Programarea calculatoarelor si limbaje de programare 1

Tuple, Fisiere, Instructiuni

Universitatea Politehnica din Bucureşti

Sumar

- ***Tuple***
- Fisiere
- Instructiuni, introducere

tuple

- Definitie: sunt colectii fixe de obiecte, reprezentate intre paranteze rotunde
- Proprietati:
  - Colectie ordonata pozitional de obiecte arbitrare
  - Accesibile prin indexare, slicing
  - Sunt secvente imuabile – nu accepta modificari in-place
  - Suporta incluziunea pe oricate nivele
  - Implementate ca vectori de referinte catre obiectele componente

tuple, expresii

Expresie *tuple*	Semnificatie	Expresie *tuple*	Semnificatie
()	Tuplu vid	len(t)	Lungime
t =(0,)	Tuplu, 1 element	t1 + t2	Concatenare
t=(0,'Hi',1.2,3)	Tuplu, 4 elemente	t * 4	Repetitie
t=0,'Hi',1.2,3	Tuplu,fara paranteze	for x in t: print(x)	Iterare
t=('Ion', ('ing','dir'))	Incluziune	'spam' in t	Apartenenta
t=tuple('spam')	Tuplu din iterabil	t.index('Hi')	Metoda, pozitie
t[i]	Indexare	t.count('Hi')	Metoda,cate aparitii
t[i][j]	Index de index	namedtuple('ang', ['nume','jobs'])	namedtuple, extensie
t[i:j]	Slicing		

Reprezentare, *tuple*

- Tuple fara paranteze:

```
>>> x = 1, 2, 3, 4          #tuple
>>> x
(1, 2, 3, 4)
```

- Parantezele sunt necesare in contexte in care parantezele – ex. apel de functie – sau virgulele – ex. intr-o lista sau dictionar – conteaza.

Operatii secventiale, *tuple*

- Tuplele suporta operatii specifice secventelor:

```
>>> (1, 2) + (3, 4)          #concatenare
(1, 2, 3, 4)
>>> (1, 2) * 3               #repetitie
(1, 2, 1, 2, 1, 2)
>>> t = (1, 2, 3, 4)
>>> t[0], t[1:3]             #indexare, slicing
(1, (2, 3))
```

- Tuple cu un singur element: *(e,)*

```
>>> x = (33)    #numar intreg
>>> x
33
```

```
>>> y = (33,)   #tuplu
>>> y
(33,)
```

Metode, *tuple*

- Tuplele nu suporta toate metodele de la str, list, dict:

```
>>> t = ('cc', 'aa', 'dd', 'bb')
>>> t.sort()                    #metoda inexistenta
AttributeError: 'tuple' object has no attribute 'sort'
>>> l = list( t )               #conversie intermediara la list
>>> l.sort()                    #list are metoda sort()
>>> t = tuple( l )
>>> t                           #tuple sortat
('aa', 'bb', 'cc', 'dd')
>>> t = tuple( sorted( t ) )    #cu functia predefinita sorted() - returneaza list!
>>> t
('aa', 'bb', 'cc', 'dd')
```

Metode...

```
>>> t = (1, 2, 3, 2, 4, 2)
>>> t.count( 2 )            #numarul de aparitii (ale lui 2) in tuplu, cu count()
3
>>> t.index( 2 )      #pozitia primei aparitii, cu index()
1
>>> t.index( 2, 1 + 1 )   #pozitia urmatoarei aparitii (dupa 1)
3
>>> t.index( 2, 3 + 1 )    #pozitia urmatoarei aparitii (dupa 3)
5
```

Iteratii, imuabilitate

```
>>> t = (1, 2, 3, 4, 5)
>>> (x ** 2 for x in t)          #cu paranteze rotunde - generator!
<generator object <genexpr> at 0x00000274F625DAC8>
>>> [x ** 2 for x in t]          #cu paranteze patrate - lista!
[1, 4, 9, 16, 25]
>>> tuple(x ** 2 for x in t)     #constructor aplicat unui iterabil - tuplu!
(1, 4, 9, 16, 25)
```

