Programarea calculatoarelor si limbaje de programare 1

Tipuri de obiecte in Python

Universitatea Politehnica din Bucureşti

Sumar

- ***Clasificare obiecte in Python***
- Numere
- Stringuri
- Liste
- Dictionare
- Tuple
- Fisiere
- Alte tipuri de baza

Tipuri de date – obiect

- In Python, toate tipurile de date sunt de tip **obiect**:
  - Obiecte predefinite (built-in)
  - Obiecte nou create de noi folosind clase sau extensii – ex. biblioteci *C*
- Chiar si numerele (ex. 33) sunt obiecte, avand valori si operatii asociate (adunare, scadere, etc)

Componente program – ierarhie

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

Fata de modelul *traditional*, bazat pe succesiunea, decizia si repetitia instructiunilor, modelul Python este *bazat pe obiecte* si modul lor de folosire.

Importanta obiectelor predefinite

Obiectele predefinite:

- Usureaza scrierea programelor, putand fi utilizate imediat (ex. liste – colectii, dictionare – cautare)
- Stau la baza claselor de obiecte nou create.
- Sunt mai eficiente decat obiectele proiectate ad-hoc (folosesc algoritmi optimizati, implementati in C – pt. viteza)
- Sunt standard, nu difera in implementare, spre deosebire de alte sisteme (ex. frameworks in C++)

Tipuri de baza

Obiect	Expresii asociate
Numere	1234, 3.14156, 3+4j, 0b111, Decimal(), Fraction()
String-uri	'spam', "Who's", b'a\x01c', u'sp\xc4m'
Liste	[1, [2, 'trei'], 4.5], list(range(10))
Dictionare	{'hrana' : 'spam', 'gust' : 'bun'}, dict(ore=2)
Tuple	(1, 'spam', 4, 'U'), tuple('spam'), namedtuple
Fisiere	open('date.txt'), open(r'C:\date.bin', 'wb')
Multimi	set('abc'), {'a', 'b', 'c'}
Alte tipuri de baza	Boolean, type, None
Unitati de program	Functii, module, clase (create cu def, import, class)
Implementari	Cod compilat, traceback

Python are *tipuri dinamice* – care nu trebuie declarate, si *tipuri bine definite (strongly typed)* – numai operatiile valide se pot executa pentru un tip dat.

Sumar

- Clasificare obiecte in Python
- ***Numere***
- Stringuri
- Liste
- Dictionare
- Tuple
- Fisiere
- Alte tipuri de baza

Numere

- Exista obiecte de tip *intreg*, in virgula mobila – cu **punct** zecimal, numere *complexe* – cu parte imaginara, *decimale* – cu precizie fixa, *rationale* – cu numarator si numitor si *multimi* – set.
- Operatorii matematici obisnuiti: +, -, *, ** (ridicare la putere), /, % (rest impartire).

Numere – exemple

```
>>> 123 + 456        #adunare intregi
579
>>> 1.5 * 3          #inmultire numere zecimale
4.5
>>> 2 ** 100         #2 la puterea 100
1267650600228229401496703205376
>>>  len( str( 2 ** 1000000 ) )   #cate cifre are 2 la puterea 1 milion
301030
>>> print( 3.14156 * 2 )      #afisare numere cu print
6.28312
```

Numere…

>>> **3.14156 * 2**

6.28312

Folosirea modulelor numerice existente:

>>> **import math**

>>> **math.pi**

3.141592653589793

>>> **math.sqrt(91)**

9.539392014169456

>>> **import random**

>>> **random.random()**

0.548287451468251

>>> **random.choice([1, 2, 3, 4 , 5, 6, 7])**

5

>>>

Sumar

- Clasificare obiecte in Python
- Numere
- ***Stringuri***
- Liste
- Dictionare
- Tuple
- Fisiere
- Alte tipuri de baza

String-uri

Reprezinta text; mai general sunt *secvente* – o colectie de obiecte ordonate pozitional (litere). Alte secvente: liste si tuple.

