

SEDINTA 2 - BUCLE LISTE DICTIONARE

Comanda print si diferente Python2.x si Python 3.x

În cele de mai jos putem vedea că putem folosi comanda print urmată de spațiu și un șir de caractere sau comanda print urmată de paranteze rotunde ce cuprind un șir de caractere. Până în prezent foloseam comanda print fără paranteze de formă:

```
>>> print "salut"
salut
>>> print ("salut")
salut
>>>
```

Fig. 1

Acum putem vedea că și comanda de afișare poate fi apelată cu paranteze. De fapt, aici ar fi una din schimbările majore și incompatibile între versiunea Python 2.7.x și Python 3.x și anume Python 2.7.x recunoaște ambele moduri de a afișa un șir de caractere, pe când Python 3.x nu recunoaște decât print cu paranteze rotunde, adică print('string').

Prin urmare, tot ce a fost scris până în prezent pentru 2.x acum trebuie portat la noua formă pentru 3.x. Portarea este un procedeu de traducere de la o formă la o altă prin schimbarea codului pentru a fi compatibil cu ambele versiuni sau cu noua versiune. Există programe python care fac traducerea Python 2.x în Python 3.x automat, deci nu trebuie să realizați voi această traducere.

Nu se recomandă utilizarea funcției print() pentru afișarea unor caractere. În Python 2.7 utilizarea comenzii print cu paranteze nu se poate folosi corespunzător când vrem să afișăm și variabile și șiruri de caractere despartite de virgulă. Prin urmare va fi folosit cât de rar este cu puțință. Figura de mai jos indică acest lucru, unde se poate vedea că se afișează parantezele în cazul în care printăm (x,"Salut").

```
>>> print "salut"
salut
>>> print ("salut")
salut
>>> x=23
>>> print (x, "Salut")
(23, 'Salut')
>>> print x, "Salut"
23 Salut
```

Fig. 2

Lucrul cu operatori decizionali și IF

Așa cum am demonstrat putem verifica dacă un șir de caractere este format numai din numere și putem converti un număr. Totuși ne trebuie un operator decizional, un lipici care să lege o condiție (spre exemplu: este șirul de caractere format numai din numere?) de expresii (acțiunea pe care dorim să o luăm).

Acest lipici este <if>. Operatorul decizional <if> ia o decizie pe baza unei condiții scrise în paranteze. Iată cum arată forma structurală a unui if:

```
if (conditie):
    expresie1
```

Fig. 3

Să presupunem că avem expresia if de mai jos:

```
if (conditie):
    expresie1
```

unde expresie1 este `print "expresie1"` sau oricare sintaxă.

If se traduce în română cu 'dacă', iar else poate traduce cu 'altfel':

Prin urmare expresia de mai sus se poate scrie mot-a-mot:

```
daca(conditie):
    expresie1
```

expresie1 va fi rulat numai cand conditia este adevarata.

Fig. 4

Sa presupunem ca am înlocuit condiția cu un True ca în sintaxa de mai jos. În acest caz tot mereu va rula **expresie1**:

if (True):

```
    expresie1
```

Putem echivala acest caz cu următoarea sintaxa:

```
expresie1
```

Avem nevoie de o metoda prin care indicam că, cand condiția este adevărată (True - boolean) , sintaxa if ar trebui să ruleze mai multe expresii.

Sa ne uitam un pic fragmentul de cod de mai jos:

if (False):

```
    print "x este mai mare ca 3"
```

```
print "Asa este!"
```

```
print "Test"
```

Cum se va comporta interpretorul? Condiția din cadrul if-ului este falsa astfel expresia **print "x este mai mare ca 3."** nu va fi rulata. În schimb expresia **print "Asa este!"** va fi rulata, ca de altfel și expresia **print "Test"**. Dorim ca și a doua expresie să fie introdusa în if sub un singur bloc. Acest lucru este posibil folosind spațierea. Aici am folosit patru spatii.

if (True):

```
    print "x este mai mare ca 3"
```

```
    print "Asa este!"
```

```
print "Test"
```

Fragmentul de cod de mai sus va rula ambele expresii de print doar în cazul în care conditia este adevărata. Deoarece conditia este adevărata va avea ca urmare rularea celor doua expresii **print "x este mai mare ca 3."** si **print "Asa este!"**. Sintaxa **print "Test"** va rula oricum deoarece nu face parte din blocul de expresii ale if-ului.

Atât IDLE, cat și Eclipse ne oferă indentare automata la scrierea codului, detectând începerea unui bloc de expresii. Începerea blocului de comenzi se face prin indentarea

unui tab, tab ce e configurat standard la 4 spatii. Prin trecerea la nivelul anterior de număr de taburi se închide un bloc de expresii .In loc de tab se pot folosi și spatii, dar e mai facil să folosim un tab fiind mult mai vizibilă trecerea de la un bloc la altul.

Regasim un alt exemplu in python in ceea ce priveste spațierea (numita si indentare).

```
>>> if (True) :
    #bloc de expresii
    #Blocul tine cat timp am aceasi indentare
    #daca apas backspace pe un rand nou voi termina indentarea
    print "salut"
#aici am terminat indentarea

salut
>>>
```

Fig. 5

Sa presupunem ca avem următoarea expresie if:

```
if (conditie):
    expresie1
else:
    expresie2
```

In cazul în care condiție este adevărată va fi rulată expresie1 , în caz contrar (condiția este falsă) expresie2 va fi rulată.

Iata si un exemplu:

```
>>> if (True) :
    print "Adevarat"
else:
    print "Fals"

Adevarat
>>> if (False) :
    print "Adevarat"
else:
    print "Fals"

Fals
>>>
```

Fig. 6

Condițiile pot fi numerice, Boolean sau compararea șirurilor de caractere.

Sa discutăm despre aceste condiții pe rând:

1. Conditii Numerice

Condițiile numerice folosesc operatori decizionali. Retine: in cazul in care conditia este adevarata, atunci va rula codul de sub if. Mai jos se poate observa un tabel de operatori decizionali:

Tabel 1 : OPERATORI DECIZIONALI

Operator	Însemnătate	Exemplu de condiție	Ce returnează
<code>==</code>	Egal cu	<code>5 == 5</code>	True
<code>!=</code>	Nu este egal cu	<code>8 != 5</code>	True
<code>></code>	Mai mare	<code>3 > 10</code>	False
<code><</code>	Mai mic	<code>5 < 8</code>	True
<code>>=</code>	Mai mare sau egal cu	<code>5 >= 10</code>	False
<code><=</code>	Mai mic sau egal cu	<code>5 <= 5</code>	True

Iata doua exemple (Fig. 5) cu operatori decizionali if ce folosesc conditii numerice.

```
>>> if (x>24) :
    print "Salut"
    # Aceasta linie nu va fi rulata deoarece 23 nu este
    #mai mare decat 24
```

```
>>> if (x>22) :
    print "Salut"
    # Aceasta linie va fi rulata deoarece 23 este
    #mai mare decat 22
```

Salut

Fig. 7

Asa cum putem vedea in Fig. 7 se verifica daca expresia x mai mare ca 24 este adevarata sau x mare decat 22. Cand conditia este adevarata atunci codul din blocul de

comenzi de sub if va rula. Prin urmare, vom vedea afisat sirul de caractere „Salut” doar cand x este mai mic decat 22.

2. Boolean

Așa cum se poate vedea și în Tabelul 1 condiția este convertita într-o variabila de tip Boolean. Deci tot mereu se folosesc expresii ce au ca rezultat o variabila Boolean.

Mai jos regasim un exemplu unde folosim o conditie de tip boolean (True/False).

```
# Manipularea sir caractere prin if statement
# Demonstreaza utilizarea sintaxei if
# Ion Studentul 1/13/03

print "\tSalut! \nAceasta e un program de conversie euro->Lei"

euro=raw_input("\nScrie te rog valoarea euro:\n")

if (euro.isdigit()):
    euro=int(euro)
    print ("Valoarea convertita este :",euro*4.5, " RON")
else:
    print("Valoarea introdusa nu este un numar!")

raw_input("\n\nApasa <enter> pt a iesi.")
```

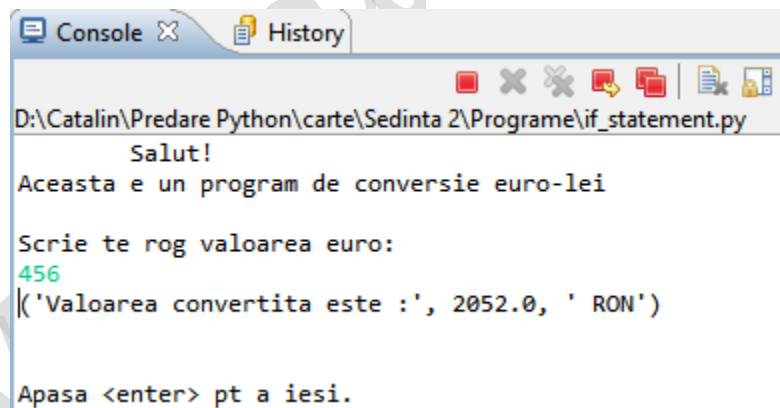


Fig.8

Sa discutăm un pic despre program.

Prima linie ne indică utilizarea functiei print() prin afisarea unui text ce indica scopul programului:

```
print "\tSalut! \nAceasta e un program de conversie euro->Lei"
```

Urmează o linie în care capturăm în variabila `euro` un șir de caractere introdus de la tastatură cu ajutorul funcției `raw_input()`.

În program regăsim o sintaxă de tip `<if>`. Sintaxele de tip `<if>` au obligația de a lua o decizie pe baza condiției din interior. Dacă utilizatorul va scrie un număr atunci `euro.isdigit()` va returna `True`. În acest caz cele două linii ce se regăsesc mai jos de acel `if` vor fi rulate. Linii rulate se regăsesc și mai jos:

```
euro=int(euro)
print ("Valoarea convertita este :",euro*4.5," RON")
```

În cazul în care utilizatorul nu introduce de la tastatură un număr, atunci `euro.isdigit()` va returna `False`. Prin urmare, blocul de expresii de sub `else` va fi rulat. În program avem doar o singură sintaxă care se regăsește sub blocul de `else`, deci ce va fi rulată în cazul în care utilizatorul nu introduce un număr:

```
print("Valoarea introdusa nu este un numar!")
```

3. Șir de caractere

Compararea unor expresii de tip `string` se face cu ajutorul ASCII într-o ordine lexicografică. Astfel într-o expresie comparativă va lua primul caracter de la stânga la dreapta din primul șir de caractere și primul caracter de la stânga la dreapta din cel de al doilea șir de caractere.

Aceste două caractere vor fi convertite în cod ASCII, apoi vor fi comparate numerele ASCII ale codului.

Pentru a obține un număr al caracterului ASCII se folosește funcția `ord()`.

```
>>> ord('a')
97
>>> ord('b')
98
>>> if ("asaMaiMerge"<"b"):
    print "a=>ASCII mai mare ca b=>ASCII"
```

```
a=>ASCII mai mare ca b=>ASCII
```

Fig. 9

În mod uzual condiția va conține doar operatorii decizionali egal cu `'=='` sau nu este egal cu `'!='`. Foarte rar se întâmplă să utilizăm alți operatori decizionali în compararea de șiruri de caractere.

Operația inversă a funcției `ord()` este `chr()`. Funcția `chr()` nu acceptă decât numere de tip `integer` ca parametru vezi Fig. 10.

```
>>> chr('a')

Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    chr('a')
TypeError: an integer is required
>>> chr(97)
'a'
>>> chr(97.0)

Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    chr(97.0)
TypeError: integer argument expected, got float
```

Fig. 10

Așa cum se poate observa în Fig. 10 funcția `ord()` nu accepta nici numere de tip `float`.

În Python nu suntem limitați în a crea un singur `if`. Putem crea o ierarhie de tip `if` în `if` așa cum se poate vedea și mai jos:

```
>>>
>>> a=1;b=2;c=3
>>> if(a==1):
    print "a = 1"
    if (b==2):
        print "b = 2"
    if (c == 4):
        print "c = 4"
    else:
        print "c = ",c

a = 1
b = 2
c = 3
>>>
```

Fig. 11

Avem posibilitatea de a avea mai multe ramuri de `if` într-un `if`, așa cum avem posibilitatea de a utiliza `elif`. Acest `elif` se adaugă cu scopul de a avea încă o ramură ce pune o condiție în vederea rularii blocului de expresii de sub acel `elif`. Operațiile se vor face de sus în jos. Pentru a explica operațiile logice ne raportăm la exemplul de mai jos.

Prin urmare, daca conditia(**conditie1**) din if este adevarata, se va rula blocul de expresii de sub if (**expresie1**). In cazul in care nu este adevarata conditia din if, atunci se va verifica conditia din elif(**conditie2**). In cazul in care **conditie2** este adevarata se va rula blocul de expresii de sub elif (**expresie2**). In cazul in care nici **conditie2** nu este adevarata se ruleaza blocul de expresii de sub else (**expresie3**).

if (conditie1):

expresie1

elif (conditie2):

expresie2

else:

expresie3

Asa cum probabil v-ati asteptat putem avea multiple conditii elif. Toate aceste conditii vor fi verificate pe rand, pana una din aceste conditii este adevarata. Pe ramura unde conditia este adevarata vom rula blocul de expresii de sub ramura respectiva.

```
If (conditie):
    expresie
elif (conditie2):
    expresie2
elif (conditie3):
    expresie3
....
elif (conditie n)
    expresien
else:
    expresie n+1
```

Fig. 12

Atentie totusi la conditii si la logica din spate. Iata un exemplu in care a doua ramura de elif nu va fi accesata niciodata deoarece conditiile respective vor fi indeplinite de ramurile de dinainte. Avem un mic programel in care definim un numar stocat de variabila x. Acest numar este 2.