- Imuabilitate – doar pe primul nivel:

```
>>> t = (1, [2, 3], 4)
>>> t[1] = 'spam'
TypeError: 'tuple' object does not support item assignment
>>> t[1][0] = 'spam'      #modificare element 0 din lista, da!
>>> t
(1, ['spam', 3], 4)
```

namedtuple

- Inregistrarile se pot reprezenta cu *dict* pentru acces prin cheie:

```
>>> ion = dict( nume='Ion', ani=33, jobs=['ing', 'dir'] )      #dict
>>> ion
{'nume': 'Ion', 'ani': 33, 'jobs': ['ing', 'dir']}
>>> ion['nume'], ion['jobs']                #indexare cu chei
('Ion', ['ing', 'dir'])
```

- Record cu *tuple* pentru acces pozitional:

```
>>> ion = ('Ion', 33, ['ing', 'dir'])          #tuple
>>> ion
('Ion', 33, ['ing', 'dir'])
>>> ion[0], ion[2]                          #indexare pozitionala
('Ion', ['ing', 'dir'])
```

namedtuple...

- Acces pozitional si prin denumire, cu *namedtuple* – o subclasa a tipului *tuple*:

```
>>> from collections import namedtuple  #import al functiei namedtuple
>>> Rec = namedtuple( 'inrg', ['nume', 'ani', 'jobs'])      #clasa noua, Rec
>>> ion = Rec( 'Ion', ani=33, jobs=['ing', 'dir'])
>>> ion
inrg(nume='Ion', ani=33, jobs=['ing', 'dir'])
>>> ion[0], ion[2]                  #acces pozitional
('Ion', ['ing', 'dir'])
>>> ion.nume, ion.jobs          #acces prin nume de atribut
('Ion', ['ing', 'dir'])
```

namedtuple...

```
>>>#acces via dict:
>>> od = ion._asdict()          #_asdict() returneaza un dictionar
>>> od
{'nume': 'Ion', 'ani': 33, 'jobs': ['ing', 'dir']}
>>> type( od )
<class 'dict'>
>>> od['nume'], od['jobs']
('Ion', ['ing', 'dir'])

>>> #cu atribuire de tuple:
>>> ion = Rec('Ion', 40.5, ['ing', 'dir'])
>>> (nume, ani, jobs) = ion  #atribuire de secvente
>>> nume, jobs
('Ion', ['ing', 'dir'])
```

Sumar

- Tuple
- ***Fisiere***
- Instructiuni, introducere

Fisiere

- Definitie: fisierele administrate de catre sistemul de operare sunt accesate cu un obiect returnat de catre functia predefinita ***open()***, ale carui metode permit transferul de date, sub forma de *str* sau *bytes*, cu aceste fisiere.
- Alte metode realizeaza repozitionarea, trunchierea, actualizarea buffer-elor de scriere pe disc, etc.

Operatii cu fisiere

Operatie	Semnificatie
o = open(r'c:\spam', 'w')	Creare fisier pentru scriere
i = open('data', 'r')	Creare fisier pentru citire
i = open('data')	Idem, 'r' este implicit
s = i.read()	Citire intreg fisier in *str*-ul s
s = i.read(n)	Citire n caractere/bytes
s = i.readline()	Citire linie (inclusiv \n)
L = i.readlines()	Citire toate liniile in *list*-ul L
o.write(s)	Scriere s (*str*/*bytes*)
o.writelines(L)	Scriere L(list de str)
o.close()	Inchidere fisier
o.flush()	Actualizare pe disc
f.seek(n)	Pozitionare la ofsetul n

Deschiderea fisierelor

Operatie	Semnificatie
for linie in open('data'):....	Iterare per linie
open('f.txt',encoding='latin-1')	Fisier Unicode, *str*
open('f.bin', 'rb')	Fisier binar, *bytes*

Deschiderea fisierelor se face cu functia *open()*:

>>> **f = open(numefisier, mod, ...)**

>>> **f.metoda()** #operatie cu fisierul

unde:

- numefisier, *str*, nume extern absolut sau relativ
- mod, *str,* 'r', 'w', 'r+', un 'b' pentru binary
- alte args pentru buffer-ing, codificare

Recomandari, fisiere

- Citirea liniilor – cu iteratori
- Citirea/scrierea se face exclusiv via *str*, cu conversii ulterioare la/de la obiecte.
- Implicit accesul este via buffere, actualizate cu *close()* sau *flush()* (fara close)
- Repozitionarea, cu seek(), offset in bytes
- Inchiderea, *close()* este optionala, se face automat la incheierea executiei sau cand obiectul este *gc*; se recomanda!