Operatii cu secvente:

```
>>> s = 'Spam'      #string de 4 litere, atribuire variabilei s
>>> len( s )        #lungimea stringului
4
>>> s[0]            #indexare, primul element este in pozitia zero
'S'
>>> s[1]            #al doilea element, de la stanga la dreapta
'p'
```

Operatii pe stringuri

```
>>> s[-1]              #ultimul element
'm'
>>> s[-2]              #penultimul, in pozitia len(s) - 2
'a'
>>> s[len( s ) - 1]    #mai simplu este s[-1]
'm'
>>> s[1:3]             #slicing (feliere), zona 1 pana la 3 - fara 3
'pa'
>>> s[1:]              #slicing, toate elementele incepand cu pozitia 1
'pam'
>>> s[0:3]             #slicing, tot, pana la pozitia 3 exclusiv
'Spa'
```

Operatii…

```
>>> s[:3]          #slicing, la fel
'Spa'
>>> s[:-1]         #slicing, si mai simplu
'Spa'
>>> s[:]           #slicing, copiere string
'Spam'
>>> s + 'xzy'      #concatenare cu +
'Spamxzy'
>>> s              #s este neschimbat – fix, imuabil
'Spam'
>>> s * 4          #repetitie, de patru ori, cu *
'SpamSpamSpamSpam'
```

Obiecte fixe (imuabile)

Stringurile, numerele, tuplele sunt imuabile; listele, dictionarele si multimile sunt modificabile.

>>> s[0] = 'Z' #stringul este fix, nu poate fi schimbat

TypeError: 'str' object does not support item assignment

>>> **s = 'Z' + s[1:] #expresie care genereaza un obiect nou**

>>> **s**

'Zpam'

>>>

Modificari ale unui string cu **list** sau ***bytearray:***

Obiecte fixe…

```
>>> s = 'mare'
>>> l = list( s )  #folosire list
>>> l
['m', 'a', 'r', 'e']
>>> l[0] = 't'      #indexare a listei
>>> ''.join( l )    #join cu string gol
'tare'
>>>
>>> b = bytearray( b'spam' )
>>> b.extend( b'eggs' )
>>> b
bytearray( b'spameggs' )
>>> b.decode()
'spameggs'
>>> #
```

Metode specifice unui obiect

Obiectele de tip string au metode proprii:

- *find()*

```
>>> s = 'Spam'
>>> s.find( 'pa' )     #pozitia de start a stringului 'pa' in s
1
```

- *replace()*

```
>>> s.replace( 'pa', 'xyz' )  #inlocuieste fiecare 'pa' cu 'xyz'
'Sxyzm'
```

Metode…

- *split()*

```
>>> line = '123;456;789;mm'
>>> line.split( ';' )        #delimitare intr-o lista de sub-stringuri
['123', '456', '789', 'mm']
```

- *upper(), isalpha(), isdigit(), rstrip()*

```
>>> s = 'Spam'
>>> s.upper()        #conversie la litere mari
'SPAM'
>>> s.isalpha()      #test, este alfabetic ?
True
```

Metode…

```
>>> line = '123;456;789;mm\n'
>>> line.rstrip()                #eliminare spatiu alb  \n, linie noua
'123;456;789;mm'
>>> line.rstrip().split( ';' )   #combinatie rstrip, apoi split
['123', '456', '789', 'mm']
```

- *format*(), cu si fara numere intre {}

```
>>> '%s, eggs, and %s' % ( 'spam', 'SPAM' )     #cu %, ca in C
'spam, eggs, and SPAM'
>>> '{0}, eggs, and {1}'.format( 'spam', 'SPAM' )
'spam, eggs, and SPAM'
>>> '{}, eggs, and {}'.format( 'spam', 'SPAM' )
'spam, eggs, and SPAM'
```

Metode…

>>> **'{:,.2f}'.format(4294967296.4567)#, ca separator, 2 zecimale**

'4,294,967,296.46'

>>> **'%.2f | %+05d' % (3.1416, -35) #cifre, semn, 0 in fata**

'3.14 | -0035'

Observatie: operatiile pe secvente se aplica tuturor secventelor, dar metodele obiectelor – stringurilor sunt specifice acestora.

