Programarea calculatoarelor si limbaje de programare 1

Iteratii, colectii iterative, documentatie

Universitatea Politehnica din Bucureşti

Sumar

- ***Iteratii***
- Colectii iterative
- Documentatia in Python

Iteratii

- Instructiunea ***for*** este aplicabila oricarei secvente – liste, tuple, stringuri:

```
>>> for x in [1, 2, 3, 4]: print( x, end=' ' )

1 2 3 4
>>> for x in (1, 2, 3, 4): print( x, end=' ' )

1 2 3 4
>>> for x in 'spam': print( x, end=' ' )

s p a m
```

Iteratii pe fisiere

- Protocolul de iterare: aplicabil oricarui obiect cu o metoda numita ***__next__()*** ce returneaza urmatorul rezultat si produce o exceptie de tipul ***StopIteration***, la sfarsit.

```
>>> f = open( 'script2.py' )
>>> f.__next__() #__next__() are acelasi efect ca f.readline()
'import sys\n'
>>> f.__next__()
'print(sys.path)\n'
>>> f.__next__()
'x = 2\n'
>>> f.__next__()
'print(x ** 32)\n'
>>> f.__next__()
StopIteration
```

Iteratii...

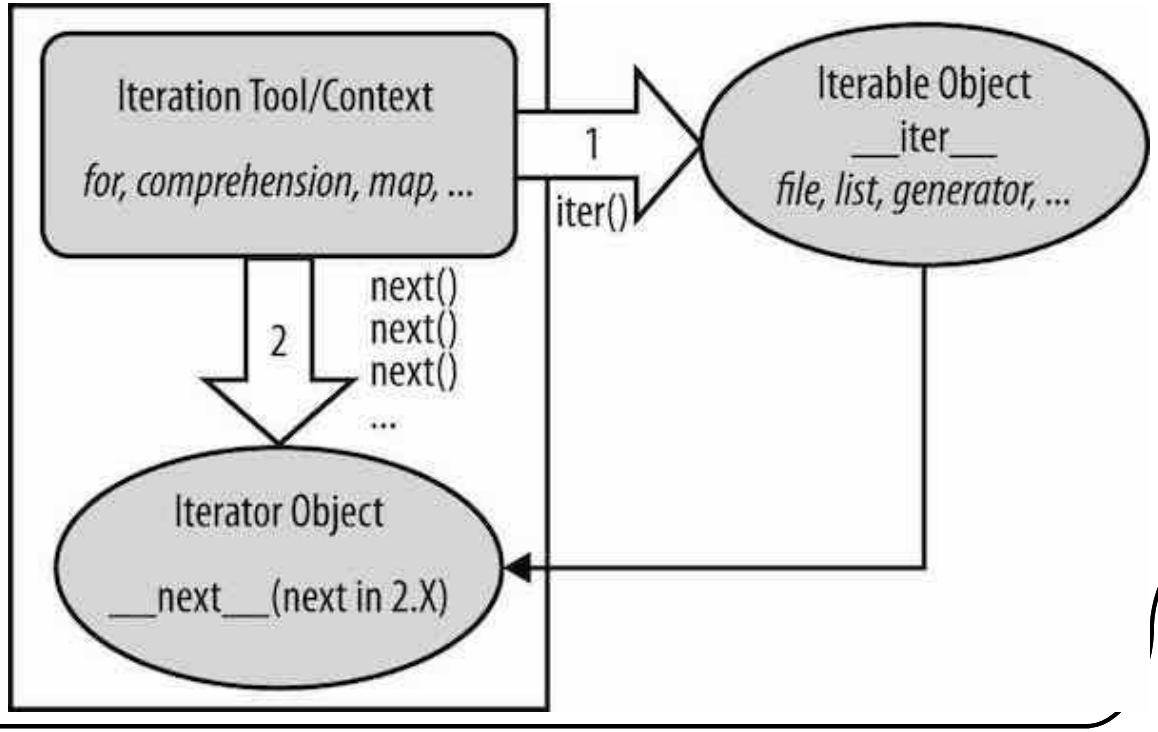
- Functia predefinita ***next()*** poate fi apelata cu argument orice obiect iterator ***x***, intern realizandu-se un apel de ***x.__next__()***:

```
>>> f = open( 'script2.py' )
>>> f.__next__()
'import sys\n'
>>> next( f )
'print(sys.path)\n'
>>> next( f )
'x = 2\n'
>>> f.__next__()
'print(x ** 32)\n'
>>> next( f )
StopIteration
```

- **next(f) este la fel ca f.__next__()**

Protocolul iterativ

- Protocolul iterativ complet include un obiect iterabil avand metoda ***__iter__()***, care odata apelata de functia predefinita ***iter()*** returneaza un obiect iterator ce are metoda ***__next__()***, anterior descrisa.