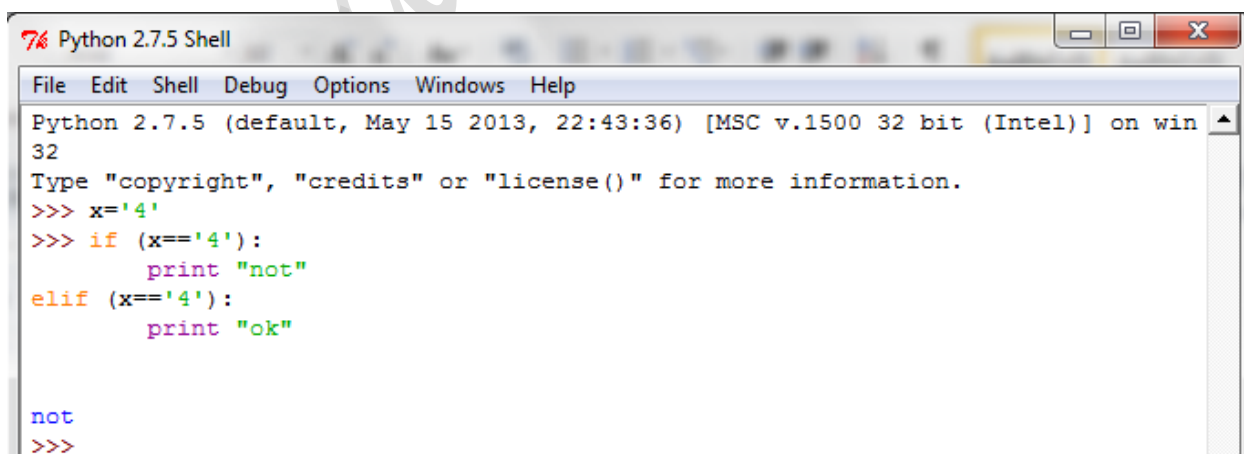
```
>>> if(x>2):
    print "x mai mare ca 2"
elif(x==2):
    print "x este egal cu 2"
elif(x>=2):
    print "Aceasta linie nu va fi afisata niciodata"
else:
    print "test"

x este egal cu 2
>>>
```

Fig. 13

In Fig.13 conditia din al doilea elif ($x \geq 2$) nu va fi verificata niciodata deoarece daca x este mai mare ca 2 va rula blocul de expresii de sub if, iar daca x este 2 va rula blocul de expresii de sub primul elif. In cazul in care numarul stocat de x va fi mai mic ca 2 atunci vom rula blocul de expresii de sub else.

Prin urmare, originalitatea conditiilor este importanta. Daca avem doua conditii identice programul nu ne va returna eroare, dar nu va ajunge niciodata pe ramura cu elif-ul ce se afla mai jos cu conditia 'duplicat'. Retine ca python prelucreaza pe rand fiecare sintaxa. Aceasta idee este intarita si de Fig.14



```
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.
>>> x='4'
>>> if (x=='4'):
    print "not"
elif (x=='4'):
    print "ok"

not
>>>
```

Fig. 14

În primele exemple cu operatori decizionali de tip if nu aveam și un else implementat. Întrebarea este dacă am putea avea și expresii de tip if-elif fără a avea un else la final.

```
If (conditie):
    expresie
elif (conditie2):
    expresie2
```

Fig. 14

Răspunsul este da. Putem avea implementări de operatori decizionali de tip if-elif fără a implementa și ramura else. Rămâne aceeași logică valabilă: toate aceste condiții vor fi verificate pe rând, până una din aceste condiții este adevărată. Pe ramura unde condiția este adevărată vom rula blocul de expresii de sub ramura respectivă. În cazul în care nici una din condițiile testate nu sunt adevărate nu vom aplica nici un bloc de expresii.

Vom trece mai departe la un program care utilizează sintaxa elif într-un cadru mai complex .

```
# Manipularea sir caractere prin if statement si elif
# Demonstreaza utilizarea sintaxei if-elif
# Ion Studentul 1/13/03

print("\tSalut! \nAceasta e un program de conversie Euro->Lei sau Lei->Euro")

alegereMoneda=raw_input("\nScrie L pt. conversia Euro->Lei și E pt. conversia Lei->Euro:\n")
alegereMoneda=alegereMoneda.upper()

if (alegereMoneda.isalpha()):
    if (alegereMoneda=='L'):
        euro=raw_input("\nScrie nr. de euro ce doresti sa-l convertesti:\n")
        if (euro.isdigit()):
            euro=int(euro)
            print "Valoarea convertita este :",euro*4.5," RON"
        else:
            print "Valoarea introdusa nu este un numar!"
    elif (alegereMoneda=="E"):
        lei=raw_input("\nScrie nr. de Lei ce doresti sa-l convertesti:\n")
        if (lei.isdigit()):
            lei=int(lei)
            print "Valoarea convertita este :",lei/4.5," EURO"
        else:
            print "Valoarea introdusa nu este un numar!"
    else:
        print "Valoarea introdusa nu este recunoscuta!"
```

```
else:
    print "Valoarea introdusa nu este un caracater alfa!"

raw_input("\n\nApasa <enter> pt a iesi.")
```

Să vedem care sunt elementele noi aduse de acest program. În primul rând, pentru a elimina eroarea umană care poate să tasteze l în loc de L sau e în loc de E, programatorul trebuie să se asigure ca există o modalitate de conversie a caracterelor primite de la tastatură în caractere mari. Linia care face acesta conversie se regaseste mai jos:

```
alegereMoneda=alegereMoneda.upper()
```

În cazul grupării de tipul if în if, pe ramura de if testăm dacă variabila `alegereMoneda` stochează caractere de tip literă exclusiv. Linia care face această testare este linia ce conține condiția `alegereMoneda.isalpha()` :

```
if (alegereMoneda.isalpha()):
    .....
    .....
else:
    print "Valoarea introdusa nu este recunoscuta!"
```

În cazul în care utilizatorul nu a introdus de la tastatură un șir de caractere de tip literă exclusiv, atunci vom merge pe ramura de else și vom afișa un text în care informăm utilizatorul de necesitatea introducerii unui șir de caractere corespunzător:

```
else:
    print "Valoarea introdusa nu este recunoscuta!"
```

În cazul în care utilizatorul a introdus de la tastatură un șir de caractere de tip literă exclusiv, atunci vom merge pe ramura de if. Acest if conține o expresie de tip if-elif:

```
if (alegereMoneda=='L'):
    .....
elif (alegereMoneda=="E"):
    .....
```

Aici fiecare condiție va fi testată individual.

Dacă șirul de caractere introdus de la tastatură este un șir de caractere ce e format doar din caracterul L atunci vom rula blocul de expresii de sub if.

Dacă șirul de caractere introdus de la tastatură este un șir de caractere ce e format doar din caracterul E atunci vom rula blocul de expresii de sub elif.

In cazul in care sirul de caractere introdus de la tastatura nu indeplineste nici una din conditiile de mai sus (nu este un sir de caractere format exclusiv din caracterul L sau din caracterul E), atunci nu vom rula nimic.

In cazul ramurii if ce testeaza daca sirul de caractere este format exclusiv din caracterul L, putem vedea urmatoarele linii:

```
if (alegereMoneda=='L'):
    euro=raw_input("\nScrie nr. de euro ce doresti sa-l convertesti:\n")
    if (euro.isdigit()):
        euro=int(euro)
        print "Valoarea convertita este :",euro*4.5," RON"
    else:
        print "Valoarea introdusa nu este un numar!"
```

In prima linie din codul de mai sus (cod extras din program) vom captura in variabila euro un text introdus de la tastatura. Intr-o expresie de tip if-else testam cu ajutorul metodei de manipulare a sirurilor de caractere isdigit() daca valoarea capturata de la tastatura in variabila euro este un sir de caractere format exclusiv doar din numere.

In cazul in care variabila euro stocheaza un sir de caractere format exclusiv doar din numere vom converti sirul de caractere intr-un numar de tip integer, captand aceasta valoare convertita in numar in variabila euro. Apoi folosim comanda print pentru a afisa mesajul "Valoarea convertita este :",euro*4.5," RON" ; mesaj in care se vor realiza si calculele matematice inainte de a afisa acest text.

In cazul in care variabila euro nu stocheaza un sir de caractere format exclusiv doar din numere vom afisa sirul de caractere "Valoarea introdusa nu este un numar!"

In cazul ramurii elif ce testeaza daca sirul de caractere este format exclusiv din caracterul E, putem vedea urmatoarele linii:

```
elif (alegereMoneda=="E"):
    lei=raw_input("\nScrie nr. de Lei ce doresti sa-l convertesti:\n")
    if (lei.isdigit()):
        lei=int(lei)
        print "Valoarea convertita este :",lei/4.5," EURO"
    else:
        print "Valoarea introdusa nu este un numar!"
```

In prima linie din codul de mai sus (cod extras din program) vom captura in variabila lei un text introdus de la tastatura. Intr-o expresie de tip if-else testam cu ajutorul metodei de manipulare a sirurilor de caractere isdigit() daca valoarea capturata de la tastatura in variabila euro este un sir de caractere format exclusiv doar din numere.

In cazul in care variabila lei stocheaza un sir de caractere format exclusiv doar din numere vom converti sirul de caractere intr-un numar de tip integer, captand aceasta valoare convertita in numar in variabila lei. Apoi folosim comanda print pentru a afisa

mesajul "Valoarea convertita este :",lei/4.5," EURO" ; mesaj in care se vor realiza si calculele matematice inainte de a afisa acest text.

In cazul in care variabila lei nu stocheaza un sir de caractere format exclusiv doar din numere vom afisa sirul de caractere "Valoarea introdusa nu este un numar!"

Tratarea unei variabile ca o condiție adevărată sau falsă este un alt mod de a scrie sintaxe if. Spre exemplificare va rog urmați Fig. 15 unde se poate vedea ca variabila x este testată ca și condiție. În acest caz variabila x stochează valori numerice. Pentru orice valoare ce este diferită de 0 va rula blocul de sintaxă de sub if deoarece condiția este considerată a fi adevărată. În cazul în care variabila x stochează valoarea 0 atunci va rula blocul de expresii de sub else deoarece condiția este considerată a fi falsă.

```
>>> x=1
>>> if (x):
        print x
else:
        print "else"

1
>>> x=0
>>> if (x):
        print x
else:
        print "else"
```

Fig. 15

```

>>> x="orice"
>>> if (x):
    print x
else:
    print "else"

orice
>>> x="0"
>>> if (x):
    print x
else:
    print "else"

0
>>> x=''
>>> if (x):
    print x
else:
    print "else"

else
>>>

```

Fig. 16

În Fig. 16 putem observa testarea unei variabile de tip sir de caractere ca fiind o conditie.

În cazul în care sirul de caractere stocat de variabila x reprezintă numărul zero sau în cazul în care sirul de caractere stocat de variabila x este un șir ce conține cuvântul orice condiția este tratată ca fiind adevărată. Șirul "0" nu este un șir vid, deci nu este un șir ce ar genera o condiție falsă. Orice sir de caractere ce nu este gol adică orice sir de caractere ce este diferit de sirul de caractere "" va fi tratat ca o condiție adevărată.

Sirul de caractere nul, adică sirul de caractere "" va fi tratat ca o condiție falsă.

În cazul în care avem mai multe condiții de făcut pentru a putea aplica anumite schimbări pentru o anumită variabilă avem la dispoziție trei soluții.

Spre exemplificare vom utiliza cazul în care avem de testat două condiții (`conditie1` și `conditie2`). În cazul în care amândouă sunt adevărate dorim să rulăm două expresii (`expresie1` și `expresie2`).

Prima soluție se referă la folosirea a două if separate cum se poate vedea mai jos:

```
If (conditie1):
```

```
expresie1
```

```
if (conditie2):
```

```
    expresie2
```

Aceasta solutie trebuie adaptata si nu functioneaza pentru toate cazurile deoarece ar putea sa ruleze doar `expresie2` fara ca `expresie1` sa ruleze.

A doua soluție implica folosirea metodei `if` în `if` cum se poate vedea mai jos:

```
If (conditie1):
```

```
    if (conditie2):
```

```
        expresie1
```

```
        expresie2
```

A doua solutie solutie nu trebuie adaptata si functioneaza pentru toate cazurile deoarece nu ar putea sa ruleze doar `expresie2` fara ca `expresie1` sa ruleze.

A treia soluție se refera la folosirea unui singur `if`, dar cu condiții multiple folosind `and`:

```
If (conditie1 and conditie2):
```

```
    expresie1
```

```
    expresie2
```

A treia solutie are acelasi efect ca si a doua solutie. Cuvântul cheie `and` are rolul de a uni doua condiții. Pt. a rula blocul de expresii format din `expresie1` si `expresie2` trebuie ca ambele condiții să fie adevărate .

In cazul în care dorim ca cel puțin una din condiții să fie adevărate pt. a rula blocul de expresii putem folosi cuvântul cheie `or`.

```
If (conditie or conditie2):
```

```
    expresie
```

```
    expresie2
```


Mai jos regăsim două tabele ce acoperă fiecare caz care îl poate avea unirea a două condiții cu **and** sau **or**.

