

# Curs 1: Incadrare, bibliografie, proces, limbajul Python

## Bibliografie

1. Jake VanderPlas, *Python Data Science Handbook - Essential tools for working with data* (<https://jakevdp.github.io/PythonDataScienceHandbook/>), O'Reilly Media Inc., 2017; [notebooks](https://github.com/jakevdp/PythonDataScienceHandbook) (<https://github.com/jakevdp/PythonDataScienceHandbook>)
2. Joel Grus, *Data Science from Scratch*, O'Reilly Media, 2015
3. Peter Bruce & Andrew Bruce, *Practical Statistics for Data Scientists*, O'Reilly Media, 2017
4. Steven S. Skiena, *The Data Science Design Manual*, Springer, 2017
5. Ron Zacharski, *A Programmer's Guide to Data Mining: The Ancient Art of the Numerati* (<http://guidetodatamining.com/>)
6. Carl Shan, William Chen, Henry Wang, Max Song, *The Data Science Handbook: Advice and Insights from 25 Amazing Data Scientists* (<http://www.thedata-sciencehandbook.com/>), Data Science Bookshelf, 2015
7. Allen B. Downey, *Think Python: How to Think Like a Computer Scientist* (<http://greenteapress.com/wp/think-python-2e/>), editia a doua, Green Tea Press
8. Shai Shalev-Shwartz, Shai Ben-David, *Understanding Machine Learning: From Theory to Algorithms* (<http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/>), Cambridge University Press, 2014
9. Kevin Markham, *Introduction to machine learning in Python with scikit-learn* (video series) (<http://www.dataschool.io/machine-learning-with-scikit-learn/>)
10. EdX DAT208x: Introduction to Python for Data Science (<https://www.edx.org/course/introduction-python-data-science-microsoft-dat208x-8>)
11. *Learn to Program Using Python* (<https://www.edx.org/course/learn-program-using-python-utarlingtonx-cse1309x>)
12. *Learn Python the Hard Way* (<https://learnpythonthehardway.org>)

## Ce este Data Science?

- cunoscuta si ca de data-driven science; un termen generic, referind practici, algoritmi, modalitati de studiu ale problemelor ce implica date
- domeniu interdisciplinar prin care se vizeaza extragerea de cunostinte sau informatii din date aflate in diverse forme
- utilizeaza: matematica, statistica, teoria informatiei, sisteme instruibile (invatare automata, machine learning), vizualizare, ...
- scop: sa se obtina din date cunostinte pe baza carora se poate actiona

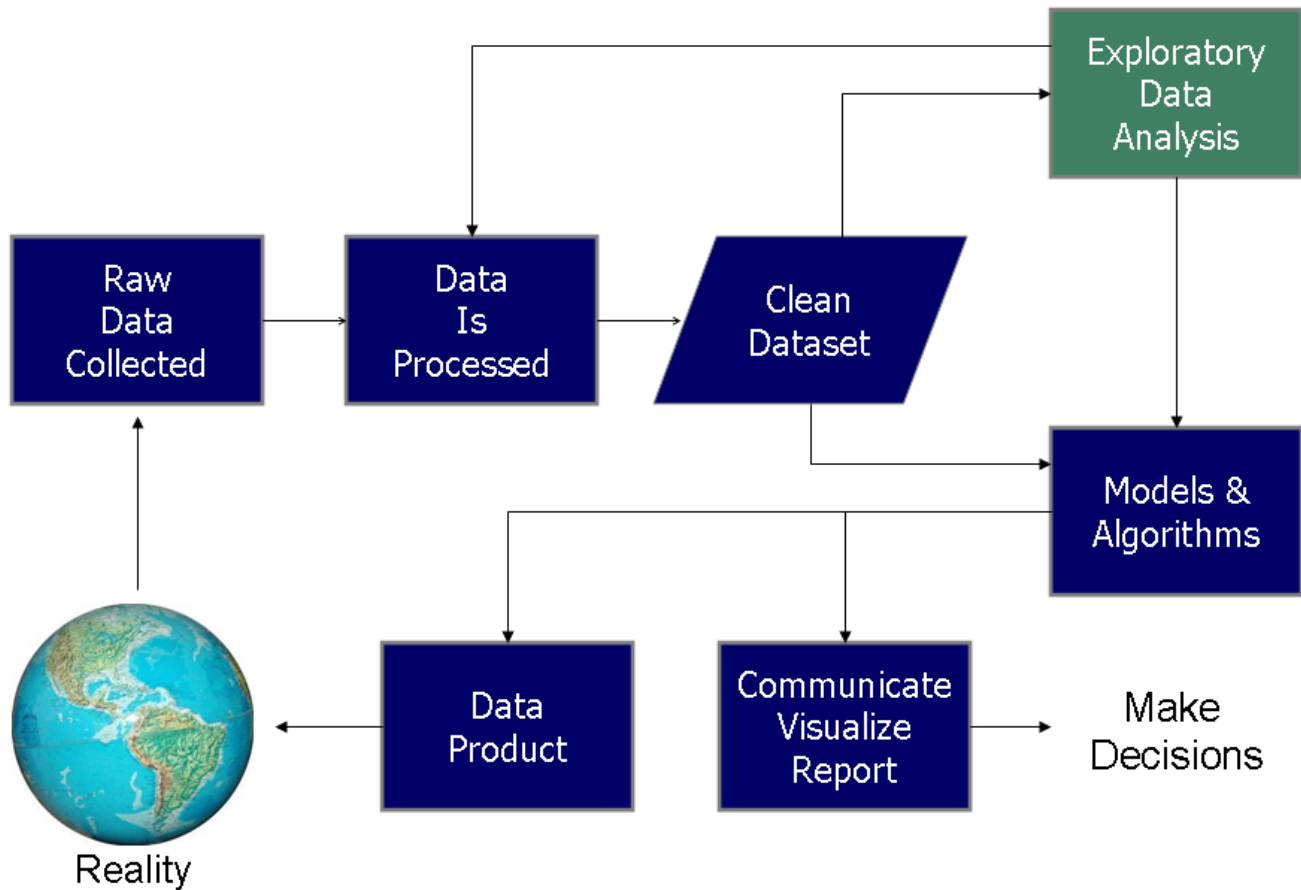
## Conferinte si jurnale dedicate:

- *European Conference on Data Analysis (ECDA)* (<http://groups.uni-paderborn.de/eim-i-fg-huellermeier/ecda2018/>)
- *IEEE International Conference on Data Science and Advanced Analytics* (<https://dsaa2018.isi.it/home>)
- *Conference on Statistical Learning and Data Science / Nonparametric Statistics* (<https://publish.illinois.edu/sldsc2018/>)
- *International Journal on Data Science and Analytics* (<https://www.springerprofessional.de/en/international-journal-of-data-science-and-analytics/11854902>)
- *Lista valabila in Februarie 2019* (<https://www.kdnuggets.com/meetings/index.html>)

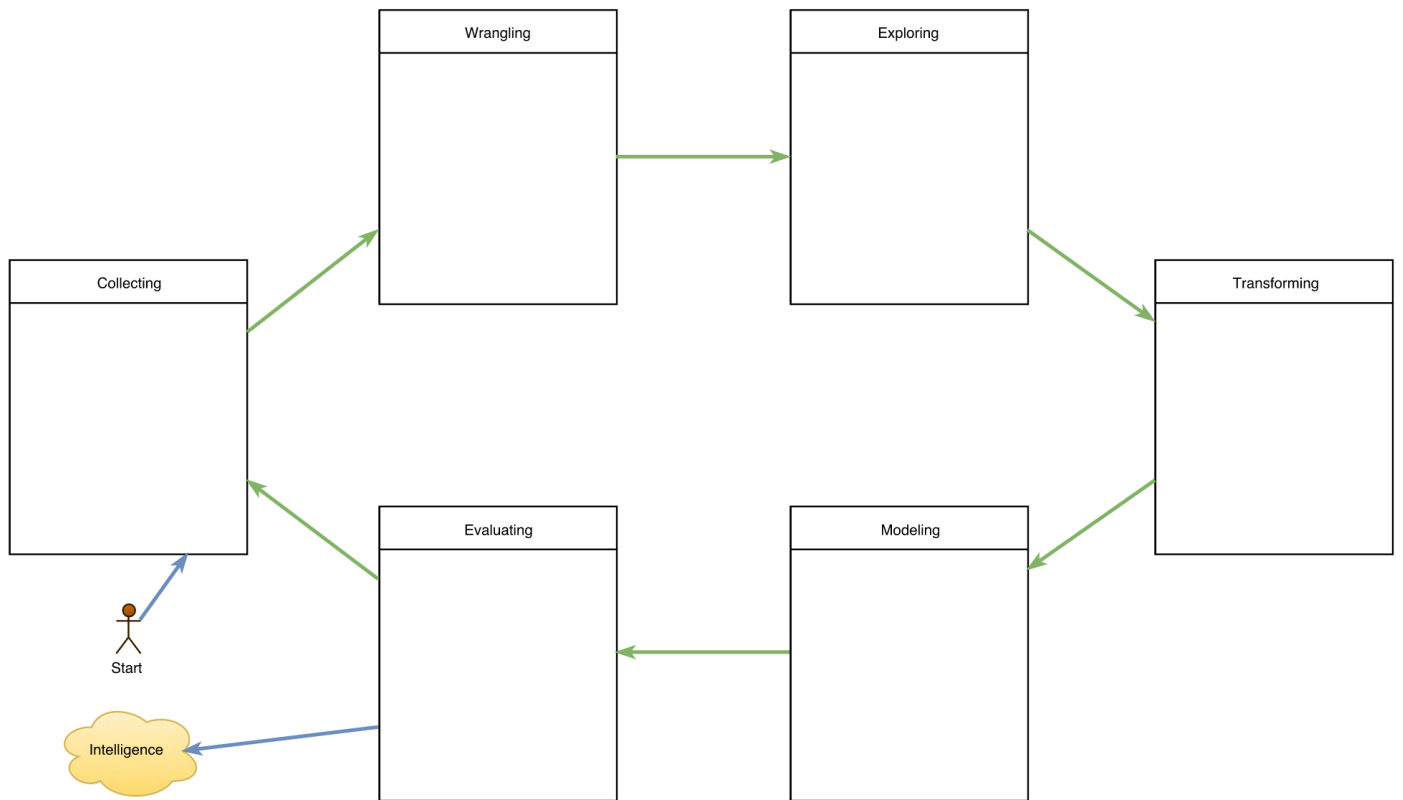
# Procesul Data Science

Sursa: Farcaster at English Wikipedia, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=40129394>  
(<https://commons.wikimedia.org/w/index.php?curid=40129394>)

## Data Science Process



Sursa: <https://github.com/authman/DAT210x/blob/master/Module1/Course%20Map.pdf>  
(<https://github.com/authman/DAT210x/blob/master/Module1/Course%20Map.pdf>)

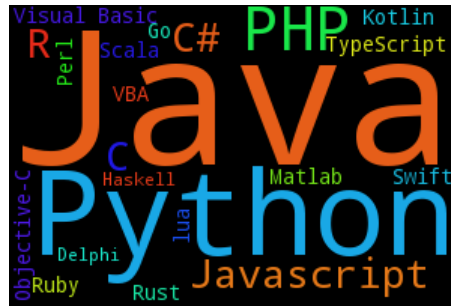


## Exemple de probleme tratate de data science

1. Google pagerank
2. Recomandari de carti pe Amazon: alti cumparatori care au achizitionat cartea X de asemenea au mai fost interesati/au cumparat ...; similar, Netflix prize
3. Clasificarea de documente de tip text - e.g. mail spam/nonspam
4. Campanii politice (Obama/Binden): gaseste oamenii indecisi
5. Reclama targetata: Nissan Motors (<https://www.warc.com/NewsAndOpinion/News/36902?>) Determinarea preferintelor pe modele in functie de regiune
6. Domeniul medical: echipament medical de monitorizare (portabil sau nu); imbunatatirea acuratetie diagnosticului; scaderea numarului de re-internari clinice (sursa: <https://www.altexsoft.com/blog/datascience/7-ways-data-science-is-reshaping-healthcare/> (<https://www.altexsoft.com/blog/datascience/7-ways-data-science-is-reshaping-healthcare/>))
7. Online customer support (chat bots)
8. Detectarea de fraude
9. Securitatea datelor, detectare automata de malware ([https://www.kaspersky.com/about/press-releases/2014\\_kaspersky-lab-is-detecting-325000-new-malicious-files-every-day](https://www.kaspersky.com/about/press-releases/2014_kaspersky-lab-is-detecting-325000-new-malicious-files-every-day)).
10. Procesarea de limbaj natural: detectarea sentimentelor din comunicari (tweets, blogs)