Protocolul...

- Exista obiecte care sunt si iterabile – au metoda __iter__() si si iterator – au metoda __next__(), de ex. fisierele; astfel de obiecte suporta o singura iteratie la un moment dat, fiind propriul lor iterator.
- Listele, range, dict sunt doar iterabile, permitand multiple iteratii simultan – cu obiecte iterator diferite.

Exemple

- Lista – obiect iterabil (nu iterator):

```
>>> L = [1, 2, 3]
>>> iter( L ) is L  #obiecte diferite!
False
>>> L.__next__()
AttributeError: 'list' object has no attribute '__next__'
```

```
>>> i = iter( L )      #obiect iterator
>>> i.__next__()
1
>>> next( i )
2
```

- Iteratie automata si manuala:

```
>>> for x in L:  #automat
        print( x ** 2, end=' ' )

1 4 9

>>> i = iter( L )  #manual, cu try
```

```
>>> while True:
        try:
            x = next( i )
        except StopIteration: break
        print( x ** 2, end=' ' )

1 4 9
```

Iterabile predefinite

- Dictionarele:

```
>>> d = { 'a': 1, 'b': 2, 'c': 3 }
>>> for cheie in d:
        print( cheie, d[cheie], end='; ' )

a 1; b 2; c 3;
```

```
>>> #cu keys(), clasic:
>>> for cheie in d.keys():
        print( cheie, d[cheie], end='; ' )

a 1; b 2; c 3;
```

- os.popen(comanda):

```
>>> import os
>>> p = os.popen( 'dir' )
>>> i = iter( p )      #iterabil dar nu si iterator!
>>> next( i )
' Volume in drive C has no label.\n'
>>> i.__next__()
' Volume Serial Number is F66F-00E9\n'
```

Iterabile...

- range():

```
>>> r = range( 5 )
>>> i = iter( r )  #iterabil, nu si iterator
>>> next( i )
0
>>> #list, pentru toate valorile deodata:
>>> list( r )
[0, 1, 2, 3, 4]
>>>
```

- enumerate():

```
>>> e = enumerate( 'spam' )
>>> e
<enumerate object at 0x00000196939C8958>
>>> next( e )          #este si iterator!
(0, 's')
>>> list( e )          #continua iteratia!
[(1, 'p'), (2, 'a'), (3, 'm')]
```

Sumar

- Iteratii
- ***Colectii iterative***
- Documentatia in Python

Lista, colectie iterativa

```
>>> L = [1, 2, 3, 4]
>>> L = [x + 10 for x in L] #colectie
>>> L
[11, 12, 13, 14]
```

```
>>> rez = []     #cod echivalent
>>> for x in L:
        rez.append( x + 10 )

>>> rez
[11, 12, 13, 14]
```

- Scriere cu paranteze patrate, ***x*** parcurge lista ***L*** si se obtine o lista cu valorile expresiei din stanga (x+10)

Colectie iterativa, fisiere

```
>>> lines = [line.rstrip() for line in open('script2.py')]
>>> lines
['import sys', 'print(sys.path)', 'x = 2', 'print(x ** 32)']
```

- Lista liniilor, ***lines***, din fisier (script2.py) – obtinuta rapid si fara incarcarea in memorie a intregului fisier ci doar a liniei curente.

```
>>> lines = [line.rstrip().upper() for line in open('script2.py')]
>>> lines
['IMPORT SYS', 'PRINT(SYS.PATH)', 'X = 2', 'PRINT(X ** 32)']
```

Sintaxa extinsa

- Filtrare cu ***if:***

```
>>> lines = [line.rstrip() for line in open('script2.py') if line[0] == 'p']
>>> lines
['print(sys.path)', 'print(x ** 32)']
```

- Iteratii incluse:

```
>>> [x + y for x in 'abc' for y in 'mnp']
['am', 'an', 'ap', 'bm', 'bn', 'bp', 'cm', 'cn', 'cp']
```

- Fiecare ***for*** poate avea ***if***-ul sau, pe oricate nivele.