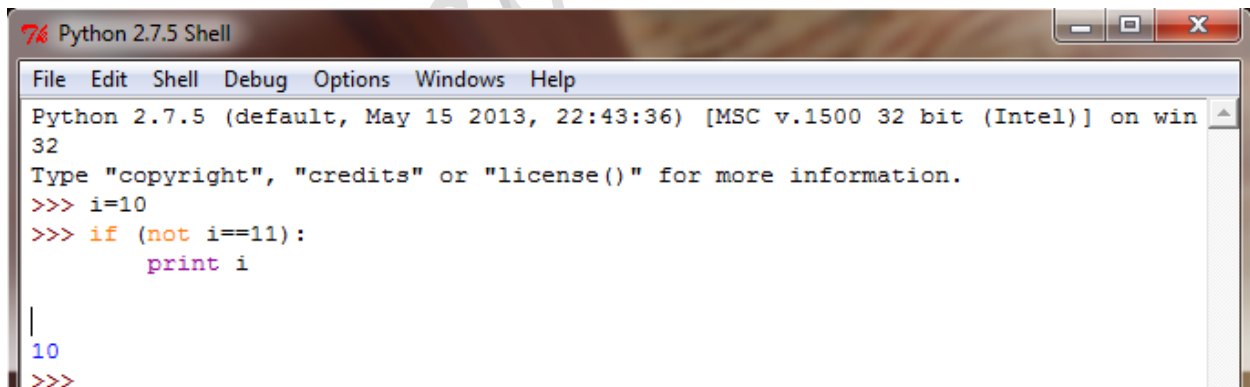
Tabel Operator decizional 'AND'

a == "conditie1"	b == "conditie2"	a == "conditie1" and b == "conditie2"
True	True	True
True	False	False
False	True	False
False	False	False

Tabel Operator decizional 'OR'

a == "conditie1"	b == "conditie2"	a == "conditie1" or b == "conditie2"
True	True	True
True	False	True
False	True	True
False	False	False

Un alt operator decizional des întâlnit este **not**. Acesta este capabil de a nega o expresie. Se presupunem ca avem următorul caz:



```
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.
>>> i=10
>>> if (not i==11):
>>>     print i
|
10
>>>
```

Fig. 17

Not funcționează prin a nega expresia **i==11**. Prin urmare **not i==11** este egal cu **i!=11**. Am completat un tabel cu rezultatele generate de operatorul not:

i==10	not i==10
-------	-----------

True	False
False	True

Iată și un exemplu în care arătăm că cele două condiții `not i==11` și `i!=11` sunt echivalente.

```
>>> x=100
>>> if(x!=10):
        print "Salut"

Salut
>>> if(not x==10):
        print "Salut"

Salut
>>>
```

Fig.18

Lucrul cu bucle de calcul: while și for

Capitolul anterior am putut vedea un program frumos de conversie monetară. Un astfel de program este foarte util și își găsește utilitatea în viață de zi cu zi. Cred că ar fi facil să putem converti mai multe sume. Spre exemplu, dacă suntem înscriși la facultate și trebuie să plătim rate inegale în euro dintr-o sursă de venit în lei, vom avea nevoie să convertim fiecare rată. Presupunând că ar trebui să plătim 3 rate, ar trebui să apelăm la programul de conversie valutară de 3 ori pentru fiecare sumă. Pot exista și alte soluții la această problemă, cum ar fi convertirea întregii sume apoi împărțirea procentuală a ratelor. Dar dacă numărul de operații crește vertiginos spre, să zicem 100, soluțiile pe care le propuneam anterior nu par atât de valide.

Python pune la dispoziție funcția implicită `while` care are rolul de a rula un bloc de sintaxă cât timp condiția este adevărată. Se va opri din rula acele sintaxe când condiția devine falsă.

`while (conditie):`

sintaxa1

sintaxa2

Iată un exemplu mai jos în care definim $x = 10$. Deoarece condiția din `while` este ca x să fie mai mare ca 1 (condiție adevărată), vom rula blocul de sintaxă de sub `while`. În acest caz vom rula două linii. Prima linie va afișa valoarea lui x , iar cea de-a doua linie va decrementa (scade cu 1) valoarea lui x .

```
>>> x = 10
>>> while (x>1):
        print "x este",x
        x = x-1

x este 10
x este 9
x este 8
x este 7
x este 6
x este 5
x este 4
x este 3
x este 2
>>>
```

Fig.19

Astfel după multiple rări succesive valoarea stocată de variabila x nu va mai fi mai mare ca 1, ci va fi egală cu 1. Din această cauză condiția va deveni falsă și interpretorul se oprește din buclarea celor două linii.

O buclă infinită este atunci când condiția este mereu îndeplinită. Mare atenție la buclele infinite. Spre exemplu dacă omiteam în exemplu de mai sus să decrementăm valoarea lui x condiția este respectată mereu, deci creăm o buclă infinită.

Putem să creăm o buclă infinită și dacă condiția este înlocuită cu variabilă boolean `True`. Și în acest caz sintaxele de sub `while` vor rula la infinit.

Pentru a opri o buclă infinită avem nevoie de comanda specială ce poartă numele de `break` ce are rolul de a implementa o terminare forțată a procesului de rulare. Programul de mai jos utilizează cuvintele cheie `break` și `pass`, explicate în cele ce urmează, și folosește o condiție de tip `True`, dar care nu poate intra într-o buclă infinită.

```
# Manipularea sir caractere prin if statement și repetarea prin while
# Demonstrează utilizarea sintaxei while
# Ion Studentul 1/13/03

print("\tSalut! \nAceasta e un program de conversie euro-Lei")

while (True):
    euro=raw_input("\nScrie te rog valoarea euro:\n")

    if (euro.isdigit()):
        euro=int(euro)
        print "Valoarea convertita este :",euro*4.5," RON"
    else:
        print"Valoarea introdusa nu este un numar!"

    quit=raw_input("\n\nApasa q pt a iesi și orice pt. a repeta conversia.\n")
    if (quit.upper()=='Q'):
        break
    else:
        pass
```

O posibilă rulare a programului se poate găsi în Fig.20.

```

<terminated> D:\Catalin\Predare Python\carte\cap 3\while.py
    Salut!
    Aceasta e un program de conversie euro-lei

    Scrie te rog valoarea euro:
    33
    Valoarea convertita este : 148.5  RON

    Apasa q pt a iesi si orice pt. a repeta conversia.

    Scrie te rog valoarea euro:
    22
    Valoarea convertita este : 99.0  RON

    Apasa q pt a iesi si orice pt. a repeta conversia.
    87

    Scrie te rog valoarea euro:
    4
    Valoarea convertita este : 18.0  RON

    Apasa q pt a iesi si orice pt. a repeta conversia.
    q
  
```

Fig. 20

Să privim un pic codul programului de mai sus cautand doar elementele noi introduse. Prima linie conține o linie ce oferă utilizatorului informații despre program. Această linie este (extrasa din program):

```
print("\tSalut! \nAceasta e un program de conversie euro-Lei")
```

A doua linie introduce o sintaxă de tip while:

```
while (True):
    bloc de comenzi
```

Această sintaxă de tip while in care conditia este valoarea boolean True nu este recomandată pentru a fi utilizată datorită posibilitatii crearii unei bucle infinite. Cu toate

acestea foarte mulți programatori o utilizează când nu găsesc o altă condiție ce poate fi utilizată în repetarea blocului de sintaxe sau când complexitatea programului este atât de mare încât nu pot determina toate condițiile în care ar trebui să se oprească buclarea.

Prin urmare această condiție spune interpretorului că blocul de sintaxe trebuie repetat la infinit.

În cadrul blocului de expresii din `while` avem următoarele linii:

```
euro=raw_input("\nScrie te rog valoarea euro:\n")

if (euro.isdigit()):
    euro=int(euro)
    print "Valoarea convertita este :",euro*4.5," RON"
else:
    print"Valoarea introdusa nu este un numar!"
```

Prima linie din codul extras din program capturează în variabila `euro` un șir de caractere introdus de la tastatură cu ajutorul funcției `raw_input()`.

În program regăsim o sintaxă de tip `<if>`. Sintaxele de tip `<if>` au obligația de a lua o decizie pe baza condiției din interior. Dacă utilizatorul va scrie un număr atunci `euro.isdigit()` va returna `True`. În acest caz cele două linii ce se regăsesc mai jos de acel `if` vor fi rulate. Linii rulate se regăsesc și mai jos:

```
euro=int(euro)
print ("Valoarea convertita este :",euro*4.5," RON")
```

În cazul în care utilizatorul nu introduce de la tastatură un număr, atunci `euro.isdigit()` va returna `False`. Prin urmare, blocul de expresii de sub `else` va fi rulat. În program avem doar o singură sintaxă care se regăsește sub blocul de `else`, deci ce va fi rulat în cazul în care utilizatorul nu introduce un număr:

```
print("Valoarea introdusa nu este un numar!")
```

Dar ce comandă face ca blocul de sintaxă să se oprească? Să studiem un pic următorul fragment extras din program. Acesta se regăsește sub blocul de expresii de sub `while`:

```
if (quit.upper()=='Q'):
    break
else:
    pass
```

Condiția aplicată este următoarea: rezultatul operației `quit.upper()` să fie egal cu `'Q'`.

`quit.upper()` are rolul de a face toate caracterele șirului litere mari. Apoi se compară acel șir cu șirul `"Q"`. În cazul în care sunt egale se aplică comanda `break`. Comanda `break` are rolul de a întrerupe o buclă în momentul apelării. Deci bucla infinită poate fi oprită

dacă tastam q sau Q deoarece apelăm `break`. Totuși aveți grijă cu utilizarea `break` deoarece poate genera erori când nu există o buclă care poate fi întreruptă.

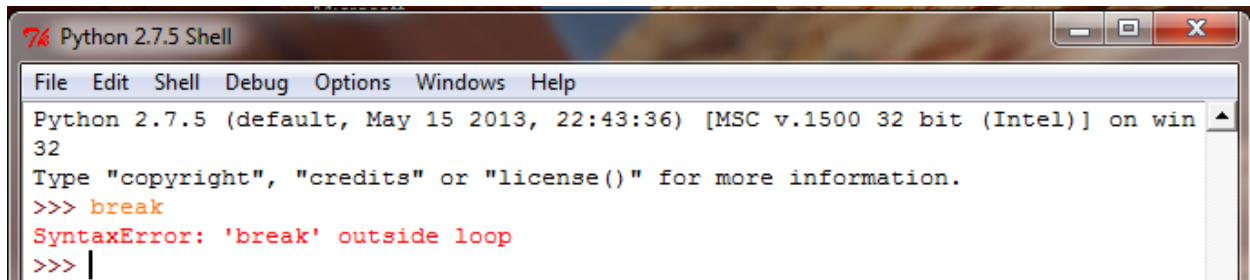


Fig. 21

Cum rămâne cu ramura de `else` ? În cazul în care șirul de caractere introdus de la tastatură nu este q sau Q rezultatul operației `quit.upper` nu va fi 'Q'. Astfel va fi apelată sintaxa `pass`. Aceasta are rolul de a nu face nimic!!! Este o sintaxă care poate fi utilizată când ești obligat să scrii cel puțin o sintaxă altfel ar da eroare. Spre exemplu un `if` fără un block de sintaxă va genera o eroare. În acest caz este inutilă această expresie deoarece am putea să ne lipsim de `else` nefiind obligatoriu. Programul poate atunci să fie sub formă de mai jos și să aibă același rezultat:

```
print("\tSalut! \nAceasta e un program de conversie euro-Lei")

while (True):
    euro=raw_input("\nScrie te rog valoarea euro:\n")

    if (euro.isdigit()):
        euro=int(euro)
        print "Valoarea convertita este :",euro*4.5," RON"
    else:
        print"Valoarea introdusa nu este un numar!"

    quit=raw_input("\n\nApasa q pt a iesi și orice pt. a repeta conversia.\n")
    if (quit.upper()=='Q'):
        break
```

Iată un alt exemplu de utilizare a `pass`. Acest exemplu a fost explicat anterior. Putem observa că cuvântul cheie `pass` nu schimbă cu nimic comportamentul anterior al sintaxei `while`.

```
>>> x = 10
>>> while (x>1):
    pass
    print "x este",x
    x = x-1
```

```
x este 10
x este 9
x este 8
x este 7
x este 6
x este 5
x este 4
x este 3
x este 2
>>>
```

Fig. 22

Tot în categoria sintaxelor cheie ce pot fi folosite în cadrul unei bucle este și comanda **continue**. Comanda **continue** are rolul de a face un salt la rularea respectiva fara a intrerupe bucla. Prin urmare va intrerupe rularea blocului de sintaxe pentru rularea dată si se va incepe rularea de la început a blocului de expresii de sub while. Iată un fragment de program care demonstrează diferența dintre **continue** și **pass** :

```
>>> x = 10
>>> while (x>1):
        print "x este",x
        continue
        x = x-1
```

[illegible]

Fig. 23

Așa cum se poate vedea în Fig. 22 sintaxa `pass` nu are nici un efect asupra rularii.

Așa cum se poate vedea în Fig. 23 sintaxa `continue` va continua cu următoarea buclare a blocului de expresii de sub `while` (va reincepe rularea a blocului de sintaxe de sub `while`). Deci nu va ajunge la sintaxa de decrementare a variabilei `x`. Prin urmare va crea o buclă infinită.

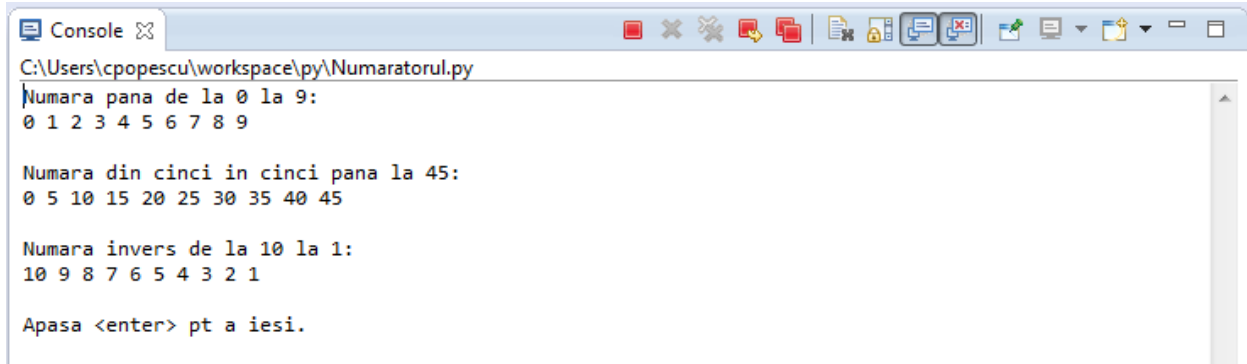
În continuare vom discuta despre o altă metodă de a repeta un bloc de sintaxe. Acest tip de apelare a unui bloc oferă posibilitatea de a rula pentru fiecare element dat și poartă numele de `for`.

