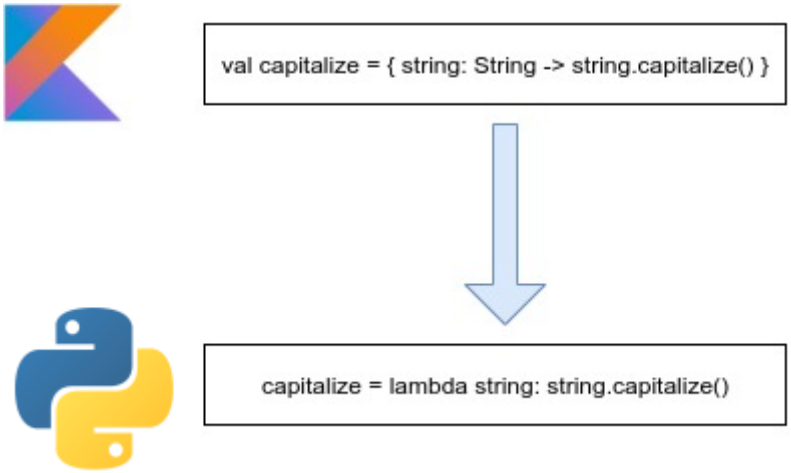
Laboratorul 13: Paradigma calculului funcțional în Python

Introducere

Pentru o imagine de ansamblu asupra paradigmei de calcul funcțional în Python, se recomandă parcurgerea următoarelor resurse:

- **„Functional Python Programming" (2nd edition) - Steven F. Lott**
- **„A functional start to computing with Python" - Ted Herman**
- Cursul 13 de la disciplina *Paradigme de programare*
- Documentația oficială:
  - » https://docs.python.org/3/howto/functional.html
  - » https://docs.python.org/3/library/itertools.html
  - » https://more-itertools.readthedocs.io/en/stable/api.html
  - » https://docs.python.org/3/library/functools.html
  - » https://docs.python.org/3/library/operator.html#mapping-operators-to-functions
  - » https://docs.python.org/3/library/functions.html
  - » https://pymonet.readthedocs.io/
  - » https://docs.pyfunctional.org/



Echivalentul expresiilor lambda din Kotlin cu cele din Python

Aşa NU:

```
map(lambda x, f=lambda x, f: (x <= 1) or (f(x - 1, f) + f(x - 2, f)):
    f(x, f),
    range(30))
```

Exemple

Modulul builtins

Vezi comentariile din cod!

```
import random

class A:
    pass

if __name__ == '__main__':
    all([True, True, False, True])  # False
    all([A(), A(), None])  # False
    all([A(), A()])  # True
    all([True] * 4)  # True
    any([False, True, False, False])  # True
    any([False] * 4)  # False

    min([5, 84, 17, 2, 20, 3, -9, 11, 7])  # -9
    max([5, 84, 17, 2, 20, 3, -9, 11, 7])  # 84

    list(reversed([1, 2, 3, 4]))  # [4, 3, 2, 1]
    ''.join(list(reversed('abcd')))  # dcba

    sum([1, 2, 3, 4])  # 10

    ''' sort vs sorted:
    sort modifica variabila
    sorted returneaza o noua colectie sortata'''
    my_list = [-9, 2, 3, 5, 7, 11, 17, 20, 84]
    my_dict = {1: (1, 43), 2: (5, 12), 3: (3, 36)}
    sorted(my_list)  # [-9, 2, 3, 5, 7, 11, 17, 20, 84]
    my_list.sort()  # my_list = [-9, 2, 3, 5, 7, 11, 17, 20, 84]
    sorted(my_list, reverse=True)  # [84, 20, 17, 11, 7, 5, 3, 2, -9]

    # sortare dictionar dupa valoare:
    {k: v for k, v in sorted(my_dict.items(), key=lambda item:
item[1])}
    # sortare dictionar dupa primul element din tupla (valoare)
    {k: v for k, v in sorted(my_dict.items(), key=lambda item:
item[1][0])}

    list0 = list(range(5))
    list1 = list('abcdefghi')
    for item in zip(list0, list1):
        print(item)  # (0, 'a'), (1, 'b'), (2, 'c'), (4, 'd')
    # sau:
    for first, second in zip(list0, list1):
        print(first, second)  # 0 a, 1 b, 2 c, 4 d

    # slice(start, end, step) -> implicit, step=1
    print(list0[0:4])  # sau print(list0[:4])  # sau
print(list0[slice(0, 4)])
    print(list0[-1])  # ultimul element
```

```
    # set comprehension:
    {random.randint(1, 10) for _ in range(15)}  # set de numere
intregi distincte

    # list comprehension:
    [item for item in range(15) if item % 2 == 0]  # [0, 2, 4, ...,
14]

    # dict comprehension:
    {n: n ** 2 for n in range(10)}  # dictionar cu patratele perfecte
pana la 81

    # generator comprehension:
    (n ** 2 for n in range(10))  # generator([0, 1, 4, 9, ..., 81])

    # filter
    filter(lambda x: x % 2 == 0 and x > 5, range(20))  # 6, 8, 10, 12,
14, 16, 18

    # map
    map(lambda x: (x+1)*(x+1), range(10))  # (x+1)^2 for x in
range(10)
```