Alte colectii iterative

- Cu ***map()***:

>>> **map(str.upper, open('script2.py')) #obiect iterabil (are__*next*__)**

<map object at 0x000001B47559F2C8>

>>> **list(map(str.upper, open('script2.py')))**

['IMPORT SYS\n', 'PRINT(SYS.PATH)\n', 'X = 2\n', 'PRINT(X ** 32)\n']

- Cu ***sorted()***:

>>> **sorted(open('script2.py')) *#sorted()* aplicat iterabilului returnat de open()**

['import sys\n', 'print(sys.path)\n', 'print(x ** 32)\n', 'x = 2\n']

- Cu ***zip()***:

>>> **list(zip(open('script2.py'), open('script2.py'))) #combinatie de iterabile**

[('import sys\n', 'import sys\n'), ('print(sys.path)\n', 'print(sys.path)\n'), ('x = 2\n', 'x = 2\n'), ('print(x ** 32)\n', 'print(x ** 32)\n')]

Alte...

- Cu ***enumerate()***:

>>> **list(enumerate(open('script2.py'))) #tuple (pozitie, valoare)**

[(**0**, 'import sys\n'), (**1**, 'print(sys.path)\n'), (**2**, 'x = 2\n'), (**3**, 'print(x ** 32)\n')]

- Cu ***filter()***:

>>> **list(filter(bool, open('script2.py'))) #filtrare cu functia bool(linie)**

['import sys\n', 'print(sys.path)\n', 'x = 2\n', 'print(x ** 32)\n']

- Cu ***reduce()***:

>>> **import functools, operator**

>>> **functools.reduce(operator.add, open('script2.py')) #rezultat, un string!**

'import sys\nprint(sys.path)\nx = 2\nprint(x ** 32)\n'

Alte...

- Cu metoda str.***join()***:

```
>>> '&'.join( open( 'script2.py' ) )
'import sys\n&print(sys.path)\n&x = 2\n&print(x ** 32)\n'
```

- Cu asignari (despachetate) de secvente, operatorul ***in***, slicing, metoda ***.extend()***:

```
>>> a, b, c, d = open( 'script2.py' )  #asignare
>>> a, c
('import sys\n', 'x = 2\n')
>>> a, *b = open( 'script2.py' )  #asignare cu despachetare, *b
>>> a, b
('import sys\n', ['print(sys.path)\n', 'x = 2\n', 'print(x ** 32)\n'])
```

Alte...

```
>>> 'x = 2\n' in open( 'script2.py' )  #in
True
>>> L = [1, 2, 3, 4]
>>> L[1:3] = open( 'script2.py' )  #slicing
>>> L
[1, 'import sys\n', 'print(sys.path)\n', 'x = 2\n', 'print(x ** 32)\n', 4]
>>> L = [1]
>>> L.extend( open( 'script2.py' ) )  #metoda extend()
>>> L
[1, 'import sys\n', 'print(sys.path)\n', 'x = 2\n', 'print(x ** 32)\n']
```

- Multimi si dictionare:

```
>>> {line for line in open( 'script2.py' ) if line[0] == 'p'}  #set
{'print(sys.path)\n', 'print(x ** 32)\n'}
```

Alte...

```
>>> {p: line for (p, line) in enumerate( open( 'script2.py' ) ) if line[0] == 'p'}  #dict
{1: 'print(sys.path)\n', 3: 'print(x ** 32)\n'}
```

- Cu functiile predefinite ***sum(), any(), all(), max(), min()***:

```
>>> sum( [1, 2, 3, 4, 5] )
15
>>> any( ['spam', '', 'hi'] )
True
>>> all( ['spam', '', 'hi'] )
False
>>> max( [1, 2, 3, 4, 5] )
5
```

```
>>> min( [1, 2, 3, 4, 5] )
1
>>> max( open( 'script2.py' ) ) #si fisiere
'x = 2\n'
>>> min( open( 'script2.py' ) )
'import sys\n'
```

Alte...