Structura `for` este :

```
for variabila_temporara in range():  
    bloc de expresii ce utilizeaza sau nu variabila_temporara
```

Valorile generate de `range()` sunt stocate pe rând de `variabila_temporara` rulând pe rând blocul de expresii cu fiecare valoare din `range`.

```
# Numaratorul  
# Demonstreaza functia range() sub for loops  
# Ion Studentul - 1/26/13  
  
print "Numara pana de La 0 La 9:"  
for i in range(10):  
    print i,  
  
print "\n\nNumara din cinci in cinci pana La 45:"  
for i in range(0, 50, 5):  
    print i,  
  
print "\n\nNumara invers de La 10 La 1:"  
for i in range(10, 0, -1):  
    print i,  
  
raw_input("\n\nApasa <enter> pt a iesi.")
```



```

C:\Users\cpopescu\workspace\py\Numaratorul.py
Numara pana de la 0 la 9:
0 1 2 3 4 5 6 7 8 9

Numara din cinci in cinci pana la 45:
0 5 10 15 20 25 30 35 40 45

Numara invers de la 10 la 1:
10 9 8 7 6 5 4 3 2 1

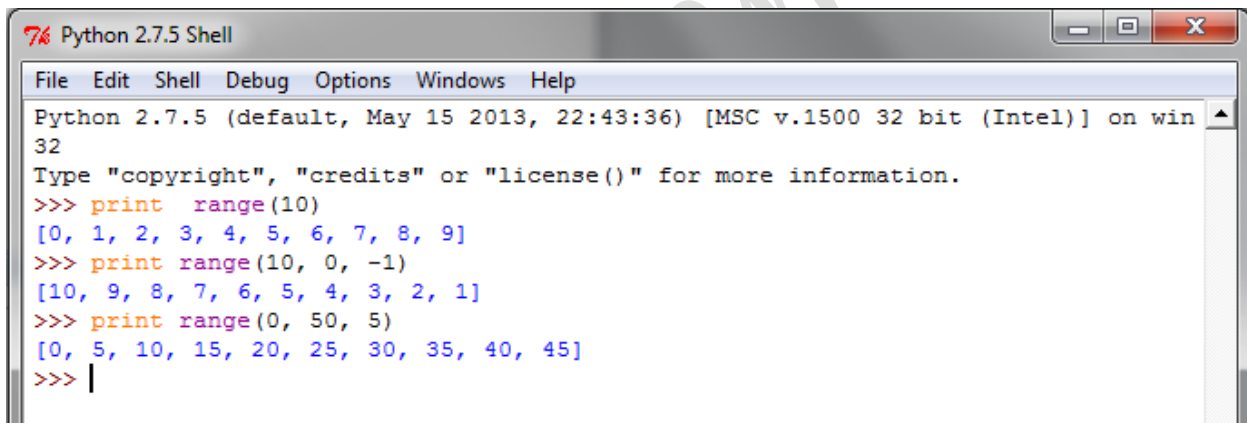
Apasa <enter> pt a iesi.

```

Fig. 24

Sintaxa `for` atribuie variabilei `i` pe rând valorile generate de `range(10)` apoi rulează pentru fiecare atribuire blocul de sintaxe. În acest caz doar afișează valoarea variabilei `i`.

Pentru a vedea ce valori va lua `i` putem folosi comanda de afisare. Se poate observa o rulare a programului în Fig. 24



```

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.
>>> print range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print range(10, 0, -1)
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> print range(0, 50, 5)
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
>>>

```

Fig. 25

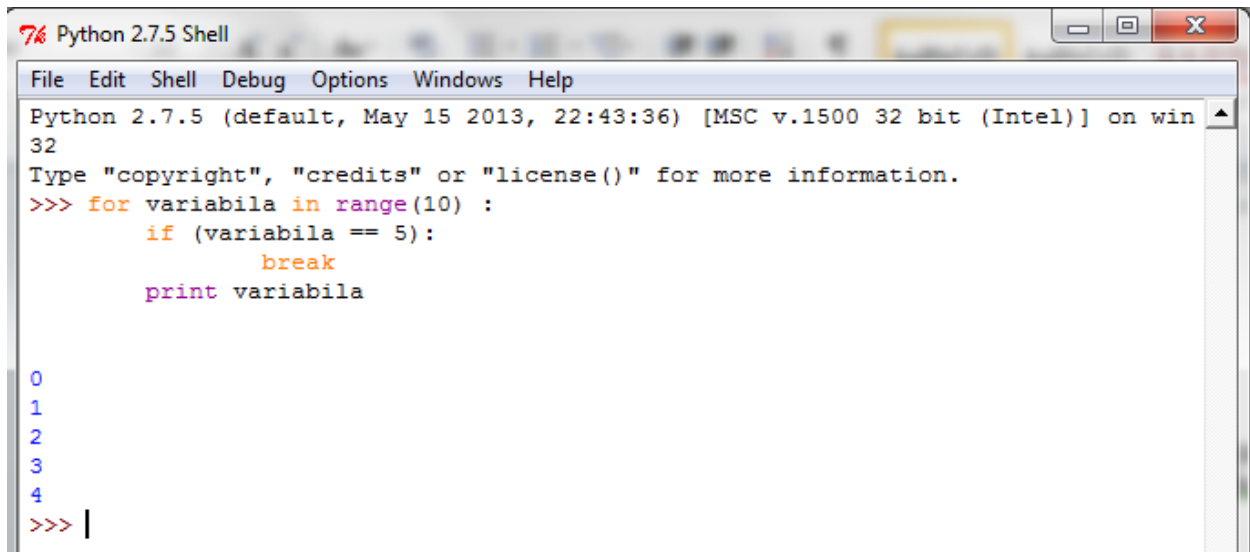
Aceste valori ce sunt înălțuite cu virgula și delimitate de paranteze pătrate poartă numele de listă. Acestea sunt studiate într-o secțiune viitoare. Deci, sintaxa `for` poate avea următoarea structura :

```

for variabila_iterare in lista:
    bloc_de_sintaxe #ce poate sau nu să foloseasca variabila iterare

```

Încă mai sunt disponibile cuvintele cheie `continue`, `pass` și `break` .



```

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.
>>> for variabila in range(10) :
        if (variabila == 5):
            break
        print variabila

0
1
2
3
4
>>> |

```

Fig. 26

Așa cum se poate vedea în Fig.26 putem introduce în interiorul unui bloc de expresii alți operatori decizionali. De asemenea `break` și `continue` funcționează și pentru `for`.

În cele ce urmează dorim să explicăm ce rol are cuvântul cheie `in` care face posibilă ciclarea pentru fiecare element din `range`. În ceea ce privește utilizarea cuvântului cheie `in` se poate utiliza și în expresii `if`. Un program se poate regăsi mai jos ca model de utilizare:

```

# Analizatorul de mesaje
# Demonstrează if cu expresia in
# Ion Studentul - 1/26/13

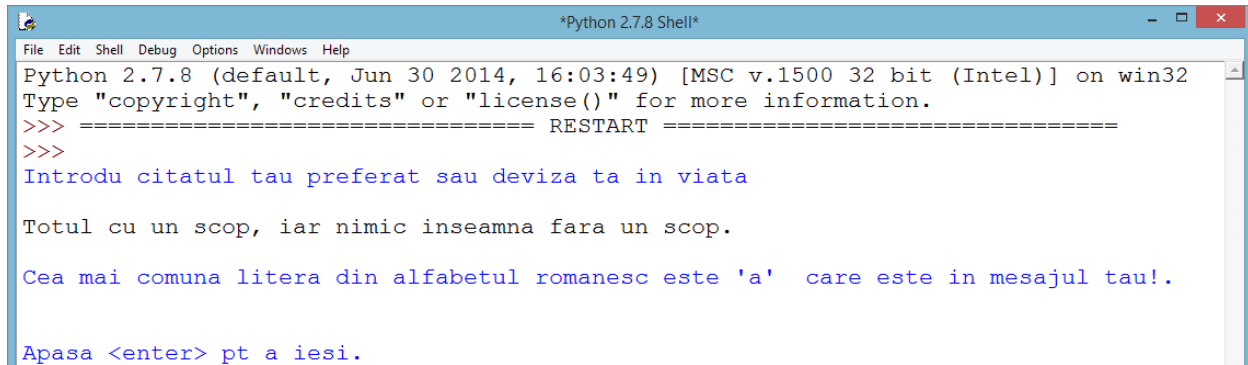
print "Introdu citatul tau preferat sau deviza ta în viața"
mesaj = raw_input("\n")

print "\nCea mai comună literă din alfabetul românesc este 'a' ",
if "a" in mesaj:
    print "care este în mesajul tau!."
else:
    print "care nu se regăsește în mesajul tau."

raw_input("\n\nApasă <enter> pt a iese.")

```

O posibilă rulare se poate vedea în Fig. 27



```

Python 2.7.8 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.8 (default, Jun 30 2014, 16:03:49) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Introdu citatul tau preferat sau deviza ta in viata

Totul cu un scop, iar nimic inseamna fara un scop.

Cea mai comuna litera din alfabetul romanesc este 'a' care este in mesajul tau!.

Apasa <enter> pt a iesi.

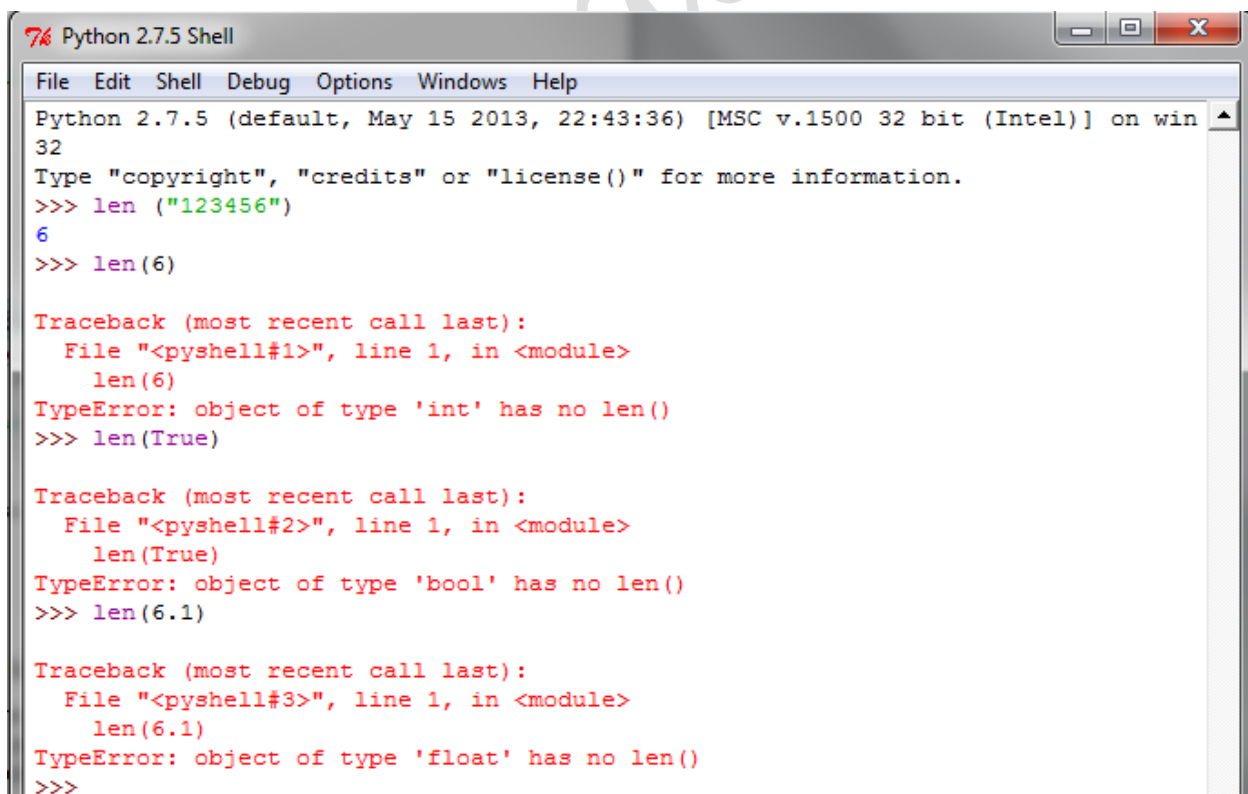
```

Fig. 27

Expresia `in` este utilizată pentru a lua fiecare caracter din șirul de caractere și va fi verificat cu șirul de caractere "a". Pentru că expresia dată de utilizator are în componență caracterul 'a' va afișa șirul de caractere "care este în mesajul tau!".

Lucrul avansat cu șiruri de caractere

Funcția `len(variabila)` returnează lungimea șirului de caractere ce e conținut de variabilă. Se poate observa ca această funcție nu se poate aplica pentru numere sau Boolean.



```

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.
>>> len("123456")
6
>>> len(6)

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    len(6)
TypeError: object of type 'int' has no len()
>>> len(True)

Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    len(True)
TypeError: object of type 'bool' has no len()
>>> len(6.1)

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    len(6.1)
TypeError: object of type 'float' has no len()
>>>

```

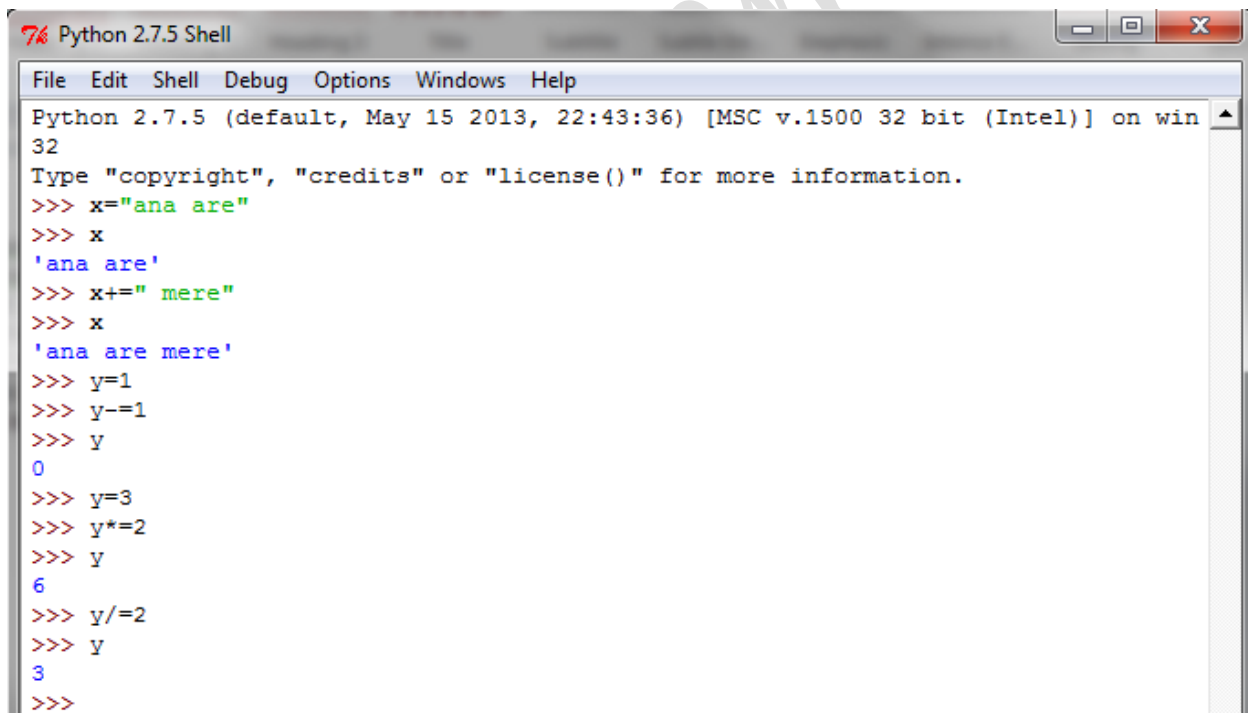
Fig. 28

Exista o formă prescurtată de calcul matematic cand trebuie sa realizez o operatie matematica cu o variabila si sa stochez rezultatul tot in aceasta variabila.

```
>>> x = 2
>>> x = x+2
>>> print x
4
>>> x = 2
>>> x+=2 # aceasta linie se poate traduce ca x=x+2 si reprezinta acelasi lucru
>>> print x
4
```

