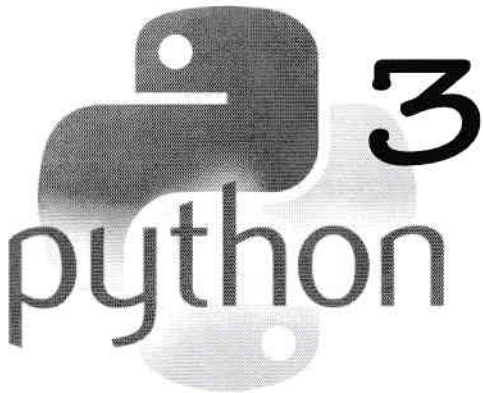
Vlad TUDOR

Curs de programare în



Volumul I

Fundamente

pentru începători

INFOBITS.RO

L&S SOFT

Cuprins

Volumul I – *Fundamente* pentru începători

Alfa. Introducere **9**
De ce Python? 9
Python, un limbaj interpretat 10
Python 2 vs. Python 3 10

Beta. Instalarea mediului de programare Python **11**
Acces rapid la Python IDLE 13

Gama. Modul de lucru în Python **15**
Crearea și salvarea unui program 16
Rularea unui program 17
Deschiderea unui program 18
Închiderea unui program 18
Fișiere recente 18
Comenzi uzuale ale editorului de text 18
Ștergerea consolei din Python IDLE 19
Executarea codului din Command Prompt 19
IDE-uri 20
Recapitulare 20

Capitolul 1. Primele noțiuni **21**
1.1. Câte ceva despre programe 21
1.2. Ce sunt variabilele? 23
1.2.1. Care este mecanismul? 24
1.2.2. Definirea variabilelor în Python 25
1.2.3. Hei!.. 2 + 3 nu fac 23!?! 28
1.3. Vocabularul limbajului 29
1.3.1. Setul de caractere 29
1.3.2. Identificatori 30
1.3.3. Separatori 30
1.3.4. Comentarii 31

1.4. Tipuri de date 31
1.4.1. Tipuri de date numerice 32
1.4.2. Șiruri de caractere 34
1.4.3. Date numerice vs. șiruri de caractere 35
1.5. Mai multe moduri de atribuire 36
1.6. Funcția **type** 37
1.7. Operații aritmetice elementare introductive 37
1.8. Ce este un algoritm? 41
Probleme propuse / exerciții 43

Capitolul 2. Expresii **46**
2.1. Introducere 46
2.2. Operatori în Python 49
2.2.1. Operatori aritmetici 50
2.2.2. Operatori relaționali 53
2.2.3. Operatori logici 55
2.2.4. Valori booleene 57
2.2.5. Operatori de atribuire 59
2.3. Erori frecvente 60
2.4. Câteva funcții utile (*built-in*) 62
2.4.1. Funcția **abs**() 62
2.4.2. Funcția **eval**() 63
2.4.3. Funcția **id**() 64
2.4.4. Funcția **len**() 64
2.4.5. Funcția **round**() 65
2.5. Mai mult despre **print**() 66
Probleme rezolvate 67
Probleme propuse / exerciții 69

Capitolul 3. Fără OOP nu putem continua... **71**
Înțelegeți la nivel de bază conceptele de *încapsulare*, *clasă*, *obiect*, *instanțiere*, *date* și *metode membru* – ***3 pagini esențiale!***

Capitolul 4. Șiruri de caractere **74**
4.1. Introducere 74
4.2. Operatorii + (*concatenarea*) și * (*repetiția*) 75

4.3. Accesul la caracterele șirului 76
4.4. Lungimea unui șir de caractere 78
4.5. Subșiruri (feliere, în engleză *slicing*) 78
4.6. Apartenența – **in** și **not in** 80
4.7. Funcțiile *built-in* **min**() și **max**() 80
4.8. O parte dintre metodele clasei **str** 81
4.9. Compararea șirurilor 85
4.10. Formatarea șirurilor 86
Probleme rezolvate/propuse 89