- Unzip de zip:

```
>>> x = (1, 2)
>>> y = (3, 4)
>>> list( zip( x, y ) )
[(1, 3), (2, 4)]
>>> a, b = zip( *zip( x, y ) ) #despachetare de argumente cu *,
                               asignare de secventa (tuple)
>>> a
(1, 2)
>>> b
(3, 4)
```

Sumar

- Iteratii
- Colectii iterative
- ***Documentatia in Python***

Documentatia in Python

- Comentarii precedate de # in programe
- Functia predefinita ***dir()*** – listeaza atributele unui modul, obiect:

```
>>> import sys
>>> dir( sys )
>>> [a for a in dir(list) if not a.startswith('__')]  #filtrare metode __M__
['append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

- Docstrings: ***__doc__***:
  - Sunt stringuri de la inceputul modulelor, functiilor/metodelor sau claselor, colectate in atributul __doc__, asadar disponibile la executie

Documentatia...

- Stringurile de documentare pot fi orice string, eventual cu triplu apostrof pentru extindere pe mai multe linii – ex. *docstrings.py*:

```
"""
Documentatie de modul
Mda, mda, mda...
"""

spam = 40

def square(x):
    """
    Documentatie de functie
    fff, fff, fff...
    """
    return x ** 2        # square

class Employee:
    "Documentatie de clasa"
    pass

print(square(4))
print(square.__doc__)
```

PyDoc

- PyDoc este un instrument standard care afiseaza documentatia si alte informatii despre structura unui obiect; se executa cu:
  - Functia predefinita ***help()*** – in mod text:

```
>>> help( sys )                    #intreg modulul sys
>>> help( sys.getrefcount )#functie din modulul sys
>>> help( ".replace )              #metoda ob. de tip str
>>> help( ord )                    #functie predefinita
>>> help( str.replace )            #metoda a str
```

Pydoc...

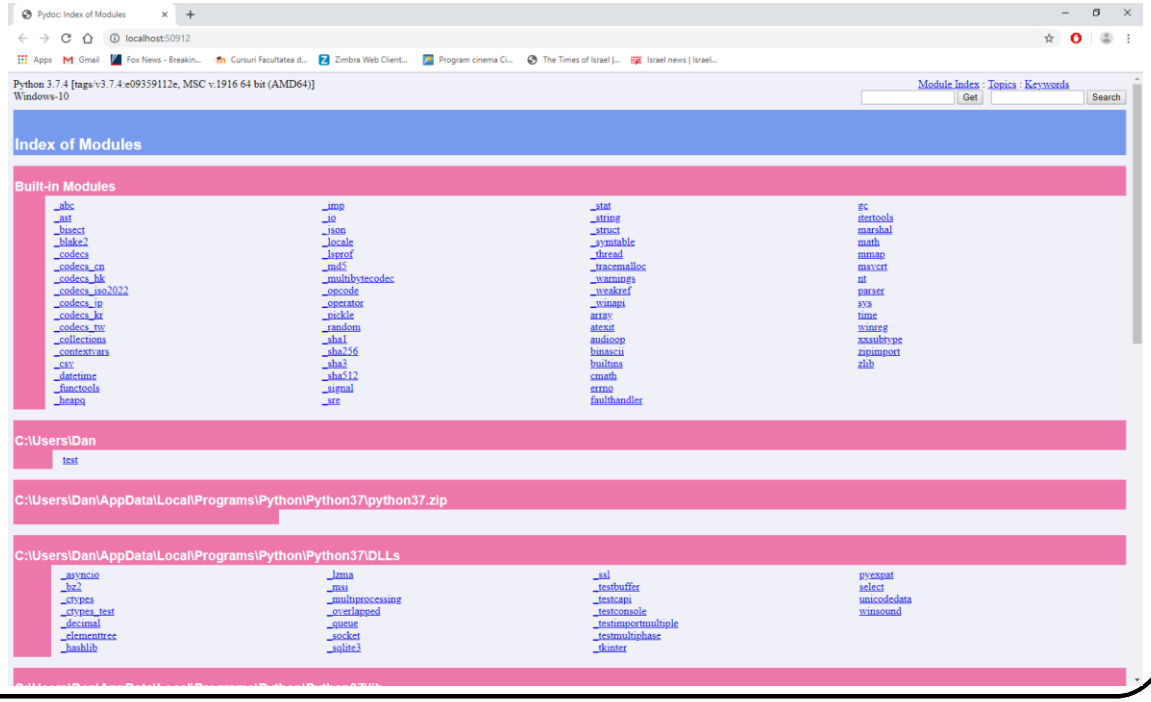
- In mod browser de web (configurabil, culori):

C:\Users\Dan>**py -m pydoc -b#modulul pydoc cu arg. -b**

Server ready at http://localhost:50912/

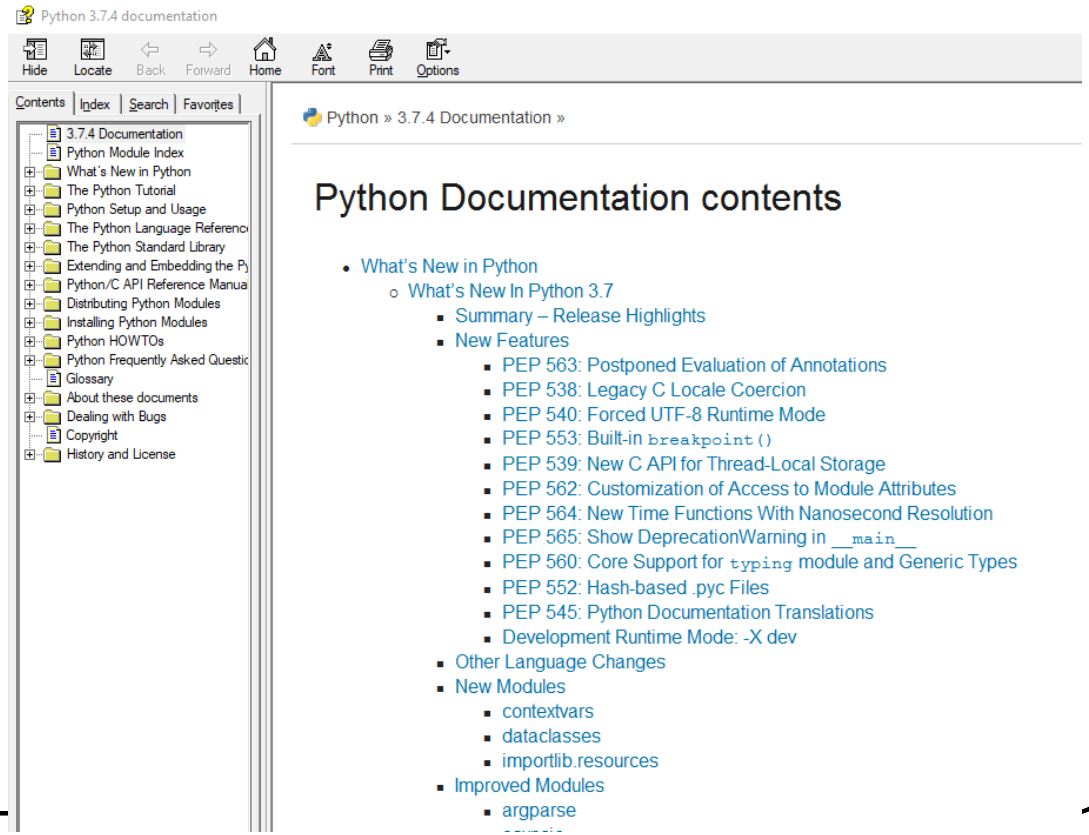
Server commands: [b]rowser, [q]uit

server>



Alte moduri de documentare

- Cu ***Sphinx*** – vezi http://sphinx-doc.org
- Cu manualul standard, din IDLE/Help(F1):



Python – recomandari

- Nu uitati de ***:*** la sfarsitul antetului instructiunilor compuse
- Nu indentati eronat – incepeti in prima coloana
- Folositi linii albe in modul interactiv pentru a incheia instructiunile compuse
- Indentati in mod consistent – nu amestecati tab cu spatiu
- Uitati de C/C++!

Python...

- Folositi *for* in loc de while sau range()
- Atentie la atribuiri extinse de obiecte modificabile
- Atentie la functiile care nu returneaza nimic (None)
- Folositi paranteze la apelul de functii, chiar fara argumente
- Folositi doar numele de modul ~~la import~~ sau reload – fara extensie

Bafta in sesiune !