Fig. 28

Aceasta metoda poate fi utilizată pentru șiruri de caractere si pentru numere, chiar dacă pentru siruri de caractere nu este o practică comună. În Fig. 29 putem vedea ca aceste prescurtări se aplică pentru șiruri de caractere, dar și alte prescurtări care le putem întâlni în programarea python, și nu numai.



```
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.
>>> x="ana are"
>>> x
'ana are'
>>> x+=" mere"
>>> x
'ana are mere'
>>> y=1
>>> y-=1
>>> y
0
>>> y=3
>>> y*=2
>>> y
6
>>> y/=2
>>> y
3
>>>
```

Fig. 29

Sa presupunem că avem o variabilă ce stochează un șir de caractere, să zicem "index" și dorim să printăm a treia literă din șirul de caractere putem să apelam comanda :

```

>>> word = "index"
>>> print word[0]
i
>>> print word[1]
n
>>> print word[2]
d
>>> print word[3]
e
>>> print word[4]
x

```

Aceste paranteze patrate aplica principiul indexarii unde fiecare element al sirului are un index. Totusi trebuie sa fim atenti la index. Daca acesta este depasit ne va genera eroare. In Fig. 30 avem un sir de caractere format din 5 caractere, deci un index de la 0 la 4. Daca solicit index de 7 imi va da eroare.

```

>>> x = "Salut"
>>> x[0]
'S'
>>> x[7]

Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    x[7]
IndexError: string index out of range
>>>

```

Fig.30

Fiecare șir de caractere poate fi prelucrat pe principiul indexării unui șir de caractere. În cazul în care dorim să prelucram dintr-un șir de caractere ultimul caracter, dar nu știm lungimea acelui șir de caractere deoarece este dat de utilizator prin introducerea de la tastatură, Python vine cu o modalitate ușoară de a manipula astfel de șiruri. Se introduce conceptul de indexare inversă. Prin urmare `word[-1]` va reprezenta ultimul caracter din șirul de caractere stocat de variabila `word`. Mai jos putem vedea fiecare index pozitiv și negativ pentru șirul de caractere ce conține cuvântul `index`.

0	1	2	3	4
i	n	d	e	x
-5	-4	-3	-2	-1

Fig. 31

În cele ce urmează vom descoperi anumite metode de a manipula un șir de caractere. Iată un program prin care se dorește să manipulăm un șir de caractere.

```
# Index Sir de caractere
# Demonstreaza indexarea sirului de caractere
# Ion Studentul - 1/27/03

iteratii=0
cuvantInvers=""

cuvant = raw_input("\n\nScrie un cuvant: \t")

print "Cuvantul tau este", cuvant, "\n"

high = len(cuvant)

print "Lungimea cuvantului este:", high

for var in cuvant:
    print cuvant[iteratii], " trebuie să fie egal cu ", var
    iteratii+=1
    cuvantInvers=var+cuvantInvers

print "\nCuvantul ales se scrie invers: ",cuvantInvers

raw_input("\n\nApasa <enter> pt a iesi.")
```

```
Scrie un cuvant:      abcdef
Cuvantul tau este abcdef

Lungimea cuvantului este: 6
a  trebuie sa fie egal cu  a
b  trebuie sa fie egal cu  b
c  trebuie sa fie egal cu  c
d  trebuie sa fie egal cu  d
e  trebuie sa fie egal cu  e
f  trebuie sa fie egal cu  f

Cuvantul ales se scrie invers:  fedcba

Apasa <enter> pt a iesi.
>>>
```

Fig.32

În prima parte a programului introducem două variabile și solicităm un șir de caractere de la tastatură.

```
iteratii=0
cuvantInvers=""

cuvant = raw_input("\nScrie un cuvânt:")
```

Următoarele linii vor afișa cuvântul scris de utilizator, vor măsura lungimea cu ajutorul `len(cuvant)` și vor stoca această lungime măsurată în variabila `high`. A treia linie extrasă din program va afișa lungimea cuvântului scris de la tastatură.

```
print "Cuvântul tau este", cuvant, "\n"

high = len(cuvant)

print "Lungimea cuvântului este:", high
```

```
for var in cuvant:
    print cuvant[iteratii], " trebuie să fie egal cu ", var
    iteratii+=1
    cuvantInvers=var+cuvantInvers
```

În următoarele linii se dorește să se afișeze cuvântul scris de la tastatură literă cu literă folosind două metode:

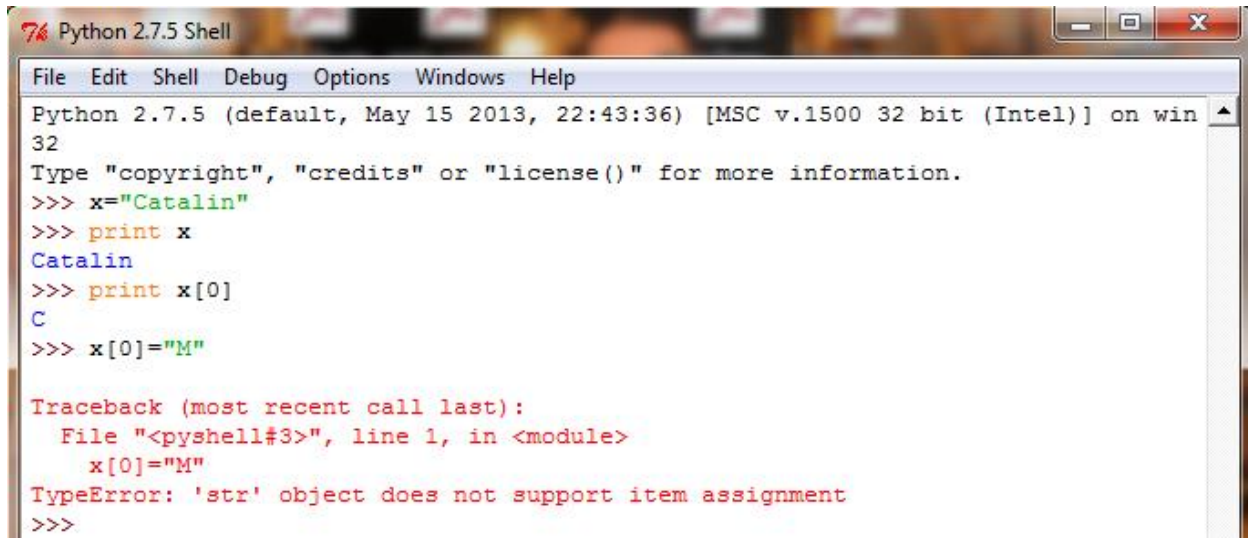
Prima metodă va ține cont de variabila `iteratii` ce inițial este setată la 0. La fiecare ciclu vom incrementa variabila `iteratii` pentru a afișa următorul caracter.

A doua metodă folosește variabila temporară din `for`, unde aici este `<<var>>`. Aceasta ia valoarea fiecărui caracter din șir, schimbându-și valoarea la fiecare ciclu.

Variabila `cuvantInvers` este o variabilă declarată inițial la un șir nul de caractere. Apoi vom concatena acest șir cu variabila temporară `var`. Pentru că am egalat variabila cu `var+cuvantInvers` vom avea cuvântul invers la final. Dacă o egalăm ca mai jos atunci variabila `cuvantInvers` stoca același lucru ca și variabila `cuvant`:

```
cuvantInvers= cuvantInvers + var
```

Un șir de caractere este un element imutabil. Presupunând că avem o variabilă `x` care susține șirul de caractere "Catalin". Dacă am greșit numele și dorim să schimbăm prima literă în `M` și a treia în `d` șirul devenind "Madalin", am putea încerca să facem acest lucru cu indexarea șirului de caractere. În Fig. 33 putem vedea că modificarea unui singur caracter din șir nu este permisă. Soluția este reatribuirea unei valori noi acelei variabile. Astfel `x="Madalin"` este soluția problemei modificării șirului de caractere.



```

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.
>>> x="Catalin"
>>> print x
Catalin
>>> print x[0]
C
>>> x[0]="M"

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    x[0]="M"
TypeError: 'str' object does not support item assignment
>>>

```

Fig. 33

Exista și alte solutii la acasta schimbare din „Catalin” in „Madalin”, metode ale șirurilor de caractere studiate in sedinta anterioara cum ar fi:

```

>>> x = "sir de caractere"
>>> x.replace("r", "c")
'sic de cacactece'
>>> print x
sir de caractere
>>>

```

Fig.34

Exista si alte metode de manipulare a sirurilor de caractere cum ar fi cele ce se regasesc mai jos:

Cu ajutorul metodei find putem găsi șiruri de caractere în interiorul altor șiruri de caractere. Apelarea se face prin variabila.find(“sir de caractere cautat”). Daca nu gaseste sirul cautat returneaza -1.

```

>>> x = "asd"
>>> x.find("c")
-1
>>>

```

Cu ajutorul metodei replice putem să înlocuim în șirul de caractere. Apelarea se face prin variabila.replace(“valoare căutata”, “valoare ce înlocuiește valoarea căutata la găsiere”)

Metoda split returnează o lista de elemente împărțite după un șir dat. Lista este un concept încă neînvățat, dar trebuie totuși subliniat ca elimina caracterul de referință.

Apelarea se face prin variabila.split("șir de caractere ce împarte șirul")

Aceste metode nu schimbă șirul inițial! Prin urmare dacă dorim să prelucram ulterior va trebui să atribuim valoarea returnată de metoda unei variabile.

```
>>> #x.replace(old,new)
>>> x.find("a")
8
>>> x[8]
'a'
>>> #prima oara unde a fost gasit elementul
>>>
>>>
>>> x.find("r")
2
>>>
#x este neschimbat pt a salva x=x.replace(old,new)
>>> print x
sir de caractere
>>> x.find("z")
-1
>>> #elementul nu este gasit
>>> #deci returneaza -1
>>> x.split(" ")
['sir', 'de', 'caractere']
>>> #returneaza o lista impartind dupa elementul din interior
>>> x.split("r")
['si', ' de ca', 'acte', 'e']
>>> # se vede ca elimina acel caracter
```

Fig. 35

Tuplu

Tuple se traduce ca tuplu având însemnătatea de pereche cu mai multe elemente.

Tuplu reprezintă o serie de secvențe care pot conține numere, șiruri, sau alte tuple. Prin urmare poți stoca tot felul de date. Inițializarea unui tuplu gol se face astfel:

```
x=()
```

Iată mai jos un tuplu care susține alte tipuri de variabile:

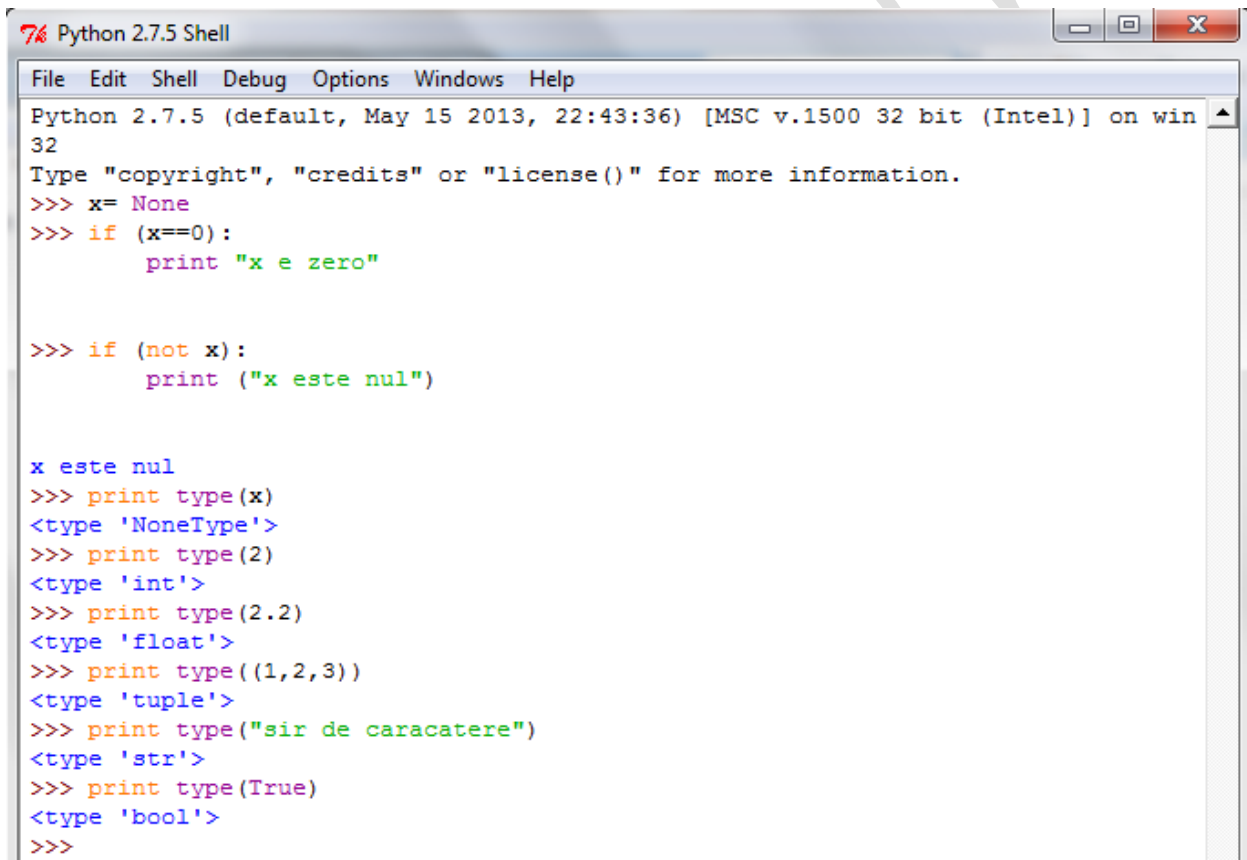
```
x=(True, False, 4,5,6,7,"sir de caractere",("tuplu 2",2))
```

Tuplu este un tip de variabilă imutabilă. Aceasta afirmație ne arată că nu putem rescrie unul din elementele unui tuplu, în schimb putem să atribuim variabilei o altă valoare, adică un alt tuplu.

```
>>> x = ("a", "b", "d")
>>> x[2]
'd'
>>> x[2] = "c"

Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    x[2] = "c"
TypeError: 'tuple' object does not support item assignment
>>> |
```

Fig. 36



```
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.
>>> x = None
>>> if (x==0):
    print "x e zero"

>>> if (not x):
    print ("x este nul")

x este nul
>>> print type(x)
<type 'NoneType'>
>>> print type(2)
<type 'int'>
>>> print type(2.2)
<type 'float'>
>>> print type((1,2,3))
<type 'tuple'>
>>> print type("sir de caractere")
<type 'str'>
>>> print type(True)
<type 'bool'>
>>>
```

Fig. 27

De altfel ne putem ajuta de funcția `type()` pentru a determina ce tip de variabilă este, adică ce tip de valoare susține variabila testată, așa cum am studiat în prima sedință.

