



Programare interactivă

Lect.univ. Victoria ALEXEI



Programare interactivă în Python

Organizarea cursului:

- ➡ Curs – 30 ore
- ➡ Lucrări de laborator – 30 ore, 6 lucrări
- ➡ Evaluarea 1 – la a 7 săptămână de studii
- ➡ Activitate curentă, Lucru individual
- ➡ Examen final

Bibliografie

<http://www.python.org/>

1.	Practical Programming (in Python) Jeffrey Elkner, Allen B. Downey, Chris Meyers Brendan McCane, Iain Hewson, Nick Meek February 9, 2015
2.	Practical Programming An Introduction to Computer Science Using Python Jennifer Campbell, Paul Gries, Jason Montojo, Greg Wilson 2009
3.	Automate the Boring Stuff with Python Practical Programming for Total Beginners
4.	Cărți în format electronic pentru limbajul Python http://pythonbooks.revolunet.com/
5.	Mihail Radu Solcan O brumă de Python, martie, 2010 http://www.ub-filosofie.ro/~solcan/cs/np/py/bp.html
6.	http://www.openbookproject.net/thinkcs/python/english2e/#

Instalarea Python

<https://www.python.org/downloads/>

➤ **Versiunea python 3.8.5**



Extensii fișiere obișnuite

.py, .pyw, .pyc, .pyo, .pyd

Paradigmă

multi-paradigmă: *object-oriented*, imperativă, funcțională, procedurală

Apărut în

1991

Dezvoltat de

Guido van Rossum

Dezvoltator actual

Python Software Foundation

Exemplu de program

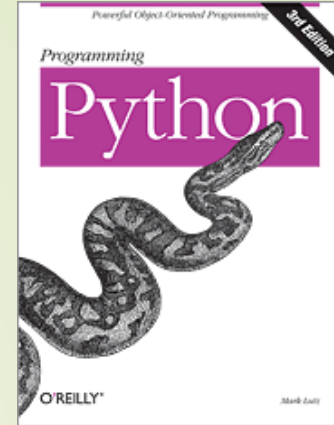
```
name = input('Enter file:')
handle = open(name, 'r')
text = handle.read()
words = text.split()

counts = dict()
for word in words:
    counts[word] = counts.get(word,0) + 1
bigcount = None
bigword = None

for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count
print (bigword, bigcount)
```

Enter file:cuvinte.txt
de
13

- **Python** este un limbaj de programare dinamic multi-paradigmă, creat în **1989** de programatorul olandez **Guido van Rossum**. Van Rossum este și în ziua de astăzi un lider al comunității de dezvoltatori de software care lucrează la perfecționarea limbajul Python și implementarea de bază a acestuia, CPython, scrisă în C.
- **Python** este un limbaj multifuncțional folosit de exemplu de către companii ca **Google** sau **Yahoo!** pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment programate parțial sau în întregime în Python.





Este omniprezent - oamenii folosesc zilnic **numeroase aplicații scrise în Python**, indiferent dacă își dau seama sau nu.

- Există miliarde de linii de cod scrise în Python, ceea ce înseamnă oportunități aproape nelimitate de **reutilizare a codului și de învățare** din exemple bine elaborate.
- Este **ușor de învățat** - timpul necesar pentru a învăța Python este mai scurt decât pentru multe alte limbaje.
- Este **ușor de utilizat** pentru scrierea de aplicații noi - este adesea posibil să scrieți cod mai repede atunci când utilizați Python.
- Este **ușor de obținut, instalat și implementat** - Python este **gratuit, deschis și multiplatform**.
- Există o **comunitate Python mare și foarte activă**.
- **Va fi distractiv!**



- **3D CAD/CAM**
- **Audio/Video Applications**
- **Console Applications**
- **Enterprise Applications**
- **File Formats**
- **Image Applications**
- **Internet Applications**
- **Mobile Applications**
- **Office Applications**
- **Personal Information Managers**
- **Science and Education Applications**
- **Software Development**
- **System Administration Applications**
- **X-Window Manager**
- **Unclassified**

