Paradigma Modulara

Cursul nr. 6

Mihai Zaharia

Ce se poate face cu Python?

ORICE:

- aplicații WEB
- frontend (inclusiv pt cloud)
- backend
- hardware low level (suportat nativ pe unele microcontrolere)
- arhitecturi complexe bazate pe microservicii
- inteligență artificială
- securitate (pentest (mai mult automatizare), criptografie dar NU treburi serioase)

Paradigme de programare suportate de PyThon

- **Funcțională**

```
import functools
my_list = [1, 2, 3, 4, 5]
def add_it(x, y):
    return (x + y)
sum = functools.reduce(add_it,
my_list)
print(sum)
```

- **Orientată obiect**

```
class ChangeList(object):
    def __init__(self, any_list):
        self.any_list = any_list
    def do_add(self):
      self.sum = sum(self.any_list)
create_sum = ChangeList(my_list)
create_sum.do_add()
print(create_sum.sum)
```

- **Imperativă**

```
sum = 0
for x in my_list:
    sum += x
print(sum)
```

- **Procedurală**

```
def do_add(any_list):
    sum = 0
    for x in any_list:
        sum += x
    return sum
print(do_add(my_list))
```

Structurarea unui program

Sintaxa

Instrucţiuni multi-linie

Exemplu 1:

```
total = item1 + \
        item2 + \
        item3
```

Exemplu 2:

```
ZileLucrat = [‘Luni',‘Marti',‘Miercuri',
              ‘Joi',‘Vineri']
```

Cuvinte cheie

And	exec	not	assert	finally	break or
For	pass	class	from	print	continue
global	raise	def	if	return	del
Import	try	elif	in	while	else
Is	with	except	lambda	yield	

Conversii explicite ale tipurilor de date

Sunt suportate: *integer, floating point, long și complex*

integer (valoare), long(valoare), complex(real,imaginar) cu caz particular Iimaginar = 0 *complex(real)*

Câteva funcții matematice (suportă mult mai multe)

sqrt(val), pow(baza,exp), minorare la întreg - *ceil(val),* majorare la întreg - *floor(val), log(val), min(val1,val2,val3,...) și max(val1,val2,val3,...)*

și trigonometrice:

acos, asin, atan, atan2, hypot, cos, sin, tan, degrees, radian

și constante precum *pi* și *e*

Control execuției

Decizie unică (simplă)

if <condition>:
    <body>

Decizie binară/dual

if <condition>:
  <statements>
else:
  <statements>

Decizie Multiplă

if <condition1>:
  <case1 statements>
elif <condition2>:
  <case2 statements>
elif <condition3>:
  <case3 statements>
…
else:
  <instrucțiuni pt default>

Similară cu *switch-case-default*

În calculul rădăcinii ecuației de gradul II
Calculează discriminantul (Δ)
  when < 0: tratează cazul fără rădăcini reale
  when = 0: tratează cazul particular al unui singur rezultat
  when > 0: tratează cazul general al celor două soluții

Excepții

```
import math

print("Rezolvare ecuatie de grad II\n")
try:
    a, b, c = [int(x) for x in input("Introdu coeficientii a b c: ").split()]
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print("\nSolutiile sunt:", root1, root2)
except ValueError as excObj:
    if str(excObj) == "math domain error":
        print("Nu are radacini reale")
    else:
        print("Numar incorect de valori")
except NameError:
    print("\nNu ai ntrodus trei valor")
except TypeError:
    print("\nnu toate valorile introduse sunt numere")
except SintaxaError:
    print("\nFormatul datelor incorect - utilizati spatiu")
except:
    print("\nEroare necunoscuta")
```

Instrucțiunea try:

```
try:
        <body>
except <ErrorType>:
        <handler>
```

Bucle cu număr de pași cunoscut - For

```
for <var> in <sequence>:
    <body>
```

```
for i in range(11):
    print(i)
```

Bucle cu număr de pași necunoscut - While

```
while <condition>:
    <body>
```

```
i = 0
while i <= 10:
    print(i)
    i = i + 1
```

Intrucţiunea pass

- **Exemplu:**

```
for litera in 'Python':
    if litera == "h":
        pass
        print("Acesta este efectul lui pass")
    print('litera curenta:', litera)
print( 'Pa Pa!')
```

Rezultate:

llitera curenta: P
litera curenta: y
litera curenta: t
Acesta este efectul lui pass
litera curenta: h
litera curenta: o
litera curenta: n
Pa Pa!