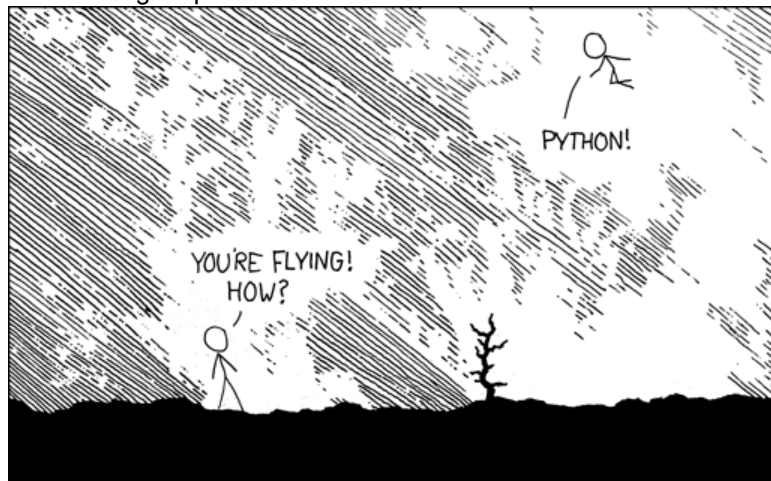
## Limbajul Python

- Limbaj open source, gratuit
- In februarie 2011: al treilea limbaj ca popularitate in TIOBE Index (<https://www.tiobe.com/tiobe-index/>), dupa Java, C, devansand C++, C#, JavaScript, R
- In aceeași luna, indexul PYPL (<http://pypl.github.io/PYPL.html>) raporteaza Python pe locul al intai



(Sursa: <http://pypl.github.io/PYPL.html> (<http://pypl.github.io/PYPL.html>), Februarie 2018)

- Doua versiuni majore: Python 2 (ultima versiune: 2.7) si 3 (versiunea curenta: 3.7).
- Python 2 are mai multe biblioteci dezvoltate, dar ele sunt treptat extinse pentru versiunea 3; diferentele de limbaj intre 2 si 3 sunt destul de mici
- Se recomanda folosirea versiunii 3 a limbajului
- Se poate instala manual sau folosind o distributie de tipul Anaconda (<https://conda.io/docs/user-guide/install/download.html>)
  - A se vedea despre distributiile Anaconda, Miniconda, Anaconda Enterprise
  - Exista o lista consistenta de distributii (<https://wiki.python.org/moin/PythonDistributions>) de Python
- Python este limbaj interpretat si netipizat
- Permite programare imperativa, orientata pe obiect, functională (<https://docs.python.org/3/library/functional.html>).
- Exista o paleta larga de biblioteci deja implementate pentru Python, usurand considerabil dezvoltarea de prototipuri sau chiar de aplicatii comerciale larg raspandite.



(Sursa: <https://xkcd.com/353/> (<https://xkcd.com/353/>))

- Peste 120000 pachete dezvoltate (<https://pypi.python.org/pypi>)
- Python este destul de mult adoptat pentru proiecte comerciale:
  - Which Internet companies use Python (<https://www.quora.com/Which-Internet-companies-use-Python>)
  - Who uses Python (<https://www.quora.com/Who-uses-Python>)

- [Is Python a Good Choice for Enterprise Projects? \(https://julien.danjou.info/is-python-a-good-choice-for-entreprise-projects/\)](https://julien.danjou.info/is-python-a-good-choice-for-entreprise-projects/)

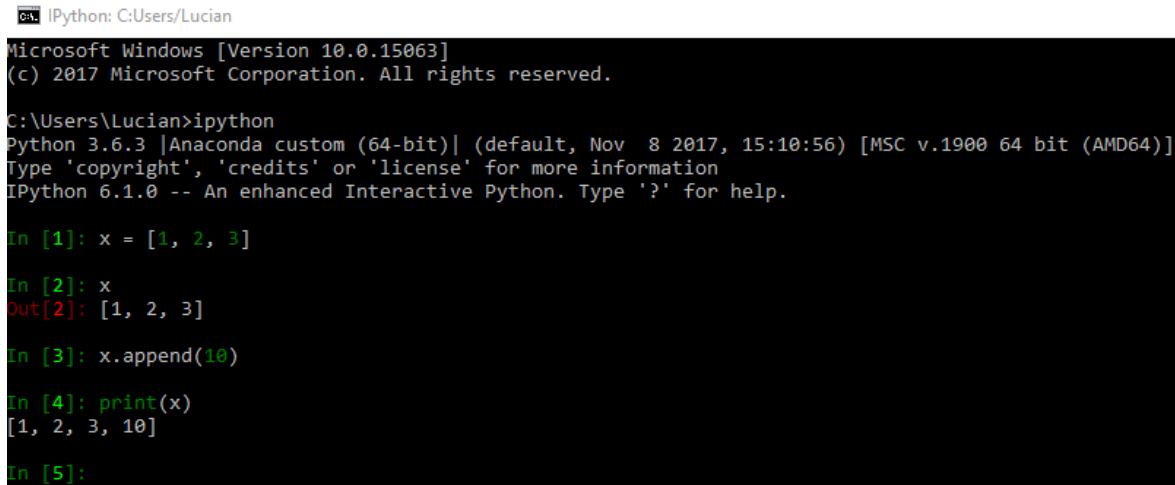
## De ce Python?

Iata cateva motive solide pentru a utiliza Python, conform [The unexpected effectiveness of Python in science \(https://lwn.net/Articles/724255/\)](https://lwn.net/Articles/724255/)

1. Limbajul vine "cu toate bateriile incluse": multitudinea de biblioteci il face perfect pentru o gama larga de proiecte. Cercetatorii care il folosesc activeaza in astronomie, fizica, bioinformatica, chimie, stiinte cognitive etc. - si pentru toate aceste domenii exista biblioteci care simplifica mult dezvoltarea de prototipuri. Prin comparatie, limbaje traditionale precum C/C++/Java/Python vin cu mai putin suport disponibil, iar uneori includerea unui pachet/biblioteci poate fi o experienta consumatoare de timp.
2. Capacitatea de interoperare cu alte limbaje este iarasi un plus. Exista puncte de comunicare (bridges) care permit apelul de cod Python din alte limbaje sau invers. Desigur, aceasta facilitate se regaseste si in alte limbaje.
3. Natura dinamica a limbajului poate fi inteligent speculata: o functie poate sa preia o multitudine de parametri (lista, tuplu, dictionar) si cateva linii de cod functioneaza la fel de bine peste toate acestea.
4. Python incurajeaza un stil de lucru experimental, via REPL: poate e nevoie de cateva incercari pentru a ajunge la instructiunile potrivite pentru o anumita functionalitate. Incercarea unei alte instructiuni nu necesita recompilarea sau rerularea codului anterior, ci vine in completare - pana cand ajungi la ce doresti sa obtii.