Source: <https://wiki.python.org/moin/PythonProjects>

3D CAD/CAM

Audio/Video Applications



- **Console Applications**
 - **Enterprise Applications**
 - **File Formats**
 - **Image Applications**
 - **Internet Applications**
 - **Mobile Applications**
 - **Office Applications**
 - **Personal Information Managers**
 - **Science and Education Applications**
 - **Software Development**
 - **System Administration Applications**
 - **X-Window Manager**
 - **Unclassified**
- **Internet Applications** (BitTorrent, Jogger Publishing Assistant, TheCircle, TwistedMatrix)
 - **3D CAD/CAM** (FreeCAD, Fandango, Blender, Vintech RCAM)
 - **Enterprise Applications** (Odoo, Tryton, Picalo, LinOTP 2, RESTx)
 - **Image Applications** (Gnofract 4D, Gogh, imgSeek, MayaVi, VPython)
 - **Mobile Applications** (Aarlogic C05/3, AppBackup, Pyrout)
 - **Office Applications** (calibre, faces, Notalon, pyspread)
 - **Personal Information Managers** (BitPim, Narval, Prioritise, Task Coach, WikidPad)

Source: <https://wiki.python.org/moin/PythonProjects>

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32

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

```
>>> x = 1
```

```
>>> print(x)
```

1

```
>>> x = x + 1
```

```
>>> print(x)
```

2

```
>>> exit()
```

Variabile, expresii și instrucțiuni

- O **variabilă** este un loc în memorie unde un programator poate stoca date și mai târziu prelua datele folosind "**numele**" **variabilei**
- Programatorii **aleg numele variabilelor**
- Aveți posibilitatea **să modificați conținutul unei variabile într-o declarație ulterioară**

x = 16.2

y = 34

x=20

Reguli de scriere a variabilelor

- Trebuie **să înceapă cu o literă** sau caracterul **underscore** _
- Poate **să conțină doar litere și numere** și caracterul **underscore**
- Case sensitive
- De ex:
 - **Bune:** spam eggs spam23 _speed
 - **Rele:** 23spam #sign var.12
 - **Diferite:** spam Spam SPAM

Exemple

```
>>> 56tyr = 34
SyntaxError: invalid syntax
>>> _tyr = 34
>>> tip = 23
>>> mai$ = 2
SyntaxError: invalid syntax
>>> class = 45
SyntaxError: invalid syntax
>>> |
```

Cuvinte rezervate

- ➔ Nu puteți utiliza cuvinte rezervate ca nume de variabile

<code>and</code>	<code>del</code>	<code>from</code>	<code>not</code>	<code>while</code>
<code>as</code>	<code>elif</code>	<code>global</code>	<code>or</code>	<code>with</code>
<code>assert</code>	<code>else</code>	<code>if</code>	<code>pass</code>	<code>yield</code>
<code>break</code>	<code>except</code>	<code>import</code>	<code>print</code>	
<code>class</code>	<code>exec</code>	<code>in</code>	<code>raise</code>	
<code>continue</code>	<code>finally</code>	<code>is</code>	<code>return</code>	
<code>def</code>	<code>for</code>	<code>lambda</code>	<code>try</code>	

Instrucțiuni

Instrucțiuni de atribuire

- Vom atribui o valoare unei variabile folosind instrucțiunea de atribuire (=)
- O declarație de atribuire constă dintr-o expresie de pe partea dreaptă și o variabilă pentru a stoca rezultatul

$x = 3.9 * x * (1 - x)$

Atribuire multiplă

Variabilele a și b iau valorile 2 și respectiv 4. Evaluarea expresiilor din partea dreaptă a unei atribuiri se face înainte de orice atribuire.