Operatori

Ordinea de precedență este *not*, *and*, *or* deci se consideră că

`a or not b and c` ⇔ `(a or ((not b) and c))`

Totuși parantezele rămân recomandate

```
(a >= 15 and a - b >= 2) or (b >= 15 and b - a >= 2)
(a >= 15 or b >= 15) and abs(a - b) >= 2
```

Datele sunt evaluate boolean ca în C (true este ≠0)

Operator	Traducere în operații logice
x and *y*	If *x* is false, return *x*. Otherwise, return *y*.
x or *y*	If *x* is true, return *x*. Otherwise, return *y*.
not *x*	If *x* is false, return `True`. Otherwise, return `False`.

Lista operatori: +, -, *, /, //, **, %, abs,

Cu observația că: 10//3 = 3 si 10%3 = 1 adică a = (a/b)(b) + (a%b)

Operatori de atribuire

Operator	Descriere	Exemplu
=	atribuire simplă	c = a + b
+=	operandului din stânga i se adună operandul din dr.	c += a -> c = c + a)
-=	din operandul stâng se scade operandul drept	c -= a -> (c = c – a)
*=	operandul din stânga se multiplică cu operandul din dr.	c *= a -> (c = c * a)
/=	împarte operandul din stânga cu cel din dreapta	c /= a -> (c = c / a)
%=	operandului din dreapta i se atribuie restul împărţirii	c %= a -> (c = c % a)
**=	operandului din stânga i se atribuie puterea	c **= a ->(c = c ** a)
//=	împărţireă întreagă este atribuită operandului stâng	c //= a ->(c = c // a)

Operatori pe biți

```
   a = 0011 1100
   b = 0000 1101
---------------
a&b = 0000 1100
a|b  = 0011 1101
a^b  = 0011 0001
~a    = 1100 0011
```

Operatori Membership (apartenență)

Operator	Descriere	Exemplu
in	verifică dacă un obiect este într-o secvență	x in y
not in	verifică dacă un obiect nu este în secvență	x not in y

Operatori de identitate

Operator	Descriere	Exemplu
is	verifică dacă 2 variabile au aceeaşi referință (adresă) de memorie	a is b
not is	verifică dacă 2 variabile nu au aceeaşi referință (adresă) de memorie	a not is b

Tipuri de date: șiruri de caractere

>>> ord("A")→65 | >>> ord("a") →97 | >>> chr(97) →'a' | >>> chr(65) →'A'

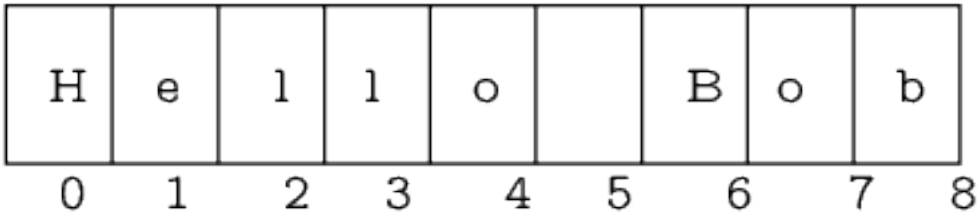
>>> str1="Hello" | >>> str2='spawn' | >>> print(str1, str2)→Hello spawn

>>> type(str1)→<class 'str'> | >>> type(str2)→<class 'str'>

- Citirea unui șir

  >>> firstName = input("Cum te cheama?: ")

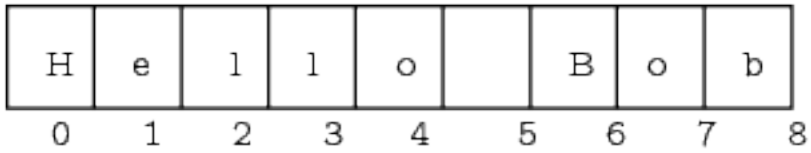
- Forma generală de indexare <string>[<expr>] (ca în C)



>>> greet[-1] →'b' | >>> greet[-3] →'B'

Tipuri de date:șiruri de caractere

• *Extragere subșiruri* (slicing)



>>> greet[0:3]→'Hel‘ | >>> greet[5:9]→' Bob' | >>> greet[:5]→'Hello' |

>>> greet[5:]→' Bob' | >>> greet[:]→'Hello Bob‘

• *Concatenare* (+) *Repetiție* (*) *Lungimea (nr caractere) len.*

>>> "spam" + "eggs"→'spameggs' | >>> "Spam" + "And"+ "Eggs"→'SpamAndEggs' |

>>> 3 * "spam"→'spamspamspam' | >>> "spam" * 5→'spamspamspamspamspam' |

>>> (3 * "spam") + ("eggs" * 5) →'spamspamspameggseggseggseggseggs‘

>>> len("spam") →4

```
>>> for ch in "Spam!":
        print (ch, end=" ")
```

Tipuri de date:șiruri de caractere

Alte metode:

