Liste

Secventa de elemente

O listă este o secvență de valori care pot avea orice tip. Valorile din cadrul listei se numesc elemente sau itemi.

Crearea unei liste se face utilizând parantezele drepte([ ]):

[10, 20, 30, 40]

['mere', 'pere', 'portocale']

Primul exemplu este o lista de patru valori întregi. A doua listă conține trei șiruri de caractere. Elementele unei liste nu trebuie să aibă același tip neapărat. Umătoarea listă conține un șir de carcatere, un nr. real, un nr. întreg și o altă listă:

['spam', 2.0, 5, [10, 20]]

O listă cuprinsă într-o altă listă se numește imbricată.

O listă care nu conține elemente se numește vidă. Aceasta poate fi create folosind paranteze drepte fără elemente, [].

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [17, 123]
>>> empty = []
>>> print(cheeses, numbers, empty)
['Cheddar', 'Edam', 'Gouda'] [17, 123] []
```

Listele sunt movibile

Pentru a accesa elementele unei liste se folosește aceeași sintaxă ca și pentru tipul string. Expresia din interiorul parantezelor specifică un indice, începând cu 0.

```
>>> print(cheeses[0])
Cheddar
```

Spre deosebire de tipul string, listele sunt movibile, putând să schimbăm ordinea elementelor din listă sau să reatribuim un element listei.

```
>>> numbers = [17, 123]
>>> numbers[1] = 5
>>> print(numbers)
[17, 5]
```

Relația dintre indici și elemente poartă denumirea de mapare. Fiecare indice mapează unul dintre elemente.

Parcurgerea unei liste

Parcurgerea cea mai comună a elementeleor unei liste este utilizând un ciclu for. Sintaxa este aceeași ca și în cazul tipului string.

```
for cheese in cheeses:
        print(cheese)
```

Această metodă funcționează bine când trebuie să citim elementele listei. Dar pentru a scrie sau a actualize elementele, avem nevoie de indici. O metodă este aceea de a combina funcțiile range și len:

```
for i in range(len(numbers)):
        numbers[i] = numbers[i] * 2
```

Un ciclu for executat asupra unei liste vide nu va executa niciodată corpul instrucțiunii:

```
for x in empty:
        print('Nu se accesează.')
```

Cu toate că o listă poate conține altă listă, aceasta va fi considerată ca un singur element. Următoarea listă are lungimea 4:

```
['spam', 1, ['Brie', 'Roquefort', 'Gouda'], [1, 2, 3]]
```

Operații cu liste

Operatorul + concatenează listele:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
```

În mod similar, operatorul * multiplică o listă de un număr de ori:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Felierea listelor

Operatorul slice este folosit și în cadrul listelor:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
>>> t[:4]
['a', 'b', 'c', 'd']
>>> t[3:]
['d', 'e', 'f']
>>> t[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

Datorită faptului că listele sunt movibile, operatorul slice folosit în stânga unei atribuiri, poate să modifice mai multe elemente:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']
>>> print(t)
['a', 'x', 'y', 'd', 'e', 'f']
```

Metode aplicate listelor

Python are definite mai multe metode pentru operarea listelor.

De exemplu, metoda append adaugă un nou element:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print(t)
['a', 'b', 'c', 'd']
```

Metoda extend primește ca și argument o listă și o adaugă unei alte liste:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print(t1)
['a', 'b', 'c', 'd', 'e']
```

Această metodă lasă lista t2 nemodificată.

sort aranjează elementele listei de la mic la mare:

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print(t)
['a', 'b', 'c', 'd', 'e']
```

Cele mai multe metode sunt vide, ele modifică lista și returnează None. Dacă veți scrie t=t.sort(), rezultatul nu va fi unul convenabil.

Ștergerea elementelor

Există mai multe moduri de a șterge elemente dintr-o listă. Dacă cunoaștem indicele elementului pe care dorim să îl ștergem, folosim metoda pop:

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> print(t)
['a', 'c']
>>> print(x)
b
```

pop modifică lista și returnează elemental care a fost șters. Dacă nu folosim un indice, va șterge ultimul element.

Dacă nu dorim ca elementul șters să fie salvat, putem utiliza del:

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> print(t)
['a', 'c']
```

Dacă știm elementul pe care dorim să îl stergem dar nu îi cunoaștem indicele, putem folosi remove:

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> print(t)
['a', 'c']
```

Remove este o metodă vidă.

Pentru a șterge mai mult de un element, putem folosi del cu operatorul slice:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> print(t)
['a', 'f']
```

