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))
```

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]
```

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))

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
```