Ajutor cu dir() si help()

>>> **dir(s)**

['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
'__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs',
'find', 'format', 'format_map', 'index', 'isalnum', **'isalpha**', 'isascii', 'isdecimal',
'isdigit', 'isidentifier', **'islower**', 'isnumeric', 'isprintable', 'isspace', 'istitle',
'isupper', **'join**', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', **'replace**', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', **'rstrip**', **'split**', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', **'upper**', 'zfill']

>>>

Ajutor…

>>> **help(s.replace)**

Help on built-in function replace:

replace(old, new, count=-1, /) method of builtins.str instance

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace.

-1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

>>> **help(*str*) #help sau dir pe tipul de date *str – string.***

Alte stringuri

- Cu secvente escape \

```
>>> s = 'A\nB\tC'          #\n linie noua, \t tab
>>> len( s )
5
>>> ord( '\n' )            #operatorul ord()
10
>>> s = 'A\0B\0C'          #\0, octetul zero, nu termina stringul
>>> len( s )
5
>>> s = 'A\x00B\x00C'      # \xNN pentru netiparibile, in hexa
>>> s
'A\x00B\x00C'
```

Alte…

- Stringuri multilinie, delimitate cu triplu apostrof:

```
>>> mesaj = '''
The rain in Spain
Stays mainly
In the plains
'''
>>> mesaj
'\nThe rain in Spain\nStays mainly\nIn the plains\n'
```

- Stringuri "raw":

```
>>> s = r'C:\dos\command.com'  #stringuri fara escape, ca atare, fara dublare a \
>>> s
'C:\\dos\\command.com'
```

Stringuri Unicode

- Sunt importante in anumite domenii, mai putin pentru programare in general.

```
>>> 'sp\xc4m'        #in 3.X, str este in Unicode
'spÄm'
 >>> 'spam'.encode( 'utf8' )
b'spam'
>>> 'spam'.encode( 'utf16' )
b'\xff\xfes\x00p\x00a\x00m\x00'
>>>
```

Expresii regulate

- Recunoasterea sabloanelor se face cu modulul **re**

```
>>> import re
>>> sablon = re.match( 'Hello[ \t]*(.*)World!', 'Hello   Beautiful World!' )
>>> sablon.group( 1 )
'Beautiful '
```

. = orice, * = de oricate ori, () = de identificat, [] = alternative

```
>>> sablon = re.match( '[/:](.*)[/:](.*)[/:](.*)', '/usr/home:mydata' )
>>> sablon.groups()
('usr', 'home', 'mydata')
>>> re.split( '[/:]', '/usr/home:mydata' )
['', 'usr', 'home', 'mydata']
```

Sumar

- Clasificare obiecte in Python
- Numere
- Stringuri
- ***Liste***
- Dictionare
- Tuple
- Fisiere
- Alte tipuri de baza

Liste

- Sunt secvente de obiecte eterogene, de dimensiune variabila, ce se pot modifica (nu ca stringurile care sunt fixe); se construiesc intre paranteze patrate [].
- **Operatii pe secvente:**

```
>>> l = [123, 'spam', 3.14]     #trei obiecte diferite intr-o lista
>>> len( l )                    #lungimea listei
3
>>> l[0]                        #indexare, de la zero
123
>>> l[:-1]                      #slicing
[123, 'spam']
```

Liste…

```
>>> l + [4, 5, 6]                  #concatenare, cu +
[123, 'spam', 3.14, 4, 5, 6]
>>> l * 2                          #repetitie, cu *
[123, 'spam', 3.14, 123, 'spam', 3.14]
>>>  l                             #lista a ramas neschimbata
[123, 'spam', 3.14]
```

