

# PYTHON FUNDAMENTALS

Curs interactiv de python

# STRUCTURA SEDINTA 3 : FUNCTII SI CLASE

▤ Tema sedinta anterioara

▤ Cunostinte avansate legate de liste, dictionare

▤ Functii

▤ Clase

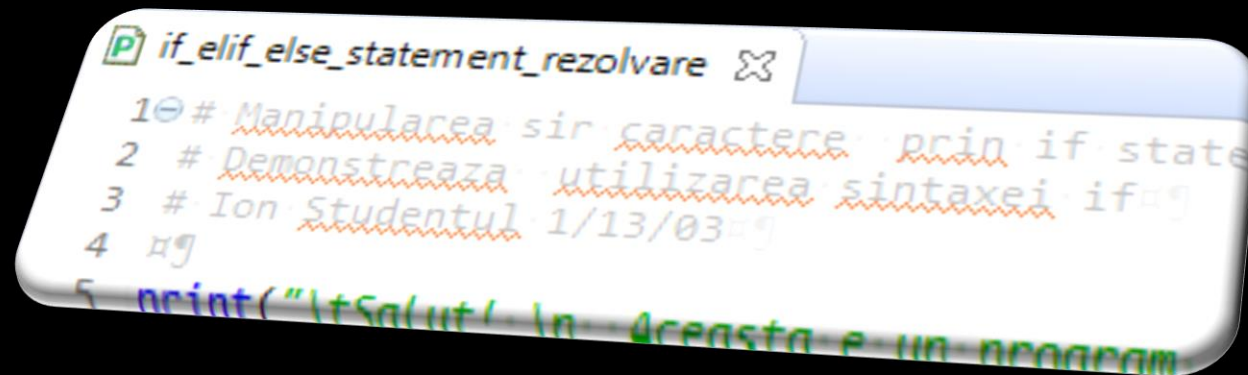
# TEMA IN CLASA SEDINTA 2

Creati un program care sa verifice daca textul introdus de un utilizator de la tastatura este un sir de tip numere sau litere.

Va rog utilizati if-elif-else.

Trebue sa afisati urmatoarele siruri de caractere:

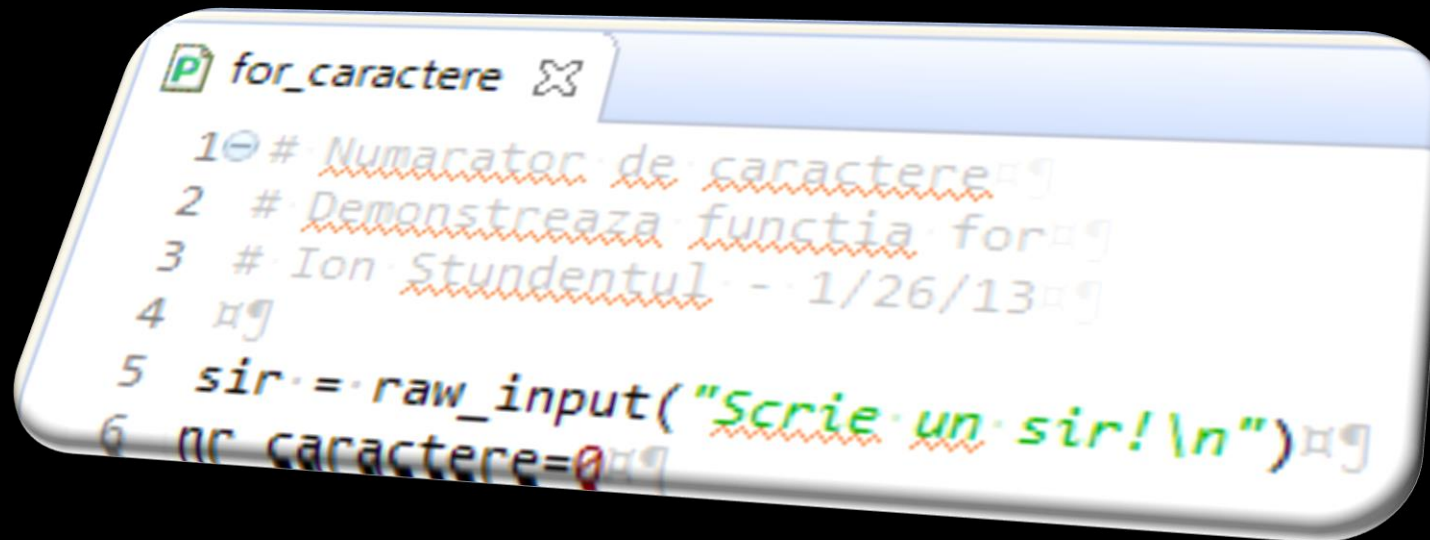
- Numar: "Sirul de caractere este format din numare"
- Litere: "Sirul de caractere este format din litere"
- Orice altceva: "Sirul de caractere este format din diferite elemente"



```
if_elif_else_statement_rezolvare ✖
1 # Manipularea sir caractere prin if state
2 # Demonstreaza utilizarea sintaxei if
3 # Ion Studentul 1/13/03
4
5 print("It's a pleasure to meet you. This is a program")
```

# TEMA IN CLASA SEDINTA 2

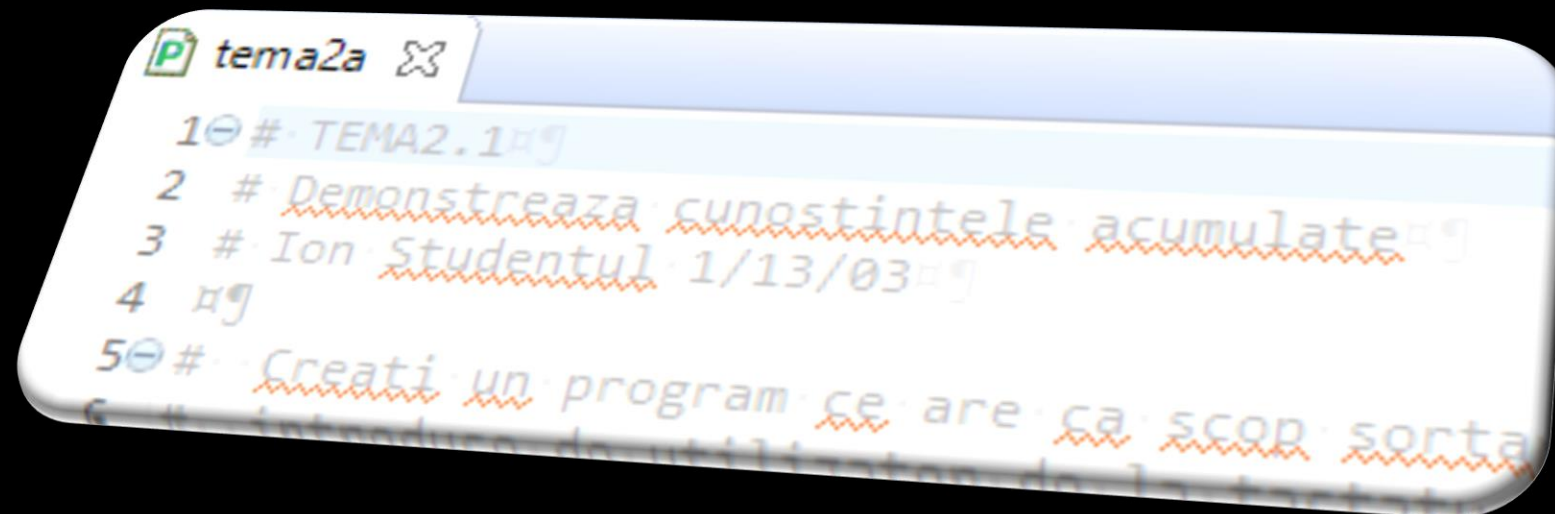
Scrieti un program ce va numara cate caractere are un sir de caractere dat de utilizator. Aceasta numarare sa se realizeze cu ajutorul unui for. La final afisati rezultatul.



```
for_caractere ✕  
1 # Numarator de caractere  
2 # Demonstreaza functia for  
3 # Ion Studentul -- 1/26/13  
4  
5 sir = raw_input("Scrie un sir!\n")  
6 nr caractere = 0
```

# TEMA ACASA SEDINTA 2

Creati un program ce are ca scop sortarea a x elemente introduse de utilizator de la tastatura printr-o bucla while. Elementele vor fi inserate pe rand intr-o lista prin acest while de catre utilizator prin capturare de text. De asemenea utilizatorul are datoria sa insereze la inceputul programului cate elemente doreste sa insereze in lista. Trebuie sa afisam lista la final.