Capitolul 5. Colecții de date **92**
5.1. Liste 92
5.1.1. Ce este o listă? 92
5.1.2. Accesul la elemente 93
5.1.3. Modificarea elementelor 94
5.1.4. Ștergerea elementelor 94
5.1.5. Adăugarea elementelor noi 95
5.1.6. Extinderea unei liste 97
5.1.7. Operatorii +, *, in și not in 97
5.1.8. Alte metode și funcții built-in utile 98
5.1.9. Exerciții propuse 100
5.2. Tupluri 101
5.2.1. Ce este un tuplu? 101
5.2.2. Accesul la elemente 102
5.2.3. Lungimea unui tuplu 102
5.2.4. Operatorul in și grupul not in 102
5.2.5. Constructorul clasei tuple 103
5.2.6. Ștergerea unui tuplu 104
5.2.7. Operatorii + și * 104
5.2.8. Metodele clasei tuple 105
5.2.9. Modificare unui tuplu – un truc interesant! 106
5.3. Seturi 107
5.3.1. Ce este un set de date? 107
5.3.2. Adăugarea de noi elemente 108
5.3.3. Ștergerea elementelor 109
5.3.4. Operații cu mulțimi 110
5.3.5. Clasa **frozenset** 111
5.4. Dicționare 112
5.4.1. Ce este un dicționar? 112

5.4.2. Clasa **dict** 113
5.4.3. Metoda **get()** 114
5.4.4. Modificarea și adăugarea perechilor de date 115
5.4.5. Din nou despre operatorii in și not in 115
5.4.6. Revenim la **frozenset**... 116

Capitolul 6. Instrucțiunile limbajului Python **117**
6.1. Instrucțiunea condițională **If** 117
6.1.1. Forma **if .. else** 117
6.1.2. Forma **if .. elif .. else** 119
6.1.3. Instrucțiunea compusă. ***Indentarea!*** 120
6.1.4. Probleme rezolvate 122
6.1.5. **Case/switch** nu există... 127
6.1.6. Exerciții propuse 129
6.2. Instrucțiunea repetitivă **for** 130
6.2.1. Forma generală 130
6.2.2. Funcția **range()**. Forma clasică pentru **for** 133
6.2.3. Probleme rezolvate 134
6.2.4. Citirea colecțiilor de date de la tastatură 139
6.2.5. Mai mult despre *user-friendly* 140
6.2.6. Exerciții propuse
6.3. Instrucțiunea repetitivă **while** 143
6.3.1. Probleme rezolvate 144
6.3.2. Exerciții propuse 148
6.4. Clauzele **break** și **continue** 149
6.3.1. Clauza **break** 149
6.3.2. Clauza **continue** 149

Capitolul 7. La voia întâmplării... **150**
7.1. Numere aleatoare 150
7.2. Cum le putem genera în Python? 150
7.2.1. Importarea primului nostru modul – **random** 150
7.2.2. Funcția **randint** 152
7.2.3. Funcția **choice** 153
7.2.4. Funcția **shuffle** 154
7.2.5. Funcția **seed** 154
7.3. Primele jocuri în Python 156
7.3.1. Ghicește numărul! 156

7.3.2. Tabla înmulțirii – crearea unui test de evaluare 158
7.3.3. Spânzurătoarea - țări din Uniunea Europeană 162

Capitolul 8. Funcții și module **169**
8.1. Introducere 169
8.2. Funcții 170
8.2.1. Crearea unei funcții 170
8.2.2. Să formăm un prim modul! 173
8.2.3. Mai mulți parametri formali 176
8.2.4. Valori implicite pentru parametri 178
8.2.5. Funcții anonime – **lambda** 179
8.2.6. Probleme rezolvate 180
8.2.7. Probleme propuse 185
8.3. Vizibilitatea variabilelor 186
8.3.1. Variabile locale și globale 186
8.3.2. Variabile nelocale 189
8.4. Transmiterea parametrilor 189
8.4.1. Introducere 189
8.4.2. Mai mult de spre variabile (mutabile / imutabile) 190
8.4.3. Stai! Cum copiez de fapt două variabile mutabile? 192
8.4.4. Revenim la transmiterea parametrilor 194
8.5. Poziția și ordinea scrierii funcțiilor 196
8.5.1. Ce înseamnă de fapt un limbaj interpretat? 196
8.5.2. Unde putem defini o funcție? 197
8.5.3. Mai multe funcții 197
8.5.4. Concluzii 198
8.6. Universul modulelor 199
8.6.1. **Python Standard Library** 199
8.6.2. Modulul **math** – fiți autodidacți! 200