Regăsim un program mai jos care creează și utilizează tupluri.

```
# inventar de joc
# Demonstreaza tuplu
# Ion Studentul - 1/26/13

# cream o variabila goala
inventar = None

print "Acest program sustine inventarul unui personaj dintr-un joc.\n"

# tratarea tuplului ca o conditie

if not inventar:
    print "Momentan nu ai nici o arma in inventar."

#crearea de tuplu cu articole in inventar
inventar = ("sabie",
            "armura",
            "scut",
            "potiune de vindecare")

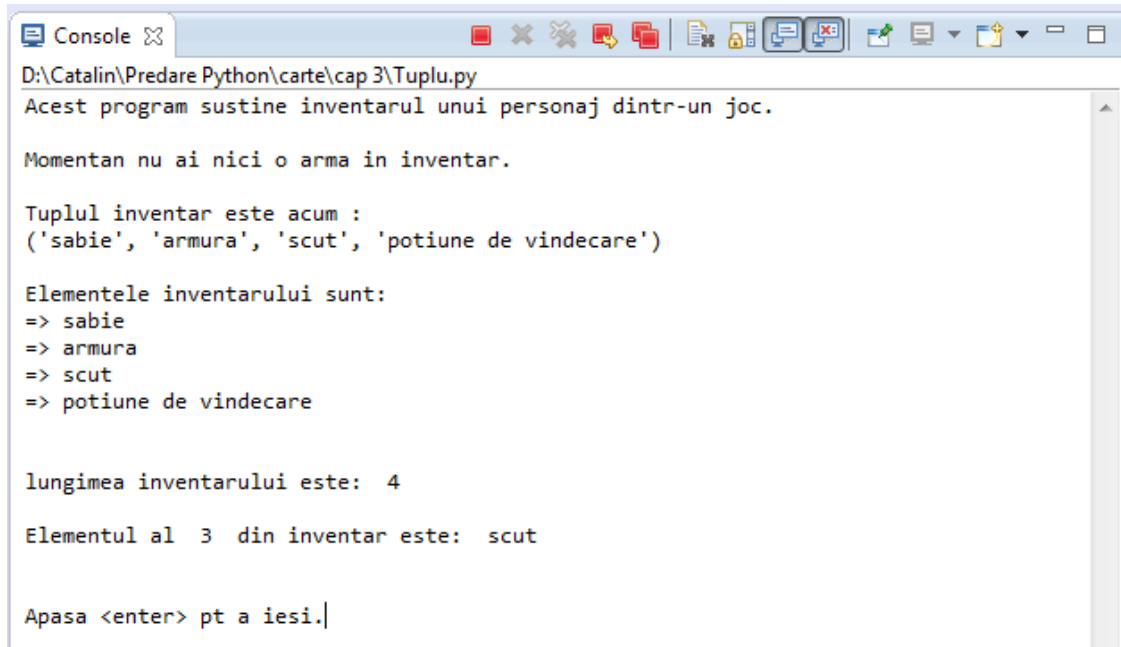
# afisarea tuplului
print "\nTuplul inventar este acum :\n", inventar

# afisarea fiecarui element din tuplu
print "\nElementele inventarului sunt:"
for item in inventar:
    print "=>",item

print "\n\nLungimea inventarului este: ", len(inventar)
jumateInv= len(inventar)/2
print "\nElementul al ", jumateInv+1 ," din inventar este: ",inventar[jumateInv]
#Folosim jumateInv+1 deoarece primul element este 0.

raw_input("\n\nApasa <enter> pt a iesi.")
```

O rulare a programului inventar se poate vedea în Fig. 38.



```

D:\Catalin\Predare Python\carte\cap 3\Tuplu.py
Acest program sustine inventarul unui personaj dintr-un joc.

Momentan nu ai nici o arma in inventar.

Tuplul inventar este acum :
('sabie', 'armura', 'scut', 'potiune de vindecare')

Elementele inventarului sunt:
=> sabie
=> armura
=> scut
=> potiune de vindecare

lungimea inventarului este: 4

Elementul al 3 din inventar este: scut

Apasa <enter> pt a iesi.

```

Fig. 38

In programul inventar observăm inițializarea unei variabile cu **None**. Această valoare are scopul de a putea inițializa o variabila, dar fără a susține nici o valoare. La testarea ei ca si condiție va returna fals.

```
inventar = None
```

In programul inventar de joc folosim variabila inventar pentru a manipula un tuplu. Putem să afișăm un tuplu cu ajutorul comenzii print.

```
print "\nTuplul inventar este acum :\n", inventar
```

Tratarea tuplului stocat în variabilă inventar ca o condiție într-o sintaxă if se poate întâlni în linia `if not inventar:` Aceasta linie tratează un tuplu gol ca fiind fals și orice tuplu ce are cel puțin o înregistrare, deci cu lungimea diferită de zero, ca fiind o condiție adevărată.

```
if not inventar:
    print "Momentan nu ai nici o arma in inventar."
```

Lungimea unui tuplu se poate obține cu funcția `len(tuplu)`:

```
print "\nLungimea inventarului este: " , len(inventar)
```

Ciclarea prin elementele unui tuplu se poate face cu ajutorul unui „for”. Astfel, variabila temporara item va lua pe rand valorile tuplului si vor fi prelucrate de blocul de comenzi de sub for. Aici vom afisa fiecare valoare a variabilei item.

```
for item in inventar:
    print "=>",item
```

Pentru a afișa cel de-al treilea caracter folosim operația de indexare, proces utilizat și la șiruri de caractere. Obținem rezultatul al treilea element deoarece împărțim lungimea la doi. Indexarea unei variabile cu cifra doi returnează al treilea element deoarece indexul pornește de la zero.(0-primul element ,1- al doilea element, 2- al treilea element).

```
jumateInv= len(inventar)/2
print "\nElementul al ", jumateInv+1 , " din inventar este: ",inventar[jumateInv]
#Folosim jumateInv+1 deoarece primul element este 0.
```

Lista

Lista este un alt tip de variabilă care are toate caracteristicile unui tuplu, exceptând imutabilitatea. Prin urmare o listă permite rescrierea unui element component, proprietate numita mutabilitate. Acesta calitate ne permite flexibilitatea de care avem nevoie în programare.

Mai jos se poate vedea un program în care utilizăm lista. Acest program este o modificare a programului inventar utilizat la explicarea tuplului.

```
# inventar de joc
# Demonstreaza lista
# Ion Studentul - 1/26/13

#crearea unei liste cu articole in inventar
inventar = ["sabie", "armura", "scut", "potiune de vindecare"]

# afisarea listei
print "\nLista inventar este :\n", inventar

# afisarea fiecarui element din listei
print "\nElementele inventarului sunt:"
for item in inventar:
    print "=>",item

print "\n\nLungimea inventarului este: " , len(inventar)
jumateInv= len(inventar)/2
print "\nElementul al ", jumateInv+1 , " din inventar este: ",inventar[jumateInv]
#Folosim jumateInv+1 deoarece primul element este 0.
```

```

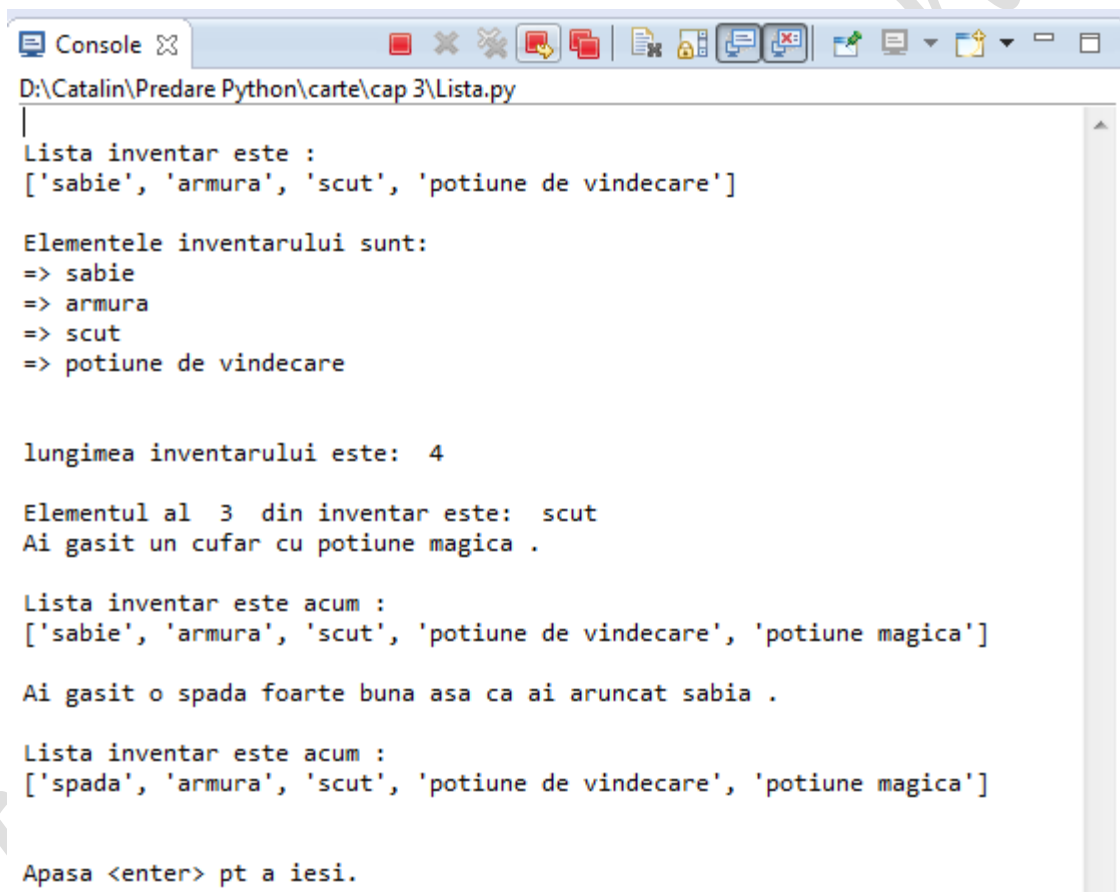
print "Ai gasit un cufar cu potiune magica ."
cufar=["potiune magica"]
inventar+=cufar

# afisarea listei
print "\nLista inventar este acum :\n", inventar

print "\nAi gasit o spada foarte buna asa ca ai aruncat sabia .\n"
inventar[0]= "spada"
# afisarea listei
print "Lista inventar este acum :\n", inventar

raw_input("\n\nApasa <enter> pt a iesi.")

```



```

D:\Catalin\Predare Python\carte\cap 3\Lista.py
|
Lista inventar este :
['sabie', 'armura', 'scut', 'potiune de vindecare']

Elementele inventarului sunt:
=> sabie
=> armura
=> scut
=> potiune de vindecare

lungimea inventarului este: 4

Elementul al 3 din inventar este: scut
Ai gasit un cufar cu potiune magica .

Lista inventar este acum :
['sabie', 'armura', 'scut', 'potiune de vindecare', 'potiune magica']

Ai gasit o spada foarte buna asa ca ai aruncat sabia .

Lista inventar este acum :
['spada', 'armura', 'scut', 'potiune de vindecare', 'potiune magica']

Apasa <enter> pt a iesi.

```

Fig. 39

În programul inventar, modificat pentru a utiliza lista în locul tuplului, putem observa că toate facilitățile de la tuplu le întâlnim și aici.

Prin urmare putem atribui unei variabile o lista.

```
#crearea unei liste cu articole in inventar
inventar = ["sabie", "armura", "scut", "potiune de vindecare"]
```

Printarea unei liste se poate realiza in mod direct.

```
# afisarea listei
print "\nLista inventar este :\n", inventar
```

Ciclarea printre elementele listei este o practica comuna în programare, unde fiecare element al unei liste este prelucrat. Aici fiecare element al listei este printat după aplicarea unui for la lista.

```
print "\nElementele inventarului sunt:"
for item in inventar:
    print "=>", item
```

Lungimea unei liste se poate obține cu funcția len(lista):

```
print "\n\nLungimea inventarului este: " , len(inventar)
```

Indexarea este posibila cu ajutorul unei liste. Așa cum putem vedea în program nu numai afișarea unui element este posibila, ci și schimbarea acelu element deoarece o lista are proprietatea de mutabilitate.