- **Metode specifice listelor** (le modifica):

```
>>> l.append( 'element nou' )      #adauga obiect la urma
>>> l
[123, 'spam', 3.14, 'element nou']
>>> l.pop( 2 )          #elimina elementul din pozitia 2
3.14
>>> l
[123, 'spam', 'element nou']
```

Metode, liste

```
>>> m = [ 'y', 'z', 'x' ]
>>> m.sort()          #sortare, in situ
>>> m
['x', 'y', 'z']
>>> m.reverse()       #inversare
>>> m
['z', 'y', 'x']
```

- Depasirea limitelor, la **indexare** si la **atribuire:**

```
>>> l[100]            #indexare depasita
IndexError: list index out of range
>>> l[100] = 7        #nu se creeaza element nou
IndexError: list assignment index out of range
>>>
```

Incluziunea

- Python suporta incluziunea pe oricate nivele:

```
>>> m = [[1, 2, 3],          #matrice 3x3
         [4, 5, 6],
         [7, 8, 9]]
>>> m
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> m[1]                     #indexare, linia din pozitia 1 (mijloc)
[4, 5, 6]
>>> m[1][2]                  #indexare, elementul din pozitia 2 al liniei 1
6
>>>
```

Colectii iterative (comprehension)

- Simplifica obtinerea unei coloane a matricii:

```
>>> col1 = [col[1] for col in m]          #coloana din pozitia 1
>>> col1
[2, 5, 8]
>>> m                                     #m este neschimbat
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> [col[1] + 1 for col in m]             #aduna 1 la coloana din pozitia 1
[3, 6, 9]
>>> [col[1] for col in m if col[1] % 2 == 0]  #elimina elementele impare din col. 1
[2, 8]
```

- Sau a unei diagonale:

```
>>> diag = [m[i][i] for i in [0, 1, 2]]      #diagonala principala
>>> diag
[1, 5, 9]
```

Colectii…

```
>>> dublu = [c * 2 for c in 'spam']          #repetare, de doua ori
>>> dublu
['ss', 'pp', 'aa', 'mm']
```

- Generarea de elemente, cu *range():*

```
>>> list( range( 4 ) )
[0, 1, 2, 3]
>>> [[x ** 2, x ** 3] for x in range( 4 )]
[[0, 0], [1, 1], [4, 8], [9, 27]]
>>> [[x, x / 2, x * 2] for x in range( -6, 7, 2 ) if x > 0]   #din 2 in 2, selectie pozitiva
[[2, 1.0, 4], [4, 2.0, 8], [6, 3.0, 12]]
```

Colectii…

- Generator – la cerere, cu paranteze rotunde ():

```
>>> g = (sum( lin ) for lin in m)    #sum(), insumare
>>> next( g )                        #next(), protocol iterativ
6
>>> next( g )                        #suma liniei urmatoare
15
>>>
>>> next( g )                        #suma ultimei linii
24
```

- Generator cu **map()**:

```
>>> list( map( sum, m ) )      #mapeaza sum() pe fiecare element (linie).
[6, 15, 24]
```

Colectii…

- Creare de multimi – set, cu {}:

```
>>> {sum( lin ) for lin in m}                    #multimea sumelor liniilor matricii m
{24, 6, 15}
```

- Creare de dictionare:

```
>>> {i : sum( m[i] ) for i in range( 3 )}        #perechi cheie:valoare, dictionar
{0: 6, 1: 15, 2: 24}
>>> [ord( x ) for x in 'spam']                   #colectii iterative de tip: liste
[115, 112, 97, 109]
>>> {ord( x ) for x in 'spam'}                   #multimi
{112, 97, 115, 109}
>>> {x: ord( x ) for x in 'spam'}                #dictionare
{'s': 115, 'p': 112, 'a': 97, 'm': 109}
>>> (ord( x ) for x in 'spam')                   #generatori
<generator object <genexpr> at 0x0000017F8B65ABC8>
```