- s.capitalize() – creează o copie a lui s cu prima literă făcută mare
- s.title() – creează o copie a lui s cu toate literele făcute mari
- s.center(width) – centrează pe s într-o zonă de o lățime dată
- s.count(sub) – numără aparițiile lui sub în s
- s.find(sub) – caută prima apariție a lui sub în s
- s.join(list) – concatenează o listă de șiruri având s ca separator
- s.ljust(width) – ca și center dar în stânga
- s.lower() – creează o copie a lui s cu toate literele făcute mici

Tipuri de date:șiruri de caractere

- s.lstrip() – creează o copie a lui s cu toate spațiile albe eliminate
- s.replace(oldsub, newsub) – înlocuieste aparițiile în s a lui oldsub cu newsub
- s.rfind(sub) – caută prima apariție a lui sub în s și întoarce cea mai din dreapta apariție
- s.rjust(width) – are efect ca și center, dar s este identat la dreapta (right-justified)
- s.rstrip() – șterge spațiile de la început și de la sfârșit
- s.split() – extrage o listă de substringuri funcție de un separator implicit ‘ ‘ sau explicit oarecare
- s.upper() – creează o copie a lui s cu toate literele făcute mari

Afișare obiecte - ca și în limbajul C

Caracter	*Argument Așteptat*
c	Șir de lungime 1
s	Șir de orice lungime
d	Întregi în baza 10
u	Întregi fără semn în baza 10
o	Întregi în baza 8
x or X	Întregi în baza 16 (uppercase for X)
e, E, f, g, G	Reale în diverse stiluri
%	Procent literal

Formatarea afișării - cam ca la C

```
capitole = {1: 5, 2: 46, 3: 52, 4: 87, 5: 90}
hexStr = "3f8"
for x in capitole:
    print("Capitole " + str(x) + \
          str(capitole[x]).rjust(15, '.'))
print("Capitol %d %15s" % (x, str(capitole[x])))
print("\nHex String: " + hexStr.upper().rjust(8, '0'))  # Right justify
print("\nHex String: " + hexStr.upper().ljust(8, '0'))  # Left justify

for x in capitole:
    print("Capitol %d %15s" % (x, str(capitole[x])))  # formatare
String
```

Exemple de formatare - consolă

```
print("Hello {0} {1}, you may have won ${2}" .format("Mr.", "Smith", 10000))
  Hello Mr. Smith, you may have won $10000

print('This int, {0:5}, was placed in a field of width 5'.format(7)) #cu ieșire

This int,     7, was placed in a field of width 5

print('This int, {0:10}, was placed in a field of witdh 10'.format(10))#cu ieșire

'This int,          10, was placed in a field of witdh 10'

print('This float, {0:10.5}, has width 10 and precision 5.'.format(3.1415926))
  This float,    3.1416, has width 10 and precision 5.

print('This float, {0:10.5f},  is fixed at 5 decimal places.'.format(3.1415926))

This float,   3.14159, has width 0 and precision 5.
```

Funcții simple pentru String

- capitalize, center, count, expandtabs, find, index, decode, encode, endswith, isalnum, isalpha, isspace, istitle, isupper, join, len, isdigit, islower, isnumeric, ljust

```
Str = 'Iar au inceput sa adoarma!!!'
suffix = '!!'
print (Str.endswith(suffix))
print (Str.endswith(suffix,20))

suffix = 'exam'
print (Str.endswith(suffix))
print (Str.endswith(suffix, 0, 19))
```

Executarea de cod din interiorul unui șir

exec(str [,globals [,locals]])

```
radius = 3
cards = ['Ace', 'King', 'Queen', 'Jack']
codeStr = 'for card in cards: print ("Card = " + card)'
exec(codeStr)
```

eval(str [,globals [,locals]])

```
areaStr = 'pi*(radius*radius)'
print("\nArea = " + str(eval(areaStr, {'pi': 3.14}, {'radius': 5})))
values = [5, 3, 'blue', 'red']
```

substitute(m, [, kwargs]).

```
s = string.Template("Variable v = $v")
for x in values:
    print(s.substitute(v=x))
```