Utlilizare fisiere

```
>>> f = open( 'file.txt', 'w' )          #creare fisier local fisier.txt
>>> f.write( 'Hello, World!\n' )         #scriere o linie - str, intotdeauna
14
>>> f.write( 'So long, goodbye!\n' )     #inca un str scris, ambele cu \n
18
>>> f.close()                            #inchidere/actualizare pe disc
>>> f = open( 'file.txt' )               #deschidere/citire, 'r' este implicit
>>> f.readline()                         #citire linie, rezultat tot str
'Hello, World!\n'
>>> f.readline()
'So long, goodbye!\n'
>>> f.readline()                         #str vid, sfarsit fisier!
''
```

Utilizare...

```
>>> open( 'file.txt' ).read()          #citire intreg fisier, intr-un str
'Hello, World!\nSo long, goodbye!\n'
>>> print( open( 'file.txt' ).read() )  #cu print, interpreteaza secventele escape,\n
Hello, World!
So long, goodbye!
```

Fisier binar, se acceseaza *bytes*, nu *str*

```
>>> f = open( 'file.bin', 'wb' )
>>> d = b'\x00\x00\x00\x07spam\x00\x08'
>>> f.write( d )
10
>>> f.close()
>>> d = open( 'file.bin', 'rb' ).read()
>>> d
b'\x00\x00\x00\x07spam\x00\x08'
>>> d[4:8]
b'spam'
>>> d[4:8][0]  #int
115
>>> bin( d[4:8][0] )  #reprezentare binara
'0b1110011'
```

Serializarea obiectelor, conversii

```
>>> x, y, z = 1, 2, 3        #int, ob. nativ
>>> s = 'spam'               #str
>>> d = {'a':1, 'b':2}       #dict
>>> l = [11, 22, 33]         #list
>>> f = open( 'date.txt', 'w' ) #scriere
>>> f.write( s + '\n' )      #str cu \n
>>> f.write( '%s,%s,%s\n' % (x, y, z) )
>>> f.write( str(l) + '$' + str(d) + '\n' )
>>> f.close()
>>> open( 'date.txt' ).read()  #tot fisierul
"spam\n1,2,3\n[11, 22, 33]${'a': 1, 'b': 2}\n"
>>> f = open( 'date.txt' )       #citire
>>> linie = f.readline()         #prima linie
>>> linie        #str
'spam\n'
>>> linie.rstrip()    #reconstituit, str!
'spam'
>>> linie = f.readline()
>>> linie
'1,2,3\n'
>>> x, y, z = [int(s) for s in linie.split(',')]
      #int() ignora \n
>>> x, y, z      #reconstituit, int
(1, 2, 3)
>>> linie = f.readline()
>>> linie        #cu eval()!
"[11, 22, 33]${'a': 1, 'b': 2}\n"
>>> ob = [eval(s) for s in linie.split('$')]
>>> ob
[[11, 22, 33], {'a': 1, 'b': 2}]
```

Serializare cu *pickle*

- **Modulul *pickle* face automat serializarea obiectelor**

```
>>> d = {'a':1, 'b':2}                  #dict, ob. nativ
>>> f = open( 'date.pkl', 'wb' )        #scriere, binar!
>>> import pickle                       #import modul pickle
>>> pickle.dump( d, f )                 #dump()
>>> f.close()
>>> f = open( 'date.pkl', 'rb' )        #citire, binar
>>> dd = pickle.load( f )               #load()
>>> dd
{'a': 1, 'b': 2}
>>> open( 'date.pkl', 'rb' ).read()     #format binar, 28 bytes...
b'\x80\x03}q\x00(X\x01\x00\x00\x00aq\x01K\x01X\x01\x00\x00\x00bq\x02K\x02u.'
```