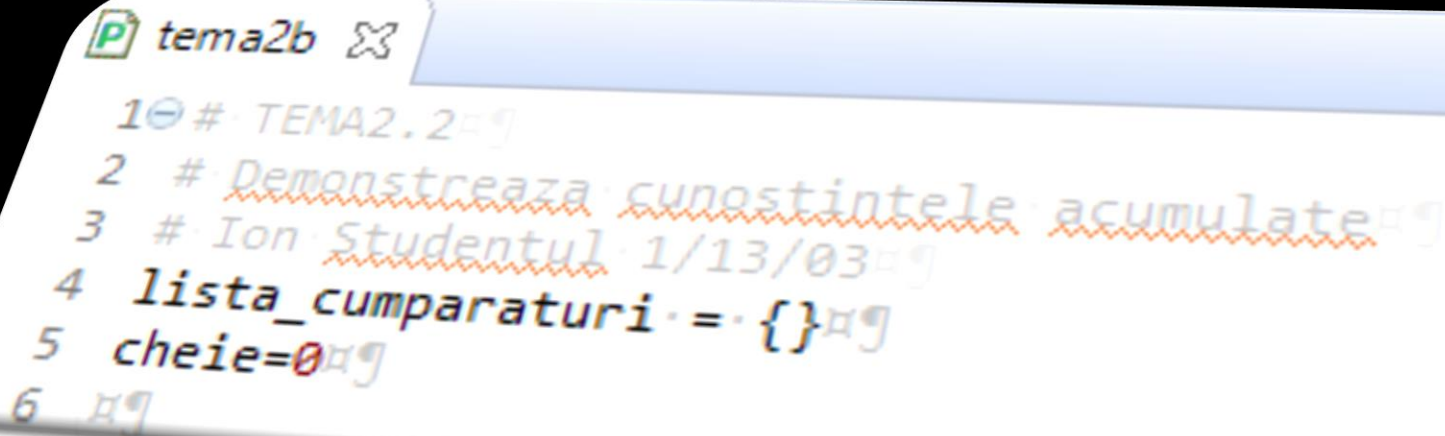


```
1 # TEMA2.1
2 # Demonstrarea cunostintelor acumulate
3 # Ion Studentul 1/13/03
4
5 # Creati un program ce are ca scop sortarea
```



# TEMA ACASA SEDINTA 2

Creati un program cu o lista de cumparaturi utilizand dictionare. Trebuie ca programul sa imite lista de cumparaturi prezentata in curs(fisierul Lista\_avansat.py transmis si pe e-mail) ca si facilitati (adaugare, stergere, afisare, iesire din program). Folositi chei numerice incepand de la 0, iar valoarea intrarii trebuie sa fie introdusa de utilizator de la tastatura sub meniul adaugare. Afisarea dictionarului se va face de forma: key=>Value



```
1 # TEMA2.2
2 # Demonstreaza cunostintele acumulate
3 # Ion Studentul 1/13/03
4 lista_cumparaturi = {}
5 cheie = 0
6
```

# TEMA SEDINTA ANTERIOARA

```
Dictionar X
1 # Dictionar de facebook
2 # Demonstreaza dictionar
3 # Ion Studentul - 1/26/13
4 dictionar={"ASAP": "As Soon As Possible-Cat mai",
5 "ASL": "Age Sex Location-Varsta, sex, locatie",
6 "BRB": "Be Right Back-Revin imediat",
7 "FYI": "For Your Info-Pentru informatia ta",
8 "LOL": "Laugh out Loud-Rad tare",
9 "PM": "Private Message-mesaj pe privat",
10 "FRUMI": "Frumos"}
11
12 print "\nAccesam dictionarul pentru a vedea ce cu
13 print dictionar.keys()
14
15 bluf qictioual kele()
16 bluf "/nAccesam qictioualul beufu a leqea ce cu
17
18
```

# CUNOSTINTE EXTRA LEGATE DE LISTE SI DICTIONARE

```
>>> lista = ["element"]*5
>>> print lista
['element', 'element', 'element', 'element', 'element']
>>> lista = ["elemnt"]+["element"]+["element"]
```

Lista poate fi concatenata si repetata



# CUNOSTINTE LEGATE DE LISTE SI DICTIONARE

```
>>> dictionar = {1:"ceva"}  
>>> dictionar*4
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#1>", line 1, in <module>  
    dictionar*4
```

```
TypeError: unsupported operand type(s) for *: 'dict' and 'int'
```

```
>>>
```

```
>>> dic1={1:"ceva"}  
>>> dic2={2:"altceva"}  
>>> dic1+dic2
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#5>", line 1, in <module>  
    dic1+dic2
```

```
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

```
>>>
```

Dictionarul  
nu poate fi  
concatenat  
si repetat

# MANIPULAREA UNUI SIR DE CARACTERE

## METODA 1

```
>>> l=[]
>>> sir = "Sir de caractere"
>>> for e in sir:
>>>     l+=e

>>> l
['S', 'i', 'r', ' ', 'd', 'e', ' ', 'c', 'a', 'r', 'a', 'c',
', 't', 'e', 'r', 'e']
>>>
```

# MANIPULAREA UNUI SIR DE CARACTERE

## METODA 2

```
>>> sir="sir de caractere"  
>>> lista=sir.split()  
>>> noul_sir=""  
>>> for elem in lista:  
    noul_sir+=elem+" "  
  
>>> print noul_sir  
sir de caractere  
>>>
```

# UTILIZAREA WHILE CU LISTE CA SI CONDITII

```
>>> scribe = ""
>>> lista = ["a", "b", "c"]
>>> while scribe not in lista:
    scribe = raw_input("Trebuie sa scrii a sau b sau c:\t")
```

```
Trebuie sa scrii a sau b sau c: d
```

```
Trebuie sa scrii a sau b sau c: qwr
```

```
Trebuie sa scrii a sau b sau c: a
```

```
>>> print scribe
```

```
a
```

# UTILIZAREA WHILE CU DICTIONARE CA SI CONDITII

```
>>> scribe = ""
>>> dic={"1":"a"}
>>> while scribe not in dic:
        scribe = raw_input("Trebuie sa scrii 1\t")
```

```
Trebuie sa scrii 1      1
>>> print scribe
1
```



# FUNCTII

Definirea unei funcții se poate face astfel:

```
def instructiuni():  
    #bloc de expresii
```

Cuvântul def este obligatoriu și indica definirea unei funcții; instructiuni este un cuvânt ce denumește noua funcție; acesta poate fi schimnat.

Nu uita să pui parantezele și doua puncte, acestea fiind vitale pentru definirea unei funcții.

În imediata apropiere a definirii numelui funcției trebuie să definim și blocul funcției, adică să-i oferim instrucțiunile de care avem nevoie. Blocul de expresii al funcției este separat prin indentare de restul codului.

Apelarea se face cu ajutorul numelui funcției urmat de paranteze:  
**instructiuni()**

# FUNCTII

```
>>>  
>>> def = "asd"  
SyntaxError: invalid syntax  
>>> def=12  
SyntaxError: invalid syntax  
>>>
```

def este un cuvant rezervat si nu putem sa-l utilizam pentru altceva

# FUNCTII

```
>>>  
>>> def nume_functie():  
    pass  
  
>>>
```

# FUNCTII

```
>>>  
>>> def nume_functie():  
    #bloc de expresii ce vor fi rulate la nevoie  
    pass  
  
>>>
```

# FUNCTII

```
>>> # exista cuvinte rezervate ca si nume de functii
>>> # deoarece sunt deja utilizate cu alt scop
>>>
>>> def del():

SyntaxError: invalid syntax
>>> def print():

SyntaxError: invalid syntax
>>> def in():