Liste simple

```
x = [1,2,3]
print("\n"+str(x))          [1, 2, 3]
y = x
print("\n"+str(y))          [1, 2, 3]
x[1] = 15
print("\n"+str(x))          [1, 15, 3]
x.append(12)
print("\n"+str(x))          [1, 15, 3, 12]
print("\n"+str(y))          [1, 15, 3, 12]
x = [1,'hello', (3 + 2j)]
print("\n"+str(x))          [1, 'hello', (3+2j)]
print("\n"+str(x[2]))       (3+2j)
print("\n"+str(x[0:2]))     [1, 'hello']

x = [1,2,3]
y=x
print("\n"+str(x))          [1, 2, 3]
x = x + [9,10]
print("\n"+str(x))          [1, 2, 3, 9, 10]
print("\n"+str(y))          [1, 2, 3]
```

Operații pe listă

Operator	Înțelesul acestuia
<seq> + <seq>	Concatenare
<seq> * <int-expr>	Repetiție
<seq>[]	Indexare
len(<seq>)	Lungime
<seq>[:]	Slicing
for <var> in <seq>:	Iterație
<expr> in <seq>	Apartenență (Boolean)

Operații pe listă

Metoda	*Întelesul acesteia*
<list>.append(x)	*Adaugă x la sfârșitul listei*
<list>.sort()	*Sortează lista (o funcție de sortare poate fi trimisă ca parametru)*
<list>.reverse()	*Inversează lista*
<list>.index(x)	*Indexul pentru prima apariție a lui x în listă*
<list>.insert(i, x)	*Iinserează x pe poziția i*
<list>.count(x)	*Numărul de apariții al lui x listă*
<list>.remove(x)	*Șterge prima aparitie a lui x în listă*
<list>.pop(i)	*Șterge element i și îi întoarce valoarea*

Exemple liste

```
lst = [3, 1, 4, 1, 5, 9]
lst.append(2)
print("\n"+str(lst))          → [3, 1, 4, 1, 5, 9, 2]
lst.sort()
print("\n"+str(lst))          [1, 1, 2, 3, 4, 5, 9]
lst.reverse()
print("\n"+str(lst))          [9, 5, 4, 3, 2, 1, 1]
print("\n"+str(lst.index(4))) 2
lst.insert(4, "Hello")
print("\n"+str(lst))          [9, 5, 4, 3, 'Hello', 2, 1, 1]
print("\n"+str(lst.count(1))) 2
lst.remove(1)
print("\n"+str(lst))          [9, 5, 4, 3, 'Hello', 2, 1]
lst.pop(3)
print("\n"+str(lst))          [9, 5, 4, 'Hello', 2, 1]
```

Tuplele

```
hexStringChars = ('A', 'B','C', 'D', 'E', 'F')
hexStringNums = ('1', '2', '3', '4', '5', '6', '7', '8', '9','0')
hexStrings = ["1FC", "1FG", "222", 'Ten"]
for hexString in hexStrings:
    for x in hexString:
        if ((not x in hexStringChars) and
                (not x in hexStringNums)):
            print(hexString+" nu este in hexa")
            break
tupleList = list(hexStringChars)
print("\n"+str(tupleList))
listTuple = tuple(hexStrings)
print("\n"+str(listTuple))
```

și ieșirea programului:

```
1FG nu este in hexa
Ten nu este in hexa

['A', 'B', 'C', 'D', 'E', 'F']

('1FC', '1FG', '222', 'Ten')
```

Dicționar

- #dicționar simplu unu la unu

```
numberDict = {1:'one', 2:'two', 3:'three', 4:'four'}
print("\n", str(numberDict))
#dicționar unu la mai mulți
letterDict = {'vowel':['a','e','i','o','u'],\
              'consonant':['b','c','d','f']}
print("\n", str(letterDict))
#dicționar mai mulți la mai mulți
numbers = (1,2,3,4,5,6,7,8,9,0)
letters = ('a','b','c','d','e','f')
punct = ('.', '!', '?')
charSetDict = {numbers:[], letters:[], punct:[]}
print("\n", str(charSetDict))
```

și rezultatul execuției

```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}

{'vowel': ['a', 'e', 'i', 'o', 'u'], 'consonant': ['b', 'c', 'd', 'f']}

{(1, 2, 3, 4, 5, 6, 7, 8, 9, 0): [], ('a', 'b', 'c', 'd', 'e', 'f'): [], ('.', '!', '?'): []}
```

Adăugare de valori

```
numbers = ('1','2','3','4','5','6','7','8','9','0')
letters = ('a','b','c','d','e','f')
punct = ('.', '!', '?')
charSetDict = {numbers:[], letters:[], punct:[]}
cSet = input("Introduceti caractere:")
for c in cSet:
    for x in charSetDict.keys():
        if c in x:
            charSetDict[x].append(c)
            break;
charSetDict["Special"] = ['%', '$', '#']
charSetDict["Special"] = '><'
print("\n"+str(charSetDict))
```