Serializare cu JSON

- **JSON – Java Script Object Notation – apropiat de cod Python:**

```
>>> nume = dict( pre='Ion', fam='Ionescu' )                    #dict
>>> rec = dict( nume=nume, job=['ing', 'dir'], ani=33 )        #incluziune
>>> rec
{'nume': {'pre': 'Ion', 'fam': 'Ionescu'}, 'job': ['ing', 'dir'], 'ani': 33}
>>> import json                    #import modul json
>>> json.dumps( rec )              #obiect ➔ str JSON, cu dumps()
'{"nume": {"pre": "Ion", "fam": "Ionescu"}, "job": ["ing", "dir"], "ani": 33}'
>>> ob = json.loads( json.dumps( rec ) )    #str JSON ➔ obiect Python
>>> ob
{'nume': {'pre': 'Ion', 'fam': 'Ionescu'}, 'job': ['ing', 'dir'], 'ani': 33}
>>> ob == rec
True
```

JSON...

```
>>> json.dump( rec, open( 'date.json', 'w' ), indent=4 )    #dump() ➔ stream/file
>>> print( open( 'date.json' ).read() )          #afisare fisier JSON
{
    "nume": {
        "pre": "Ion",
        "fam": "Ionescu"
    },
    "job": [
        "ing",
        "dir"
    ],
    "ani": 33
}
```

Date binare cu *struct*

- **Modulul *struct* impacheteaza/despacheteaza date binare – ex. din retea, scrise de alt program (in C)**

```
>>> f = open( 'date.bin', 'wb' )
>>> import struct  #import modul struct
>>> #impachetare, big-endian,int,str,short
>>> bd = struct.pack( '>i4sh', 7, b'spam', 8 )
>>> bd
b'\x00\x00\x00\x07spam\x00\x08'
>>> f.write( bd )
10
>>> f.close()
```

```
>>> f = open( 'date.bin', 'rb' )
>>> bd = f.read()
>>> bd
b'\x00\x00\x00\x07spam\x00\x08'
>>> #despachetare,acelasi format:
>>> obs = struct.unpack( '>i4sh', bd )
>>> obs
(7, b'spam', 8)
```

Manageri de context

- Efectueaza operatii cu fisiere, asigurand inchiderea/actualizarea pe disc, eliberarea resurselor de sistem, fara *gc* (garbage collection)

```
>>> with open( 'date.txt' ) as f:
        for linie in f:
            ...prelucrari...
```

- Idem, cu instructiunea *try/finally*, mai generala:

```
>>> f = open( 'date.txt' )
>>> try:
        for linie in f:
            ...prelucrari...
    finally:
        f.close()
```

Recapitulare, tipuri de date

Tipuri de obiecte	Categorie	Modificabil
Numere – int,float,complex,Decimal,Fraction	Numeric	Nu
str, bytes	Secventa	Nu
list	Secventa	Da
dict	Mapare/asociere	Da
tuple	Secventa	Nu
Fisiere	Extensie	N/A
set	Multime	Da
frozenset	Multime	Nu
bytearray	Secventa	Da

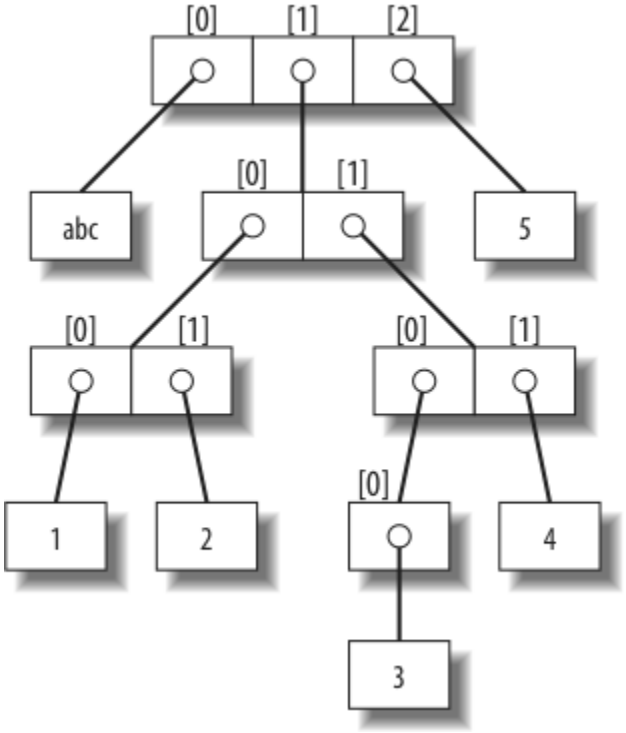
- Obiectele suporta operatii specifice categoriei din care fac parte, dar si metode specifice tipului caruia ii apartin