SyntaxError: invalid syntax
>>>
```



## FUNCTII

```
>>>
```

```
>>> def nume:
```

```
SyntaxError: invalid syntax
```

```
>>> def nume() 
```

```
SyntaxError: invalid syntax
```

```
>>>
```

# FUNCTII

Un docstring este un șir de caractere definit cu trei ghilimele ce are rolul de a informa ce rol are funcția în program sau ce returnează. Aceasta trebuie să fie prima linie din bloc, dar se poate răspândi peste mai multe linii. Funcțiile pot funcționa foarte bine fără docstring, dar documentarea poate ajuta colegii sau clientul ce cumpăra produsul.

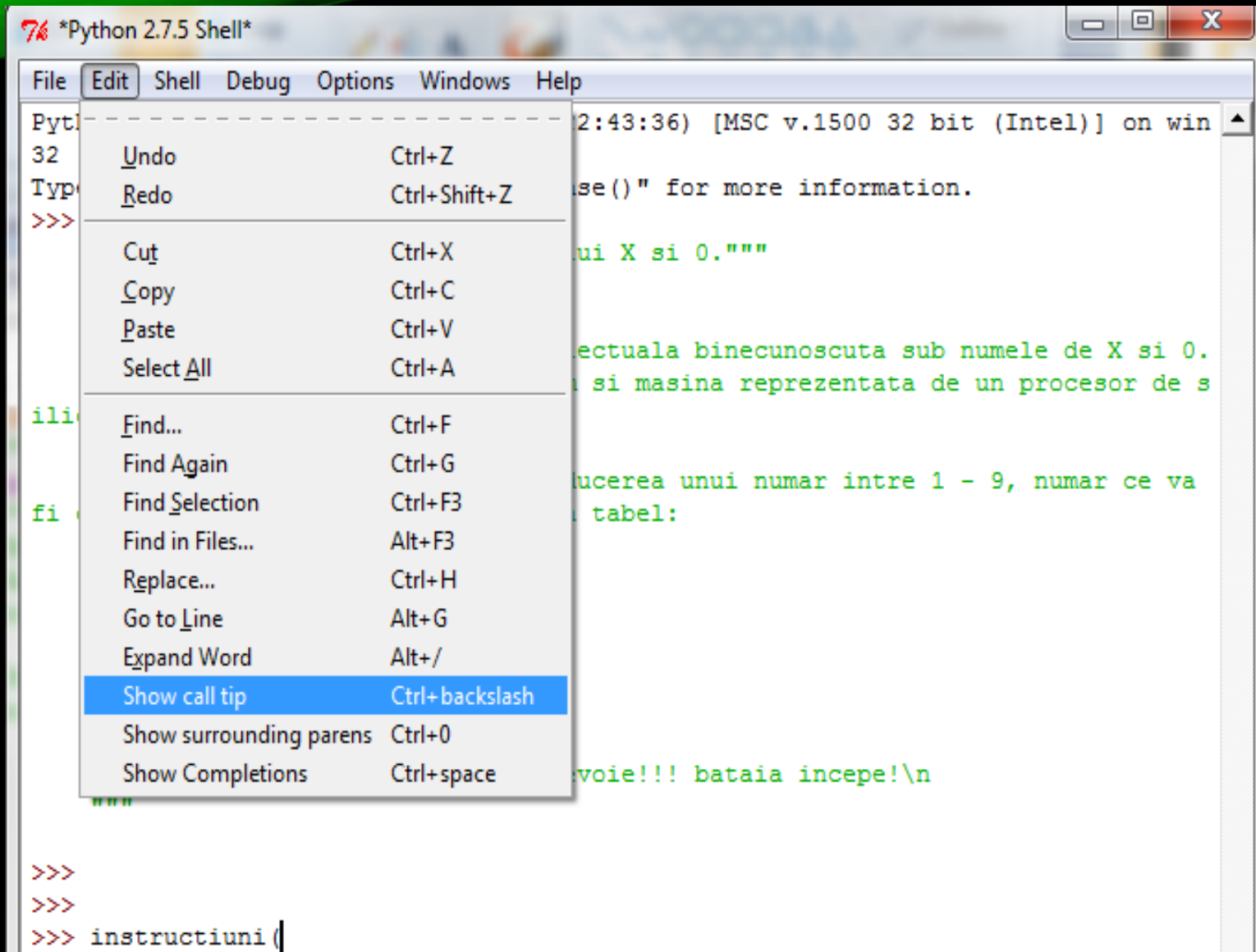
```
>>> def nume_functie():  
    """docstring"""  
    #bloc de expresii ce vor fi rulate la nevoie  
    pass  
  
>>> nume_functie.func_doc  
'docstring'  
>>>
```

# FUNCTII

Docstring-ul poate apărea ca popout în IDLE sau chiar poate fi apelat, utilizând `instructiuni.func_doc`

```
>>> instructiuni.func_doc  
_   
' Afiseaza instructiunile jocului X si 0.'
```

# FUNCTII



Pentru a face vizibila apariția acestui popul va trebui scrieți numele funcției urmat de o paranteza rotonda, apoi să apăsați succesiunea de taste CTRL+\ (backslash). O alta modalitate ar fi să înlocuiți succesiunea de taste cu apelarea din meniul Edit a opțiunii "Show call tip" așa cum se poate observa și în figura alăturata.

# FUNCTII

Este firesc ca funcțiile să se definească la începutul programului imediat sub secțiunea de declarare a variabilelor. Aceasta ordine ar trebui respectata deoarece modificarea programului de către un programator ce nu are cunoștințe despre cod devine greoaie în alte cazuri.

Prin scrierea și apelarea de funcții încercăm să aplicăm și conceptul de abstractizare. Abstractizarea ne lasă să vedem imaginea de ansamblu fără a ne face griji despre detalii. Abstractizarea este un lucru comun în viață de zi cu zi. Spre exemplu, într-un supermarket raioanele sunt numerotate pt. a fi mai rapid de a fi identificat de lucrătorii centrului comercial.



Rularea unei functii se poate realiza cu nume\_functie()

# FUNCTII

```
funcție X
1 # Functie meniu
2 # Demonstreaza utilizarea functiei
3 # Ion Studentul - 1/26/13
4
5 def instructiuni():
6     """ Afiseaza instructiunile jocului X si 0. """
7     print \
8     """
9     Bine ati venit la incercarea intelectuala binecunoscuta
10    de a gasi cele mai bune strategii pentru a castiga.
11    """
12     print /
```

# FUNCTII

Un alt lucru pe care o functie poate sa-l faca este sa aiba parametrii de intrare. Parametrii captează valoarea trimisa către functie si mai poarta numele de argument. O functie poate avea multipli parametri. Astfel parametrii param1 si param2 pot fi utilizati ca orice variabilă, afișând sau modificând valoarea acesteia. Astfel parametrii param1 si param2 pot fi utilizati ca orice variabilă, afișând sau modificând valoarea acesteia.

```
>>>
>>> def adunam(param1,param2):
    """In functie de acesti parametrii vom face altceva.
    Spre exemplificare vom crea o functie care aduna doua numere"""
    print param1+param2

>>> adunam(1,2)
3
>>> x=adunam(2,3)
5
```

# FUNCTII

Apelarea functiei cu mai multi parametrii sau mai putini parametrii returneaza eroare. In corpul erorii putem vedea ca adunam() primeste exact 2 parametrii, conditie neindeplinita.

```
>>> def adunam(param1,param2):  
    """In functie de acesti parametrii vom face altceva.  
    Spre exemplificare vom crea o functie care aduna doua numere"""  
    print param1+param2  
  
>>> adunam(1)  
  
Traceback (most recent call last):  
  File "<pyshell#240>", line 1, in <module>  
    adunam(1)  
TypeError: adunam() takes exactly 2 arguments (1 given)  
>>> adunam(1,2,3)  
  
Traceback (most recent call last):  
  File "<pyshell#241>", line 1, in <module>  
    adunam(1,2,3)  
TypeError: adunam() takes exactly 2 arguments (3 given)
```

# FUNCTII

Functia `adunam()` nu returneaza nimic. Numerele pe care noi le-am vazut la apelarea functiei se datoreaza sintaxei `print` din blocul de sintaxe al functiei `adunam`. Cu ajutorul cuvintului `return` putem returna calculul matematic dintre cele doua numere introduse ca parametrii, calcul ce va fi stocat de variabila `x`; deci putem sa realizam `x+2`, rezultat ce ne da 7.