Evaluarea se face de la stânga la dreapta.

```
>>> a, b = 2, 4
>>> a
2
>>> b
4
```

Atribuire multiplă

```
>>> a, b, c, d = 3, 4, 5, 7
>>> a
3
>>> b
4
>>> c
5
>>> d
7
```

Atribuire multiplă

```
>>> a, b, c, d = 3, 4, 5, 7
```

```
>>> a
```

```
3
```

```
>>> b
```

```
4
```

```
>>> c
```

```
5
```

```
>>> d
```

```
7
```

Interschimbarea valorilor variabilor

```
>>> a, c = c, a
```

```
>>> a
```

```
5
```

```
>>> c
```

```
3
```

Atribuire multiplă

```
>>> a, b, c, d = 12.8, 34, "Salut", [2, 3.5, 4, 1.]  
>>> a  
12.8  
>>> b  
34  
>>> c  
'Salut'  
>>> d  
[2, 3.5, 4, 1.0]  
>>> |
```

Expresiile numerice

```
>>> x = 4
>>> xx = x+6
>>> print (xx)
10
>>> y = 230* 30
>>> print (y)
6900
>>> z= y / 1000
>>> z
6.9
>>> print (11**11)
285311670611
>>> print (4**3)
64
>>> g=30
>>> d = g % 5
>>> d
0
>>> g = 23 % 5
>>> g
3
>>> |
```

Operator	Operația
+	Adunare
-	Scădere
*	Înmulțire
/	Împărțire
**	Ridicarea la putere
%	Restul împărțirii

Expresiile numerice

Operatorul //

```
>>>  
>>> 234/6.3  
37.142857142857146  
>>> 234//6.3  
37.0  
>>> |
```

Operator	Operația
+	Adunare
-	Scădere
*	Înmulțire
/	Împărțire
**	Ridicarea la putere
%	Restul împărțirii


Ordinea expresiilor matematice

Atunci când mai mult de un operator apare într-o expresie, **ordinea de evaluare depinde de normele de prioritate.**

Python urmează **aceleași reguli de prioritate pentru operatorii matematici ca în matematică.**

Ordinea operațiilor:

- **1. Parantezele** au cea mai mare și poate fi de prioritate utilizate pentru a forța o expresie să fie evaluată în ordinea dorită. Din moment ce expresiile din paranteze sunt evaluate mai întâi, $2 * (3-1)$ rezultatul este 4, și $(1 + 1) ** (5-2)$ este 8. Puteți utiliza, de asemenea paranteze pentru a face o expresie mai ușor de citit, la fel ca în $(\text{minut} * 100) / 60$, chiar dacă aceasta nu schimbă rezultatul.
- **2. Ridicarea la putere** are următoarea cea mai mare precedență, așa că $2 ** 1 + 1$ este 3 și nu 4 și $3 * 1 ** 3$ este 3 și nu 27.

- 
- **3. Înmulțirea și împărțirea** au aceeași precedență, care este mai mare decât adunarea și scăderea, care au, aceeași prioritate. Așa că $2 * 3 - 1$ face 5 în loc de 4.
 - Operatorii cu aceeași importanță sunt evaluați de la **stânga la dreapta**. Deci, în expresia $59 * 100 / 60$, înmulțirea se efectuează în primul rând, rezultând $5900 / 60$, care, la rândul său dă rezultatul 98. În cazul în care operațiunile au fost evaluate de la dreapta la stânga, rezultatul ar fi fost de $59 * 1$, care este de 59, ce este greșit.
 - În mod similar, în evaluarea $17 - 4 - 3$, $17 - 4$ este evaluată mai întâi.
 - Dacă aveți îndoieli, utilizați paranteze.

Exemplu

```
>>> x = 1 + 2 ** 3 / 4 * 5
```

```
>>> print (x)
```

```
11
```

```
>>>
```

Parenthesis


Power

Multiplication

Addition

Left to Right



- 
- Atunci când efectuați o operațiune în cazul în care un operand este un **număr întreg**, iar celălalt este un operand **număr real** rezultatul este **un număr real**
 - Numărul întreg este transformat într-un număr real înainte de operație

```
>>> print (99/100)
```

```
0.99
```

```
>>> print (99.0/100)
```

```
0.99
```

```
>>> print(1 + 2 * 3 / 4.0 - 5)
```

```
-2.5
```

Tipul datelor. Funcția type()

