Paradigma Calculul Functional

Cursul nr. 13

Mihai Zaharia

Lambda

- redefinim true, false și iff astfel:

```
true = lambda x, y: x
false = lambda x, y: y
iff = lambda p, x, y: p(x, y)
```

- Cum ar arata o structură elementară de date și anume o pereche

```
pair = lambda x, y: lambda f: f(x, y)
```

- Se introduc următoarele două funcții:

```
first = lambda p: p(true)
second = lambda p: p(false)
```

“P este o pereche de două obiecte x și y dacă exista două funcții first și second astfel încât first(P)=x și second(P)=y

argumente și keyword arguments

```
def func(a, *args, **kw):
    print(a)
    print(args)
    print(kw)
func('valoare A', 'valoare B', 'valoare C', argumentA = 'valoare D',
argumentB = 'valoare D')
```

si rezultatul executiei

```
valoare A
('valoare B', 'valoare C')
{'argumentA': 'valoare D', 'argumentB': 'valoare D'}

Process finished with exit code 0
```

Funcții de prim nivel

- def exemplu(a, b, **kw): return a * b
- print(type(exemplu))
- print(exemplu.__code__.co_varnames)
- print(exemplu.__code__.co_argcount)

si rezultatul executiei

```
<class 'function'>
('a', 'b', 'kw')
2

Process finished with exit code 0
```

Funcții pure

```
from functools import reduce
x=int(input("numar="))
calculNumereMersenne = lambda x: 2**x-1
print("Tipul variabilei calculNUmereMersene este:
"+str(type(calculNumereMersenne)))
print(str(calculNumereMersenne(x)))
print("mersene(%i)=%i"%(x, calculNumereMersenne(x)))
lista = [50, 71, 11, 97, 54, 62, 77]
rezultat = list(filter(lambda x: (x%2 == 0) , lista))
print(rezultat)
rezultat = list(map(lambda x: (x*2) , lista))
print(rezultat)
suma = reduce((lambda x, y: x + y), lista)
print("Suma elementelor din lista este %d" %suma)
```

si rezultatul executiei

```
numar=10
Tipul variabilei calculNUmereMersene este:
<class 'function'>
1023
mersene(10)=1023
[50, 54, 62]
[100, 142, 22, 194, 108, 124, 154]
Suma elementelor din lista este 422

Process finished with exit code 0
```

Funcții de nivel superior

```
from functools import reduce

lista = [50, 71, 11, 97, 54, 62, 77]

rezultat=min(max(list(filter(lambda x: (x%2 == 0) ,
lista))),min(list(map(lambda x: (x*2) , lista))),reduce((lambda x, y: x + y),
lista))

print("rezultatul unei functii de nivel superior este %d"%rezultat)
```

si rezultatul executiei

```
rezultatul unei functii de nivel superior este 22

Process finished with exit code 0
```

împachetează ->procesează->despachetează

```
preturiCarne = [(2000,30), (2001, 35), (2002, 40),(2003, 45),
                (2004, 48), (2005, 52), (2006, 57),(2007, 65),
                (2008, 70), (2009, 75), (2010, 80)]
snd= lambda x: x[1]
print(snd(max(map(lambda yc: (yc[1], yc), preturiCarne))))
```

si rezultatul executiei

```
(2010, 80)

Process finished with exit code 0
```

Evaluare la cerere

```
True and print("and cu true")
False and print("and cu false")
1 and print("and cu unu")
0 and print("and cu zero")
True or print("or cu true")
False or print("or cu false")
```

si rezultatul executiei

```
and cu true
and cu unu
or cu false

Process finished with exit code 0
```

Evaluare la cerere

```
from typing import Iterator
def numere(n: int) -> Iterator[int]:
    for i in range(n):
        print(f"= {i}")
        yield i
def sumaPrimelorN(n: int):
    suma: int = 0
    for i in numere(n):
        print(f"= {i}")
        if i == n: break
        suma += i
    return suma
def sumaPrimelorN1(n: int):
    suma: int = 0
    for i in range(n):
        suma += i
    return suma
```

```
def sumaPrimelorN2(n: int):
    suma: int = 0
    i:int =0
    while True:
        print(f"= {i}")
        suma += i
        i+=1
        if(i>=n):break
    return suma
x = 6
print("Sumare cu generator
%d" %sumaPrimelorN(x))
print("Sumare cu range %d"
%sumaPrimelorN1(x))
print("Sumare clasica %d"
%sumaPrimelorN2(x))
```

si rezultatul executiei

```
= 0
= 0
= 1
= 1
= 2
= 2
= 3
= 3
= 4
= 4
= 5
= 5
Sumare cu generator 15
Sumare cu range 15
= 0
= 1
= 2
= 3
= 4
= 5
Sumare clasica 15

Process finished with exit code 0
```

Generatoare recursive

```
from typing import Iterator
def pfactorsr(x: int) -> Iterator[int]:
    def factor_n(x: int, n: int) ->
Iterator[int]:
        if n*n > x:
            yield x
            return
        if x % n == 0:
            yield n
            if x//n > 1:
                yield from factor_n(x // n, n)
        else:
            yield from factor_n(x, n+2)
    if x//2 > 1:
        yield from pfactorsr(x//2)
    return
yield from factor_n(x, 3)
print(list(pfactorsr(1455560)))
```

si rezultatul executiei

```
[2, 2, 2, 5, 36389]
[
Process finished with exit code 0
```