Capitolul 9. I/O. Fișiere text **201**
9.1. Introducere 201
9.1.1. Modelul **black-box** 201
9.1.2. Fișiere text 201
9.2. Deschiderea și închiderea unui fișierelor text 202
9.2.1. Un prim exemplu de citire 202
9.2.2. Întotdeauna închidem fișierul deschis... 203
9.2.3. Funcția **open()** 204

9.3. Citirea fișierelor text 205
9.3.1. Metoda **read()** 205
9.3.2. Conversia explicită spre o listă. Metoda **readlines()** 207
9.3.3. Metoda **readline()** 208
9.4. Scrierea fișierelor text 209
9.5. Cum ștergem un fișier? 212

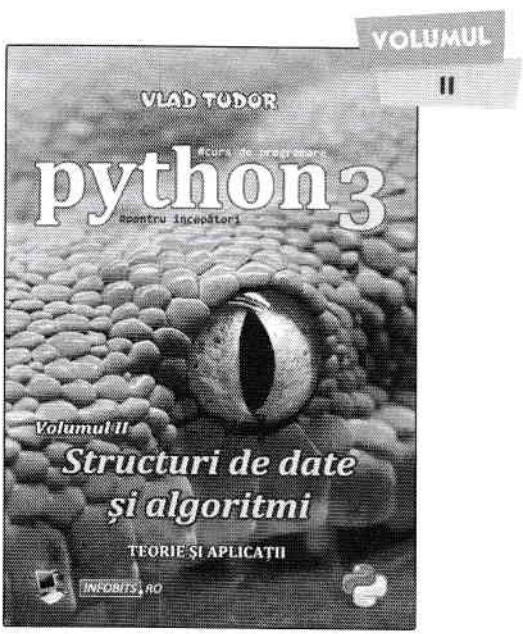
Anexa 1. Diferențe dintre versiunile 2 și 3 213

Anexa 2. Tabela codurilor ASCII 214

Bibliografie 215

Vă recomandăm și volumul al II-lea

(disponibil în format letric și digital)



Capitolul 1
Primele noțiuni

1.1. Câte ceva despre programe

Bun... Avem totul pregătit, deci să trecem la treabă! :)

Cel mai simplu program afișează ori un text, mai exact *un șir de caractere*, ori rezultatul unei *operații aritmetice*. De exemplu, executând

```
print("Primul meu program!")
print(7*3+2-10/2)
```

obținem imediat:

```
========== RESTART: C:\Users\Vlad\Desktop\primul.py
Primul meu program!
18.0
>>>
```

O primă observație ar fi că putem să delimităm, *fără a fi însă necesar*, fiecare instrucțiune cu semnul ";", ca mai jos:

```
print("Primul meu program!");
print(7*3+2-10/2);
```

Rezultatul este același. *Testați!*

De asemenea, puteți scrie cele două instrucțiuni pe aceeași linie:

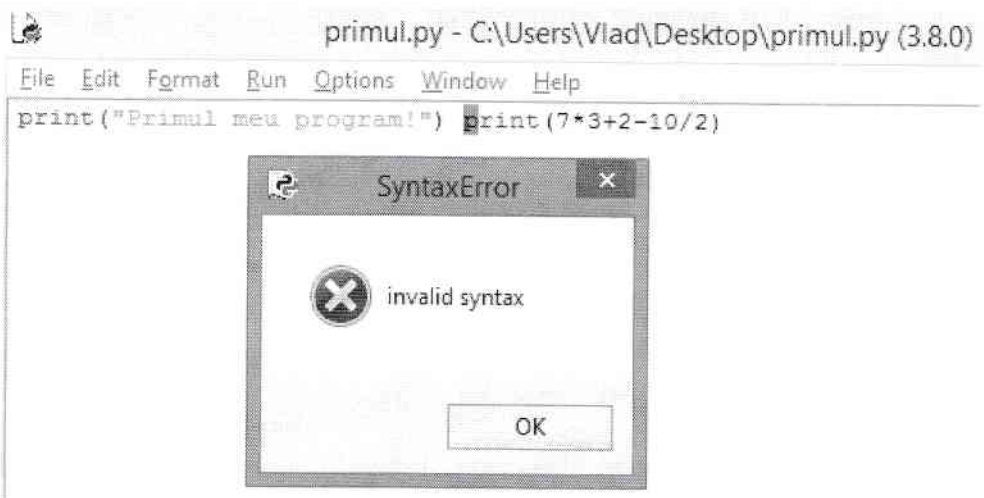
```
print("Primul meu program!"); print(7*3+2-10/2);
```

Totuși, limbajul Python este unul orientat puternic spre o redactare care să permită înțelegerea ușoară a codului altui programator căruia îi parvine programul, așadar ***este de preferat ca pe fiecare linie să fie scrisă o singură instrucțiune***.