Liste și funcții

Există o serie de funcții interne ce pot fi folosite asupra listelor:

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
```

```
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25
```

Putem rescrie un program anterior care calculează media unei liste de numere introduse de către utilizator:

Fară liste	Cu liste

total = 0
count = 0
while (True):
  inp = input('Dati un nr: ')
  if inp == 'gata': **break**
  val = float(inp)
  total = total + val
  count = count + 1
media = total / count
print('Media:', media)
	numlist = list()
	while (True):
	  inp = input('Enter a number: ')
	  if inp == 'done': **break**
	  value = float(inp)
	  numlist.append(value)
	media = sum(numlist) / len(numlist)
	print('Media:', media)

Folosim variabilele total și count pentru a memora suma și numărul de elemente introduse.	Folosim o listă vidă numlist, în care adăugăm elemente folosind append. Calculăm suma și lungimea listei folosind funcțiile sum și len.

Liste și șiruri de caractere

O listă de caractere nu este același lucru cu un șir de caractere. Pentru a face conversia între string și lista, folosim list:

```
>>> s = 'spam'
>>> t = list(s)
>>> print(t)
['s', 'p', 'a', 'm']
```

Este bine să evităm atribuirea lui list ca și denumire de variabilă, aceasta fiind o funcție internă. Funcția list separă șirul de caractere în litere. Dacă dorim separarea unei fraze în cuvinte, putem folosi metoda split:

```
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> print(t)
['pining', 'for', 'the', 'fjords']
>>> print(t[2])
the
```

```
>>> s = 'spam-spam-spam'
```

```
>>> delimiter = '-'
>>> s.split(delimiter)
['spam', 'spam', 'spam']
```

Inversul lui split este join. Pentru a folosi join trebuie sa ii asociem un delimitator ce va fi utilizat pentru construirea sirului de caractere.

```
>>> t = ['Ana, 'are', 'mere', 'multe']
>>> delimiter = ' '
>>> delimiter.join(t)
'Ana are mere multe'
```

Parsarea liniilor

Pentru prelucrarea fișierelor putem utliza operațiile cu liste. Având fișierul mbox-short.txt să presupunem că dorim să extragem ziua săptămânii din liniile de forma:

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

Metoda split este foarte eficientă în acest caz:

```
fhand = open('mbox-short.txt')
for line in fhand:
   if not line.startswith('From '): continue
   words = line.split()
   print(words[2])
```

Obiecte și valori

Pentru a verifica dacă două variabile se referă la același obiect, putem folosi operatorul is.

```
>>> a = 'banana'
>>> b = 'banana'
>>> a is b
True
```

În acest exemplu Python a creat un singur obiect string, spre care fac referință ambele variabile.

Atunci însă când avem două liste similare, vor fi create două obiecte:

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
```

În acest caz vom spune că cele două liste sunt echivalente, pentr că au aceleași elemente, însă nu sunt identice, deoarece nu se referă la același obiect.

Dacă a se referă la un obiect și facem atribuirea b=a, atunci ambele variable se vor referi la același obiect:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```

Asocierea unei variabile la un obiect se numește referință. În ultimul exemplu avem două referințe la același obiect.

Dacă obiectul este unul movibil, o modificare a unei variabile o va modifica și pe cea care se referă la același obiect:

```
>>> b[0] = 17
>>> print(a)
[17, 2, 3]
```

Exercițiu:

Scrieți un program care deschide fișierul romeo.txt și îl citește linie cu linie. Pentru fiecare linie, separati linia într-o listă de cuvinte folosind funcția split. Extrageți într-o altă listă cuvinte, astfel încât acestea să nu se repete (Verificați fiecare cuvânt daca este intr-o listă, iar dacă nu apare, adăugați-l). Când programul se termină, sortați și printați cuvintele din listă în ordine alfabetică.

```
Enter file: romeo.txt
['Arise', 'But', 'It', 'Juliet', 'Who', 'already',
'and', 'breaks', 'east', 'envious', 'fair', 'grief',
'is', 'kill', 'light', 'moon', 'pale', 'sick', 'soft',
'sun', 'the', 'through', 'what', 'window',
'with', 'yonder']
```

Dicționare

Un dicționar se aseamănă cu o listă, dar este mai general. Într-o listă, indicele are o valoare întreagă. Într-un dicționar, indicia pot avea aproape orice tip.

Dicționarele pot fi privite ca o mapare între un set de indici (numiți chei) și un set de valori. Fiecare cheie are mapată o valoare. Asocierea dintre cheie și valoare se numește pereche cheie-valoare.