Flexibilitate

- *list*, *dict*, *tuple* – pot contine obiecte de orice tip
- *set* – contine orice obiect fix/imuabil
- *list*, *dict*, *tuple* – pot fi incluse pe oricate nivele
- *list*, *dict*, *set* – pot fi modificate (cresc/scad) dinamic

Incluziune

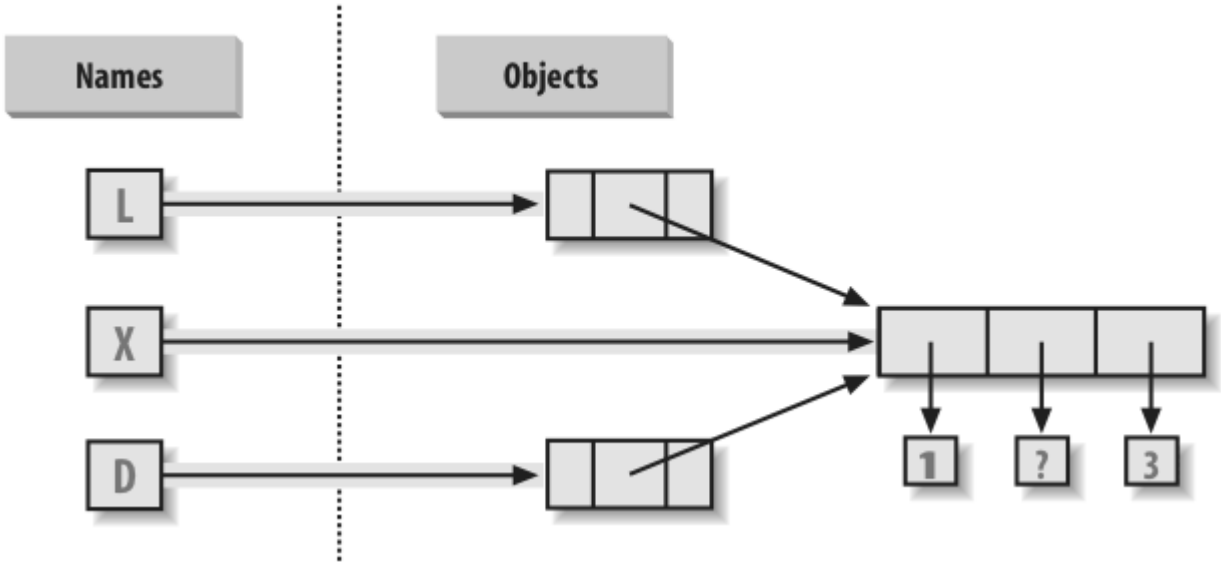
```
>>> L = ['abc', [(1, 2), ([3], 4)], 5]
>>> L[1]        #indexari in adancime
[(1, 2), ([3], 4)]
>>> L[1][1]
([3], 4)
>>> L[1][1][0]
[3]
>>> L[1][1][0][0]
3
```



Referinte vs. copii

```
>>> X = [1, 2, 3]          #L, D contin referinte catre X
>>> L = ['a', X, 'b']      #L, D afectate de modificari ale lui X
>>> D = {'x': X, 'y':2}
>>> X[1] = 'ceva'
>>> L
['a', [1, 'ceva', 3], 'b']
>>> D
{'x': [1, 'ceva', 3], 'y': 2}

>>> #Copieri:
>>> X = [1, 2, 3]
>>> L = ['a', X[:], 'b']
>>> D = {'x': X[:], 'y': 2}
```



Copiere

Copiere, in loc de referire, se face cu:

- Expresii *slice*, cu limite neprecizate, **L[:]**
- Metoda ***.copy()*** de la *dict*, *set*, *list*
- Functii predefinite **list(L)**, **dict(D)**, **set(S)**
- Functiile *copy()*, *deepcopy()* din modulul standard **copy**

```
>>> L = [1, 2, 3]
>>> D = {'a': 1, 'b': 2}
>>> LL = L[:]  #slicing
>>> DD = D.copy() #metoda copy()
>>> LL[1] = 'Hi'
>>> DD['c'] = 'spam'
```

```
>>> L, LL  #doar copia este modificata
([1, 2, 3], [1, 'Hi', 3])
>>> D, DD  #idem
({'a': 1, 'b': 2}, {'a': 1, 'b': 2, 'c': 'spam'})
>>> import copy  #import modulul copy
>>> Y = copy.deepcopy( X )
```

Comparatii

- In Python, comparatia este recursiva – de la stanga la dreapta si in adancime
- Operatorul **==** testeaza egalitatea de valori
- Operatorul ***is*** testeaza identitatea de obiecte (aceeasi referinta de memorie)

```
>>> L1 = [1, ('a', 3)]  #aceeasi valoare
>>> L2 = [1, ('a', 3)]  #obiecte diferite
>>> L1 == L2, L1 is L2
(True, False)
```

```
>>> s1 = 'spam'          #caching pentru
>>> s2 = 'spam'          #str mic, deci
>>> s1 == s2, s1 is s2   #acelasi obiect
(True, True)
```

Comparatii...

- Numerele se compara dupa conversia la tipul mai cuprinzator
- *str* se compara lexicographic (pozitia returnata de *ord()*) – 'abc' < 'ac'
- *list* si *tuple* se compara recursiv – [2] > [1, 2]
- *dict* sunt egale daca listele de chei/valori sortate sunt egale; comparatii nesuportate in 3.X
- Comparatii nenumerice cu tipuri mixte sunt **erori** in 3.X (ex. 1 < 'spam')

Comparatii...

```
>>> 11 == '11'        #Egalitate, da
False
>>> 11 >= '11'        #Comparatie, NU, intre tipuri mixte
TypeError: '>=' not supported between instances of 'int' and 'str'
>>> 11 > 9.123        #Comparatie, da, intre tipuri numerice diferite
True
>>> str( 11 ) >= '11', 11 >= int( '11' )    #Cu conversii prealabille, da!
(True, True)
>>> D1 = {'a': 1, 'b': 2}          #dict
>>> D2 = {'a': 1, 'b': 3}
>>> D1 == D2                       #egalitate, da
False
>>> D1 < D2                        #comparatie, NU!
TypeError: '<' not supported between instances of 'dict' and 'dict'
```

dict, comparatii

- In Python 3.X, comparatia dictionarelor se face cu sortari de **dict.items():**

```
>>> list( D1.items() )
[('a', 1), ('b', 2)]
>>> sorted( D1.items() )
[('a', 1), ('b', 2)]
>>> sorted( D1.items() ) < sorted( D2.items() )
True
>>> sorted( D1.items() ) > sorted( D2.items() )
False
```

True, *False*, *None*, *bool*

Obiect	Valoare True/False
'spam'	True
''	False
[1, 2]	True
[]	False
{'a': 1}	True
{}	False
1	True
0.0	False
None	False