Sumar

- Clasificare obiecte in Python
- Numere
- Stringuri
- Liste
- ***Dictionare***
- Tuple
- Fisiere
- Alte tipuri de baza

Dictionare

- Reprezinta asocieri (mapari) de obiecte cu chei – in loc de pozitie (cum era la secvente)
- Nu exista deci ordine de la stanga la dreapta
- Sunt modificabile, se pot schimba si extinde/restrange
- Cheile ofera avantaje mnemonice – ex. coloane ale unei tabele dintr-o baza de date.
- Se construiesc cu acolade {}

Asocieri/mapari

```
>>> d = {'fel': 'Spam', 'cantitate': 4,
     'culoare': 'roz'}
>>> d['fel']      #indexare pe cheie
'Spam'
>>> d['cantitate'] += 1   #increment
>>> d
{'fel': 'Spam', 'cantitate': 5, 'culoare': 'roz'}
>>> d = {}       #constructie
>>> d['nume'] = 'John'
>>> d['job'] = 'ing'
>>> d['varsta'] = 40
>>> d
{'nume': 'John', 'job': 'ing', 'varsta': 40}
>>> print( d['nume'] )
John
>>> d1 = dict( nume = 'John', job = 'ing',
     varsta = 40 )     #cu dict()
>>> d1
{'nume': 'John', 'job': 'ing', 'varsta': 40}
>>> d2 = dict( zip( ['nume', 'job', 'varsta'],
     ['John', 'ing', 40] ) )  #cu zip() intre lista
     de chei si cea de valori
>>> d2
{'nume': 'John', 'job': 'ing', 'varsta': 40}
```

Incluziune

- Flexibilitate:

```
>>> rec = {'nume': { 'prenume': 'John',
'nume familie': 'Connors'},
        'jobs': ['ing', 'sef'],
        'varsta': 40.0}     #trei chei, cu valori de dictionar, lista, numar
>>> rec['nume']     #indexare
{'prenume': 'John', 'nume familie': 'Connors'}
>>> rec['nume']['nume familie']
'Connors'
>>> rec['jobs']
['ing', 'sef']
>>> rec['jobs'][-1]  #ultimul job din lista, cu index -1
'sef'
>>> rec['jobs'].append( 'director' )
>>> rec
{'nume': {'prenume': 'John', 'nume familie': 'Connors'}, 'jobs': ['ing', 'sef', 'director'], 'varsta': 40.0}
>>> rec = 3     #garbage collection, automat memoria ocupata de dictionar se elibereaza cand nu mai este referita.
```

Chei lipsa/testare

```
>>> d = { 'a': 1, 'b': 2, 'c': 3}
>>> d
{'a': 1, 'b': 2, 'c': 3}
>>> d['f']        #cheie inexistenta
KeyError: 'f'
>>> d['e'] = 33#adaugare cheie:valoare
>>> d
{'a': 1, 'b': 2, 'c': 3, 'e': 33}
 >>> 'f' in d     #expresie logica
False
>>> if not 'f' in d:          #test cu if
            print( 'cheie lipsa' )


cheie lipsa
```

```
>>> if not 'f' in d:
            print( 'cheie lipsa' )
            print( 'eroare...' )

cheie lipsa
eroare...
>>> val = d.get( 'x', 25 )        #default 25
>>> val
25
>>> val = d['x'] if 'x' in d else 25   #else
>>> val
25
>>>
```

Sortarea cheilor, cu *for*

- Sortare in trei pasi cu metoda *keys()*

```
>>> ks = list( d.keys() )
>>> ks                  #ordinea nu este garantata
['a', 'c', 'b']
>>> ks.sort()
>>> ks
['a', 'b', 'c']
>>> for k in ks:
        print( k, '=>', d[k] )


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

Sortare…

- Sortare directa, cu *sorted()* – predefinita

```
>>> for k in sorted( d ):
        print( k, '=>', d[k] )