## Moduri de utilizare

1. Codul poate fi scris intr-un fisier text cu extensia py. Lansarea codului se face cu comanda `python nume_script.py`
2. Se poate folosi interpretorul python sau ipython, in care modul de lucru este REPL: read-evaluate-print-loop. REPL permite o prototipizare rapida si scrierea codului in mod incremental. Interpretorul ipython este o varianta imbunatatita a interpretorului python.



```

C:\Users\Lucian>ipython
Python 3.6.3 [Anaconda custom (64-bit)] (default, Nov  8 2017, 15:10:56) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: x = [1, 2, 3]

In [2]: x
Out[2]: [1, 2, 3]

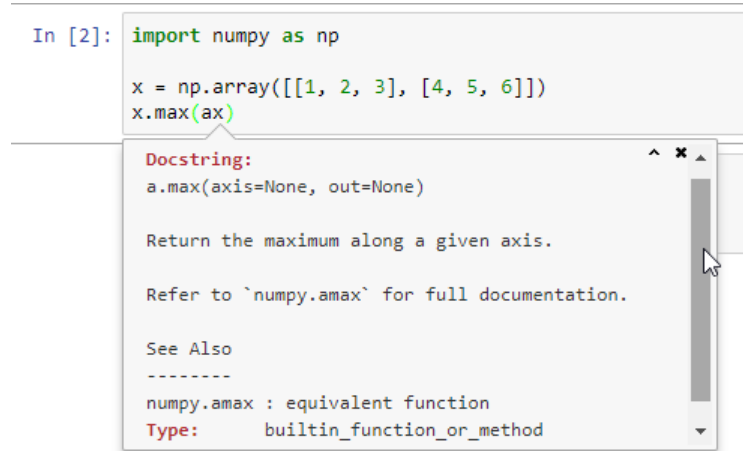
In [3]: x.append(10)

In [4]: print(x)
[1, 2, 3, 10]

In [5]:
  
```

Mai multe detalii pentru lucrul cu interpretorul ipython se gasesc in resursa bibliografica [1], capitolul 1: debug, magic commands, lucru cu interpretorul de comenzi etc.

3. Jupyter notebook - codul este scris in browser, cu suport pentru completare automata de cod. Fisierelor rezultate sunt cu extensia ipynb.



O lista de notebooks se gaseste [aici](http://nbviewer.jupyter.org/) (<http://nbviewer.jupyter.org/>).

4. Folosind diverse framework-uri (de ex Flask), codul Python poate fi folosit pentru crearea de pagini Web, REST endpoints - sau mai simplu, functiile definite in Jupyter notebook pot fi expuse ca servicii web REST, etc.

## Variabile, tipuri de date

Clasica afisare de mesaj "Hello world" se obtine cu *functia* `print()`:

```

In [1]: print("Hello world!")
#sau: print('Hello world')

Hello world!
  
```

Pentru Python versiunea 2, afisarea se face folosind *instructiunea* print:

```
print "Hello world!"
```

- se observa lipsa parantezelor. In ambele cazuri delimitarea sirurilor de caractere se poate face cu apostroafe sau ghilimele.

Variabilele nu se declara in prealabil impreuna cu tipul lor - Python este un limbaj slab tipizat. Natura unei variabile se deduce din valoarea care ii este asociata:

```
In [2]: x = 3 #x e variabila de tip intreg
        y = "abcd" #y este variabila de tip sir de caractere
        #se obisnuieste ca numele compuse ale variabilelor sa fie despartite prin _:
        nume_complet = "Popescu Ion"
```

Python este case sensitive: Nume\_complet si nume\_complet refera variabile diferite. Atribuirea se face folosind semnul egal, asa cum s-a vazut mai sus.

Tipurile concrete ale variabilelor poate fi aflat la rulare:

```
In [3]: #functia type
        print(type(x))
        print(type(y))

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

Variabilelor li se pot da nume incepand cu caracter sau , *continuand cu caractere*, sau cifre. Urmatoarele cuvinte rezervate nu pot fi folosite ca nume de variabile:

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

## Tipuri numerice





```
In [8]: z = 1 + 4j
print(z)
print(type(z))
print(z.conjugate())
z2 = z * z.conjugate()
print(z2)

(1+4j)
<class 'complex'>
(1-4j)
(17+0j)
```

Sunt admise conversii intre tipuri numerice:

```
In [9]: a = 3.997
b = int(a)
print(b)
print(float(b))

3
3.0
```

```
In [10]: #Alte exemple: https://www.tutorialspoint.com/python/python\_numbers.htm
```

## Siruri de caractere

Pentru variabile de tip sir de caractere, exista posibilitatea de a scrie valori care nu necesita secvente escape:

```
In [11]: a = r'adresa \\michel\protect\....' #remarcam prefixul r
print(a)

#the hard way...
a = 'adresa \\\michel\\protect\\....'
print(a)

#caractere Unicode
a = u"ĂÎÂȘȚ ăăîâșț" #se remarca prefixul u
print(a)

adresa \\michel\protect\....
adresa \\\michel\\protect\\....
ĂÎÂȘȚ ăăîâșț
```

..sau stringuri multilinie, folosind delimitare cu trei ghilimele sau trei apostroafe:

```
In [12]: versuri = '''If you can keep your head when all about you
    Are losing theirs and blaming it on you,
    If you can trust yourself when all men doubt you,
    But make allowance for their doubting too; '''
print(versuri)

If you can keep your head when all about you
    Are losing theirs and blaming it on you,
    If you can trust yourself when all men doubt you,
    But make allowance for their doubting too;
```

Sirurile de caractere se pot concatena cu + si suporta urmatoarele operatii si functii:

```
In [13]: sir = "Ana " + "are " + "mere"  
print(sir)
```

Ana are mere

```
In [14]: print('Lungimea sirului: ', len(sir))  
print('Primul element din sir: ', sir[0])  
print('Ultimul element din sir: ', sir[len(sir)-1])  
print("Ultimul element, in stil Python: ", sir[-1])  
print("Penultimul element din sir, in stil Python: ", sir[-2]) #!!!
```

Lungimea sirului: 12  
Primul element din sir: A  
Ultimul element din sir: e  
Ultimul element, in stil Python: e  
Penultimul element din sir, in stil Python: r

```
In [15]: #Conversia unei variabile de tip non-string in string:  
x = 3  
mesaj = "numarul este " + str(x)  
print(mesaj)
```

numarul este 3

```
In [16]: #metode si operatii posibile pe obiect string:  
print(sir.lower())  
print('slicing:', sir[4:7])  
print(sir.find('nu are'))  
print(sir.replace('Ana', 'Ion'))
```

ana are mere  
slicing: are  
-1  
Ion are mere

```
In [17]: multi_ana = 'Ana' * 10  
print(multi_ana)
```

AnaAnaAnaAnaAnaAnaAnaAnaAnaAna

```
In [18]: cuvantul_pere_este_in_sir = 'pere' in sir  
print(cuvantul_pere_este_in_sir)  
cuvantul_pere_nu_este_in_sir = 'pere' not in sir  
print(cuvantul_pere_nu_este_in_sir)
```

False  
True

```
In [19]: tokens = sir.split(' ')  
print(type(tokens))  
print(tokens)  
print(", ".join(['Ana', 'Vasile', 'Dana', 'Ion']))
```

<class 'list'>  
['Ana', 'are', 'mere']  
Ana,Vasile,Dana,Ion

Demna de mentionat este functia eval, care preia un string de caractere si returneaza un obiect rezultat prin evaluare:

```
In [20]: a = 3
b = 4
expresie = '(a+b)/(a**2 + b**2 + 1)'
print(eval(expresie))
```

```
0.2692307692307692
```

Recomandam citirea documentatiei oficiale (<https://docs.python.org/3/library/string.html>) pentru tipul de date string.

Python contine tipul `bool`, util pentru reprezentarea valorilor de adevar `True` si `False`:

```
In [21]: x = True
print(type(x))
```

```
<class 'bool'>
```

Se pot face conversii de la tipuri numerice la `bool` si viceversa: valoarea 0 este asociata lui `False`, orice non-zero lui `True`. Invers, `True` se traduce in 1 si `False` in 0:

```
In [22]: x = bool(-1)
print('-1 ca bool: ', x)
x = bool(0.0)
print('0.0 ca bool: ', x)
b = True
print('True ca int:', int(b))
b = False
print('False ca int:', int(b))
```

```
-1 ca bool: True
0 ca bool: False
True ca int: 1
False ca int: 0
```

Operatorii aplicabili pe valori `bool` sunt:

```
In [23]: print(True and False)
print(True or False)
print(not False)
#nu exista xor, dar...
a, b = True, False
print(a != b)
```

```
False
True
True
True
```

## Liste

In interiorul unei liste se pot pune oricate elemente, nu neaparat de acelasi tip. Elementele se despart prin virgula; capetele listei sunt marcate prin paranteze drepte:

```
In [24]: lista = [10, 20, 30, 40]
        print('Lungimea listei este', len(lista))
        print('Tipul listei este', type(lista))
        #Lista eterogena: string si int
        lista2 = ['Ana', 234]
```

```
Lungimea listei este 4
Tipul listei este <class 'list'>
```

Elementele unei liste se acceseaza pe baza de indici, incepand de la 0 si terminand cu `len(lista)-1`. Ultimul element se referentia cu indicele `-1`:

```
In [25]: lista = [10, 20, 30]
        print('Ultimul element este:', lista[-1])
        print('Penultimul element este:', lista[-2])
```

```
Ultimul element este: 30
Penultimul element este: 20
```

O lista se poate obtine prin repetarea unei constructii de un numar dorit de ori:

```
In [26]: lista = [1, 2, 3] * 5 #efect similar demonstrat anterior pe string 'Ana'
        print(lista)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Putem prelua portiuni intregi din lista, nu doar de pe pozitii individuale, folosind 'decuparea' (slicing), sub forma: `lista[k:l]`. Va rezulta o lista formata din elementele `lista[k]`, `lista[k+1]`, ..., `lista[l-1]` (se remarca marginea din dreapta a sirului: nu e l, ci l-1). Daca se doreste ca ordinea elementelor sa fie inversata, se foloseste `lista[l:k:-1]`; elementul de indice l va fi inclus in rezultat, elementul de indice k se va omite:

```
In [27]: lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        print(lista)
        sliced = lista[3:8]
        print(sliced)
        sliced_reversed = lista[8:3:-1]
        print(sliced_reversed)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[4, 5, 6, 7, 8]
[9, 8, 7, 6, 5]
```

```
In [28]: lista = [10, 20, 30, 40]
```

Formele acceptate pentru slicing sunt:

- `a[start:end]` # items start through end-1
- `a[start:]` # items start through the rest of the array
- `a[:end]` # items from the beginning through end-1
- `a[:]` # a copy of the whole array

In special ultima forma este interesanta: se obtine o clona a listei a. In lipsa acestui mecanism, doua variabile care indica spre aceeasi lista pot duce la accese concurente nedorite:

```
In [29]: #aliasing
a = [1, 2, 3]
b = a
print('a=', a)
print('b=', b)
a[0] *= -1
print('Dupa modificare via a: b=', b)
#a si b refera aceeasi zona de memorie

a= [1, 2, 3]
b= [1, 2, 3]
Dupa modificare via a: b= [-1, 2, 3]
```

```
In [30]: #copiere de lista
a = [1, 2, 3]
b = a[:]
print('a=', a)
print('b=', b)
a[0] *= -1
print('Dupa modificare: a=', a)
print('Dupa modificare: b=', b)
#a si b refera zone de memorie diferite

a= [1, 2, 3]
b= [1, 2, 3]
Dupa modificare: a= [-1, 2, 3]
Dupa modificare: b= [1, 2, 3]
```