și rezultatul execuției:

```
Introduceti caractere:12hjh78

{('1', '2', '3', '4', '5', '6', '7', '8', '9', '0'):
['1', '2', '7', '8'], ('a', 'b', 'c', 'd', 'e', 'f'): [
], ('.', '!', '?'): [], 'Special': '><'}
```

Adăugare valori la dicționar

```
numbers = ('1','2','3','4','5','6','7','8','9','0')
letters = ('a','b','c','d','e','f')
punct = ('.', '!', '?')
charSetDict = {numbers:[], letters:[], punct:[]}
def display_cset (cset):
    #print
    for x in cset.items():
        if x[0] == numbers:
            print("Numere:")
        elif x[0] == letters:
            print("Cifre:")
        elif x[0] == punct:
            print("Puctuatie:")
        else:
            print("Necunoscut:")
        print(x[1])
cSet = input("Introduceti caractere: ") #se adauga noi valori la chei
for c in cSet:
    for x in charSetDict.keys():
        if c in x:
            charSetDict[x].append(c)
            break;
display_cset(charSetDict)
charSetDict["Special"] = ['%', '$', '#'] #se adaugă o cheie și valoare nouă
display_cset(charSetDict)
charSetDict["Special"] = '><' #se schimbă valoarea unei chei existente
display_cset(charSetDict)
```

```
Introduceti caractere: 1,d,4d,%,<3,4>
Numere:
['1', '4', '3', '4']
Cifre:
['d', 'd']
Puctuatie:
[]
Numere:
['1', '4', '3', '4']
Cifre:
['d', 'd']
Puctuatie:
[]
Necunoscut:
['%', '$', '#']
Numere:
['1', '4', '3', '4']
Cifre:
['d', 'd']
Puctuatie:
[]
Necunoscut:
><
```

Modificare de elemente din dicționar

```
validkeys = (1,2,3)
keyGenDict={'keys': [1,2,3], 1: 'blue',
            2: 'fast', 3: 'test','key': 2}
print("Cheile:")
print(keyGenDict.keys())
print("\nValorile")
print(keyGenDict.values())
print("\nElemente:")
print(keyGenDict.items())
val = keyGenDict['key']
keyGenDict['key'] = 1
print("\nValorile")
print(keyGenDict.values())
keyGenDict.clear()
print("\nElemente:")
print(keyGenDict.items())
```

```
Cheile:
dict_keys(['keys', 1, 2, 3, 'key'])

Valorile
dict_values([[1, 2, 3], 'blue', 'fast', 'test', 2])

Elemente:
dict_items([('keys', [1, 2, 3]), (1, 'blue'), (2, 'fast'),
(3, 'test'), ('key', 2)])

Valorile
dict_values([[1, 2, 3], 'blue', 'fast', 'test', 1])

Elemente:
dict_items([])
```

Extragerea unei valori din dicționar

```
validkeys = (1,2,3)
keyGenDict={'keys':[1,2,3],1:'blue',
2:'fast', 3:'test','key':2}

def show_key (key):
    if(key in validkeys):
        keyVal = (keyGenDict["keys"])[key-1]
        print("Key = " + keyGenDict[keyVal])
    else:
        print("Invalid key")

#lista cheilor din dicționar
print(keyGenDict.keys())
```

→ dict_keys(['keys', 1, 2, 3, 'key'])

```
#valorile dicționar
print(keyGenDict.items())
```

dict_items([('keys', [1, 2, 3]), (1, 'blue'), (2, 'fast'), (3, 'test'), ('key', 2)])

```
#valoare din cheie
val = keyGenDict["key"]
show_key(val)
```

Key = fast
Key = blue

Gestiunea fișierelor în Python

- open(path [,mode [,buffersize]])
- Infile = open("numbers.dat", "r")
- <file>.read()
- <file>.readline()
- <file>.readlines()

Atributele unui obiect fișier

- **Atributele unui obiect fisier:**

  După ce s-a deschis un fișier se obține o referință (obiect fișier) care are mai multe atribute.