- Tipul datelor - Un set de valori. Tipul de valoare determină modul în care poate fi utilizată variabila în expresii. Valorile sunt de **tip numere întregi** (de tip **int**), **numere reale** (de tip **float**), numere complexe, **șiruri de caractere** (de tip **string**).

```
>>> x=2
```

```
>>> type(x)
```

```
<class 'int'>
```

```
>>> type(1.0)
```

```
<class 'float'>
```


Tip de date șir de caractere

```
>>> n= "omleta"
>>> n
'omleta'
>>> print("cartof")
cartof
>>> print('rosii')
rosii
>>> type(n)
<class 'str'>
>>> |
```

```
>>> """Acesta este un sir de
caractere pe mai
multe rinduri"""
'Acesta este un sir de\ncaractere pe mai\nmulte rinduri'
>>> |
```

Efectul expresiilor matematice asupra șirurilor

- De exemplu, "+" înseamnă „**adună**” dacă operanzii sunt numere și „**concatinează**” dacă variabila este de tip șir de caractere
- Înmulțire „*” înseamnă **concatinare multiplă(replicare)**

```
>>> ee="hello"+"there"  
>>> ee  
'hellothere'  
>>> type(ee)  
<class 'str'>
```

```
>>> print('hi'*5)  
hihihihihi  
>>>
```

Nu putem adăuga un număr la șir

```
>>> ee="hello"+"there"
```

```
>>> ee
```

```
'hellothere'
```

```
>>> type(ee)
```

```
<class 'str'>
```

```
>>> f=ee +1
```

Traceback (most recent call last):

File "<pyshell#12>", line 1, in <module>

f=ee +1

TypeError: Can't convert 'int' object to str implicitly

Conversia tipurilor de date

Conversia poate fi efectuată cu funcțiile predefinite **int()** și **float()**, **str()**

```
>>> i=40
```

```
>>> type(i)
```

```
<class 'int'>
```

```
>>> f=float(i)
```

```
>>> f
```

```
40.0
```

```
>>> print (1 + 2 * float(3) / 4 - 5)
```

```
-2.5
```

Conversia șirului de caractere

```
>>> sir= "235"
>>> type(sir)
<class 'str'>
>>> adun = sir + 1
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    adun = sir + 1
TypeError: Can't convert 'int' object to str implicitly
>>> adun = int(sir) + 1
>>> adun
236
>>> type(adun)
<class 'int'>
>>> |
```

```
>>> sir2="salut"
>>> int(sir2)
Traceback (most recent call last):
  File "<pyshell#47>", line 1, in <module>
    int(sir2)
ValueError: invalid literal for int() with base 10: 'salut'
>>> |
```

Conversia șirului de caractere introdus de utilizator

- ➡ Dacă dorim să citim un număr scris de utilizator, trebuie să-l **transformăm dintr-un șir de caractere într-un număr** utilizând o funcție de tip de conversie

```
>>> n = input("Scrie un numar: ")
Scrie un numar: 23
>>> type(n)
<class 'str'>
>>> numar = int(n) + 5
>>> numar
28
>>> |
```


Comentarii în Python

- Comentarii pe un rând după semnul #
- Comentarii pe mai multe rânduri """ ... """

```
>>> #Acesta este un comentariu  
>>> |
```

```
""" Scriu comentariu  
pe mai multe rânduri  
|  
"""  
  
...  
alt tip de comentariu  
  
...
```

Scriem codul explicit

```
x1q3z9ocd = 35.0
```

```
x1q3z9afd = 12.50
```

```
x1q3p9afd = x1q3z9ocd * x1q3z9afd
```

```
print (x1q3p9afd)
```

```
a = 35.0
```

```
b = 12.50
```

```
c = a * b
```

```
print (c)
```

```
hours = 35.0
```

```
rate = 12.50
```

```
pay = hours * rate
```

```
print pay
```