Spre exemplu, putem construi un dicționar care mapează cuvinte din limba engleză în română, atât cheile cât și valorile fiind de tip string.

Funcția dict crează un nou dicționar, fără itemi. Deoarece dict este o funcție internă, trebuie evitat să fie dat ca nume de variabilă.

```
>>> engro = dict()
>>> print(engro)
{}
```

Parantezele acolade, {}, reprezintă un dicționar vid. Pentru a adăuga elemente, vom folosi paranteze drepte:

engro[‘one’]=’unu’

Instrucțiunea crează un item care mapează cheia ‘one’ către o valoare ‘unu’. Dacă vom printa dicționarul din nou, acesta va afișa perechea definită:

```
>>> print(engro)
{'one': 'unu'}
```

Acest format poate fi folosit și ca metodă de input. Spre exemplu, putem crea un dicționar cu trei itemi. Printarea acestuia însă, poate fi surprinzătoare. Perechile nu vor fi neaparat afișate în ordinea în care au fost introduse. Pe calculatoare diferite, ordinea poate fi diferită.

```
>>> engro={'one':'unu', 'two':'doi', 'three':'trei'}
>>> print(engro)
{'one': 'unu', 'three': 'trei', 'two': 'doi'}
```

Elementele unui dicționar nefiind identificate printr-un indice, ordinea lor în cadrul dicționarului nu are importanță.

```
>>> print(engro['two'])
doi
```

Să presupunem că dorim să calculăm numărul de apariții a unei litere într-un text. Putem folosi dicționarul pentru a memora perechile caractere-numar_aparitii. Folosind dictionare nu trebuie sa stim dinainte ce caractere ar putea aparea.

```
word = 'abracadabra'
d = dict()
for c in word:
   if c not in d:
      d[c] = 1
   else:
      d[c] +=1
print(d)
```

Metoda get ne va afișa valoare asociată cheii, în caz contrara afișând valoarea implicită.

```
>>>print (d.get('a',0))
5
```

Programul de mai sus poate fi scris cu ajutorul metodei get, întrucât aceasta gestionează cazul în care cheia nu se află în dicționar:

```
word = 'portocala'
d = dict()
for c in word:
   d[c] = d.get(c,0) + 1
print(d)
```

Dicționare și fișiere

Vom folosi dicționarul pentru a număra aparițiile fiecărui cuvânt într-un fișier. Pentru aceasta ne vom folosi de două cicluri for, pentru liniile din fișier și pentru cuvintele din fiecare linie.

```
fname = input('Nume fisier: ')
try:
        f = open(fname)
except:
        print('Fisierul nu a fost gasit:', fname)
        exit()
counts = dict()
for line in f:
        words = line.split()
        for word in words:
                if word not in counts:
                counts[word] = 1
        else:
                counts[word] += 1
print(counts)
```

Bucle și dicționare

Pentru a parcurge un dicționar putem folosi un ciclu for care parcurge cheile dicționarului:

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for key in counts:
        print(key, counts[key])
```

Dacă dorim să afișam cheile dicționarului in ordine alfabetică, vom crea o listă a cheilor dicționarului, vom sorta acea listă, iar apoi vom parcurge dicționarul folosindu-ne de lista sortată.

```
        counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
        lst = list(counts.keys())
        print(lst)
```

```
lst.sort()
for key in lst:
        print(key, counts[key])
```

Exerciţii

1. Scrieți un program care numără câte emailuri au fost primite în fiecare zi a săptămânii, în fișierul “mbox-short.txt” . Pentru aceasta trebuie identificate liniile care încep cu “From ” și căutat al treilea cuvânt din linie. Un contor va fi incrementat la fiecare apariție a unei zile din săptămână. La final se afișează conținutul dicționarului.

```
Enter a file name: mbox-short.txt
{'Fri': 20, 'Thu': 6, 'Sat': 1}
```

2. Scrieți un program care parcurge fișierul de mai sus și calculează câte emailuri au fost primite de la fiecare adresă, iar apoi afișează continutul dicționarului.