Copierea unei liste se poate face si cu metoda `copy`:

```
In [31]: #copiere de lista cu metoda copy
a = [1, 2, 3]
b = a.copy()
print('a=', a)
print('b=', b)
a[0] *= -1
print('Dupa modificare: a=', a)
print('Dupa modificare: b=', b)
#a si b refera zone de memorie diferite

a= [1, 2, 3]
b= [1, 2, 3]
Dupa modificare: a= [-1, 2, 3]
Dupa modificare: b= [1, 2, 3]
```

Listele pot fi concatenate:

```
In [32]: a = [1, 2, 3]
b = [10, 20, 30]
c = a+b
print(c)

[1, 2, 3, 10, 20, 30]
```

La o lista se pot adauga elemente:

```
In [33]: a = [1, 2, 3]
a.append(100)
print(a)

[1, 2, 3, 100]
```

Se recomanda ca daca pentru o lista se cunoaste capacitatea maxima sau chiar exacta, sa se prealoce, in loc sa se foloseasca adaugari repetate:

```
In [34]: a = [None] * 10
#....
a[3] = 'Ana are mere'
print(a)

[None, None, None, 'Ana are mere', None, None, None, None, None, None]
```

Sortarea unei liste se face cu functia `sorted`, care optional poate sa preia:

- un parametru numit `reverse` de tip boolean, care precizeaza daca se face sortare descrescatoare
- un parametru numit `key`, care face referinta la o functie ce determina politica de sortare

```
In [35]: #Sortare crescatoare
a = [5, 1, 4, 3]
print(sorted(a))

[1, 3, 4, 5]
```

```
In [36]: # Sortare descrescatoare
print(sorted(a, reverse=True))

[5, 4, 3, 1]
```

```
In [37]: #Sortare cu cheie specificata
siruri = ['ccc', 'aaaa', 'd', 'bb']
print(sorted(siruri, key=len))

['d', 'bb', 'ccc', 'aaaa']
```

Functia `sorted` returneaza o noua lista, lasand pe cea originala nemodificata. Daca se doreste ca sortarea sa se faca in cadrul liste originale, se va folosi metoda `sort` a tipului lista. Metoda `sort` returneaza `None`.

```
In [38]: siruri = ['ccc', 'aaaa', 'd', 'bb']
siruri.sort()
print(siruri)

['aaaa', 'bb', 'ccc', 'd']
```

Stergerea unui element dintr-o lista, de pe o pozitie (indice) specificat se face cu `del`:

```
In [39]: lista = [10, 20, 30]
del lista[1]
print(lista)

[10, 30]
```

Testarea existentei unui element intr-o lista se face cu operatorul `in`:

```
In [40]: lista = ['Ana', 'are', 'mere']  
print('portocale' in lista )
```

False

Minimul si maximul unei liste se obtin cu functiile `min` respectiv `max`.

Alte metode apartinand tipului `list` sunt exemplificate mai jos:

```
In [41]: lista = [10, 20, 30, 10]  
#De cate ori apare elementul 10 in lista?  
print('De cate ori apare elementul 10 in lista', lista.count(10))  
#Care e primul index in care un anumit element apare in lista?  
print('Care e primul index in care un anumit element apare in lista?', lista.index(20))  
#daca pentru metoda index se specifica un element care nu exista, se arunca eroare:  
# print(lista.index(100000))  
#ValueError: 100000 is not in list  
#Inserare elementului 100 pe indexul 2 in lista:  
lista.insert(2, 100)  
print('Inserare elementului 100 pe indexul 2 in lista', lista)  
#Inversarea ordinii elementelor dintr-o lista, in-place  
lista.reverse()  
print('Inversarea ordinii elementelor dintr-o lista, in-place', lista)  
#Stergerea unui element din lista, prima aparitie  
lista.remove(10)  
print('Stergerea unui element din lista, prima aparitie', lista)  
lista.clear()  
print('Lista a fost golita:', lista)
```

```
De cate ori apare elementul 10 in lista 2  
Care e primul index in care un anumit element apare in lista? 1  
Inserare elementului 100 pe indexul 2 in lista [10, 20, 100, 30, 10]  
Inversarea ordinii elementelor dintr-o lista, in-place [10, 30, 100, 20, 10]  
Stergerea unui element din lista, prima aparitie [30, 100, 20, 10]  
Lista a fost golita: []
```

Pentru testarea faptului ca toate elementele (respectiv: macar un element al) unei liste de valori boolene sunt cu valoarea `true`, se va folosi functia `all` (respectiv: `any`).

```
In [42]: lista = [True, False, True]  
print('Macar unul e True:', any(lista))  
print('Toate sunt True:', all(lista))
```

```
Macar unul e True: True  
Toate sunt True: False
```

Daca lista este goala, functia `all` returneaza `True`, iar `any` - `False`.

```
In [43]: print('Macar unul e True:', any([]))  
print('Toate sunt True:', all([]))
```

```
Macar unul e True: False  
Toate sunt True: True
```

O metoda deosebit de eleganta de procesare a elementelor unei liste este prin `list comprehension`, ce se va prezenta in cadrul sectiunii de instructiuni.

## Tupluri



În timp ce o listă permite modificarea conținutului său, un tuplu reprezintă o colecție ordonată imuabilă. Elementele se separă cu virgulă, capetele tuplului se marchează de regulă cu paranteze rotunde, sau pot lipsi.

```
In [44]: tuplu1 = ('informatica', 3, 'dimineata')
tuplu2 = 'chimie', 2, 'seara'
print(tuplu1)

('informatica', 3, 'dimineata')
```

```
In [45]: print(tuplu1[0])

informatica
```

```
In [46]: print(tuplu1[-1])

dimineata
```

```
In [47]: print(len(tuplu1))

3
```

```
In [48]: tuplu_gol = ()
print(len(tuplu_gol))

0
```

```
In [49]: la_tuplul_cu_doar_o_valoare_trebuie_adaugata_virgula_la_sfarsit = (42,)
print(len(la_tuplul_cu_doar_o_valoare_trebuie_adaugata_virgula_la_sfarsit))

1
```

Elementele unui tuplu pot fi preluate individual în variabile:

```
In [50]: print(tuplu1)
a, b, c = tuplu1
print(a)
print(b)
print(c)

('informatica', 3, 'dimineata')
informatica
3
dimineata
```

```
In [51]: #Tuplurile se pot concatena
tuplu1 + tuplu2
```

```
Out[51]: ('informatica', 3, 'dimineata', 'chimie', 2, 'seara')
```

```
In [52]: #Convertirea unei liste în tuplu:
lista = [1, 2, 3, 4, 5]
tuplu = tuple(lista)
print(type(tuplu))
print(tuplu)

<class 'tuple'>
(1, 2, 3, 4, 5)
```