Funcțiile în Python

- Există două tipuri de funcții în python.
- Funcțiile predefinite în python de ex. - `input()`, `type()`, `float()`, `int()`, `str()`, `print()`, etc.
- Funcțiile care noi le creem și le utilizăm
- În python o funcție reprezintă cod ce ia ca argumente date de intrare și după executare returnează un rezultat sau mai multe
- Definim funcția prin cuvântul rezervat `def`
- Apelăm funcția prin numele ei, paranteze și argumentele într-o expresie
- Odata definită funcția poate fi apelată ori de câte ori doriți



Funcțiile în Python. Definirea funcțiilor



```
def name( lista_parametri ):
    instructiuni
```

Apelul funcției

```
>>> def functie():  
        print("Salut")  
        print('Tuturor')
```

```
>>> functie()  
Salut  
Tuturor  
>>> |
```

Argumentele funcției

Un **argument** este o **valoare introdusă în funcție ca date de intrare** atunci când vom apela funcția

- Noi folosim argumente astfel încât să putem direcționa funcția de a executa diferite operații, atunci când le apelăm

`big = max('Salut')`

- Un parametru este o variabilă pe care o folosim în definiția funcției care permite în funcție de a avea acces la argumentele pentru o anumită invocare a funcției.

```
def greet(lang):  
    if lang == 'es':  
        print ('Hola')  
    elif lang == 'fr':  
        print ('Bonjour')  
    else:  
        print ('Hello')
```

```
greet('es')  
greet('fr')  
greet('orice')
```

```
Hola  
Bonjour  
Hello
```

Returnarea funcției

Adesea, o funcție va lua argumentele sale, va face unele manipulări și va **returna o valoare** care urmează să fie utilizată ca valoarea apelului funcției.

Cuvântul cheie **return** este folosit pentru acest lucru.

```
def greet():  
    return ("Salut")
```

```
print (greet(), "Ion")  
print (greet(), "Maria")
```

```
Salut Ion  
Salut Maria  
>>> |
```


Parametri multipli

```
>>> def aduna(a, b):  
        adun = a + b  
        return adun
```

```
>>> aduna(34, 45)
```



```
79
```

```
>>> x=aduna(2, 5)
```

```
>>> print(x)
```

```
7
```

```
>>> |
```

- 
- 
- Atunci când o funcție **nu returnează** o valoare, o numim funcție "**void**"
 - Funcții **care returnează valori** sunt funcții "**fructuoase/ productive**"

Intruțiunea condițională

if BOOLEAN EXPRESSION:
STATEMENTS

Exemplu

```
x = 5
if x < 10:
    print ('Mai mic')
if x > 20:
    print('mai mare')
print('Atat')
```

Rezultat

```
Mai mic
Atat
>>>
```

Operatorii de comparare

- **Expresiile booleene** pun o întrebare și produc rezultate de tip **Da sau Nu**, pe care le folosim pentru a controla fluxul din program
- Expresiile booleene folosind operatorii de comparare evaluează prin - **Adevărat / Fals** - Da / Nu
- **Operatorii de comparare se uită la variabile, dar nu schimbă variabilele**

Python	Meaning
<	Less than
<=	Less than or Equal
==	Equal to
>=	Greater than or Equal
>	Greater than
!=	Not equal

x	y	z	Boolean Expression	Value
"windy"	"rainy"	-	<code>x=="windy"</code>	
"windy"	"rainy"	-	<code>y=="rainy"</code>	
"windy"	"rainy"	-	<code>(x=="windy") and (y=="rainy")</code>	
"windy"	"sunny"	-	<code>(x=="windy") and (y=="rainy")</code>	
"windy"	"sunny"	-	<code>(x=="windy") or (y=="rainy")</code>	
True	-	-	<code>x</code>	
True	True	-	<code>x or y</code>	
True	False	-	<code>x and y</code>	
True	False	True	<code>x and (y or z)</code>	
10	5	7	<code>(x<y) or (y<z)</code>	
10	5	7	<code>(x<y) and (y<z)</code>	
10	5	7	<code>(x<y) and (y<x)</code>	
10	5	7	<code>(x<y) or (y<z)</code>	
10	10	7	<code>(x<y) or (y<z)</code>	
10	10	7	<code>((x<y) or (y<z)) and ((x<20) and (y>5))</code>	