```
Enter file name: mbox-short.txt
{'gopal.ramasammycook@gmail.com': 1, 'louis@media.berkeley.edu': 3,
'cwen@iupui.edu': 5, 'antranig@caret.cam.ac.uk': 1,
'rjlowe@iupui.edu': 2, 'gsilver@umich.edu': 3,
'david.horwitz@uct.ac.za': 4, 'wagnermr@iupui.edu': 1,
'zqian@umich.edu': 4, 'stephen.marquard@uct.ac.za': 2,
    'ray@media.berkeley.edu': 1}
```

Tuple

Definiție

Tuplele reprezintă o secvență de valori, asemănător cu o listă, având proprietatea că sunt însă inamovibile.

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Chiar dacă nu este obligatoriu, se practică includerea tuplei între paranteze rotunde ():

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Pentru a crea o tuplă cu un singur element se folosește sintaxa următoare, ce include o virgulă la final:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

Fără a folosi virgula la final, atribuirea se va face cu o expresie de tip string:

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
```

O altă modalitate este de a utiliza funcția tuple:

```
>>> t=tuple('abracadabra')
>>> print(t)
('a', 'b', 'r', 'a', 'c', 'a', 'd', 'a', 'b', 'r', 'a')
```

Cei mai mulți operatori ai listelor funcționează și pentru tuple.

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print(t[0])
'a'

>>> print(t[1:3])
('b', 'c')
```

Dacă încercăm să modificăm unul din elementele tuplei, ni se va afișa o eroare:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

Tupla poate fi înlocuita cu o alta:

```
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c', 'd', 'e')
```

Compararea tuplelor

Operatorii de comparație funcționează cu tuple. Python compara primul element din fiecare secvență. Dacă sunt egale, trece mai departe, element cu element, până când găsește prima diferență. Următoarele elemente nu mai sunt luate în considerare.

```
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
```

Funcția de sortare funcționează similar. Sortează întâi primul element, dar în caz de egalitate trece la următorul.

Exemplu: Sortarea unei liste de cuvinte, de la cel mai lung la cel mai scurt.

```
txt = “Ana are mere mari”
words = txt.split()
t = list()
for word in words:
        t.append((len(word), word))
t.sort(reverse=True)
res = list()
for length, word in t:
        res.append(word)
print(res)
```

Prima buclă construiește o listă de tuple, fiecare tuplă fiind un cuvânt precedat de lungimea sa.

sort compară primul element (lungimea) și numai în caz de egalitate și cuvântul. Cuvântul cheie reverse=True realizează sortarea în ordine descrescătoare.

Al doilea for parcurge lista de tuple și construiește o listă de cuvinte în ordine descrescătoare a lungimii.

Cuvintele de patru caractere vor apărea în ordine invers alfabetică, datorită sortării cu reverse=True.

Atribuirea tuplelor

O caracteristică specifică limbajului Python este aceea de a putea utiliza o tuplă în partea stângă a atribuirii. Acest lucru dă posibilitatea atribuirii a mai mult de o variabilă la un moment dat.

Exemplu:

```
>>> m = [ 'have', 'fun' ]
>>> x, y = m
>>> x
'have'
>>> y
'fun'
```

Este similar cu următoarele:

```
>>> m = [ 'have', 'fun' ]
>>> x = m[0]
>>> y = m[1]
>>> x
'have'
>>> y
'fun'
```

O altă aplicație, este aceea de a inversa valorile a două variabile într-o singură declarație:

```
>>> a, b = b, a
```

De exemplu, pentru a extrage numele si domeniul dintr-o adresa de email, putem folosi urmatoarele instructiuni:

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
```

Dicționare și tuple

Dicționarele au o metodă denumită items care returnează o listă de tuple, unde fiecare tuplă este o pereche cheie-valoare:

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = list(d.items())
>>> print(t)
[('b', 1), ('a', 10), ('c', 22)]
```

Folosind transformarea dicționarului în listă de tuple, putem sorta conținutul dicționarului:

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = list(d.items())
>>> t
[('b', 1), ('a', 10), ('c', 22)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

Deasemenea, prin combinarea tuplelor și a ciclului for, ptutem realiza o traversare într-un singur ciclua dicționarului:

```
for key, val in list(d.items()):
        print(val, key)
```

Exercitii

Realizati un program care calculeaza distributia emailurilor din fisierul mbox-short.txt in functie de ora, pentru fiecare mesaj transmis. Ora poate fi extrasa din linia ce incepe cu “From” prin identificarea sirului de caractere ce contine ora si apoi impartirea lui folosind caracterul “:”. Dupa ce ati numarat emailurile pe fiecare ora, afisati numarul lor sortat in functie de ora, asa cum arata in exemplul de mai jos:

```
Enter a file name: mbox-short.txt
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
```

Bibliografie:

Charles R. Severance, Python for Everybody – Exploring Data Using Python 3, 2016, www.py43.com