Un alt exemplu combinare liste de caractere

```
def combinare(ADTiterabil, index=0, lungime=1):
    it = iter(ADTiterabil)
    for contor in range(index):
        yield next(it)
    combinata = next(it)
    for count in range(lungime-1):
        combinata += next(it)
    yield combinata
    for element in it:
        yield element

l1 = ['11', '22', '3', '4']
l2 = []
l2 = list(combinare(l1,0,len(l1)))
print(l2)
```

si rezultatul executiei

```
['112234']

Process finished with exit code 0
```

Afișare fără conversie la listă

```
def printIterator(it):
    s=''
    for x in it:
        s=s+' '+str(x)
    print("%s\n"%s)
print(l2)
l1 = [1, 2, 3, 4]
t1 = (5, 6, 7, 8, 9, 10)
m = map(lambda x, y: x * y, l1, t1)
printIterator(m)
print(list(m))#pentru ca iteratorul a fost consumat deja
```

si rezultatul executiei

```
5 12 21 32

[]

Process finished with exit code 0
```

Funcții din biblioteca itertools

- **accumulate(iterable[, func])** furnizează o serie de serii ale elementelor dintr-o structura iterabilă
- **chain(*iterables)** parcurge mai multe structuri iterabile una după alta fără a crea o listă intermediară a tuturor elementelor
- **combinations(iterable, r)** generează toate combinațiile de lungime **r** pornind de la o structură iterabilă
- **compress(data, selectors)** va aplica o mască logică (booleană) furnizată de selectori asupra elementelor și ne furnizează numai acele valori care corespund criteriilor de selecție din selectori.
- **count(start, step)** generează o secvență infinită de valori începând cu cea de start si incrementând-o cu step la fiecare apel
- **cycle(iterable)** parcurge în mod repetat (într-o buclă) valorile dintr-o structură iterabilă
- **repeat(elem[, n])** repetă un element de **n** ori
- **dropwhile(predicate, iterable)** extrage o submulțime de elemente pornind dela primul și continuând până ce predicatul devine Fals
- **groupby(iterable, keyfunc)** crează un iterator care grupează elemente în funcție de rezultatul furnizat de functia keyfunc().
- **permutations(iterable[, r])** furnizează permutări succesive de dimensiune **r** ale elementelor dintr-o structură iterabilă.

itertools - iteratori infiniți - count

```
import itertools as it
contorReal = it.count(start=0.5, step=0.75)
print(list(next(contorReal) for _ in range(5)))
contorRealPrecizie= (0.5+x*.75 for x in it.count())
print(list(next(contorRealPrecizie) for _ in range(5)))
contorDescrescator=it.count(start=-1, step=-0.5)
print(list(next(contorDescrescator) for _ in range(5)))
#simulare functie enumerate
print(list(zip(it.count(), ['a', 'b', 'c'])))
numaraDinTreiInTrei=it.count(step=3)
print(list(next(numaraDinTreiInTrei) for _ in range(5)))
```

si rezultatul executiei

```
[0.5, 1.25, 2.0, 2.75, 3.5]
[0.5, 1.25, 2.0, 2.75, 3.5]
[-1, -1.5, -2.0, -2.5, -3.0]
[(0, 'a'), (1, 'b'), (2, 'c')]
[0, 3, 6, 9, 12]

Process finished with exit code 0
```

calculul erorii prin acumulare

```
import itertools as it
#functie utila
def until(terminate, iterator):
    i = next(iterator)
    if terminate(*i):
        return i
    return until(terminate, iterator)
source = zip(it.count(0, .1), (.1*c for c in it.count()))
neq = lambda x, y: abs(x-y) > 1.0E-12
print(until(neq, source))
#dapa cati pasi apare deja diferenta
print(until(lambda x, y: x != y, source))
```

si rezultatul executiei

```
(92.799999999999, 92.80000000000001)
(92.899999999999, 92.9)

Process finished with exit code 0
```

itertools - iteratori infiniți - cycle

```
sinus=it.cycle([1, -1])
print(list(next(sinus) for _ in range(6)))
ceva=it.cycle([3,1,0,-1,-3])
print(list(next(ceva) for _ in range(9)))
sir=it.cycle(['a', 'b', 'c'])
print(list(next(sir) for _ in range(6)))
```

si rezultatul executiei

```
[1, -1, 1, -1, 1, -1]
[3, 1, 0, -1, -3, 3, 1, 0, -1]
['a', 'b', 'c', 'a', 'b', 'c']

Process finished with exit code 0
```

itertools - accumulate

```
import operator as op
import itertools as it
rezultat = list(it.accumulate([1, 2, 3, 4, 5], op.add))
print(rezultat)
rezultat = list(it.accumulate([1, 2, 3, 4, 5]))
print(rezultat)
rezultat = list(it.accumulate([1, 2, 3, 4, 5], lambda x, y: (x + y) / 2))
print(rezultat)
#ordinea argumentelor
ordine1Arg = list(it.accumulate([1, 2, 3, 4, 5], lambda x, y: x - y))
print(ordine1Arg)
ordine2Arg = list(it.accumulate([1, 2, 3, 4, 5], lambda x, y: y - x))
print(ordine2Arg)
#pentru s_i = P*s_i-1 + Q, ∀ι∈N vom avea:
def generareSecventa(p, q, valoareInitiala):
    return it.accumulate(it.repeat(valoareInitiala), lambda s, _: p * s + q)
pare = generareSecventa(1,2,0)
primeleOpt = list(next(pare) for _ in range(8))
print(primeleOpt)
```

si rezultatul executiei

```
[1, 3, 6, 10, 15]
[1, 3, 6, 10, 15]
[1, 1.5, 2.25, 3.125, 4.0625]
[1, -1, -4, -8, -13]
[1, 1, 2, 2, 3]
recursivitate
[0, 2, 4, 6, 8, 10, 12, 14]

Process finished with exit code 0
```