```
>>> def adunam(param1,param2):  
    """In functie de acesti parametrii vom face altceva.  
    Spre exemplificare vom crea o functie care aduna doua numere"""  
    print param1+param2  
    #pentru a returna ceva trebuie sa utilizam cuvantul cheie return  
    return param1+param2  
  
>>> x=adunam(2,3)  
5  
>>> x  
5  
>>> x+2  
7
```

# FUNCTII

## Exemplu

funcție2

```
1 # Functie input si output
2 # Demonstreaza parametrii si valorile returnate
3 # Ion Studentul - 1/26/13
4
5 Nrlist={1:True,2:False,3:False,4:False,5:True,6:False,7:False}
6 raspDaNu=None
7
8
9 def Afiseaza(mesaj):
10     """ Afiseaza un mesaj dat. """
11     print "Acesta este mesajul: " + str(mesaj).upper() + "!"
12
13
14 def Afiseaza(mesaj):
15     """ Afiseaza un mesaj dat. """
16     print "Acesta este mesajul: " + str(mesaj).upper() + "!"
```



Prin apelarea acestei funcții vedem ca ordinea parametrilor contează și trebuie să apelăm tot mereu în ordinea în care au fost definiți acești parametri. În caz contrar, observăm ca în loc de nume regăsim vârsta și în loc de vârsta regăsim numele. Imaginea de mai jos utilizează parametri normali.

```
>>> # parametrii pozitionali
>>> def ziNasterel(name, age):
    print "Salut,", name, "!", "Am auzit ca ai", age, "ani!.\n"

>>> ziNasterel("Catalin",20)
Salut, Catalin ! Am auzit ca ai 20 ani!.

>>> ziNasterel(20,"Catalin")
Salut, 20 ! Am auzit ca ai Catalin ani!.

>>> |
```

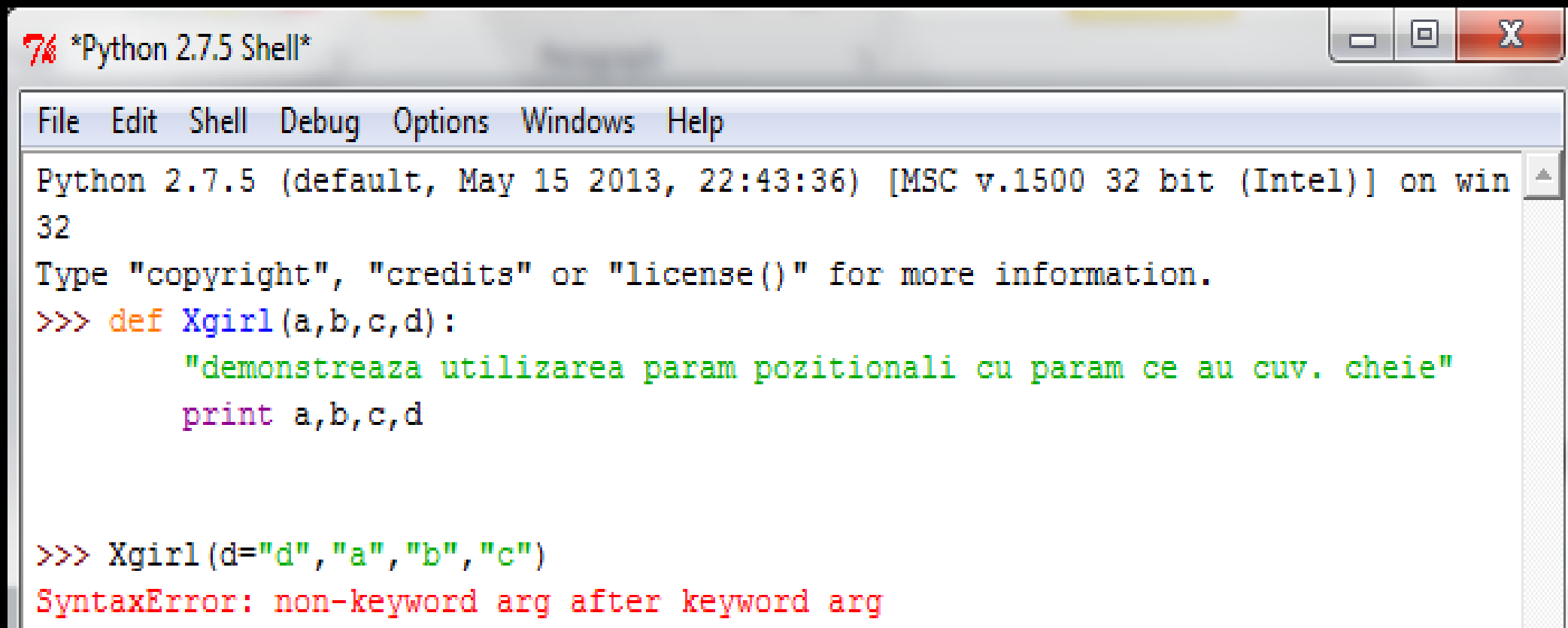
# FUNCTII

În cazul în care nu reținem ordinea parametrilor pe care i-am utilizat putem să folosim parametrii ce au cuvinte cheie. Prin aceștia stabilim valorile parametrilor fără a ține cont de ordine. Aceasta utilizare folosește parametrii poziționali; prin urmare poziția parametrilor în cadrul apelării este importantă.

```
>>> def ziNasterel(name, age):  
    print "Salut,", name, "!", "Am auzit ca ai", age, "ani!.\n"  
  
>>> ziNasterel(name = "Catalin", age = 20)  
Salut, Catalin ! Am auzit ca ai 20 ani!.  
  
>>> ziNasterel(age = 20, name = "Catalin")  
Salut, Catalin ! Am auzit ca ai 20 ani!.  
  
>>>
```

## FUNCTII

Nu poti combina parametrii pozitionali cu parametrii normali



```
*Python 2.7.5 Shell*  
File Edit Shell Debug Options Windows Help  
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win  
32  
Type "copyright", "credits" or "license()" for more information.  
>>> def Xgirl(a,b,c,d):  
    "demonstreaza utilizarea param pozitionali cu param ce au cuv. cheie"  
    print a,b,c,d  
  
>>> Xgirl(d="d","a","b","c")  
SyntaxError: non-keyword arg after keyword arg
```

O funcție poate stabili în definirea funcției parametrii standard (default). Prin acești parametrii default putem să omitem la apelarea funcției anumiți parametrii sau chiar pe toți, valorile acestor parametrii omiși vor fi valorile default definite în funcție. Așa cum e normal putem folosi fie parametrii poziționali, fie parametrii ce conțin cuvinte cheie la apelarea funcției.

```
>>> # parametrii cu valori standard
>>> def ziNastere2(name = "Ion Studentul", age = 20):
    print "Salut,", name, "!", "Am auzit ca ai", age, "ani!.\n"

>>> ziNastere2()
Salut, Ion Studentul ! Am auzit ca ai 20 ani!.

>>> ziNastere2(name = "Maria")
Salut, Maria ! Am auzit ca ai 20 ani!.

>>> ziNastere2(age = 18)
Salut, Ion Studentul ! Am auzit ca ai 18 ani!.
```

# FUNCTII

```
>>>
>>> def ziNastere2(name = "Ion Studentul", age = 20):
    print "Salut,", name, "!", "Am auzit ca ai", age, "ani!.\n"

>>> ziNastere2(name = "Maria", age = 18)
Salut, Maria ! Am auzit ca ai 18 ani!.

>>> ziNastere2("Maria", 18)
Salut, Maria ! Am auzit ca ai 18 ani!.

>>> ziNastere2(18, "Maria")
Salut, 18 ! Am auzit ca ai Maria ani!.