- **Atribut** **Descriere**

  file.closed returnează true dacă fișierul este închis, false altfel.

  file.mode returnează modul de acces în care a fost deschis fișierul.

  file.name returnează numele fișierului.

  file.softspace returnează false dacă este necesar spațiu explicit pentru afișare (print), true altfel

Tratare simplă fișiere

```
fobj_in = open("/home/bugs/PycharmProjects/fisiere1/otopeni.txt")
fobj_out =
open("/home/bugs/PycharmProjects/fisiere1/otopeni1.txt","w")
i = 1
for line in fobj_in:
    print(line.rstrip())
    fobj_out.write(str(i) + ": " + line)
    i = i + 1
fobj_in.close()
fobj_out.close()
```

Poziționare în fișier

```
fo= open("/home/bugs/PycharmProjects/fisiere1/otopeni.txt","r+")
str = fo.read(10);
print("Sirul citit este: "+str)
pos = fo.tell()  # verifică poziția curentă
print("Pozitia curenta:",pos)
pos = fo.seek(0, 0);  # repoziționarea cursorului la inceputul fișierului
str = fo.read(25);
print("din nou citire:"+str)
fo.close()
```

Citește o anume linie

```
import linecache
filePath = "input.txt"
print(linecache.getline(filePath, 2))
print(linecache.getline(filePath, 4))
linecache.clearcache()
```

Citește câte un cuvânt

```
filePath = "input.txt"
wordList = []
wordCount = 0
#citeste linii intr-o lista
file = open(filePath, 'r')
for line in file:
    for word in line.split():
        wordList.append(word)
        wordCount += 1
print(wordList)
print("Total words = %d" % wordCount)
```

```
L=[]
infile = open(filePath, "r")
for line in infile:
    L=L+[str(s) for s in line[:-1].split(' ')]
it=iter(L)
for cuvant in it:
   print("\n"+ cuvant)
infile.close()
```

Fișiere - comenzi sistem

- de unde : import os, sys
- Metoda rename():
  - are 2 argumente: numele curent și noul nume al fișierului;

  Sintaxa:

  os.rename(current_file_name, new_file_name)

  Exemplu:

  import os

  os.rename("test1.txt", "test2.txt")
- Metoda delete()
  - metoda se găsește în modulul os și are un argument
- Sintaxa:

  os.delete(file_name)

  Exemplu:

  Ștergem fișierul f2.txt

  import os

  os.delete("f2.txt")

Fișiere - comenzi sistem

Metoda mkdir()

Are un argument și creează un director cu numele argumentului

Sintaxa:

```
os.mkdir("newdir")
```

Exemplu:

```
import os
os.mkdir("test") # Create a directory "test"
```

Metoda chdir()

Are un argument care conține numele noului director curent

Sintaxa:

```
os.chdir("newdir")
```

Exemplu:

În exemplul care urmează ne poziționam pe directorul "/home/newdir“:

```
import os
os.chdir("/home/newdir")
```

Fișiere - comenzi sistem

Metoda getcwd()

Afișează numele directorului curent

Sintaxa:

```
os.getcwd()
```

Exemplu:

```
import os # This would give location of the current directory
os.getcwd()
```

Metoda rmdir()

Are un argument care conține numele directorului fișierului ce urmează a fi șters.

Sintaxa:

```
os.rmdir('dirname')
```

Exemplu:

Vom șterge directorul test; calea completă"/tmp/test".

```
import os
os.rmdir( "/tmp/test" )
```

Metode asociate fișierelor

file.close()
file.flush()
file.fileno()
file.isatty()
file.next()
file.read([size])
file.readline([size])
file.readlines([sizehint])
file.seek(offset[, whence])
file.tell()
file.truncate([size])
file.write(str)
file.writelines(sequence)

Tratare corectă operații I/O fișiere

```
inPath = input("Fisierul de intrare: ")
outPath = input("Fisierul de iesire: ")
try:
    file = open(inPath, 'r')#deschide pentru citire
     # incepe citirea
    file.close()
except FileNotFoundError:
    print("Erroare deschidere fisier")
try:
    file = open(outPath, 'wb') #deschide pentru scriere
    # scrie în fișier
    file.close()
except FileNotFoundError:
    print("Eroare scriere in fisier")
```

Print to File

```
import sys

wordList = ["Red", "Blue", "Green"]
filePath = "output.txt"


def printToFile(filePath, mode, text):
    original = sys.stdout
    sys.stdout = open(filePath,mode)
    print(text)
    sys.stdout = original

for word in wordList:
    printToFile(filePath,'a', "\n"+str(word)+" Color Adjust")
```

Funcții

- Definiția:
  def <name>(<formal-parameters>):
  <body>

```
def dictToBinary(the_dict):
    str = json.dumps(the_dict)
    binary = ' '.join(format(ord(letter), 'b') for letter in str)
    return binary
```

- Apelul funcției
  <name>(<actual-parameters>)

```
ceva=dictToBinary(dictonar)
```

Funcții care întorc mai multe valori

- Câte odată o funcție trebuie să întoarcă mai mult de o valoare.