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

- Cicluri *for*, *while*:

```
>>> for c in 'Spam':
        print( c.upper() )

S
P
A
M
>>> x = 3
>>> while x > 0:
        print( 'spam' * x )
        x -= 1

spamspamspam
spamspam
spam
```

Iterare/ciclare

- Un obiect este iterabil daca suporta protocolul de enumerare, adica fie este o secventa in memorie fie un generator.
- Echivalenta intre iteratie si ciclare:

```
>>> patrat = [x ** 2 for x in [1, 2, 3, 4, 5]]    #iteratie
>>> patrat
[1, 4, 9, 16, 25]
>>> patrat = []
>>> for x in [1, 2, 3, 4, 5]:                     #ciclu for
        patrat.append( x ** 2 )
```

<--acelasi rezultat-->

```
>>> patrat
[1, 4, 9, 16, 25]
>>>
```

Sumar

- Clasificare obiecte in Python
- Numere
- Stringuri
- Liste
- Dictionare
- ***Tuple***
- Fisiere
- Alte tipuri de baza

Tuple

- **Tuplele sunt secvente, ca listele, dar sunt fixe, ca stringurile; se codifica cu paranteze rotunde ()**

```
>>> t = (1, 2, 3, 4)   #tuplu cu 4 elemente
>>> len( t )
4
>>> t + (5, 6 )        #concatenare
(1, 2, 3, 4, 5, 6)
>>> t[0]               #indexare
1
>>> t.index( 3 )       #pozitia lui 3
2
>>> t.count( 3 )       #o aparitie a lui 3
1
```

```
>>> t[0] = 7           #obiect fix
TypeError: 'tuple' object does not support
    item assignment
>>> t = (2,) + t[1:]   #tuplu nou, slicing
>>> t
(2, 2, 3, 4)
>>> t = 'spam', 3.14, [1, 2, 3]        #fara ()
>>> t[2][1]
2
```

Obiectele fixe sunt importante in programare!

Sumar

- Clasificare obiecte in Python
- Numere
- Stringuri
- Liste
- Dictionare
- Tuple
- ***Fisiere***
- Alte tipuri de baza

Fisiere

- **Sunt obiecte care asigura interfata cu sistemul de fisiere.**

```
>>> f = open( 'data.txt', 'w' )
      #deschidere pentru scriere, 'w'
>>> f.write( 'Hello\n' )    #scriere o linie incheiata cu \n
6
>>> f.write( 'World!\n' ) #inca o linie
7
>>> f.close()   #inchidere fisier, actualizare pe disc
>>> for line in open( 'data.txt' ): print( line )      #citire cu iterator
Hello

World!
```

```
>>> f = open( 'data.txt' )
      #deschidere pt. citire, 'r' este implicit
>>> txt = f.read()
>>> txt    #string citit
'Hello\nWorld!\n'
>>> print( txt )       #formatare cu print()
Hello
World!
>>> txt.split()        #separator spatiu alb
['Hello', 'World!']
```

Fisiere binare

- **Contin stringuri *bytes*; cele text contin stringuri *str***

```
>>> import struct  #pt. pack()/unpack()
>>> packed = struct.pack('>i4sh', 7,
    b'spam', 8) #big endian, int, string
    cu len 4, short
>>> packed
b'\x00\x00\x00\x07spam\x00\x08'
>>> f = open( 'data.bin', 'wb' )      #pt.
    scriere, fisier binar cu 'wb'
>>> f.write( packed )
10
>>> f.close()
>>>
```

```
>>> data = open( 'data.bin', 'rb' ).read()
    #citire binara 'rb'
>>> data
b'\x00\x00\x00\x07spam\x00\x08'
>>> data[4:8]          #slicing
b'spam'
>>> list( data )       #lista octeti
[0, 0, 0, 7, 115, 112, 97, 109, 0, 8]
>>> struct.unpack( '>i4sh', data )
    #decodificare
(7, b'spam', 8)
>>>
```