itertools - iteratori infiniți - repeat

```
import itertools as it
print(list(tuple(it.repeat(i, times=i)) for i in range(5)))
print(list(sum(it.repeat(i, times=i)) for i in range(10)))
def generareSecventa(p, q, valoareInitiala):
    return it.accumulate(it.repeat(valoareInitiala), lambda s, _: p * s + q)
print("serii")
numerePare = generareSecventa(1,2,0)
secvNumerePare = list(next(numerePare) for _ in range(8))
print(secvNumerePare)
numereImpare = generareSecventa(1,2,1)
secvNumereImpare = list(next(numereImpare) for _ in range(8))
print(secvNumereImpare)
numereDinTreiInTrei = generareSecventa(1,3,0)
secvNumereDinTreiInTrei = list(next(numereDinTreiInTrei) for _ in range(8))
print(secvNumereDinTreiInTrei)
numaiUnu = generareSecventa(1,0,1)
secvNumaiUnu = list(next(numaiUnu) for _ in range(8))
print(secvNumaiUnu)
numereAlternative = generareSecventa(-1,1,1)
secvNumereAlternative = list(next(numereAlternative) for _ in range(8))
print(secvNumereAlternative)
```

si rezultatul executiei

```
[(), (1,), (2, 2), (3, 3, 3), (4, 4, 4, 4)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
serii
[0, 2, 4, 6, 8, 10, 12, 14]
[1, 3, 5, 7, 9, 11, 13, 15]
[0, 3, 6, 9, 12, 15, 18, 21]
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 1, 0, 1, 0, 1, 0]

Process finished with exit code 0
```

itertools - creare funcții de ordin doi

- pentru reamintire: s(n) = p * s(n-1) + q * s(n-2) + r

```
import itertools as it
def secventaOrdinDoi(p, q, r, initial_values):
    intermediate = it.accumulate(
        it.repeat(initial_values),
        lambda s, _: (s[1], p*s[1] + q*s[0] + r) )
    return map(lambda x: x[0], intermediate)
numereFibonacci = secventaOrdinDoi(1, 1, 0, (0, 1))
secvNumereFibonacci = list(next(numereFibonacci) for _ in range(8))
print(secvNumereFibonacci)
numerePell = secventaOrdinDoi(2, 1, 0, (0, 1))
secvNumerePell = list(next(numerePell) for _ in range(6))
print(secvNumerePell)
numereLucas = secventaOrdinDoi(1, 1, 0, (2, 1))
secvNumereLucas = list(next(numereLucas) for _ in range(6))
print(secvNumereLucas)
numereFibonacciAlternative = secventaOrdinDoi(-1, 1, 0, (-1, 1))
secvNumereFibonacciAlternative = list(next(numereFibonacciAlternative) for _ in range(6))
print(secvNumereFibonacciAlternative)
```

si rezultatul executiei

```
[0, 1, 1, 2, 3, 5, 8, 13]
[0, 1, 2, 5, 12, 29]
[2, 1, 3, 4, 7, 11]
[-1, 1, -2, 3, -5, 8]

Process finished with exit code 0
```

alte functii din itertools

```
import itertools as it
numere = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def grupareOptimizata(lista, numarTuple):
    iteratori = [iter(lista)] * numarTuple
    return zip(*iteratori)
grupate1 = list(grupareOptimizata(numere, 2))
print(grupate1)
grupate2 = list(grupareOptimizata(numere, 4))
print(grupate2)
def grupareOptimizataCuPadding(lista, numarTuple, valoareCompletare=None):
    iteratori = [iter(lista)] * numarTuple
    return it.zip_longest(*iteratori, fillvalue=valoareCompletare)
grupate3 = list(grupareOptimizataCuPadding(numere, 4))
print(grupate3)
```

alte functii din itertools

```
combinare = list(it.combinations(numere, 3))
print(combinare)
seturiCuSuma12 = []
for i in range(1, len(numere) + 1):
    for combinatie in it.combinations(numere, i):
        if sum(combinatie) == 12:
            seturiCuSuma12.append(combinatie)
print(seturiCuSuma12)
seturiUniceCuSuma12 = set(seturiCuSuma12)
print(seturiUniceCuSuma12)
seturiCuSuma12SiInlocuire = []
for i in range(1, len(numere) + 1):
    for combinatie in it.combinations_with_replacement(numere, i):
        if sum(combinatie) == 12:
            seturiCuSuma12SiInlocuire.append(combinatie)
print(seturiCuSuma12SiInlocuire)
```

Clonare iteratori -tee()

```
import itertools as it
numere = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def calcul(lista):
    it0, it1 = it.tee(lista,2)
    s0= sum(1 for x in it0)
    s1= sum(x for x in it1)
    return s1/s0

print(calcul(numere))
```

list comprehension

• lista = [expresie for element in lista_intrare if conditie]

```
x=[]#cod clasic
for i in (1,2,3):#la intrare o tupla
    x.append(i)
print(x)
#utilizand list comprehension
x = [i for i in [1, 2, 3]]
print(x)#incepe exemplu 3
oLista = (1, "4", 9, "a", 0, 4)
for i in oLista:
    print(type(i))
listaIntregi = [ i**2 for i in oLista if str(i.__class__)=='<class \'int\'>' ]
print("Lista filtrata%s" %' '.join(map(str, listaIntregi)))
```

si rezultat executie

```
[1, 2, 3]
[1, 2, 3]
<class 'int'>
<class 'str'>
<class 'int'>
<class 'str'>
<class 'int'>
<class 'int'>
Lista filtrata1 81 0 16

Process finished with exit code 0
```

procesare avansată liste

```
oLista = (1, "4", 9, "a", 0, 4)

print(type(oLista))

rezultat =map(lambda el:el**2, filter(lambda el: str(type(el))=='<class
\'int\'>',oLista))

print("Lista filtrata %s" %str(list((rezultat))[0:4]))
```

si rezultatul

```
<class 'tuple'>
<class 'tuple'>
Lista filtrata [1, 81, 0, 16]

Process finished with exit code 0
```