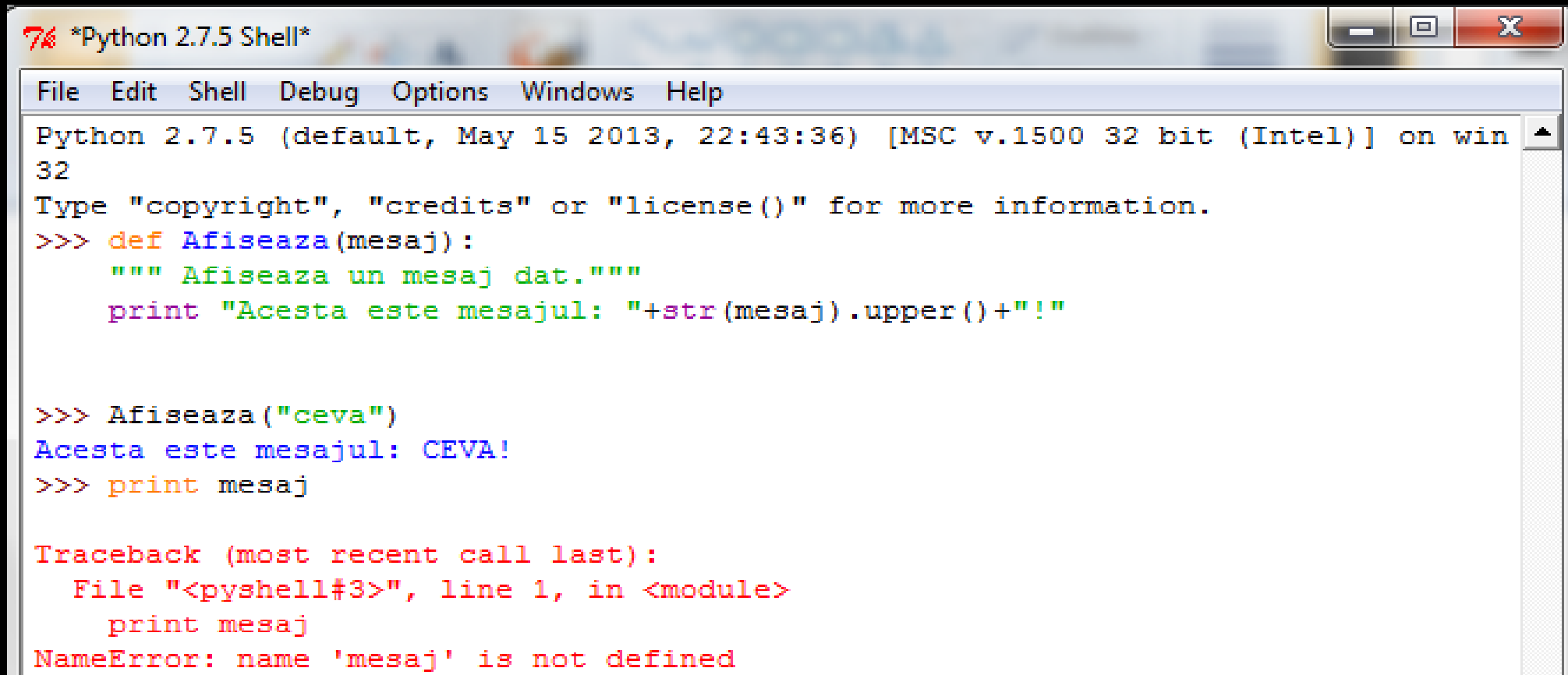
>>>
```

## FUNCTII

funcție3

```
1 # Functie parametrii default si keyword args
2 # Demonstreaza parametrii standard si argumente cheie
3 # Ion Studentul -- 1/26/13
4
5 # parametrii pozitionali
6 def ziNastere1(name, age):
7     print "Salut, ", name, "!", "Am auzit ca ai", age, "ani!"
8
9     print "Salut, ", name, "!", "Am auzit ca ai", age, "ani!"
10 def ziNastere2(name, age):
```

Un parametru nu este disponibil si in exteriorul unei functii. Prin apelarea funcției Afiseaza() aceasta va afisa mesajul dat ca parametru de intrare. Totusi variabila mesaj nu exista dupa terminarea rularii functiei.



```
*Python 2.7.5 Shell*
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> def Afiseaza(mesaj):
    """ Afiseaza un mesaj dat."""
    print "Acesta este mesajul: "+str(mesaj).upper()+"!"

>>> Afiseaza("ceva")
Acesta este mesajul: CEVA!
>>> print mesaj

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print mesaj
NameError: name 'mesaj' is not defined
```



Imposibilitatea unei variabile locale să fie accesata din exterior se numește încapsulare și are rolul de tine codul independent.

Dacă avem o funcție care face sute de calcule matematice pe baza a 3 parametrii și are nevoie de zeci de variabile locale, dar returnează un singur număr ca răspuns putem izola acele variabile locale, codul rulând mult mai rapid deoarece după terminarea funcției acestea sunt eliminate.

Deci dacă avem o sută de funcții cu o sută de variabile locale fiecare programul nu va încarcă decât cate 100 de variabile pe rând neafectând rapiditatea programului.

Încapsularea este un principiu al abstractizării, abstractizare care te salvează de a înțelege toate detaliile, privind în ansambluri ușor de schematizat.

# FUNCTII

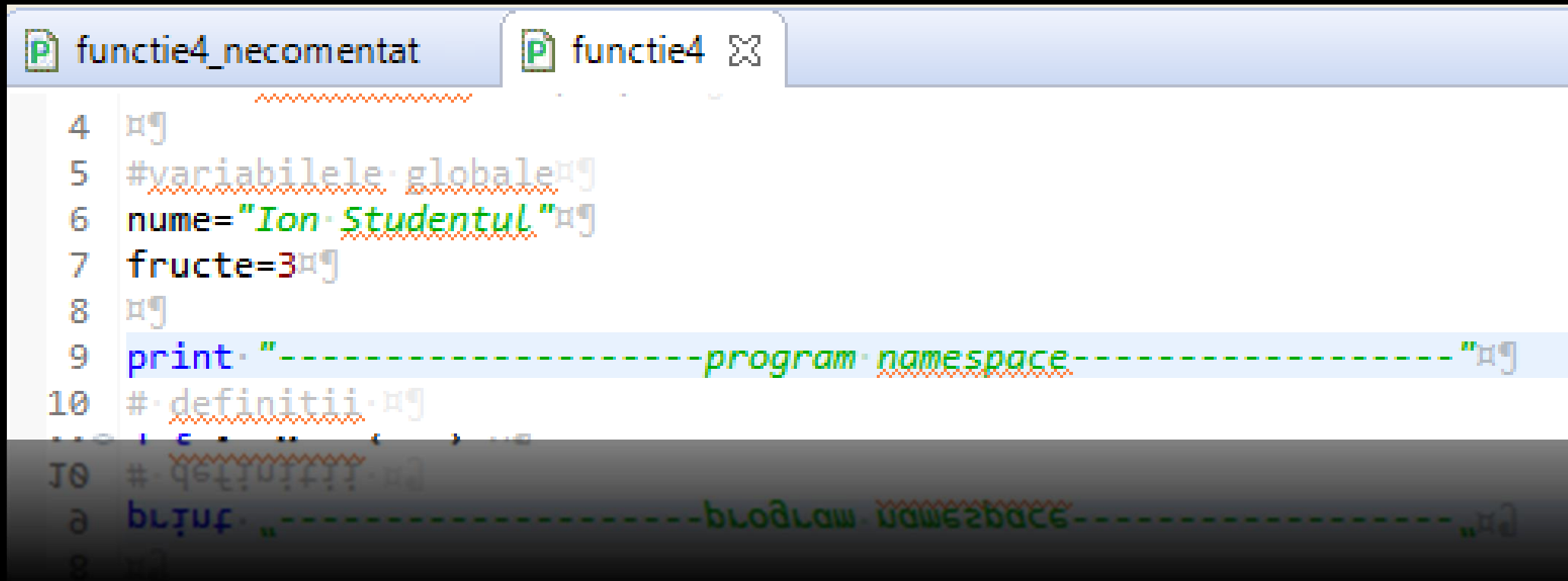
- Reducerea codului duplicat
- Descompunerea problemelor complexe în bucăți mai simple
- Îmbunătățirea clarității codului
- Reutilizarea de cod
- Ascunderea de informații(incapsulare)

# FUNCTII

Se definește *namespace* (sau *scope*) anumite zone diferite ale programului care sunt separate una de alta. Aceste zone spre exemplu sunt doua funcții distincte, și este motivul pt. care variabilele locale nu pot fi modificate direct(datorita zonelor namespace diferite) .

Programul în sine are un namespace. O funcție poate vedea toate variabilele din namespace-ul programului (denumite variabile globale), dar și toate variabilele definite în namespace-ul propriu (denumite variabile locale). De asemenea poate modifica doar variabilele definite în namespace-ul propriu.

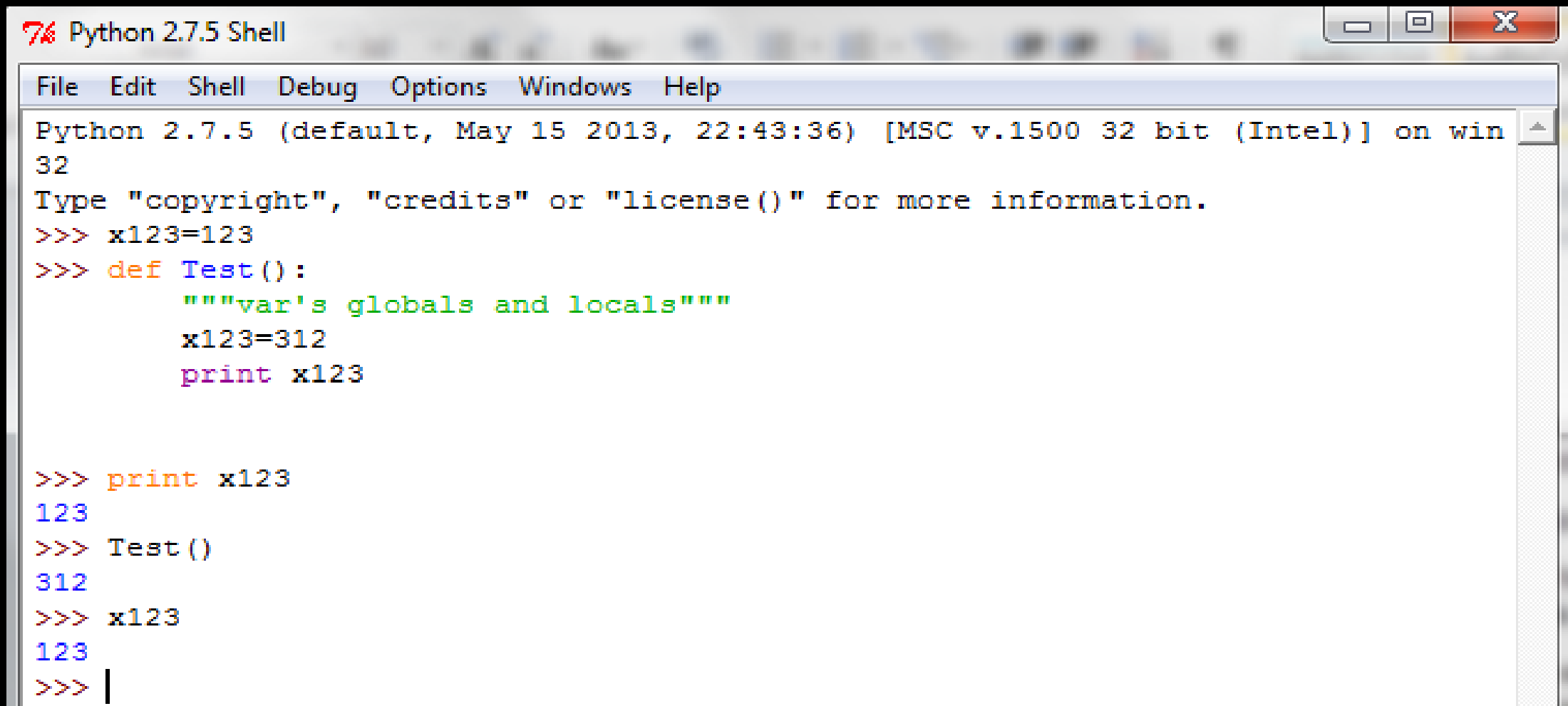
## FUNCTII



The screenshot shows a code editor with two tabs: 'functie4\_necomentat' and 'functie4'. The 'functie4' tab is active, displaying the following Python code:

```
4  #  
5  #variabilele globale  
6  nume="Ion Studentul"  
7  fructe=3  
8    
9  print("-----program namespace-----")  
10 #definitii  
11   
12 #definitii  
13 def bluc("-----bluc namespace-----")  
14   
15
```

## FUNCTII



```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> x123=123
>>> def Test():
    """var's globals and locals"""
    x123=312
    print x123

>>> print x123
123
>>> Test()
312
>>> x123
123
>>> |
```

## FUNCTII

```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> var1=1
>>> var2=2
>>> var3=3
>>> def Test ():
    """modifica variabile globale"""
    global var1,var3
    print "var1:",var1,"\nvar3:",var3
    var1=104
    var3=312
    print "var1:",var1,"\nvar3:",var3

>>> print var1,var2,var3
1 2 3
>>> Test()
var1: 1
var3: 3
var1: 104
var3: 312
>>> print var1,var2,var3
104 2 312
>>>
```

```

>>> sss=7
>>> #am definit variabila sss
>>> def test():
    """variabila globala"""
    global ss #am omis un s
    pass

>>> print ss

Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print ss
NameError: name 'ss' is not defined
>>> test()
>>> print ss

Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print ss
NameError: name 'ss' is not defined
>>> def test():
    """variabila globala"""
    global ss #am omis un s
    ss=123

>>> print ss

Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    print ss
NameError: name 'ss' is not defined
>>> test()
>>> print ss
123

```



Numar diferit de parametrii:

```
>>> # numar diferit de parametrii
>>> def Aduna(a,b,*c):
    """cel putin 2 parametrii"""
    x = a+b
    if(c):
        print c
        for e in c:
            x +=e
    return x

>>> Aduna(1,2)
3
>>> Aduna(1,2,3,5)
(3, 5)
11
```

## FUNCTII

## Object Oriented Programming (OOP)

De ce am folosi OOP?

- structurare - fiecare din intrări are propriile caracteristici
- încapsulare - izolează informația pentru a putea rula eficient
- scalabilitate - poate face fata cu ușurința la schimbările codului-  
exemplu: adăugarea unui tip special inexistent pana în prezent

## Object Oriented Programming (OOP)

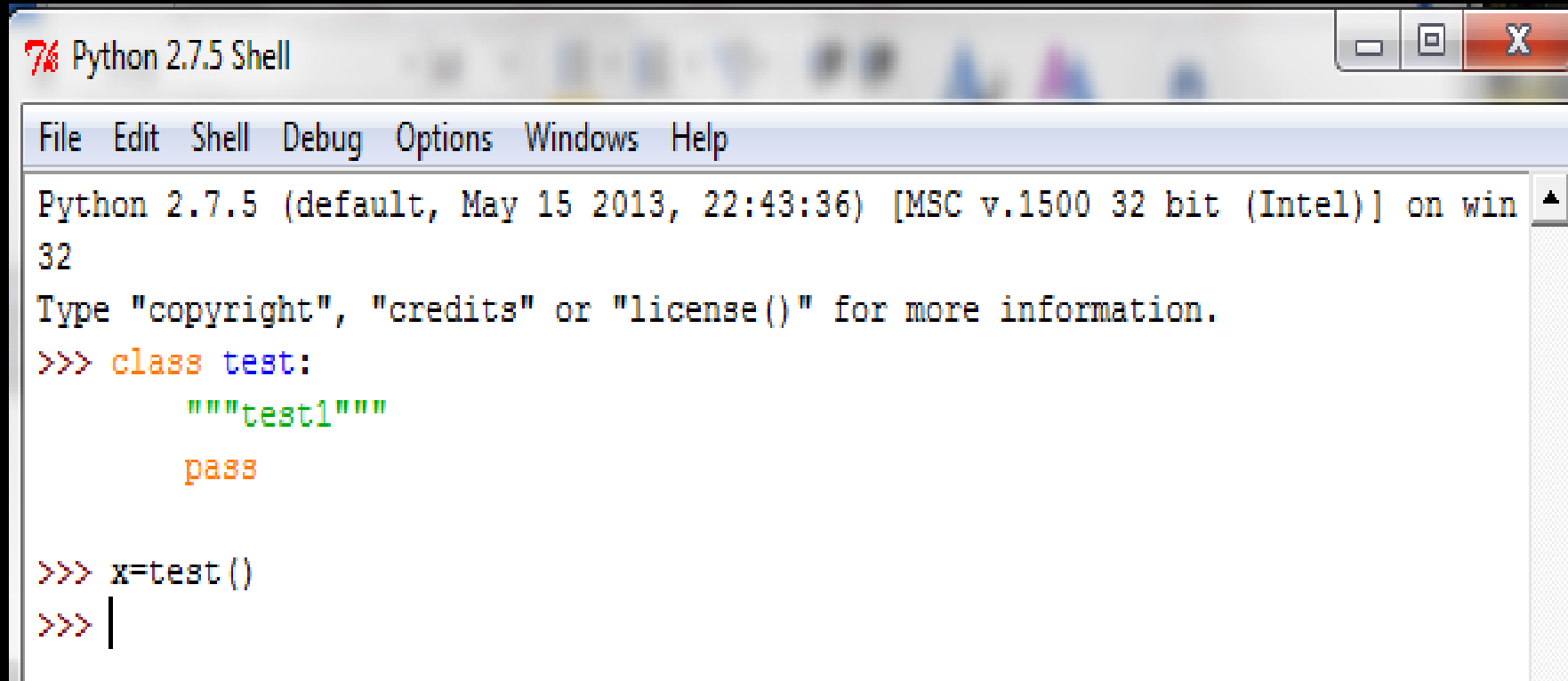
De ce am folosi OOP?

- Un Obiect este rularea( o instanta) unei clase. Fiecare obiect are propriul set de caractere sitici.
- Putem spune ca un set de attribute ale unui obiect formează, de asemenea, un namespace. Deci, încapsularea în cazul claselor ajuta ca programul să devina scalabil, nu contează mărimea lui.
- Fiecare obiect poate avea un set proprietati distincte numite attribute si acestea reprezinta reprezintă orice numele obiectului urmat de un punct, apoi numele atributului- de exemplu , în expresia obiectulMeu.atributulNostru, atributulNostru este un atribut al obiectului obiectulMeu.

## Object Oriented Programming (OOP)