Nu întâmplător nu au fost adăugate în programul de inventar cu tuplu aceste linii:

```
print "Ai gasit un cufar cu potiune magica ."
cufar=["potiune magica"]
inventar+=cufar

# afisarea listei
print "\nLista inventar este acum :\n", inventar

print "\nAi gasit o spada foarte buna asa ca ai aruncat sabia .\n"
inventar[0]= "spada"
# afisarea listei
print "Lista inventar este acum :\n", inventar
```

Cufar este o variabilă noua inițializată pentru a susține o noua lista ce are un singur element "potiune magica".

Lista variabilei cufar este adunată la lista inventar. Acest lucru este posibil deoarece ambele variabile sunt de același tip și lista este un element mutabil. Deci putem să adunăm la o lista o alta lista. Modificarea unui element al unei liste este de asemenea posibil deoarece lista este mutabila; astfel variabilă inventar își schimbă primul element din "sabie" în "spada".


```
>>> cufar=["lotiune magica"]
>>> type(cufar)
<type 'list'>
>>> inventar = ["sabie","armura","scut", "potiune de vindecare"]

>>> type(inventar)
<type 'list'>
```

Fig. 40

Similar cu metodele de manipulare ale unui șir de caractere exista metode, diferite ce-i drept, și pentru liste. Mai jos se regăsesc într-un tabel toate metodele uzuale pentru un sir:

Metoda	Descriere
append(<i>value</i>)	adauga <i>value</i> la finalul listei.
sort()	Sorteaza elementele dupa valoarea cea mai mica.
reverse()	Reinverseaza ordinea listei. Primul element devine ultimul, al doilea devine penultimul...
count(<i>value</i>)	Indexeaza de cate ori va gasi acel sir de caractere (<i>value</i>).Zero inseamna ca nu a gasit nimic, 1 inseamna ca a gasit o potrivire
index(<i>value</i>)	Returneaza prima pozitie(indexul) cand <i>value</i> apare în lista. Returneaza eroare daca nu este intalnita.
insert(<i>i</i> , <i>value</i>)	Insereaza <i>value</i> la pozitia <i>i</i> .
pop([<i>i</i>])	Returneaza <i>value</i> pozitia <i>i</i> și sterge valoarea <i>value</i> din lista. E optionala numarul pozitie <i>i</i> și poate fi omis ; în acest caz ultimul element va fi sters și returnat.
remove(<i>value</i>)	Sterge prima intrare pe care o intalneste cu valoarea <i>value</i> din lista.

Vom utiliza câteva metode în programul următor:

```
# Lista de cumparaturi
# Demonstreaza optiuni avansate lista
# Ion Studentul - 1/26/13
lista=[]

print("\t\t Bine ati venit la programul Lista de cumparaturi")

while (True):
    print """
0-Pentru afisare lista actuala
1-Pentru introducerea unui elent nou
2 pentru stergerea unui element existent
3- pentru stergerea intregii liste
q- pentru iesire
"""
```

```

alegere=raw_input("\nIntrodu o cifra:\t")

if (alegere=="0"):
    if (lista):
        lista.sort()
        print "\nLista cumparaturi este :\n"
        for e in lista:
            print "=>",e
    else:
        print "Lista este goala"
elif (alegere=="1"):
    elementNou=raw_input("\nElementul nou este:\t")
    lista.append(elementNou)
elif (alegere=="2"):
    if (lista):
        sterge=raw_input("scrie element ce doresti sa-l stergi:")
        if (sterge in lista):
            lista.remove(sterge)
            print "Elementul ", sterge , "a fost sters"
        else:
            print sterge , "nu a fost gasit in Lista"
    else:
        print "Lista e goala"
elif (alegere=="3"):
    lista=[]
    print("Lista a fost stearsa")
elif(alegere.upper()=="Q"):
    break

print "Va multumim ca ati ales acest program"

```

În următoarea secțiune vom urmări fiecare fragment la programul "lista de cumparaturi" :

```
lista=[]
```

La începutul programului se inițializează variabilele și printăm mesajul de început. Dacă variabila lista nu era inițializată existau erori la afișare în cazul în care doream să afișăm lista de cumpărături imediat după intrarea în program. În cazul în care era inițializată după expresia while de fiecare dată când iteram lista devenea nula.

```
print "\t\t Bine ati venit La programul Lista de cumparaturi"
```

Afișăm un mesaj introductiv înaintea expresiei while pentru a nu apărea de fiecare dată când iterează.

Expresia while are o condiție de tip boolean ceea ce indică ca va cicla blocul de sintaxe la infinit sau pana la întâlnirea unei sintaxe break.

```
while (True):
```

```

print """
0-Pentru afisare Lista actuala
1-Pentru introducerea unui elent nou
2 pentru stergerea unui element existent
3- pentru stergerea intregii Liste
q- pentru iesire
"""

```

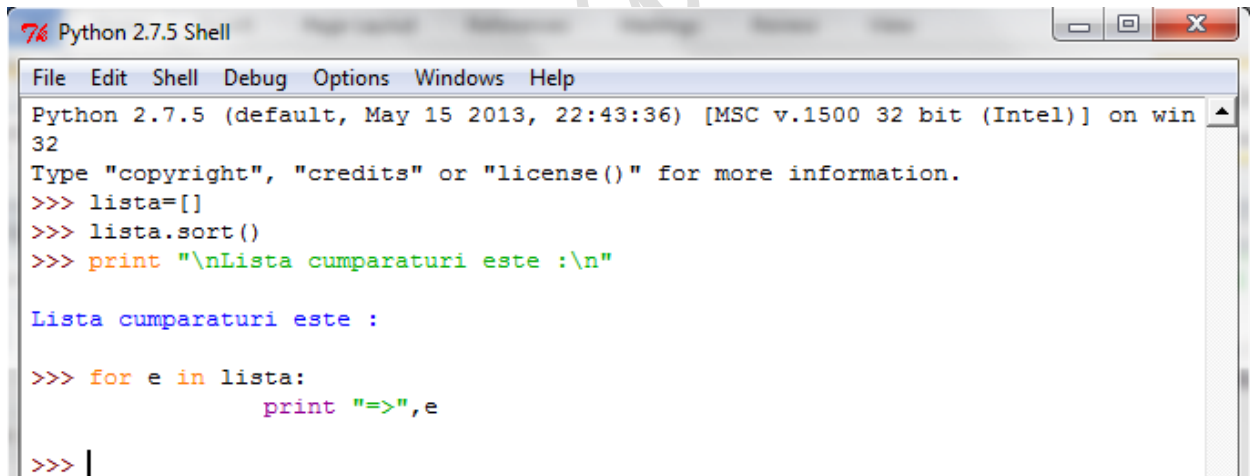
Prima comanda ce se regăsește în blocul de sintaxe while este afișarea unui meniu. Aceasta are rolul de a afișa meniu ori de cate ori utilizatorul ar trebui să ia o decizie .

```

if (alegere=="0"):
    if (lista):
        lista.sort()
        print "\nLista cumparaturi este :\n"
        for e in lista:
            print "=>",e
    else:
        print "Lista este goala"

```

Prima alegere conform cu meniul este afișarea listei actuale de cumpărături. Aceasta se poate apela prin tastarea cifrei zero. Se verifica dacă lista este nula sau nu pentru a afișa un mesaj corespunzător. În cazul în care eliminăm if- else-ul ce verifica dacă lista este nula vom avea un mesaj ambiguu la o lista nula așa cum se poate vedea și în figura de mai jos:



```

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.
>>> lista=[]
>>> lista.sort()
>>> print "\nLista cumparaturi este :\n"

Lista cumparaturi este :

>>> for e in lista:
            print "=>",e

>>> |

```

Fig. 41

Acest mesaj dat de situatia in care nu am avea nici un element in lista de cumaraturi nu ar fi potrivit. Prin urmare, ar trebui să corectam aceste comportamente. In acest caz acest mesaj fost corectat prin condiționarea cu un if-else.

Funcția sort aplicata listei în sintaxă lista.sort() are rolul de a sorta elementele listei într-o ordine alfanumerica. Aranjarea unei liste se realizează astfel mai întâi numerele, apoi

caractere speciale sau numere de tip șir de caractere, apoi celelalte șiruri de caractere ce încep cu litere în ordine alfabetică.

```
>>>
>>> lista = ['a', "c", 'b', 1, "1", "@", "!"]
>>> lista.sort()
>>> print lista
[1, '!', '1', '@', 'a', 'b', 'c']
>>>
```

Fig. 42

```
for e in lista:
    print ">",e
```

Pentru a afișa lista am recurs la un for pentru o afișare facilă din punctul de vedere al utilizatorului, care nu ar dori să vadă paranteze și virgule.

```
elif (alegere=="1"):
    elementNou=raw_input("\nElementul nou este:\t")
    lista.append(elementNou)
```

Urmatoarea alegere pe care utilizatorul o poate face este introducerea unui element nou apăsând tasta 1. Introducerea elementului nou se face prin metoda lista.append("element_nou"), metoda ce introduce la finalul listei elementul nou ca o noua intrare.

```
>>> lista=["primul element", "al doilea element"]
>>>
>>> lista.append("al treilea element")
>>> print lista
['primul element', 'al doilea element', 'al treilea element']
>>>
```

Fig. 43

În cazul în care utilizatorul ar apăsa tasta 2 pentru a introduce o nouă intrare s-ar rula codul de mai jos:

```
elif (alegere=='2'):
    if (lista):
        sterge=raw_input("\nscrie element ce doresti sa-l sterghi:")
        if (sterge in lista):
            lista.remove(sterge)
            print "Elementul ", sterge , "a fost sters"
        else:
            print sterge , "nu a fost gasit in lista"
    else:
        print "Lista e goala"
```

Inițial trebuie să verificam dacă lista este goală. Nu putem șterge ceva dacă nu avem ce șterge. Astfel adaptăm programul prin mesajul “Lista e goală”, decizie ce se ia printr-un if-else ce testează dacă lista este vidă.

În cazul în care lista are cel puțin un element utilizatorul trebuie să scrie elementul care îl dorește să-l șteargă, element ce este stocat în variabila șterge.

Pasul următor este să verificăm dacă elementul este găsit în lista; facem acest pas cu el-if ul ce are ca și condiție aceasta verificare.

În cazul în care elementul e găsit în lista putem să ștergem acel element cu metoda:

```
lista.remove(sterge)
```

Am adăugat și posibilitatea de a reseta lista când utilizatorul apasă tasta 3:

```
elif (alegere=='3'):  
    lista=[]  
    print("Lista a fost stearsa")
```

Lista este ștearsă prin rescrierea variabilei.

Propun un exercițiu de creativitate și anume să găsiți voi alte variante diferite!

Bineînțeles că ar mai fi varianta unu prin care iterăm lista printr-un for și ștergem fiecare element din lista sau varianta a doua vedem cât de mare este lista apoi iterăm în funcție de asta aplicând spre exemplu un pop la lista:

Varianta 1:

```
for e in lista:  
    lista.remove(e)
```

Varianta 2:

```
x=len(lista)  
while (x>=1):  
    x-=1  
    lista.pop()
```

Ultima alegere pe care poate să o facă utilizatorul este să iasă din program utilizând tasta ‘q’ sau ‘Q’:

```
elif(alegere.upper()=='Q'):  
    break
```

Prin aceasta varianta se poate ieși din program apelând secvența de întrerupere a buclei și anume break.

Am fi putut să adăugăm și următoarele două linii :

```
else:
    pass
```

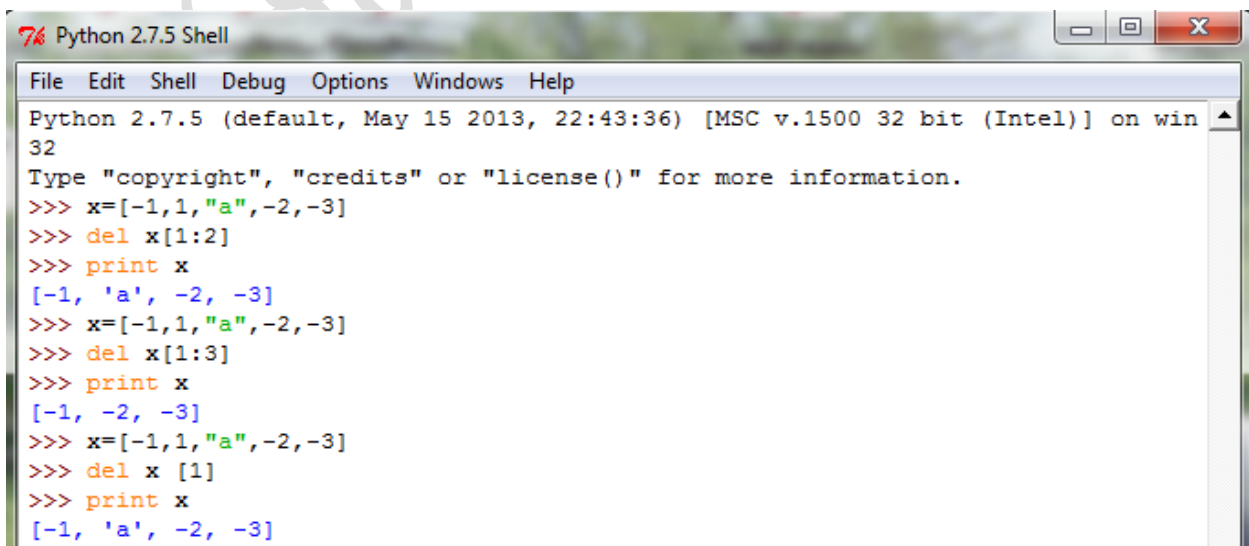
Dar aceste două linii nu schimbau cu nimic modul de rulare a programului 'Lista de cumparaturi', astfel ca nu au fost introduse.