Alt exemplu

```
#aplicare partiala calcul functional
numere = [1, 2, 3, 4, 5]
def doiX(x):
    return x + x
def laPatrat(x):
    return x * x
listaProcesata =[]
for i in numere:
    listaProcesata = listaProcesata+list(map(lambda x: x(i), (doiX, laPatrat)))
print(listaProcesata)
#daca nu am nevoie de reutilizarea celor doua functii
listaProcesata1 = []
for i in numere:
    listaProcesata1 = listaProcesata1+list(map(lambda x: x(i), (lambda x:x+x, lambda x:x*x)))
print(listaProcesata1)
```

set & dictionary comprehensions

```
nume = ['Ion', 'BULA', 'sile', 'cucu', 'GODAC', 'J&k', 'CORO']
submultime = {element[0].upper() + element[1:].lower() for element in
nume if len(element) > 2 and len(element) < 4}
print(list(submultime))


dictionar = {'a': 10, 'b': 34, 'A': 7, 'Z': 3}


frecv_dictionar = {k.lower(): dictionar.get(k.lower(), 0) +
dictionar.get(k.upper(), 0) for k in dictionar.keys()}

print(frecv_dictionar)
```

si rezultatul executiei

```
['Ion', 'J&k']
{'a': 17, 'b': 34, 'z': 3}

Process finished with exit code 0
```

comprehensie Dictionar

```
#intersectie
x = {'a': 1, 'b': 2}
y = {'b': 3, 'c': 4}
z = {**x, **y}
print(z)
#comprehensie
valoriGradeF = {'t1': -32,'t2': -24,'t3': -13,'t4': 0}
valoriGradeC = {k:round((float(5)/9)*(v-32)) for (k,v) in valoriGradeF.items()}
print(valoriGradeC)
dict1 = {'e': 1, 'f': 2, 'k': 3, 'p': 4, 'l': 5}
# aplicare conditie (valoare element > 2)
filtrareDictCuCond = {k:v for (k,v) in dict1.items() if v>2}
print(filtrareDictCuCond)
# aplicare conditii multiple
filtrareDictCuCond = {k:v for (k,v) in dict1.items() if v>2 if v%2 == 0}
print(filtrareDictCuCond)
#aplicare conditii una intr-alta
nestedDict = {'primulEl':{'a':1}, 'alDoileaEl':{'b':2}}
conversieFloatDict = {exterior_k: {float(interior_v) for (interior_k, interior_v) in exterior_v.items()}
          for (exterior_k, exterior_v) in nestedDict.items()}
print(conversieFloatDict)
```

si rezultatul executiei

```
{'a': 1, 'b': 3, 'c': 4}
{'t1': -36, 't2': -31, 't3': -25, 't4': -18}
{'k': 3, 'p': 4, 'l': 5}
{'p': 4}
{'primulEl': {1.0}, 'alDoileaEl': {2.0}}

Process finished with exit code 0
```

Nested comprehensions

```
numarLinii=5

numarColoane=5

matrice = [ [ 1 if elementMatrice == linie else 0 for elementMatrice in range(0, numarColoane) ] for linie in range(0, numarLinii) ]

for i in range(0,numarLinii):
    print(matrice[i])


for i in range(0,numarLinii):
    for j in range(0, numarColoane):pass
        #print(matrice[i][j])
```

si rezultatul executiei

```
[1, 0, 0, 0, 0]
[0, 1, 0, 0, 0]
[0, 0, 1, 0, 0]
[0, 0, 0, 1, 0]
[0, 0, 0, 0, 1]

Process finished with exit code 0
```

Operatorii any si all

```
listaNumere=(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103)
listaNumereNormale = [i for i in range(25)]
listaNumerePare = [2*i for i in range(15)]
def isprime(n):
    n = abs(int(n))
    if n < 2:
        return False
    if n == 2:
        return True
    if not n & 1:
        return False
    for x in range(3, int(n**0.5)+1, 2):
        if n % x == 0:
            return False
    return True
if all(isprime(x) for x in listaNumere):print("Lista de numere prime")
if not all(isprime(x) for x in listaNumereNormale):print("Lista de numere")
if any(not isprime(x) for x in listaNumerePare):print("Lista de numere")
```

si rezultatul executiei

```
Lista de numere prime
Lista de numere
Lista de numere

Process finished with exit code 0
```