Atenție, dacă scriem

```
print("Primul meu program!") print(7*3+2-10/2)
```

vom întâmpina *prima eroare*:



Pe aceeași linie nu putem scrie două instrucțiuni fără a le separa prin ";", iar interpretorul Python ne anunță imediat!

Mai mult, Python face diferența dintre literele mari (*majuscule*) și cele mici. Testând codul de mai jos:

```
priNt("Primul meu program!")
```

obținem:

```
========== RESTART: C:\Users\Vlad\Desktop\primul.py ==
Traceback (most recent call last):
  File "C:\Users\Vlad\Desktop\primul.py", line 1, in <module>
    priNt("Primul meu program!")
NameError: name 'priNt' is not defined
>>>
```

deci mediul de programare nu recunoaște instrucțiunea "**priNt**", deoarece se consideră ca fiind încă nedefinită.

Definiția 1.1. *Sintaxa limbajului este dată de totalitatea regulilor de scriere corectă* (în sensul acceptării sale de programul traducător (interpretor în cazul Python), care are rolul de a îl executa. Dar un program corect din punct de vedere sintactic nu este automat un program bun, iar corectitudinea sintactică este numai o cerință a programelor, așa cum vom vedea ulterior.

Definiția 1.2. Prin **semantica** unui limbaj se înțelege semnificația construcțiilor sintactice corecte (ce anume realizează instrucțiunile, etc).

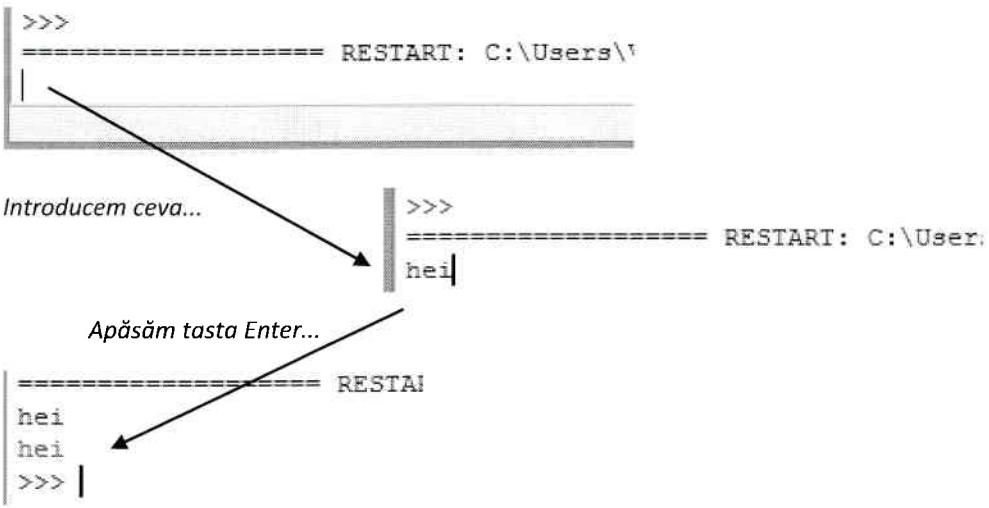
Cel mai dificil este ca programul să execute întocmai ceea ce și-a propus cel care l-a realizat, iar a verifica corectitudinea nu este deloc un lucru simplu.

1.2. Ce sunt variabilele?

Exemplul 1.1. E mai simplu să analizăm inițial programul de mai jos:

```
x = input()
print(x)
```

Acest program citește de la tastatură o dată de intrare (un șir de caractere) prin intermediul funcției **input**, apoi o afișează cu ajutorul lui **print**:



Ce observăm? Apare în program *variabila* **x** care reține data introdusă de la tastatură (șirul de caractere "**hei**") la rularea programului.