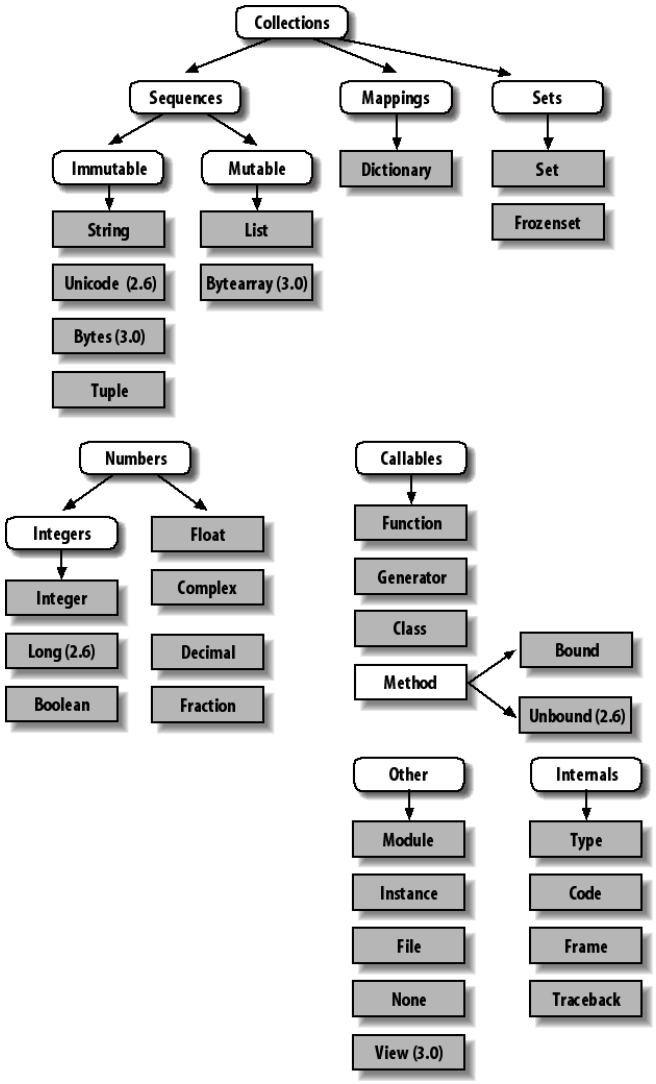
- Numerele nenule sunt *True*, altfel *False*
- Alte obiecte sunt *False* daca vide, altfel *True*
- Obiectul **None** este *False*; se foloseste pentru initializari, prealocari de spatiu:

```
>>> L = [None] * 4
>>> L
[None, None, None, None]
```

- Tipul **bool**, subclasa a lui *int*, cu doua instante, *True* si *False*, echivalentele lui 1 si 0

```
>>> bool( 0 ), bool( 1 )  #functie predef. bool()
(False, True)
>>> bool( 'spam' ), bool( {} )
(True, False)
```

Clasificare, tipuri de obiecte



- Si tipurile sunt tipuri de obiecte!

Exemple cu func. predef. type():

>>> type([1]) == type([]) #acelasi tip, list

True

>>> type([1]) == list #nume tip, list

True

>>> isinstance([1], list) #instanta list

True

>>> import *types* #nume pt. alte tipuri

>>> def f(): pass #o functie

>>> type(f) == types.*FunctionType* #nume de tip nepredefinit

True

Repetitia, cu referinte

- Un exemplu interesant:

```
>>> L = [1, 2, 3]
>>> X = L * 3                    #repetitie obisnuita
>>> Y = [L] * 3                  #o lista cu o referinta la L, repetata
>>> X
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> Y
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
>>> L[1] = 9                     #modificare L
>>> X                            #X neafectat
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> Y                            #Y modificat, contine referinte la L
[[1, 9, 3], [1, 9, 3], [1, 9, 3]]
```

Sumar

- Tuple
- Fisiere
- ***Instructiuni, introducere***

Instructiuni, introducere

Python este un limbaj procedural, bazat pe instructiuni – combinate pentru a specifica o procedura care indeplineste scopul programului.

Ierarhia Python – reamintire:

- Programele sunt formate din **module.**
- Modulele contin **instructiuni**.
- Instructiunile contin **expresii**.
- Expresiile creeaza si proceseaza **obiecte**.

Instructiuni Python

Instructiune	Rol	Exemplu
Atribuirea	Creeaza referinte	a, b = 1, 2
Apeluri	Executa functii	log.write('spam')
Apel print	Afiseaza obiecte	print('spam')
if/elif/else	Selectie	if 'python' in text: print(text)
for/else	Iteratie	for x in L: print(x)
while/else	Repetitie	while x > y: print('Hi!')
pass	Instructiune vida	while True: pass
break	Iesire din cicluri	while True: if exittest(): break
continue	Continua ciclul	while True: if skiptest(): continue

Instructiuni...

Instructiune	Rol	Exemplu
def	Functii, metode	def f(a, b, c=1, *d): print(a+b+c+d[0])
return	Rezultat functie	def f(a, b, c=1, *d): return a+b+c+d[0]
yield	Functie generator	def gen(n): for i in n: yield i*2
global	Spatiu nume	x = 'old' def f(): global x; x='new'

nonlocal	Spatiu nume (3.X)	def outer():
		  x = 'old'
		  def function():
		  nonlocal x; x = 'new'

import	Acces la module	import sys
from	Acces la atribute	from sys import stdin

Instructiuni...