Zip si unzip

```
listaNumere=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
        41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
        89, 97, 101, 103]
listaNumereNormale = [i for i in range(28)]
fermoar=zip(listaNumereNormale,listaNumere)
print(list(fermoar))
print(list(fermoar))
stanga=(x[0] for x in fermoar)
dreapta=(x[1] for x in fermoar)
print(list(stanga))
print(list(dreapta))
```

si rezultatul executiei

```
[(0, 2), (1, 3), (2, 5), (3, 7), (4, 11),
(5, 13), (6, 17), (7, 19), (8, 23), (9,
29), (10, 31), (11, 37), (12, 41), (13,
43), (14, 47), (15, 53), (16, 59), (17,
61), (18, 67), (19, 71), (20, 73), (21,
79), (22, 83), (23, 89), (24, 97), (25,
101), (26, 103)]
[]
[]
[]

Process finished with exit code 0
```

zip si unzip

```
listaNumere=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
        41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
        89, 97, 101, 103]
listaNumereNormale = [i for i in range(28)]
fermoar=zip(listaNumereNormale,listaNumere)
listaDinObjZip=list(fermoar)
listaFinala = listaDinObjZip[:]
listaFinala1 = list(listaDinObjZip)
print(list(listaFinala))
print(list(listaFinala1))
stanga=(x[0] for x in listaFinala)
dreapta=(x[1] for x in listaFinala1)
print(list(stanga))
print(list(dreapta))
print(list(stanga))
print(list(dreapta))
```

Data Flatten - ex listă cu tuple

```
listaValori=[('5', '3', '5', '7', '11'), ('13', '17', '19', '23', '29'), ('31', '137', '41', '143',
'47'), ('53', '59', '61', '67', '71'), ('173', '9', '83', '89', '97'), ('101', '103', '107', '109',
'113'), ('12', '131', '17', '139', '19'), ('151', '157', '163','167', '173'), ('179', '18',
'191', '199', '197'), ('194', '21','123', '225', '229')]
def flatten(aList):
    t = []
    for i in aList:
        if not isinstance(i, tuple):
            t.append(i)
        else:
            t.extend(flatten(i))
    return t
listaCurata=flatten(listaValori)
print(list(listaCurata))
```

si rezultatul executiei

```
['5', '3', '5', '7', '11', '13', '17', '19', '23', '29', '31', '137', '41',
'143', '47', '53', '59', '61', '67', '71', '173', '9', '83', '89', '97',
'101', '103', '107', '109', '113', '12', '131', '17', '139', '19', '151',
'157', '163', '167', '173', '179', '18', '191', '199', '197', '194', '21',
'123', '225', '229']

Process finished with exit code 0
```

Data Flatten - ex cap tabel extras din json

```
inregistrare = {'Nume':'Bula', 'Locatia':{'Oras':'Pocreaca','Tara':'ROM'}, 'hobi':['Manea', 'Bautura',
'Femei']}
def flatten_tabel(y):
    iesire = {}
    def flatten(x, nume=''):
        if type(x) is dict:
            for a in x:
                flatten(x[a], nume + a + '_')
        elif type(x) is list:
            i = 0
            for a in x:
                flatten(a, nume + str(i) + '_')
                i += 1
        else:
            iesire[nume[:-1]] = x
    flatten(y)
    return iesire
listaCurata=flatten_tabel(inregistrare)
print(listaCurata)
```

si rezultatul executiei

```
{'Nume': 'Bula', 'Locatia_Oras': 'Pocreaca', 'Locatia_Tara':
'ROM', 'hobi_0': 'Manea', 'hobi_1': 'Bautura', 'hobi_2': 'Femei'}

Process finished with exit code 0
```

enumerate, slice & reverse

```
listaSimpla=['5', '3', '5', '7', '11', '13', '17', '19', '23', '29', '31', '137', '41', '143',
 '47', '53', '59', '61', '67', '71', '173', '9', '83', '89', '97', '101', '103']
student = ('manaca','doarme','femei')
#slice
n=len(listaSimpla)
listaSliced=zip((listaSimpla[2*i::n] for i in
range(round(n/2))),(listaSimpla[2*i+1::n] for i in range(round(n/2))))
ptListat=list(listaSliced)
print(ptListat)
listaReversed = reversed(listaSimpla)
print(list(listaReversed))
print(list(enumerate(student)))
```

```
[(['5'], ['3']), (['5'], ['7']), (['11'], ['13']), (['17'], ['19']), (['23'],
['29']), (['31'], ['137']), (['41'], ['143']), (['47'], ['53']), (['59'],
['61']), (['67'], ['71']), (['173'], ['9']), (['83'], ['89']), (['97'],
['101']), (['103'], [])]
['103', '101', '97', '89', '83', '9', '173', '71', '67', '61', '59', '53',
'47', '143', '41', '137', '31', '29', '23', '19', '17', '13', '11', '7', '5',
'3', '5']
[(0, 'manaca'), (1, 'doarme'), (2, 'femei')]

Process finished with exit code 0
```

map() echivalent pentru for

```
for e in it:
        func(e)
map(func, it)
```

- **Versiunea 2**

```
compunereFunctii = lambda f, *args: f(*args)
map(compunereFunctii, [f1, f2, f3])
```

- **Versiunea 3**

```
compunereFunctii = lambda fns, *args: [list(map(fn, *args)) for fn in fns]
```

și exemplul de aplicare

```
ista1=list(range(11, 17))
f1=lambda x:x*2
f2=lambda x:x+2
f3=lambda x:round(x/2)
tabelFunctii=[f1, f2, f3]
compunereFunctii = lambda tabelFunctii, *args: [list(map(functie, *args)) for functie in tabelFunctii]
lista=compunereFunctii([f1,f2,f3],lista1)
print(lista)
```

si rezultatul executiei

```
[[22, 24, 26, 28, 30, 32], [13, 14, 15, 16, 17, 18], [6, 6, 6, 7, 8, 8]]

Process finished with exit code 0
```

atenție la următoarele diferențe:

```
map(lambda for v: in map(lambda for w: v + w, in y), in x) #este "pseudo" lambda
```

iar codul echivalent

```
map(lambda v : map(lambda w : v + w, y), x)          #este lambda real
```

echivalare if elif în calcul funcțional sau scurtcircuit

- # structură standard de control a fluxului de date

if <cond1>: func1()

elif <cond2>: func2()

else: func3()