Fisiere text Unicode

- **Realizeaza automat encode() la scriere si decode() la citire.**

```
>>> s = 'sp\xc4m'
>>> s
'spÄm'
>>> f = open( 'unicode.txt', 'w',
      encoding = 'utf-8' ) #scriere/encode
      UTF-8 text
>>> f.write( s )
4
>>> f.close()
```

```
>>> txt = open( 'unicode.txt', encoding
      = 'utf-8' ).read()   #citire/decode
      UTF-8 text
>>> txt
'spÄm'
>>> len( txt )
4
>>> f.close()
```

- **Deci Python suporta nu numai text ASCII ci si date binare (ex. fisiere media) si si text in alfabete internationalizate.**

Sumar

- Clasificare obiecte in Python
- Numere
- Stringuri
- Liste
- Dictionare
- Tuple
- Fisiere
- ***Alte tipuri de baza***

Multimi (set)

- **Reprezinta colectii neordonate, de obiecte unice, fixe; nu sunt nici secvente si nici mapari(asocieri)**

```
>>> x = set( 'spam' )  #din string
>>> y = { 'h', 'a', 'm' } #din litere
>>> x, y   #neordonate
({'s', 'm', 'p', 'a'}, {'h', 'm', 'a'})
>>> x & y #intersectie
{'m', 'a'}
>>> x | y #reuniune
{'s', 'p', 'a', 'h', 'm'}
>>> x - y #diferenta
{'s', 'p'}
>>> x > y #superset?
False
```

```
>>> list( set( [1, 2, 1, 3, 1] ) ) #eliminare
    duplicate
[1, 2, 3]
>>> set( 'spam' ) - set( 'ham' ) #gasirea
    diferentelor
{'s', 'p'}
>>> set( 'spam' ) == set( 'mpsa' ) #testarea
    egalitatii, neordonat
True
>>>
```

Fractii si decimali

- **Fractiile sunt numere rationale cu numarator si numitor; decimalii sunt zecimali cu precizie fixa**

```
>>> from fractions import Fraction
>>> f = Fraction( 2, 3 )
>>> f + 1
Fraction(5, 3)
>>> f + Fraction( 1, 2 )
Fraction(7, 6)
>>> #Util in clasele primare !!
```

```
>>> 1 / 3  #zecimal
0.3333333333333333
>>> import decimal
>>> d = decimal.Decimal( '3.141' )
>>> d + 1
Decimal('4.141')
>>> decimal.getcontext().prec = 2
>>> decimal.Decimal( '1.00' ) /
      decimal.Decimal( '3.00' )
Decimal('0.33')
>>>
```

Alte obiecte

- ***bool*: au doar valorile predefinite *True* si *False***

```
>>> 1 > 2, 1 < 2      #expresii logice
(False, True)
>>> bool( 'spam' ) #valoare booleana
True
```

- ***None:* pentru initializari**

```
>>> x = None
>>> print( x )
None
>>> l = [None] * 3
>>> l
[None, None, None]
```

- ***type*: returneaza tipul altui obiect, cu functia *type()*, predefinita**

```
>>> type( l )
<class 'list'>
>>> type( type( l ) )
<class 'type'>
>>> if type( l ) == list:
        print( 'yes' )

yes
>>> #poate infrange ideea de
    polimorfism!
```

Clase, definite de utilizatori

- **Clasele definesc tipuri de obiecte noi care extind obiectele de baza ale Python:**

```
>>> class Angajat:
        def __init__( self, nume, salariu ):
                self.nume = nume
                self.salariu = salariu
        def numeFamilie( self ):
                return self.nume.split()[-1]
        def prima( self, procent ):
                self.salariu *= (1.0 + procent)


>>> joe = Angajat( 'Joe Smith', 50000 )
>>> sue = Angajat( 'Sue Ellen', 50000 )
```

```
>>> joe.numeFamilie()
'Smith'
>>> sue.numeFamilie()
'Ellen'
>>> sue.prima( 0.15 )
>>> sue.salariu
57500.0
>>>
```