Instructiune	Rol	Exemplu

class	Construire obiecte	class Subclass(Superclass):
		  staticData = []
		  def method(self): pass

try/except/finally	Tratare exceptii	try:
		  action()
		except:
		  print('action error')

raise	Declansare exceptie	raise EndSearch(location)
assert	Verificari, debugger	assert x>y, 'x too small'
with/as	Manageri de context	with open('data') as f: process(f)
del	Stergere referinte	del data[x] del data[i:j] del obiect.atribut del variabila

Sintaxa instructiunilor

- Fata de limbaje ca *C*, Python foloseste doua puncte ***:*** in instructiuni compuse

Antet Instructiune Compusa:

Bloc cu Instructiuni Indentate

- Omiterea simbolului ***:*** este o eroare frecventa la inceput.
- Parantezele ***()*** sunt optionale – **if x < y:**
- Sfarsitul liniei marcheaza sfarsitul instructiunii – **x = 1** asa incat ***;*** nu mai este necesar
- Blocul/instructiunile succesive, egal indentate, se incheie prin revenirea la indentarea anterioara

Exceptii, sintaxa...

- Indentarea, in orice limbaj, contribuie la usurarea citirii si scrierii programelor; in Python este obligatorie.
- Exceptii sintactice:
  - ***;*** se poate folosi pentru separarea instructiunilor simple
  - Parantezele ***(), [], {}*** permit extinderea unei expresii pe mai multe randuri, fara restrictii de indentare
  - Blocul unei instructiuni compuse se poate scrie pe acelasi rand cu antetul sau, daca e alcatuit din instructiuni simple

Ex. 1 de program

```
while True:
    raspuns = input( 'Introduceti text: ' )   #citire cu input()
    if raspuns == 'gata': break
    print( raspuns.upper() )
```

Introduceti text: asdf

ASDF

Introduceti text: 123

123

Introduceti text: gata

Ex. 2, calculator

- Se testeaza prezenta cifrelor cu *isdigit()*

```
while True:
    raspuns = input( 'Introduceti text: ' )
    if raspuns == 'gata':
        break
    if not raspuns.isdigit():
        print( 'Gresit!' * 3 )
    else:
        print( int( raspuns ) ** 2 )
```

```
Introduceti text: 3
9
Introduceti text: dsa
Gresit!Gresit!Gresit!
Introduceti text: 10
100
Introduceti text: gata
```

Ex. 3, cu try/except

- *try/except/else* sunt aliniate, reprezinta o instructiune compusa
- blocul dupa *try*: este actiunea care se probeaza
- blocul dupa *except*: este modul de tratare al eventualei erori
- blocul dupa *else*: este executat daca nu sunt erori captate

```
while True:
    raspuns = input( 'Introduceti text: ' )
    if raspuns == 'gata':
        break
    try:
        nr = int( raspuns )
    except:
        print( 'Gresit!' * 3 )
    else:
        print( nr ** 2)
```

```
Introduceti text: 10
100
Introduceti text: qwe
Gresit!Gresit!Gresit!
Introduceti text: gata
```

Ex. 3, mai concis

- Pe ramura *try* se afla intraga executie, putandu-se introduce date de intrare variate, cat si greseli, captate la rulare:

```
while True:
    raspuns = input( 'Introduceti text: ' )
    if raspuns == 'gata':
        break
    try:
        print( float(raspuns ) ** 2 )
    except:
        print( 'Gresit!' * 3 )
```

```
Introduceti text: 22.7
515.29
Introduceti text: 1.23e-3
1.5129e-06
Introduceti text: spam
Gresit!Gresit!Gresit!
Introduceti text: gata
```

Ex. 4, indentare pe 3 nivele

- Se indenteaza spre dreapta, inca un nivel:

```
while True:
    raspuns = input( 'Introduceti text: ' )
    if raspuns == 'gata':
        break
    if not raspuns.isdigit():
        print( 'Gresit!' * 3 )
    else:
        nr = int( raspuns )
        if nr < 20:
            print( 'Prea mic' )
        else:
            print( nr ** 2 )
```

```
Introduceti text: 15
Prea mic
Introduceti text: 22
484
Introduceti text: spam
Gresit!Gresit!Gresit!
Introduceti text: gata
```