- # Și echivalentul ei funcțional numit câteodată și expresie scurtcircuit

(<cond1> and func1()) or (<cond2> and func2()) or (func3())

- În sfârșit vine și lambda cu scurtcircuit pentru aceași structură:

lambdascurtcircuit = lambda x: (cond1 and func1(parlist)) or (cond2 and func2(parlist)) or (func3(parlist))

Expresii scurtcircuit

lista2 = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103]

```
val1 = lambda x: x in range(0, 3)
val2 = lambda x: x in range(3, 100)
val3 = lambda x: x > 100
f1 = lambda x: x * 2
f2 = lambda x: x + 2
f3 = lambda x: round(x / 2)
eval = (lambda x, y: y if x == True else False)
scurtcircuit = lambda x: eval(val1(x), f1(x)) or eval(val2(x), f2(x)) or eval(val3(x), f3(x))
map1 = map(scurtcircuit, lista2)
print(list(map1))
```

si rezultatul executiei

```
[4, 5, 7, 9, 13, 15, 19, 21, 25, 31, 33, 39, 43, 45, 49, 55, 61, 63, 69, 73, 75, 81, 85, 91, 99, 50, 52]

Process finished with exit code 0
```

Closure

```
def fabricaDeFunctii(x):
    def contine(lst):
        return x in lst
    return contine
```

si exemplu de executie

```
<function fabricaDeFunctii.<locals>.contine at 0x7fd15d496d90>
False
True
True

Process finished with exit code 0
```

```
amProcent100 = fabricaDeFunctii(100)
print(amProcent100)
print(amProcent100([1,2,30,40,50]))
print(amProcent100([1,2,30,40,50,100]))
print(amProcent100(range(1, 200)))
```

contor ca o closure - ver 1

```
def fabricaDeContoare():
    contorvar = 0
    def contor():
        nonlocal contorvar
        contorvar += 1
        return contorvar
    return contor
numarator1 =fabricaDeContoare()
print(numarator1())
print(numarator1())
print(numarator1())
numarator2 =fabricaDeContoare()
print(numarator2())
print(numarator2())
print(numarator2())
```

contor ca o closure - ver 2

```
def fabricaDeContoare():
    contorvar = 0
    def contor():
        nonlocal contorvar
        contorvar += 1
        return contorvar
    def valoareCurentaContor():
        nonlocal contorvar
        return contorvar
    return contor, valoareCurentaContor
contor,valoareContor = fabricaDeContoare()
print(contor())
print(contor())
print(contor())
print(valoareContor())
```

si rezultatul executiei

```
1
2
3
3

Process finished with exit code 0
```

Analiză de performanțe

```
import time
nrIteratii = 10000000
def f(k):
    return 2*k
def benchmark(functie, numeFunctie):
    start = time.time()
    functie()
    stop = time.time()
    print("calculul a durat {0} seconde pentru {1}".format((stop - start), numeFunctie))
def listaA():
    listaA = []
    for i in range(nrIteratii):
        listaA.append(f(i))
def listaB():
    listaB = [f(i) for i in range(nrIteratii)]
def listaC():
    listaC = map(f, range(nrIteratii))
benchmark(listaA, "structurata")
benchmark(listaB, "caclul functional")
benchmark(listaC, "operator dedicat")
```

si rezultatul executiei

```
calculul a durat 1.2460236549377441 seconde
pentru structurata
calculul a durat 0.9806721210479736 seconde
pentru caclul functional
calculul a durat 6.198883056640625e-06 seconde
pentru operator dedicat

Process finished with exit code 0
```

pentru de zece ori mai multe operatii

```
calculul a durat 11.70402979850769 seconde
pentru structurata
calculul a durat 9.303850650787354 seconde
pentru caclul functional
calculul a durat 5.722045898437e-06 seconde
pentru operator dedicat

Process finished with exit code 0
```

Excepții personalizate

```
#versiune simplista
def cereVarsta(varsta):
  try:
     assert int(varsta) > 18
  except ValueError:
     return 'Interzis minorilor'
  else:
     return 'batran dar nebunatic'


print(cereVarsta('ciaia bai'))
print(cereVarsta(60))
print(cereVarsta(6))
```

si rezultatul executiei

```
Interzis minorilor
Traceback (most recent call last):
batran dar nebunatic
  File "/home/bugs/PycharmProjects/exceptii
personalizate/exceptiii personalizate.py", line 12, in
<module>
    print(cereVarsta(6))
  File "/home/bugs/PycharmProjects/exceptii
personalizate/exceptiii personalizate.py", line 4, in
cereVarsta
    assert int(varsta) > 18
AssertionError

Process finished with exit code 1
```

Excepții personalizate

```
def cereVarsta(varsta):
  try:
     if(int(varsta) > 18):
        raise ZeroDivisionError
  except ValueError:
     return 'Interzis minorilor'
  else:
     return 'batran dar nebunatic'

print(cereVarsta('ciaia bai'))
print(cereVarsta(60))
print(cereVarsta(6))
```

si rezultatul executiei

```
Interzis minorilor
Traceback (most recent call last):
  File "/home/bugs/PycharmProjects/exceptii personalizate/exceptiii
personalizate.py", line 12, in <module>
    print(cereVarsta(60))
  File "/home/bugs/PycharmProjects/exceptii personalizate/exceptiii
personalizate.py", line 5, in cereVarsta
    raise ZeroDivisionError
ZeroDivisionError
```

