pot avea opțional și o clauză *else*. Instrucțiunile din blocul *else* sunt executate atunci când blocul *try* nu generează nicio excepție.

Exemplu implementare in Python clasele din C++ Timp, Cerc, Carte

```
from time import sleep

class Carte:
    nume = str    #atribute
    autor = str
    nrpg = int
    pret = float

    def __init__(self):  #constructor
        pass             #pass e folosit cand in functie nu exista operatii

    def citire_carte(self, numecarte, autorcarte, np, p):
        self.nume = numecarte        #self se refera la instanta curenta,
un fel de this (cred)
        self.autor = autorcarte
        self.nrpg = np
        self.pret = p

    def afisare_carte(self):
        print('Numele cartii: ' + self.nume)
        print('Autorul cartii: ' + self.autor)
        print('Numar pagini: ' + str(self.nrpg))
        print('Pretul cartii: ' + str(self.pret))

carte = Carte()
numecarte = input('Nume carte: ')
autorcarte = input('Autor carte: ')
numarpagini = int(input('Numar pagini: '))   #am pus int in fata pentru ca
input returneaza un string si
pretcarte = int(input('Pret carte: '))       #trebuia convertit la int
pentru a putea face operatile aferente
carte.citire_carte(numecarte, autorcarte, numarpagini, pretcarte)
carte.afisare_carte()

sleep(4)


////////////////////////////////////////////////////////////////////

from time import sleep

class Circle:
    raza = float   #atribut

    def __init__(self):     #constructor
        pass                #pass e folosit cand in functie nu exista
operatii
```

```
    def citire_raza(self, r):
        self.raza = r       #self se refera la instanta curenta, un fel de
this (cred)

    def afisate_arie(self):
        pi = 3.14
        arie = self.raza ** 2 * pi
        print('Aria cercului este: ' + str(arie))

    def afisare_circumferinta(self):
        pi = 3.14
        circumferinta = 2 * pi * self.raza
        print('Circumferinta cercului este: ' + str(circumferinta))

cerc = Circle()
cerc.citire_raza(float(input('Raza: ')))    #am pus float in fata pentru ca
input returneaza un string si
cerc.afisate_arie()                         #trebuia convertit la float
pentru a putea face operatile aferente
cerc.afisare_circumferinta()

sleep(4)



////////////////////////////////////////////////////////////////////////////
//////////////////////////

import datetime
from time import sleep

class Time:
    hour = int
    minute = int
    second = int
    time = datetime.datetime.now()

    def __init__(self):
        self.hour = int(self.time.hour)
        self.minute = int(self.time.minute)
        self.second = int(self.time.second)

    def get_hour(self):
        print('Ora: ' + str(self.hour))

    def get_minute(self):
        print('Minutul: ' + str(self.minute))

    def get_second(self):
        print('Secunda: ' + str(self.second))

    def set_hour(self, h):
```

```
        if h < 0 or h > 23:
            print('Reia clasa a 2-a!')
        else:
            self.hour = h

    def set_minute(self, m):
        if m < 0 or m > 59:
            print('Ț Ț Ț...')
        else:
            self.minute = m

    def set_second(self, s):
        if m < 0 or m > 59:
            print('Cronos ar fi dezamagit :(')
        else:
            self.second = s

    def set_time(self, o = 0, h = 0, m = 0, s = 0):
        if o == 0:
            self.hour = int(self.time.hour)
            self.minute = int(self.time.minute)
            self.second = int(self.time.second)
        else:
            if h < 0 or h > 23:
                self.hour = int(self.time.hour)
                print('Am setat eu ora.')
            else:
                self.hour = h
            if m < 0 or m > 59:
                self.minute = int(self.time.minute)
                print('Am setat eu minutul.')
            else:
                self.minute = m
            if s < 0 or s > 59:
                self.second = int(self.time.second)
                print('Am setat eu secunda.')
            else:
                self.second = s

    def print(self):
        print(str(self.hour) + ':' + str(self.minute) + ':' +
str(self.second))

    def next_second(self):
        while 1:
            self.print()
            sleep(1.0)
            if self.second == 59:
                self.second = 0
                if self.minute == 59:
                    self.minute = 0
                    if self.hour == 23:
```

```
                        self.hour = 0
                    else:
                        self.hour += 1
                else:
                    self.minute += 1
            else:
                self.second += 1

time = Time()
time.get_hour()
time.get_minute()
time.get_second()
print('Daca doriti sa setati dumneavoarstra ceasul...cu eroare
umana...apasati tasta 1.\nIar daca vreti sa il setez eu cu precizie de
chirurg apasati tasta 0.\nOptiune: ')
optiune = int(input())
if optiune == 0:
    time.set_time(0)
    time.print()
elif optiune == 1:
    h = int(input('Seteaza ora: '))
    time.set_hour(h)
    m = int(input('Seteaza minutul: '))
    time.set_minute(m)
    s = int(input('Seteaza secunda: '))
    time.set_second(s)
    time.set_time(1, h, m, s)
    time.print()
```

Exerciții propuse:

1) Implementați clasa Complex, împreună cu operaţiile de adunare, înmulţire şi calculul modulului unui număr complex.
2) Implementați o ierarhie de clase pentru următoarele figuri geometrice:
   α) Triunghi din care sunt derivate clasele triunghi isoscel și triunghi echilateral.
   β) Patrulater – din care sunt derivate clasele paralelogram, dreptunghi, romb si pătrat.
   χ) Cerc.

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