## Dictionare

Dictionarele sunt colectii de asocieri intre chei si valori. Colectia de tip dictionar se demarcheaza cu acolade. Elementele se acceseaza specificand intre paranteze drepte valoarea cheii.

```
In [53]: geografie = {} #dictionar gol
geografie = {'Romania': 'Bucuresti', 'Serbia': 'Belgrad'}
print('Lungimea este:', len(geografie))
key = 'Romania'
print('Valoarea pentru cheia', key, 'este', geografie[key])
geografie['Grecia'] = 'Atena' #adaugare de pereche cheie-valoare in dictionar
#daca se foloseste o cheie care exista, valoarea asociata va fi suprascrisa cu cea noua
geografie['Grecia'] = 'Athens'
#acesarea folosind o cheie invalida duce la eroare (KeyError)
# geografie['Franta']
#pentru a evita eroare la rulare, se poate folosi metoda get; daca cheia nu e gasita in dictionar,
#se returneaza None
print(geografie.get('Franta'))
#daca se doreste returnarea unei valori prestabilite in cazul lipsei cheii, metoda get accepta un
#parametru suplimentar
print(geografie.get('Franta', '<Nu exista in dictionar>'))
```

Lungimea este: 2  
 Valoarea pentru cheia Romania este Bucuresti  
 None  
 <Nu exista in dictionar>

Colectiile cheilor unui dictionar se determina cu metoda keys, respectiv values:

```
In [54]: print('Chei:', geografie.keys())
print('Valori:', geografie.values())
```

Chei: dict\_keys(['Romania', 'Serbia', 'Grecia'])  
 Valori: dict\_values(['Bucuresti', 'Belgrad', 'Athens'])

Oricare din colectii poate fi transformata intr-o lista prin apelul constructorului list() care poate prelua o colectie oarecare:

```
In [55]: print(list(geografie.keys()))
print(list(geografie.values()))
```

['Romania', 'Serbia', 'Grecia']  
 ['Bucuresti', 'Belgrad', 'Athens']

Stergerea unei chei din dictionar, impreuna cu valoarea asociata, se face cu instructiunea del:

```
In [56]: del geografie['Grecia']
print(geografie)
```

{'Romania': 'Bucuresti', 'Serbia': 'Belgrad'}

Testarea faptului ca o anumita cheie se afla intr-un dictionar se face cu operatorul in:

```
In [57]: print('Grecia' in geografie)
```

False

Golirea unui dictionar se face, precum la alte tipuri colectie cu metoda clear:

```
In [58]: # geografie.clear()
```

Colectia perechilor (cheie, valoare) a unui dictionar se obtine cu metoda `items()`:

```
In [59]: print(geografie.items())
         print(list(geografie.items()))

         dict_items([('Romania', 'Bucuresti'), ('Serbia', 'Belgrad')])
         [('Romania', 'Bucuresti'), ('Serbia', 'Belgrad')]
```

Daca la un dictionar se doreste adaugarea perechilor (cheie-valoare) ale altui dictionar se foloseste metoda `update`:

```
In [60]: geografie_asia = {'China':'Beijing', 'India':'New Delhi'}
         geografie.update(geografie_asia)
         print(geografie)

         {'Romania': 'Bucuresti', 'Serbia': 'Belgrad', 'China': 'Beijing', 'India': 'New Delhi'}
```

Clonarea unui dictionar se face cu metoda `copy`:

```
In [61]: geografie_copie = geografie.copy()
         geografie_copie['India'] = 'New----Delhi'
         print(geografie['India'], ', ', geografie_copie['India'])

         New Delhi , New----Delhi
```

## Alte colectii

Se foloseste foarte frecvent functia `range` care produce o secventa de numere. Formele de utilizare sunt:

- `range(n)` produce colectia 0, 1, ..., n-1
- `range(start, stop)` produce colectia start, start+1, ..., stop-1
- `range(start, stop, step)` produce secventa in functie de semnul lui `step`:
  - start, start+step, start+2\*step, ... k, unde k este cel mai mare intreg mai mic decat stop, care se obtine prin adaugarea unui multiplu intreg al pasului step la start
  - start, start+step, start+2\*step, ....k unde k este cel mai mare intreg mai mare decat stop, obtinut prin adaugarea unui multiplu intreg al pasului step la start

```
In [62]: for i in range(2, 10):
         print(i, end=' ')
```

2 3 4 5 6 7 8 9

```
In [63]: for i in range(2, 10, 3):
         print(i, end=' ')
```

2 5 8

```
In [64]: for i in range(10, 2, -2):
         print(i, end=' ')
```

10 8 6 4

Functia `enumerate` porneste de la o colectie si da acces simultan atat la indicele elementului din colectie, cat si la elementul curent:

```
In [65]: lista1 = ['a', 'b', 'c']
         for i, item in enumerate(lista1):
             print(i, item)

0 a
1 b
2 c
```

Funcția zip permite 'împerecherea' a două liste

```
In [66]: lista2 = ['m', 'n', 'p']

         for pereche in zip(lista1, lista2):
             print(pereche)

('a', 'm')
('b', 'n')
('c', 'p')
```

## Instructiuni, comentarii

### Blocuri de instructiuni

Instructiunile se scriu de regula cate una pe linie. Daca pentru o instructiune prea lunga se doresc continuarea ei pe linia urmatoare, se pune la finalul liniei caracterul \ si se continua pe rand nou:

```
In [67]: a = 1 + 2 + 3 \
          + 4
          print(a)

10
```

Pentru colectii, continuarea pe rand nou nu necesita caracter backslash:

```
In [68]: lista_lunga = [1, 2, 3,
                        4, 5, 6]
```

Se pot scrie mai multe instructiuni pe o linie, despartindu-se cu caracterul ';'. Acest stil insa e nerecomandat

```
In [69]: print('1'); print('2')

1
2
```