Excepții personalizate

```
class EsteMinor(Exception):
  pass

def areVarsta(varsta):
  if int(varsta) < 18:
     raise EsteMinor

try:
   areVarsta(23)
   areVarsta(17)
except EsteMinor:
   print("Va rugam sa reveniti cand imbatraniti")
```

si rezultatul executiei

```
Va rugam sa reveniti cand imbatraniti

Process finished with exit code 0
```

Decorator - stil macro

```
# decorator fara argumente
def ornament(oFunctie):
    def impachetezFunctia():
        print("blocul predecesor al apelului functiei originale")
        oFunctie()
        print("blocul succesor al apelului functiei originale")
    return impachetezFunctia

# aplicam decoratorul asupra unei functii si o definim
@ornament
def functie1():
    print("se executa functia originala din decorator")

#apelul decoratorului
functie1()
```

si rezultatul executiei

```
blocul predecesor al apelului functiei originale
se executa functia originala din decorator
blocul succesor al apelului functiei originale

Process finished with exit code 0
```

Decorator - stil PeOO

```
class ornament(object):

   def __init__(self, f):
      print("sunt in constructorul decoratorului")
      f()
   def __call__(self):
      print("sunt in decorator")

@ornament
def functie1():
   print("sunt in interiorul functiei decorate")

functie1()
```

si rezultatul decorarii

```
sunt in constructorul decoratorului
sunt in interiorul functiei decorate
sunt in decorator
```

Decorator POO versiunea mai serioasă

```
#versiune mai serioasa
class ornament(object):
   def __init__(self, f):
      self.f = f
   def __call__(self):
      print("Intru in corpul decorator", self.f.__name__)
      self.f()
      print("ies din corpul decoratorului", self.f.__name__)
@ornament
def functia1():
   print("sunt in functia 1")
@ornament
def functia2():
   print("sunt in functia 2")

functia1()
functia2()
```

si rezultatul executiei

```
Intru in corpul decorator functia1
sunt in functia 1
ies din corpul decoratorului functia1
Intru in corpul decorator functia2
sunt in functia 2
ies din corpul decoratorului functia2

Process finished with exit code 0
```

Decorator cu parametri

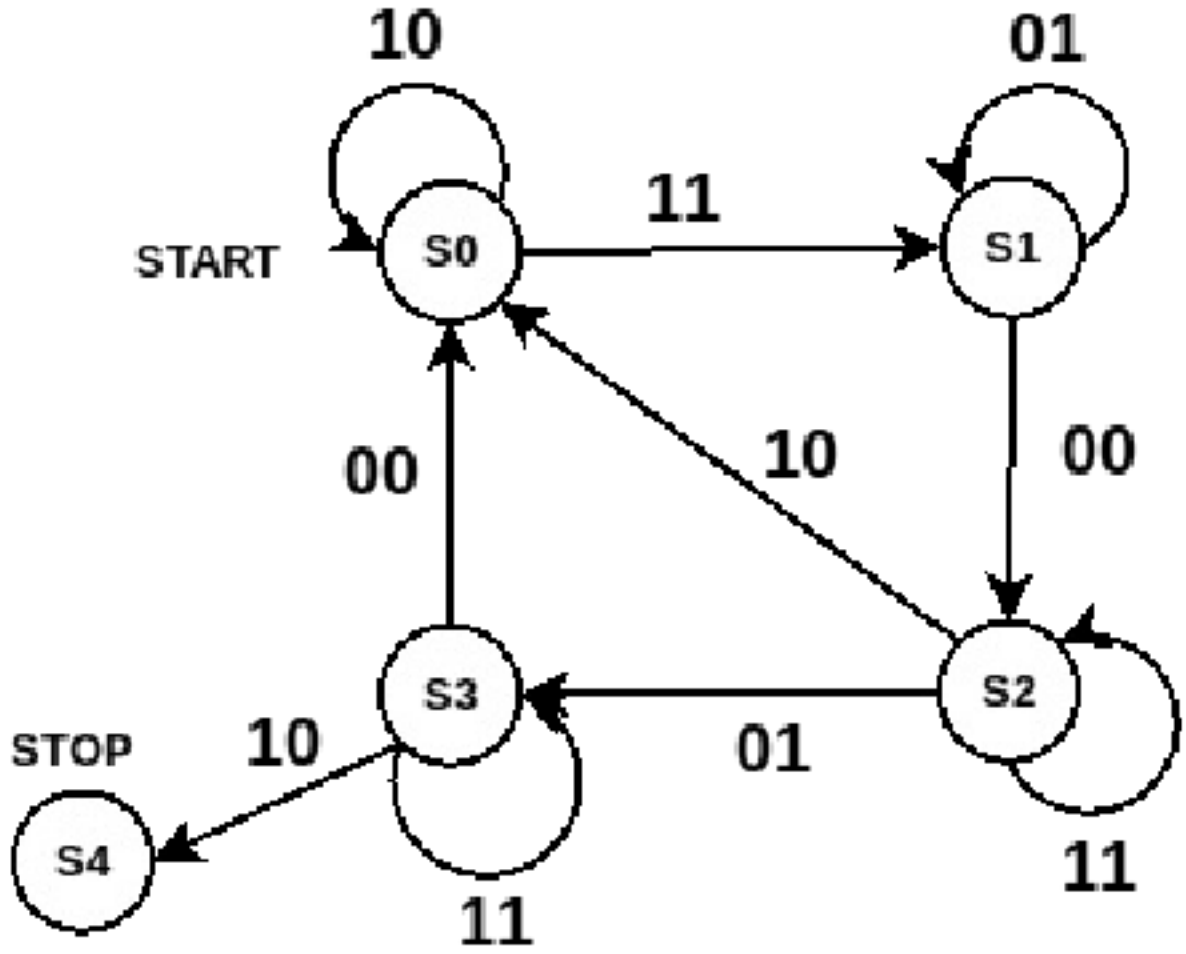
```
class ornamentCuParametri(object):
   def __init__(self, par1, par2, par3):
      #print("nu mai pot apela functia decorata in constructorul decoratorului")
      self.par1 = par1
      self.par2 = par2
      self.par3 = par3
   def __call__(self, f):
      #print("in interiorul decoratorului")
      def functieImpachetata(*args):
         #print("in interiorul functiei de impachetare")
         print("Argumentele decoratorului sunt", self.par1, self.par2, self.par3)
         f(*args)
        #print("dupa apelul functiei decorate")
      return functieImpachetata
@ornamentCuParametri("vreau", "sa merg", 42)
def sayHello(a1, a2, a3, a4):
   print("%s %s %s %s" %(a1, a2, a3, a4))

sayHello("ma","duc","la","bere")
```