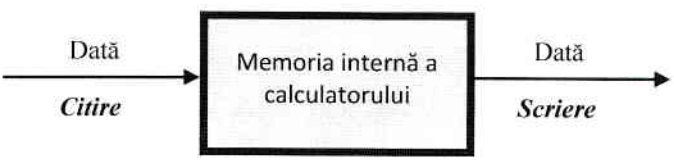
1.2.1. Care este mecanismul prin care se realizează acest lucru?

Calculatorul preia data citită (în acest caz, "**hei**") şi o depune / reține în memoria internă. De acolo, aceasta este preluată şi afişată în cadrul consolei cu ajutorul instrucțiunii **print**.

Definiția 1.3. Prin **citire** se înțelege operaţia prin care calculatorul preia o dată din exterior şi o depune în memoria internă. Pentru citire, mediul exterior va fi tastatura prin intermediul instrucțiunii **input** (vom vedea că există şi alte dispozitive / medii de pe care se poate citi).

Definiția 1.4. Prin **scriere** se înțelege operaţia prin care calculatorul preia o dată din memoria internă şi o depune în exterior. Pentru scriere, mediul exterior va fi pentru moment monitorul / consola (ca şi la citire, vom vedea că există şi alte dispozitive pe care se poate scrie).

Schema de mai jos sintetizează cele prezentate:



A spune că o dată este depusă în memoria internă este mult prea puțin. Memoria internă conține o mulțime de alte informații. Trebuie să ştim **locul** (**adresa**) în care aceasta este depusă. Pentru a putea realiza aceasta, se foloseşte un procedeu specific. În ce constă el?

Pe scurt, se declară o **variabilă** (în programul nostru se numeşte **x**), iar aceasta are rolul de a reține cele introduse de utilizator. În exemplul nostru, am introdus numele variabilei, apoi semnul egal și valoarea cu care *am inițializat-o* (ceea ce a introdus utilizatorul cu ajutorul funcției **input**, mai exact: "**hei**").

În esență, să ne imaginăm variabila **x** ca și o "*cutiuță*" aflată în memoria internă a calculatorului nostru, unde reținem "**hei**".

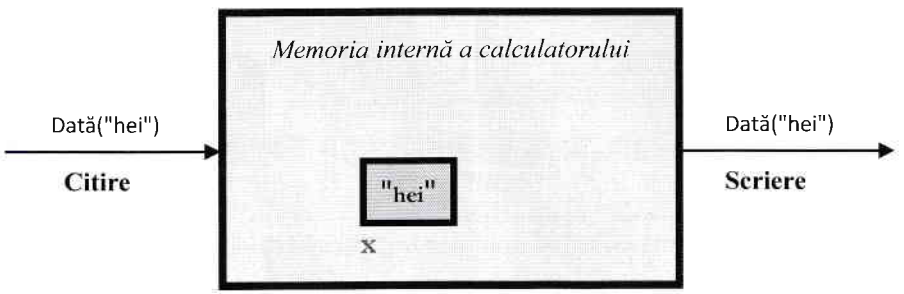
Reprezentarea intuitivă ar fi următoarea:

x

Dacă am fi citit "**wow**", am fi avut:

x

Revenind, dacă ţinem cont că variabila **x** se găseşte în memoria internă, **sch**ema devine:



Să observăm că în cazul citirii lui "**wow**", variabila **x** reţine "**wow**", iar în cazul citirii textului "**hei**", variabila **x** reţine "**hei**". Ne-am imaginat variabila ca pe o cutiuţă pe care, în cazul nostru, am numit-o **x**, ce reţine la un anumit moment o valoare.

Tocmai de aici provine numele de **variabilă** - deşi are un unic nume **x**, reţine *diverse date*, deci are un **conţinut variabil**.

1.2.2. Definirea variabilelor în Python

... este extrem de simplă. Alegem un nume potrivit, apoi scriem semnul egal și introducem valoarea cu care o inițializăm.

Testați codul de mai jos:

```
n = 123
print(n)
```

Avantajul utilizării limbajului Python este faptul că **orice variabilă poate reține dinamic, pe parcursul unui program, valori de tipuri diferite**, fără a impune tipul la declarare, precum *inițializarea* în C++.