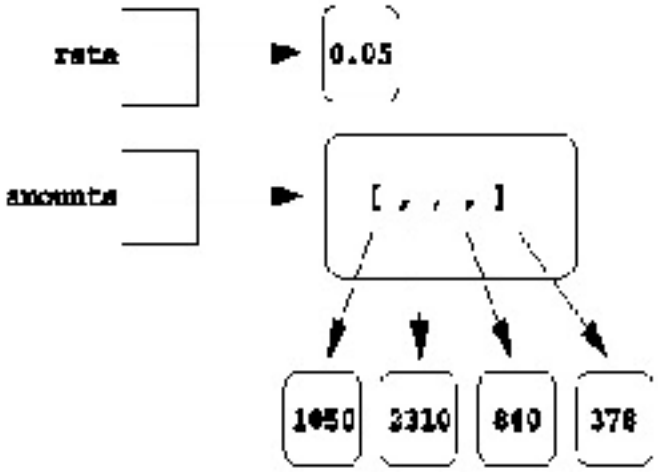
```
def sumDiff(x, y):
    sum = x + y
    diff = x – y
    return sum, diff
```

- Python suportă return multiplu precum și inițializare multiplă

```
num1, num2 = eval(input("Enter two numbers(num1, num2)"))

s, d = sumDiff(num1, num2)

print("The sum is", s, "and the difference is", d)
```

Transfer parametri prin liste/tablouri

```
def addInterest(balances, rate):
    for i in range(len(balances)):
        balances[i] = balances[i] * (1+rate)

def test():
    amounts = [1000, 2200, 800, 360]
    rate = 0.05
    addInterest(amounts, 0.05)
    print(amounts)

test()
```



Cuvinte cheie ca argumente

- Argumentele de apel sunt de forma "*keyword* = *value*".
- De exemplu pentru funcția

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print "-- This parrot wouldn't", action,
    print "if you put", voltage, "Volts through it."
    print "-- Lovely plumage, the", type
    print "-- It's", state, "!"
```

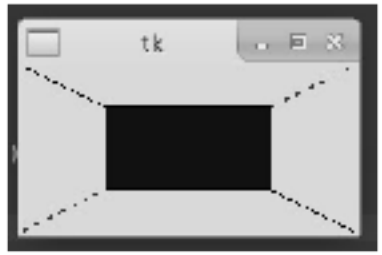
- Putem avea un apel de forma

```
parrot(1000)
parrot(action = 'VOOOOOM', voltage = 1000000)
parrot('a thousand', state = 'pushing up the daisies')
parrot('a million', 'bereft of life', 'jump')
```

- Dar nu putem avea un apel de forma

```
parrot() # required argument missing
parrot(voltage=5.0, 'dead') # non-keyword argument following keyword
parrot(110, voltage=220) # duplicate value for argument
parrot(actor='John Cleese') # unknown keyword
```

Grafică simplă




```
from tkinter import *
master = Tk()
w = Canvas(master, width=200, height=100)
w.pack()
w.create_line(0, 0, 200, 100)
l=w.create_line(0, 100, 200, 0, fill="red", dash=(4, 4))
r=w.create_rectangle(50, 25, 150, 75, fill="blue")
mainloop()
```

dacă doresc modificări directe am

```
w.itemconfig(r, fill="red")
w.delete(l)
```

sau

```
w.delete(ALL)
```

Exemplu Python vs CPP

```
import time
class now:
    def __init__(self):
        self.t=time.time()
        self.year, \
        self.month, \
        self.day, \
        self.hour, \
        self.minute, \
        self.second, \
        self.dow, \
        self.doy, \
            self.dst =
time.localtime(self.t)
n=now()
print("The year is" + str(n.year))
```

```
#include <stdio.h>
 2   #include <time.h>
 3   class now
 4   {
 5       public:
 6       time_t t;
 7       int year;
 8       int month;
 9       int day;
11        int hour;
12        int minute;
13        int second;
14        int dow;
15        int doy;
17
31   };
33
```

```
now()
18  {   time(&t);
20  struct tm * ttime;
21  ttime = localtime(&t);
22  year = 1900 + ttime->tm_year;
23  month = ttime->tm_mon;
24  day = ttime->tm_mday;
25  hour = ttime->tm_hour;
26  minute = ttime->tm_min;
27  second = ttime->tm_sec;
28  dow = ttime->tm_wday;
29  doy = ttime->tm_yday; }

main ( int argc, char ** argv )
34  {
35  now n ;
36  fprintf ( stdout, "The year is %d\
n", n.year ) ;
37  }
```

CPP vs Python

- În C++, avem terminator de instrucțiune “; “ nu și în Python.
- În C++, marcajul de bloc este cu {}, în Python, prin identare.
- În C++, clasa are un membru ascuns numit "this", care este vizibil pentru orice metodă din clasă dacă este nevoie. În Python, echivalentul este "self", iar folosirea lui este obligatorie .
- În C++ (și in C), trebuie adăugat 1900 la anul întors de localtime() ; În Python este rezolvată problema.
- În C++, metoda apelată atunci când se creează o instanță a clasei now poarta numele clasei: *now()*. Când o clasă este instanțiată în Python, acesta caută o metodă numită *__init__()* și o apelează
- Pentru copierea unui obiect se folosește metoda clone()

OOP în Python - structurarea datelor