Un bloc de instructiuni va folosi aceeasi indentare. Se poate folosi orice pentru indentare (tab, sau acelasi numar de spatii), dar stilul de indentare trebuie sa fie unitar. Un astfel de bloc de instructiuni se termina cu prima linie care nu e indentata in acelasi stil ca si blocul.

```
In [70]: if 1 + 1 == 3:
         print('Nu se afiseaza')
         print('Linia aceasta nu face parte din blocul corespunzator instructiunii if')

Linia aceasta nu face parte din blocul corespunzator instructiunii if
```

### Comentarii

Comentariile sunt fie pe o singura linie, incepand de la caracterul # pana la finalul liniei, fie folosind apostroafe sau ghilimele, de trei ori la inceput si la sfarsit de comentariu:

```
In [71]: """Comentariu
         foarte lung"""

Out[71]: 'Comentariu\nfoarte lung'
```

## Atribuirea

Atribuirea a fost exemplificata mai sus. Mai avem variantele:

```
In [72]: #Atribuire multipla cu o valoare
         a = b = c = 3
```

```
In [73]: #Atribuire multipla cu mai multe valori simultan
         a, b = 2, 3
         print(a, b)

         2 3
```

Ultima forma poate fi speculata astfel: daca a si b sunt doua variabile si se doreste interschimbarea valorilor lor, atunci putem scrie:

```
In [74]: print('Inainte de interschimbare: ', a, b)
         a, b = b, a
         print('Dupa interschimbare: ', a, b)

         Inainte de interschimbare: 2 3
         Dupa interschimbare: 3 2
```

## Instructiunea if

Formele uzuale sunt:

```
if expresie:
    bloc 1
else
    bloc 2
```

Partea else poate sa lipseasca. Blocurile de instructiuni ce urmeaza dupa if si else sunt indentate.

```
In [75]: var1 = 100
         var2 = 200
         if var1 > var2:
             print("In if - Got a true expression value")
             print(var1)
         else:
             print("In else - Got a false expression value")
             print(var2)

         In else - Got a false expression value
         200
```

O instructiune `if` poate conine multiple teste, folosind `elif`:

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```

Se evalueaza in ordine conditiile `expression1`, `expression2` etc. pana la prima care e gasita ca fiind adevarata; blocul ei este executat si restul se ignora

```
In [76]: var = 100
        if var == 200:
            print("1 - Got a true expression value")
            print(var)
        elif var == 150:
            print("2 - Got a true expression value")
            print(var)
        elif var == 100:
            print("3 - Got a true expression value")
            print(var)
        else:
            print("4 - Got a false expression value")
            print(var)
```

```
3 - Got a true expression value
100
```

Instructiunea `if` se poate folosi si inline, in forma:

```
expression_if_true if condition else expression_if_false
```

Exemplu:

```
In [77]: CNP = '5678901234567'
        gen_masculin = True if int(CNP[0]) % 2 == 1 else False
        print('gen masculin:', gen_masculin)
```

```
gen masculin: True
```

## Ciclare: `for`

Ciclarea cu `for` este folosita pentru a itera o instructiune sau un bloc de instructiuni peste o colectie cunoscuta.

```
In [78]: for i in range(0, 3):
        print('i=', i)
```

```
i= 0
i= 1
i= 2
```

```
In [79]: #Exemplu: calculul factorialului unui numar
n = 100
p = 1
for i in range(1, n+1):
    p *= i
print(n, '!=', p)
```

100 != 9332621544394415268169923885626670049071596826438162146859296389521759993229915608941463976156518286253697920827223758251185210916  
864000000000000000000000

```
In [80]: # %%timeit
# n = 100
# list_numbers = [str(i) for i in range(1, n+1)]
# expression = '.*'.join(list_numbers)
# print(eval(expression))
```

```
In [81]: lista_nome = ['Ana', 'Dan', 'Rares', 'Dana']
         for nome in lista_nome:
             print(nome)
```

Ana  
Dan  
Rares  
Dana

```
In [82]: geografie = {'Romania': 'Bucuresti', 'Serbia': 'Belgrad', 'Grecia': 'Atena'}
         for key in geografie:
             print(geografie[key])
```

Bucuresti  
Belgrad  
Atena

```
In [83]: #iterare cu doua elemente, peste colectie de perechi
for key, value in geografie.items():
    print('Capitala pentru', key, 'este', value)
```

Capitala pentru Romania este Bucuresti  
Capitala pentru Serbia este Belgrad  
Capitala pentru Grecia este Atena

```
In [84]: for index, element in enumerate(lista_num):
          print(index, element)
```

```
0 Ana
1 Dan
2 Rares
3 Dana
```

```
In [85]: #enumerare peste doua colectii 'impreunate'
        lista_anii = [20, 21, 22, 23]
        for nume, anii in zip(lista_nume, lista_anii):
            print(nume, 'are', anii, 'anii')
```

Ana are 20 ani  
Dan are 21 ani  
Rares are 22 ani  
Dana are 23 ani

Se poate forta iesirea dintr-un ciclu `for` cu instructiunea `break`. Se poate sari peste o parte din blocul unui ciclu `for` si trece la ciclarea urmatoare folosind `continue`:

```
In [86]: for i in range(100):  
        if i > 5:  
            break  
        print(i)
```

```
0  
1  
2  
3  
4  
5
```

```
In [87]: for i in range(10):  
        if i % 2 == 0:  
            continue  
        print(i)
```

```
1  
3  
5  
7  
9
```

O forma aparte a instructiunii `for` in Python este aceea in care la final se pune `else`: daca nu s-a executat `'break'` in corpul ciclului, atunci se executa blocul de dupa `else`:

```
In [88]: for n in range(2, 10):  
        for x in range(2, n):  
            if n % x == 0:  
                print( n, 'equals', x, '*', n//x)  
                break  
        else:  
            # Loop fell through without finding a factor  
            print(n, 'is a prime number')
```

```
2 is a prime number  
3 is a prime number  
4 equals 2 * 2  
5 is a prime number  
6 equals 2 * 3  
7 is a prime number  
8 equals 2 * 4  
9 equals 3 * 3
```

## Ciclare: while

Folosind `while` se cilceaza peste un bloc de instructiuni atata timp cat o anumita conditie este adevarata:

```
while test_expression:  
    body of while
```



```
In [89]: n = 10

# initialize sum and counter
my_sum = 0
i = 1

while i <= n:
    my_sum = my_sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", my_sum)

The sum is 55
```

Se poate ca o instructiune `while` sa se termine cu `else`, al carei bloc se executa cand expresia de test devine falsa, dar numai daca nu s-a ajuns la un `break`.