Continuăm cu un alt caz...

```
n = 123
print(n)
n = "hei"
print(n)
n = 48
print(n)
```

```
========== RESTART: C:\
123
hei
48
>>>
```

Rezultatul este

Ce observăm? Inițial variabila **n** a reținut valoarea numerică **123**, apoi un șir de caractere "**hei**", iar la final, din nou o valoare, **48**. După fiecare atribuire am tipărit ceea ce reține variabila **n**.

Exemplul 1.2. Să analizăm programul de mai jos:

```
y = input('Numele tau este: ')
print('Salut, ' + y)
```

Acest program citește de la tastatură un șir de caractere:

```
>>>
========== RESTART: C:\Use
Numele tau este:
```

Introducem numele...

```
>>>
========== RE
Numele tau este: Vlad
```

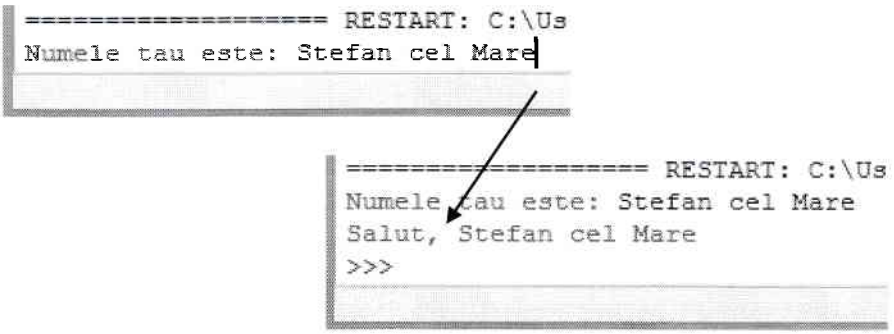
Apăsăm tasta Enter...

apoi se afișează șirul de caractere format din "**Salut,** " și numele introdus:

```
>>>
========== RESTART: C:\U
Numele tau este: Vlad
Salut, Vlad
>>>
```

Observații

1. Un șir de caractere poate fi format din mai multe cuvinte:



2. În exemplul al doilea:

   - funcția **input** are între paranteze un text care este afișat atunci când trebuie să introducem datele de la tastatură; acesta apare în linia de comandă / consolă și ne oferă *un indiciu* asupra a ceea ce trebuie să scriem (**"Numele tau este: "**);

   - am introdus special un spațiu după caracterul ":" mai sus a.î. la rularea programului să respectăm spațierea convențională (implicit, cursorul rămâne imediat după textul ales ca *indiciu*);

   - în cadrul instrucțiunii (*funcției*) **print** am folosit operatorul "+" care *concatenează* (alătură) cele două șiruri de caractere:

     "Salut, " + "Vlad" ⟶ **"Salut, Vlad"**

     (*veți învăța detaliat aceste noțiuni puțin mai târziu*)

Exemplul 1.3. Fără alte comentarii, analizați și rulați programul:

```
print("Salutare, draga vizitator!")
print("-------------------------")
x = input('Numele tau este: ')
y = input('Prenumele tau este: ')
print("-------------------------")
print('Bine ai venit, ' + x + " " + y + "!")
```

În cazul meu, am obținut în consolă ca mai jos:

```
========== RESTART: C:\Users\Vlad\Deskto
Salutare, draga vizitator!
--------------------------
Numele tau este: Tudor
Prenumele tau este: Vlad
--------------------------
Bine ai venit, Tudor Vlad!
>>>
```

Super, nu?

Observăm un alt aspect important.
Când funcția **input** este executată, programul se oprește și așteaptă date de la utilizator / tastatură, deci până la apăsarea tastei *Enter*, totul este în așteptare.

1.2.3. Hei!.. 2 + 3 nu fac 23!?!

Da, aparent pare ciudat. Să presupunem că avem următorul program:

```
x = input("x=")
y = input("y=")
print(x+y)
```

```
========== RESTART: C:\Users\Vlad\Desk
x=2
y=3
23
>>>
```

Răspunsul este simplu. Funcția **input** oferă implicit ca **șir de caractere orice informație introdusă**, iar operatorul "**+**" pur și simplu a alăturat cele două caractere și le-a afișat în consolă, deci **tipul de date reținut este esențial. Pentru a le folosi corect, datele trebuie convertite după caz**!

Veți învăța în curând tipurile de date și modul de a le converti corespunzător.