```
class NumeClasa(object):  
    <sintaxa-1>  
    .  
    .  
    .  
    < sintaxa-N>
```

## Object Oriented Programming (OOP)



```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> class test:
    """test1"""
    pass

>>> x=test()
>>> |
```

```
>>>  
>>> class Nume(object):  
    """Prima mea clasa"""  
    def Metoda(self):  
        print "ceva"  
  
>>> obj1=Nume()  
>>> obj1.Metoda()  
ceva  
>>> |
```

Fiecare metoda ar trebui să fie însoțită de o descriere

# INFOACADEMY.NET

## CLASA IN PYTHON

```
*Python 2.7.5 Shell*
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> class Test(object):
    """Test metoda fara self"""
    def metodal():
        pass

>>> x=Test()
>>> x.metodal()

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    x.metodal()
TypeError: metodal() takes no arguments (1 given)
>>> class Test(object):
    """Test metoda fara self"""
    def metodal(x):
        pass

|
>>> x=Test()
>>> x.metodal()
>>>
>>>
```



# INFOACADEMY.NET

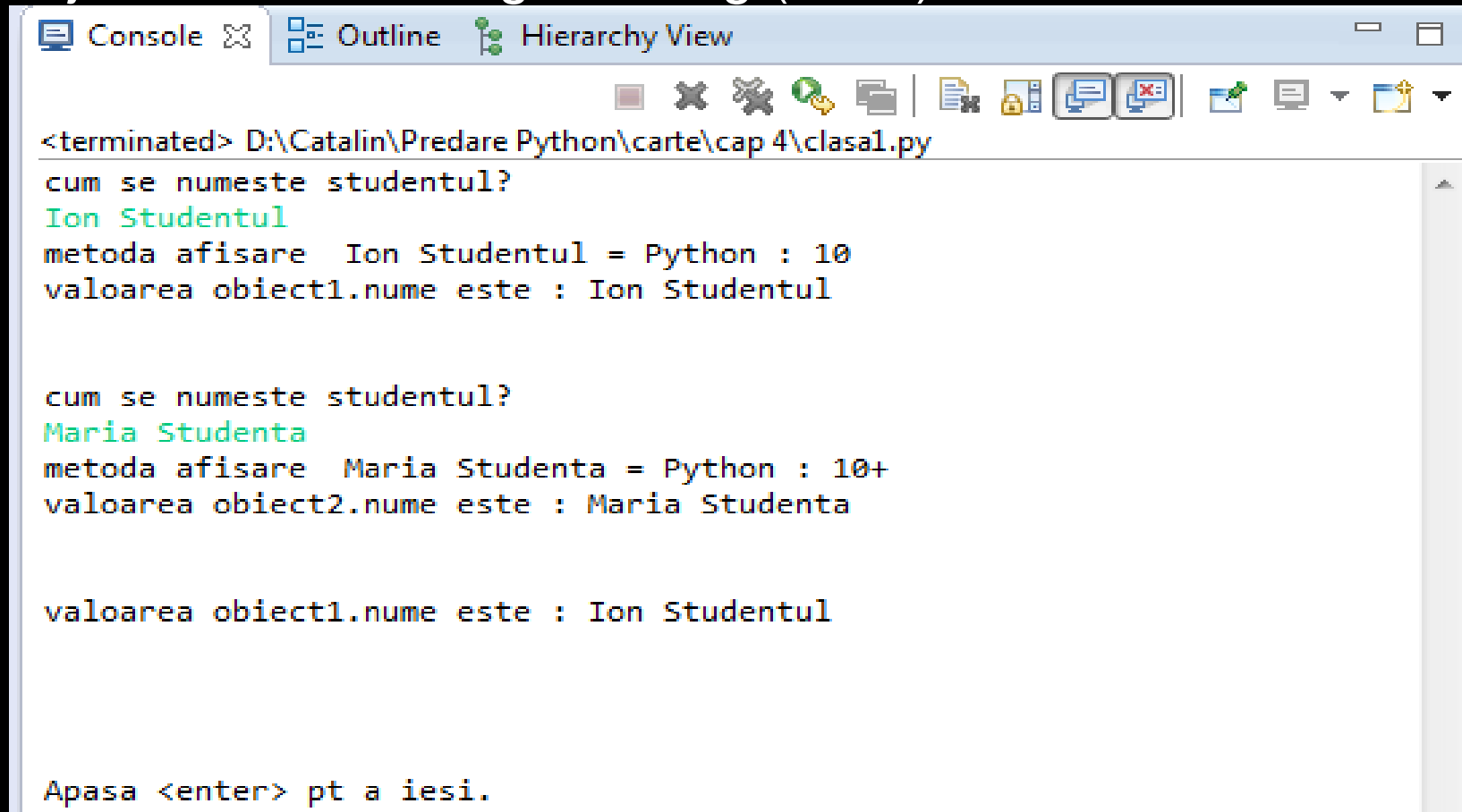
## CLASA IN PYTHON

```
>>> class Z(object):  
    """test"""  
    def Test(self):  
        x=1  
  
>>> a=Z()  
>>> a.x  
  
Traceback (most recent call last):  
  File "<pyshell#117>", line 1, in <module>  
    a.x  
AttributeError: 'Z' object has no attribute 'x'  
>>> a.Test()  
>>> a.x  
  
Traceback (most recent call last):  
  File "<pyshell#119>", line 1, in <module>  
    a.x  
AttributeError: 'Z' object has no attribute 'x'  
>>> |
```

## Object Oriented Programming (OOP)

```
clasa1
1 # Program clasa1
2 # Demonstreaza utilizarea clasei
3 # Ion Studentul -- 1/26/13
4
5 class NumeClasa(object):
6     """clasa mea"""
7     def metodaNume(self):
8         """nume student"""
9         self.nume = raw_input("Cum se numeste studentul? ")
10        self.universitate = raw_input("Care este universitatea? ")
11        """nume student"""
12        """universitate"""
13    def metodaNume(self):
14        """nume student"""
15        """universitate"""
```

## Object Oriented Programming (OOP)



The screenshot shows a Python IDE window with three tabs: 'Console', 'Outline', and 'Hierarchy View'. The 'Console' tab is active, displaying the output of a Python script. The script defines a class 'Student' with a constructor and a class method 'afisare'. It then creates two objects, 'Ion Studentul' and 'Maria Studenta', and prints their names. The output shows the names of the objects and the result of the class method.

```
<terminated> D:\Catalin\Predare Python\carte\cap 4\clasa1.py
cum se numeste studentul?
Ion Studentul
metoda afisare Ion Studentul = Python : 10
valoarea obiect1.numa este : Ion Studentul

cum se numeste studentul?
Maria Studenta
metoda afisare Maria Studenta = Python : 10+
valoarea obiect2.numa este : Maria Studenta

valoarea obiect1.numa este : Ion Studentul

Apasa <enter> pt a iesi.
```

File Edit Shell Debug Options Windows Help

Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win32

Type "copyright", "credits" or "license()" for more information.