Spuneam în subcapitolul Tuplu ca un tuplu poate conține un alt tuplu. Bineînțeles ca și o lista poate conține o alta lista. Se poate merge mai departe de atât, astfel ca o lista poate conține o alta lista care conține o alta lista... Acest procedeu se numește secvențe imbricate (imbricat=suprapus parțial) (eng. nested sequence). Accesarea unui astfel de lista din lista se face prin indexarea repetată.

```
>>> lista=[1,["al doilea",["infoacademy"]]]
>>> len(lista)
2
>>> lista[1]
['al doilea', ['infoacademy']]
>>> lista[0]
1
>>> lista[1][0]
'al doilea'
>>> lista[1][1]
['infoacademy']
>>> lista[1][1][0]
'infoacademy'
```

Fig. 44

Trebuie menționată și o modalitate prin care putem să ștergem doar un element fără a returna nici o valoare sau o modalitate prin care ștergem doar a doua și a treia valoare din lista. Aceste tipuri de operații se realizează cu comanda 'del':



```
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.
>>> x=[-1,1,"a",-2,-3]
>>> del x[1:2]
>>> print x
[-1, 'a', -2, -3]
>>> x=[-1,1,"a",-2,-3]
>>> del x[1:3]
>>> print x
[-1, -2, -3]
>>> x=[-1,1,"a",-2,-3]
>>> del x [1]
>>> print x
[-1, 'a', -2, -3]
```

Fig. 45

Asa cum se poate vedea in Fig. 45 putem da un range de indecsi separandu-i cu ajutorul a doua puncte `:`. Daca dorim sa prelucram de la un index pana la sfarsitul unui range nu mai completam valoarea finala. Simiar si pentru prelucrarea primelor elemente ale listei.

```
>>> x = [1,2,3,4,5]
>>> print x[1:]
[2, 3, 4, 5]
>>> print x[:3]
[1, 2, 3]
>>>
```

Fig. 46

In Fig. 46 putem vedea ca daca apelam `print x[1:]` va afisa de la indexul 1 in sus (inclusiv index 1). Daca apelam `print x[:3]` va afisa de la indexul 0 la indexul 3 (fara index 3).

Dicționar

Probabil ca se vede deja ca programatorilor le place să organizeze datele. Astfel tipul de variabilă dicționar face și el același lucru ca și o lista, dar merge la etapa următoare!

Un dicționar este format din perechi de tip "cheie": „valoare” ; unde cheia și valoarea pot fi de asemenea și numere liste etc. .

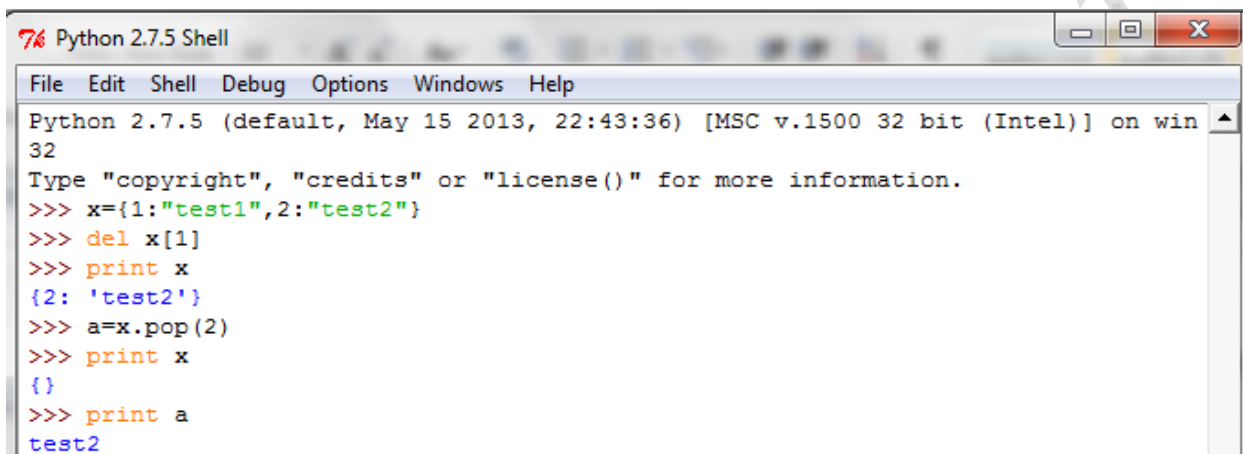
Un dicționar este are proprietatea de mutabilitate.

Mai jos putem vedea un tabel cu toate metodele mai uzuale utilizate pentru manipularea unui dicționar:

Metoda	Descriere
<code>has_key(key)</code>	Returneaza True daca <i>key</i> se gaseste în dicționar ca și cheie, altfel returneaza False.
<code>get(key, [default])</code>	Returneaza valoarea cheii <i>key</i> .Daca cheia nu este gasita atunci se returneaza cuvantul optional <i>default</i> . Daca cheia nu exista și cuvantul <i>default</i> nu este specificat, atunci se va returna None.
<code>keys()</code>	Returneaza o lista cu toate cheile din dicționar.
<code>values()</code>	Returneaza o lista cu toate valorile din dicționar.
<code>items()</code>	Returneaza o lista cu toate elementele din dicționar Fiecare element este un tuplu de doua elemnete de tip (cheie,valoare)..

Metoda	Descriere
pop(key)	Sterge elementul key:value din dictionar daca key exista. In cazul în care nu exista va returna eroare. Returneaza value.
del dictionar[key]	Sterge perechea key:value din dictionar daca key exista. In cazul în care nu exista va returna eroare. Nu returneaza nimic.

Mai jos se regăsește o figura unde se poate deosebi care e diferența dintre pop și del.



```

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.
>>> x={1:"test1",2:"test2"}
>>> del x[1]
>>> print x
{2: 'test2'}
>>> a=x.pop(2)
>>> print x
{}
>>> print a
test2

```

Fig. 47

Sa vedem dicționarul la treaba în următorul program și anume Dicționarul de Facebook:

```

# Dictionar de facebook
# Demonstreaza dictionar
# Ion Studentul - 1/26/13

dictionar={"ASAP": "As Soon As Possible-Cat mai repede posibil",
"ASL": "Age Sex Location-Varsta, sex, Locatie",
"BRB": "Be Right Back-Revin imediat",
"FYI": "For Your Info-Pentru informatia ta",
"LOL": "Laugh out Loud-Rad tare",
"PM": "Private Message-mesaj pe privat",
"FRUMI": "Frumos"}

print"\nAccesam dictionarul pentru a vedea ce cuvinte cheie avem:"
print dictionar.keys()

print"\nAccesam dictionarul pentru a vedea ce valori avem:"
print dictionar.values()

print "\nToate elementele dictionarului sunt:"
for elem in dictionar.keys():
    print "cheie:",elem , ">=> valoare:",dictionar[elem]

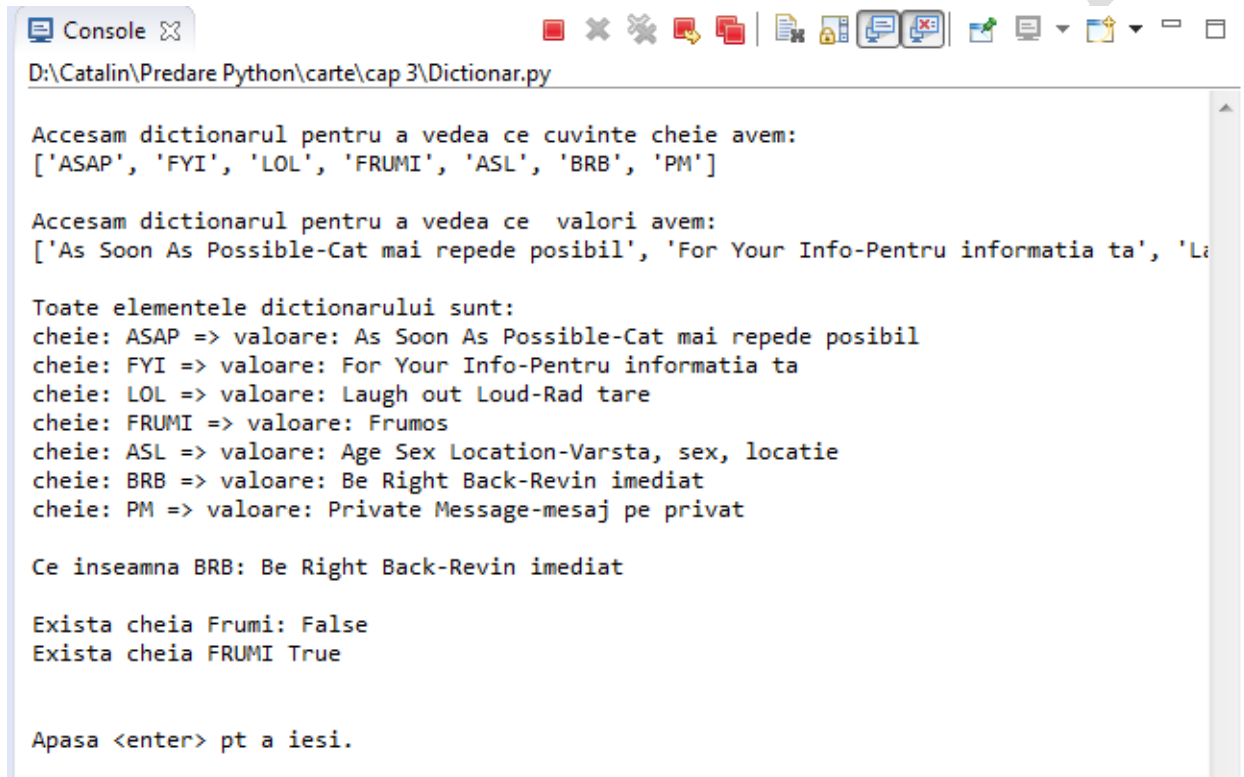
```



```
#apelarea directa trebuie testat inainte pentru a nu genera eroare
if "BRB" in dictionar:
    print "\nCe inseamna BRB:",dictionar["BRB"],"n"

#Exista cheia Frumi testarea modul alternativ
print "Exista cheia Frumi:",dictionar.has_key("Frumi")
print "Exista cheia FRUMI",dictionar.has_key("FRUMI")

raw_input("\n\nApasa <enter> pt a iesi.")
```



```
Console
D:\Catalin\Predare Python\carte\cap 3\Dicționar.py

Accesam dictionarul pentru a vedea ce cuvinte cheie avem:
['ASAP', 'FYI', 'LOL', 'FRUMI', 'ASL', 'BRB', 'PM']

Accesam dictionarul pentru a vedea ce valori avem:
['As Soon As Possible-Cat mai repede posibil', 'For Your Info-Pentru informatia ta', 'Laugh out Loud-Rad tare']

Toate elementele dictionarului sunt:
cheie: ASAP => valoare: As Soon As Possible-Cat mai repede posibil
cheie: FYI => valoare: For Your Info-Pentru informatia ta
cheie: LOL => valoare: Laugh out Loud-Rad tare
cheie: FRUMI => valoare: Frumos
cheie: ASL => valoare: Age Sex Location-Varsta, sex, locatie
cheie: BRB => valoare: Be Right Back-Revin imediat
cheie: PM => valoare: Private Message-mesaj pe privat

Ce inseamna BRB: Be Right Back-Revin imediat

Exista cheia Frumi: False
Exista cheia FRUMI True

Apasa <enter> pt a iesi.
```

Fig. 49

Sa analizam fiecare sintaxă din programul Dicționar de Facebook

```
dictionar={"ASAP": "As Soon As Possible-Cat mai repede posibil",
"ASL": "Age Sex Location-Varsta, sex, locatie",
"BRB": "Be Right Back-Revin imediat",
"FYI": "For Your Info-Pentru informatia ta",
"LOL": "Laugh out Loud-Rad tare",
"PM": "Private Message-mesaj pe privat",
"FRUMI": "Frumos"}
```

In sintaxă de mai sus se inițializează variabila dicționar de același tip ca și numele (dictionar).

Perechile se inițializează sub paranteze de tip acolade de formă cheie: valoare;

Fiecare pereche cheie valoare se separa prin virgula.

```
Print"\nAccesam dicționarul pentru a vedea ce cuvinte cheie avem:"
print dicționar.keys()
```

Pentru a accesa cheile disponibile putem apela metoda variabila.keys(). Acesta returnează o lista cu toate cheile pe care dicționarul le are.

```
Print"\nAccesam dicționarul pentru a vedea ce valori avem:"
print dicționar.values()
```

Pentru a vedea valorile pe care dicționarul le are putem apela metoda variabila.values(). Aceasta returnează o lista cu toate valorile pe care le are.

```
Print "\nToate elementele dicționarului sunt:"
for elem in dicționar.keys():
    print "cheie:",elem, ">= valoare:",dicționar[elem]
```

Cea mai răspândită prelucrare a unui dicționar este iterarea cu for după cheile dicționarului. Aceasta permite să prelucram fiecare cheie. Pentru a afișa o valoare putem apela variabila["cheie"]. Pentru a scrie o noua valoare folosim:

```
dicționar[cheie_noua]= "valoare noua"
```

Pentru a verifica dacă o cheie este în dicționar exista o metoda mai ușoară decât un for după cheile dicționarului și anume un if ca mai jos:

```
#apelarea directa trebuie testat înainte pentru a nu genera eroare
if "BRB" in dicționar:
    print "\nCe înseamnă BRB:",dicționar["BRB"],"n"
```

Acesta sintaxă are rolul de a ne păzi de erori deoarece dacă dam o cheie care nu exista interpretorul va returna o eroare.

O altă modalitate de a verifica existența unei valori este prin metoda has_key cu se poate vedea mai jos:

```
#Exista cheia Frumi testarea modul alternativ
print "Exista cheia Frumi:",dicționar.has_key("Frumi")

print "Exista cheia FRUMI",dicționar.has_key("FRUMI")
```

Pentru cheia "Frumi" metoda has_key returnează un boolean de tip False deoarece cheia este case-sensitive (sensibil la diferențe alfanumerice de tip litere mari/mici).

Pentru cheia "FRUMI" metoda has_key returnează un boolean de tip True.

Mai exista si o alta modalitate de a cicla printre cheile unui dictionar. Aceasta foloseste un for aplicat direct la dictionar. Asa cum se poate vedea in Fig.50 variabila temporara i va returna pe rand fiecare cheie a dictionarului.

```
>>> x = {1:"test",2:"test2"}
>>> for i in x :
        print i
```

```
1
2
>>> .
```

Fig. 50