Decorator utilizat ca adapter

```
class ornamentCuParametri(object):
   def __init__(self, par1, par2, par3):
      #print("nu mai pot apela functia decorata in constructorul decoratorului")
      self.par1 = par1
      self.par2 = par2
      self.par3 = par3
   def __call__(self, f):
      #print("in interiorul decoratorului")
      def functieImpachetata(*args):
         print("Argumentele decoratorului sunt", self.par1, self.par2, self.par3)
         #if(f.__code__.co_varnames[0]=='ma'):f(*args)
         if(not args[0]):
            f(self.par1,args[1],args[2],args[3])
         else:
            f(*args)
       return functieImpachetata
@ornamentCuParametri("vreau", "sa merg", 42)
def sayHello(a1, a2, a3, a4):
   print("%s %s %s %s" %(a1, a2, a3, a4))
sayHello("","duc","la","bere")
sayHello("ma","duc","la","bere")
```

```
Argumentele decoratorului sunt vreau sa merg 42
vreau duc la bere
Argumentele decoratorului sunt vreau sa merg 42
ma duc la bere
```

Să implementăm un automat



Implementare state machine

```
class StateMachine():
    def __init__(self):
        self.handlers = {}
        self.startState = None
        self.endStates = []
        self.cargo1 = -1
        self.cargo2 = -1
        self.handler=0
    def setCargo(self, cargo1, cargo2):
        self.cargo1 = cargo1
        self.cargo2 = cargo2
    def add_state(self, name, handler, start_state=0,
end_state=0):
        name = name.upper()
        self.handlers[name] = handler
        if end_state:
            self.endStates.append(name)
        if start_state:
            self.startState = name.upper()
        self.handler = self.handlers[self.startState]
```

Implementare state machine

```
    def run(self):
        (newState, self.cargo1, self.cargo2) =
self.handler(self.cargo1, self.cargo2)
        if newState.upper() in self.endStates:
            print("Am ajuns in stare terminala")
        else:
            self.handler = self.handlers[newState.upper()]
def stare0(x1, x2):
    print("S0")
    while True:
        if (x1 == 1) and (x2 == 1):
            newState = "S1"
            break
        elif (x1 == 1) and (x2 == 0):
            newState = "S0"
            break
        else:
            newState = "S4"
            print("inrare eronata - STOP")
            break
    print(">>")  # pe post de actiune in starea curenta
    return (newState, x1, x2)
```

```
def stare1(x1, x2):
    print("S1")
    while True:
        if (x1 == 0) and (x2 == 1):
            newState = "S1"
            break
        elif (x1 == 0) and (x2 == 0):
            newState = "S2"
            break
        else:
            newState = "S4"
            print("inrare eronata - STOP")
            break
    print(">>")  # pe post de actiune in starea curenta
    return (newState, x1, x2)
def stare2(x1, x2):
    print("S2")
    while True:
        if (x1 == 1) and (x2 == 1):
            newState = "S2"
            break
        elif (x1 == 1) and (x2 == 0):
            newState = "S0"
            break
        elif (x1 == 0) and (x2 == 1):
            newState = "S3"
            break
        else:
            newState = "S4"
            print("inrare eronata - STOP")
            break
    print(">>")  # pe post de actiune in starea curenta
    return (newState, x1, x2)
```

Implementare state machine

```
def stare3(x1, x2):
    print("S3")
    while True:
        if (x1 == 1) and (x2 == 1):
            newState = "S3"
            break
        elif (x1 == 0) and (x2 == 0):
            newState = "S0"
            break
        elif (x1 == 1) and (x2 == 0):
            newState = "S4"
            break
        else:
            newState = "S4"
            print("inrare eronata - STOP")
            break
    print(">>")  # pe post de actiune in starea curenta
    return (newState, x1, x2)
```

```
def stare4(x1, x2):
    print("S4-STOP")
    newState = "S4"
    return (newState, x1, x2)
if __name__ == "__main__":
    m = StateMachine()
    m.add_state("S0", stare0,1,0)
    m.add_state("S1", stare1)
    m.add_state("S2", stare2)
    m.add_state("S3", stare3)
    m.add_state("S4", None, 0,1)

    x1x2=[(1,0),(1,1),(0,0),(1,1),(0,1),(1,1),(1,0)]
    for i in x1x2:
        print(i)
        m.setCargo(*i)
        m.run()
```