```
>>> class TestMe(object):  
    """Argument sters sau modificat"""  
    def met1(self, valArgument):  
        """cream un argument"""  
        self.valArgument=3+valArgument
```

```
>>> obj1=TestMe()  
>>> obj1.met1(5)  
>>> print obj1.valArgument  
8  
>>> obj1.valArgument=9  
>>> print obj1.valArgument  
9  
>>> del obj1.valArgument  
>>> print obj1.valArgument
```

Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

print obj1.valArgument

AttributeError: 'TestMe' object has no attribute 'valArgument'

```
>>> obj1.argNou = 12  
>>> print obj1.argNou  
12
```

# INFOACADEMY.NET

## CLASA IN PYTHON

```
>>> class Z(object):  
    pass
```

```
>>> c=Z()
```

```
>>> d=Z()
```

```
>>> c.a="N-am dar am"
```

```
>>> d.a
```

```
Traceback (most recent call last):
```

```
  File "<pysHELL#85>", line 1, in <module>
```

```
    d.a
```

```
AttributeError: 'Z' object has no attribute 'a'
```

```
>>> c.a
```

```
'N-am dar am'
```

```
>>> |
```

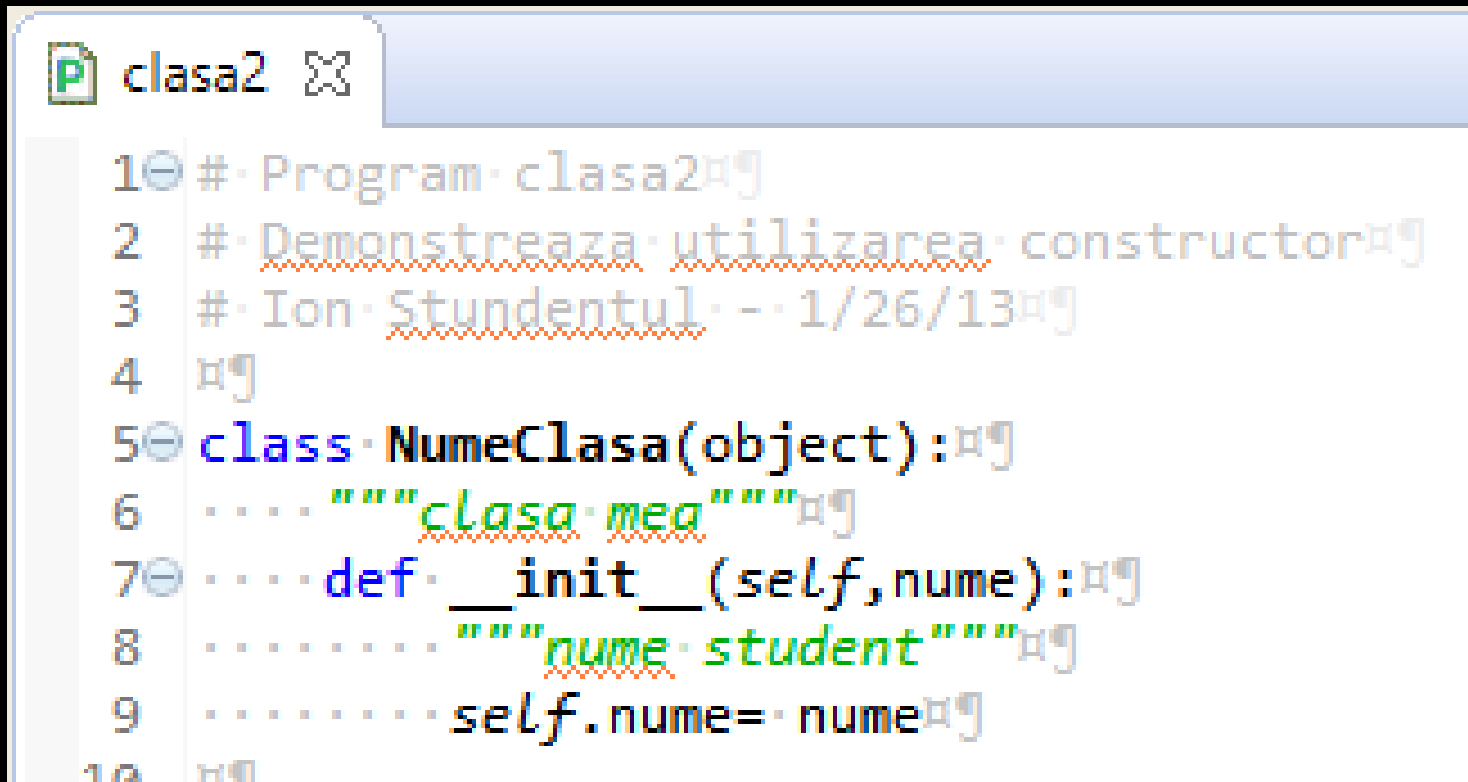
## Clasa cu parametrii de intrare

Metode speciale:

- Metoda `__init__` este o metoda speciala care este rulata de fiecare data când un obiect este creat.
- In programul de mai sus folosim metoda `__str__`. Aceasta permite prin return să afișăm obiecte.

## CLASA IN PYTHON

## Clasa cu parametrii de intrare



```
1 # Program: clasa2.py
2 # Demonstreaza utilizarea constructorului
3 # Ion Studentul - 1/26/13
4
5 class NumeClasa(object):
6     """clasa mea"""
7     def __init__(self, nume):
8         """nume student"""
9         self.nume = nume
10
```



## Clasa cu parametrii de intrare

```
clasa3  ✖
1 # Program: clasa3.py
2 # Demonstreaza utilizarea metodei speciale __str__
3 # Ion Studentul - 1/26/13
4
5 class NumeClasa(object):
6     """clasa mea"""
7     def __init__(self, nume):
8         """nume student"""
9         self.nume = nume
```

# CUNOSTINTE EXTRA LEGATE DE LISTE SI DICTIONARE

Creati un program ce imparte sirul de caractere “Avem ce face in Python” intr-o lista. Foloseste lista pentru a prelucra caracterele din interior astfel:

- Inlocuieste e cu i
- Inlocuieste spatiu cu \_ (underline)

Uneste apoi lista intr-un sir de caractere ce trebuie afisata

# TEMA SEDINTA ANTERIOARA

Creati un dictionar cu elementele:

1-"Unu"

2-"Doi Doi"

3-"Trei Trei Trei"

Printati fiecare pereche de tip cheie valoare din dictionar printr-un for.

Solicitati userului sa introduca text de la tastatura. Acest text devine valoarea unei noi intrari in dictionar ce va avea cheia 4.

Printati fiecare pereche de tip cheie valoare din dictionar printr-un for.

Stergeti intrarea ce are cheia cu numarul 2. Printati fiecare pereche de tip cheie valoare din dictionar printr-un for.

# FUNCTII

## Exercitiu

- Creati o functie care sa calculeze si sa returneze operatia matematica de mai jos exclusiv pentru 3 numere. Daca nu sunt numere returnati sirul de caractere "Dati va rog 3 numere":

$$[a(a+3)/b]*c$$

Rulati functia pentru 1,2,3 si afisati rezultatul

Rulati functia pentru "1","2","3" si afisati rezultatul

# FUNCTII

## Exercitiu

- Creati o functie care sa inlocuiasca caracterele de tip spatiu cu underline si sa inlocuiasca sirul de caractere "e" cu sirul de caractere "i" dintr-un sir de caractere primit ca parametru. Verificati ca este sir de caractere inainte de a face aceasta operatie. Returnati sirul de caractere modificat sau returnati sirul urmator daca nu a fost primit un sir de caractere:  
"Dati va rog un sir de caractere"

Rulati codul pentru "Merele, perele si pestele nu au E-uri"

Creati o clasa ce va reprezenta un catalog:

- La initializare trebuie sa oferim doi parametrii de intrare nume si prenume
- Avem o metoda care afiseaza absente implementat cu `__str__`
- Avem o metoda care incrementeaza cu 1 nr. de absente
- Avem o metoda care sterge un nr. (exclusiv un numar - verifica) de absente dat (pentru cazurile in care avem o scutire medical) fara a deveni negativ
- Creati 1 student numit Ion Roata
- Modificati argumentul absente sa fie incrementat de 3 ori prin metoda creata
- Stergeti doua absente prin metoda specificata
- Creati al doilea student numit George Cerc
- Modificati argumentul absente sa fie incrementat de 4 ori prin metoda creata
- Stergeti doua absente prin metoda specificata
- Afisati absentele fiecarui student

VA MULTUMESC PENTRU PARTICIPARE

**La revedere!**