Ce rezultate ne vor da următoarele expresii

```
True or False  
True and False  
not(False) and True  
not(False or True)  
not(False and True)  
False or not(True)  
True and True  
True or True  
False or False
```

Exemplu

```
x = 5
if x == 5:
    print( 'Egal')
if x > 4:
    print ("mai mare de 4 ")
if x >= 5:
    print( "mai mare sau egal cu 5 ")
if x < 6:
    print ("Mai mic de 6 ")
if x <= 5:
    print("Mai mic sau egal cu 5 ")
if x != 6:
    print(" Nu este egal cu 6 ")
```

➡ Rezultat

Egal

mai mare de 4

mai mare sau egal cu 5

Mai mic de 6

Mai mic sau egal 5

Nu este egal cu 6

>>>

Intruțiuni imbricate

```
x = 42
```

```
if x > 1:
```

```
    print ('mai mult decit unu')
```

```
        if x < 100:
```

```
            print ('mai putin decit 100')
```

```
print('Atat')
```


Rezultat

mai mult decit unu

mai putin decit 100

Atat

>>>



x = 4

if x > 2:

 print ('Bigger')

else:

 print('Smaller')

print ('All done')

Resultat

Bigger

All done

Condiții multiple

```
if x < 2:
    print( 'Small')
elif x < 10:
    print( 'Medium')
else:
    print( 'LARGE')
print( 'Atat')
```

Medium

Atat

>>> |

Example

```
if x < 2:
    print ('Small')
elif x < 10:
    print ('Medium')
elif x < 20:
    print ('Big')
elif x < 40:
    print ('Large')
elif x < 100:
    print ('Huge')
else:
    print ('Enorm')
```

Fără instrucțiunea else

```
if x < 2:
    print('Small')
elif x < 10:
    print('Medium')

print('Atat')|
```

Structura **try / except**

- Se înconjoară o secțiune periculoasă a codului cu **try** și cu **except**.
- În cazul în care codul în **try** funcționează, se sare peste **except**
- În cazul în care codul din **try** eșuează - acesta trece la secțiunea **except**

Exemplu

```
astr = 'Salut Ion'
```

```
istr = int(astr)
```

```
print ('First', istr)
```

```
astr = '123'
```

```
istr = int(astr)
```

```
print ('Second', istr)
```

```
Traceback (most recent call last):  
  File "C:\Users\Vica\Desktop\PI2016\T2\21.py.txt", line 90, in <module>  
    istr = int(astr)  
ValueError: invalid literal for int() with base 10: 'Hello Bob'  
>>> |
```

Exemplu continuare

```
astr = 'Salut Ion'
```

```
try:
```

```
    istr = int(astr)
```

```
except:
```

```
    istr=-1
```

```
print ('First', istr)
```

```
astr = '123'
```

```
try:
```

```
    istr = int(astr)
```

```
except:
```

```
    istr=-1
```

```
print ('Second', istr)
```

```
First -1
```

```
Second 123
```

```
>>> |
```

Exemplu

```
rawstr = input('introdu un numar:')
```

```
try:
```

```
    ival = int(rawstr)
```

```
except:
```

```
    ival = -1
```

```
if ival > 0 :
```

```
    print('Bine lucrat')
```

```
else:
```

```
    print('Nu este un numar')
```

```
introdu un  numar:u
```

```
Nu este un numar
```

```
>>>
```

```
===== RESTART: C:/Us
```

```
introdu un  numar:8
```

```
bine lucrat
```

```
>>> |
```

Sumar

- Variabile
- Cuvinte rezervate
- Operatori. Expresiile matematice
- Ordinea operatorilor
- Funcțiile: `type()`, `int()`, `float()`, `str()`
- Conversia dintre tipuri de date
- Funcția `Input()`
- Comentarii (`#`)
- Funcții. Definirea. Apelarea.
- Operatorii de comparare `==` `<=` `>=` `>` `<` `!=`
- Identarea
- Condiții unilaterale
- `if` : și `else` :
- Decizii imbricate
- Decizii multiple `elif`
- `Try / Except` pentru compensarea erorilor