```
class Person:
    def __init__(self, name, age, pay=0, job=None):
        self.name = name
        self.age  = age
        self.pay  = pay
        self.job  = job
if __name__ == '__main__':

#zona pentru testare independenta a modului
    bob = Person('Bob Smith', 42, 30000, 'sweng')
    sue = Person('Sue Jones', 45, 40000, 'music')
    print(bob.name, sue.pay)
    print(bob.name.split( )[-1])
    sue.pay *= 1.10
    print(sue.pay)
```

OOP în Python – adăugare comportament

```
class Person:
    def __init__(self, name, age, pay=0, job=None):
        self.name = name
        self.age  = age
        self.pay  = pay
        self.job  = job
    def lastName(self):
        return self.name.split( )[-1]
    def giveRaise(self, percent):
        self.pay *= (1.0 + percent)
if __name__ == '__main__':
    bob = Person('Bob Smith', 42, 30000, 'sweng')
    sue = Person('Sue Jones', 45, 40000, 'music')
    print(bob.name, sue.pay)
    print(bob.lastName())
    sue.giveRaise(.10)
    print(sue.pay)
```

OOP în Python – adăugare mostenire

• Abordarea este cea cunoscută deja:

```
from person import Person

class Manager(Person):
    def giveRaise(self, percent, bonus=0.1):
        self.pay *= (1.0 + percent + bonus)

if __name__ == '__main__':
    tom = Manager(name='Tom Doe', age=50, pay=50000)
    print(tom.lastName())
    tom.giveRaise(.20)
    print(tom.pay)
```

OOP în Python – adăugare mostenire

```
from person import Person
from manager import Manager
bob = Person(name='Bob Smith', age=42, pay=10000)
sue = Person(name='Sue Jones', age=45, pay=20000)
tom = Manager(name='Tom Doe',  age=55, pay=30000)
db = [bob, sue, tom]
for obj in db:
        obj.giveRaise(.10)
for obj in db:
        print(obj.lastName( ), '=>', obj.pay)
```

```
Smith => 11000.0
Jones => 22000.0
Doe => 36000.0
```

OOP în Python – adăugare persistență

```
import shelve

from person import Person

from manager import Manager

bob = Person('Bob Smith', 42, 30000,
'sweng')

sue = Person('Sue Jones', 45, 40000,
'music')

tom = Manager('Tom Doe',  50, 50000)

db = shelve.open('class-shelve')

db['bob'] = bob

db['sue'] = sue

db['tom'] = tom

db.close( )
```

```
import shelve
db = shelve.open('class-shelve')
for key in db:
    print key, '=>\n  ', db[key].name, db[key].pay

bob = db['bob']
print(bob.lastName())
print(db['tom'].lastName())

#O modificare a valorilor existente în zona de persis
import shelve
db = shelve.open('class-shelve')
sue = db['sue']
sue.giveRaise(.25)
db['sue'] = sue
tom = db['tom']
tom.giveRaise(.20)
db['tom'] = tom
db.close( )
```

Clase Exemplu 2

```
class Student:
    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
    def getName(self):
        return self.name
    def getHours(self):
        return self.hours
    def getQPoints(self):
        return self.qpoints
    def gpa(self):
        return self.qpoints/self.hours
def makeStudent(infoStr):
    name, hours, qpoints = infoStr.split("\t")
    return Student(name, hours, qpoints)
def main():
    filename = input("Fisierul cu notele: ")
```

```
Adams, Henry      127     228
Comptewell, Susan         100     400
DibbleBit, Denny  18      41.5
Jones, Jim        48.5    155
Smith, Frank      37      125.33
```

```
    try:#protectia anti prost
        infile = open(filename, 'r')
        best = makeStudent(infile.readline())
        for line in infile:
            s = makeStudent(line)
            if s.gpa() > best.gpa():
                best = s
        infile.close()
    except FileNotFoundError:
        print("Erroare deschidere fisier")
    print("TCel mai bun student este:",
best.getName())
    print ("ore:", best.getHours())
    print("GPA:", best.gpa())
```