Exemplu de extensii

```
class A:
    pass


def custom_extension(self):
    print("whatever")


class int(int):
    def is_even(self):
        return self % 2 == 0


if __name__ == '__main__':
    # se pot crea extensii DOAR pe clase custom
    A.whatever = custom_extension
    a_instance = A()
    a_instance.whatever()  # afiseaza whatever

    print(int(6).is_even())  # True
    print(int(5).is_even())  # False
```

Modulul itertools

Vezi comentariile din cod!

```
import os
import operator
import itertools
from collections import namedtuple

City = namedtuple('City', ['city_name', 'country_code',
'city_zip_code'])

if __name__ == '__main__':
    ''' itertools.count(start, step) -> returneaza un flux de date
infinit '''
    itertools.count()  # 0, 1, 2, 3, 4, ...
    itertools.count(10)  # 10, 11, 12, 13, 14, ...
    itertools.count(10, 5)  # 10, 15, 20, 25, 30, ...

    ''' itertools.cycle(iter) -> repeta la infinit iterabilul
primit'''
    itertools.cycle([1, 2, 3])  # 1, 2, 3, 1, 2, 3, 1, 2, 3, ...

    ''' itertools.repeat(elem, [n]) -> returneaza elementul primit de
n ori (sau infinit daca n lipseste) '''
    itertools.repeat('abc')  # abc, abc, abc, abc, ...
    itertools.repeat('abc', 3)  # abc, abc, abc

    ''' itertools.accumulate(iterable[, func, *, initial=None) ->
creeaza un iterator care returneaza sumele acumulate, sau rezultatul
acumularii altor functii binare specificate prin argumentul func'''
    itertools.accumulate([1, 2, 3, 4, 5])  # 1, 3, 6, 10, 15
    itertools.accumulate([1, 2, 3, 4, 5], operator.mul)  # 1, 2, 6,
24, 120

    ''' itertools.chain(iterA, iterB, ...) -> primeste un numar
oarecare de iteratori si returneaza pe rand, toate elementele
iteratorilor primiti'''
    itertools.chain(['a', 'b', 'c'], (1, 2, 3))  # a, b, c, 1, 2, 3
    ''' de asemenea, se poate folosi si operatorul de despachetare:'''
    [*['a', 'b', 'c'], *(1, 2, 3)]  # ['a', 'b', 'c', 1, 2, 3]

    ''' itertools.islice(iter, [start], stop, [step]) -> returneaza un
stream (flux de date) compus dintr-o portiune a iteratorului
initial'''
    itertools.islice(range(10), 8)  # 0, 1, 2, 3, 4, 5, 6, 7
    itertools.islice(range(10), 2, 8)  # 2, 3, 4, 5, 6, 7
    itertools.islice(range(10), 2, 8, 2)  # 2, 4, 6

    ''' itertools.tee(iter, [n]) -> replica iteratorul initial in n
iteratori independenti; implicit, n=2 (daca nu este precizat)'''
    itertools.tee(itertools.repeat('abc', 5))  # (iter1, iter2)
    itertools.tee(itertools.repeat('abc', 5), 3)  # (iter1, iter2,
iter3)

    ''' itertools.zip_longest(*iterables, fillvalue=None) -> creeaza
un iterator compus din tuple cu cate un element din fiecare
```

```
iterabil'''
    itertools.zip_longest('ABCD', 'xy', fillvalue='-')
    # ('A', 'x'), ('B', 'y'), ('C', '-'), ('D', '-')

    ''' itertools.starmap(func, iter) -> primeste un iterator care va
fi un stream de tuple si apeleaza func folosind elementele din tupla
ca argumente'''
    itertools.starmap(os.path.join, [('/bin', 'python'),
                                     ('/usr', 'bin', 'java'),
                                     ('/usr', 'bin', 'perl'),
                                     ('/usr', 'bin', 'ruby')])
    # /bin/python, /usr/bin/java, /usr/bin/perl, /usr/bin/ruby
    itertools.starmap(operator.add, zip([1, 2, 3], [4, 5, 6]))  # 5,
7, 9

    ''' itertools.filterfalse(predicate, iter) -> opusul functiei
filter, returneaza elementele pentru care predicatul returneaza
False'''
    itertools.filterfalse(lambda x: x % 2 == 0, list(range(10)))  # 1,
3, 5, 7, 9
    # echivalent cu:
    filter(lambda x: x % 2 != 0, list(range(10)))

    ''' itertools.takewhile(predicate, iter) -> returneaza elemente
cat timp predicatul returneaza True'''
    itertools.takewhile(lambda x: x < 5, itertools.count())  # 0, 1,
2, 3, 4
    itertools.takewhile(lambda x: x % 2 == 0, itertools.count())  # 0

    ''' itertools.dropwhile(predicate, iter) -> elimina elemente cat
timp predicatul returneaza True, dupa care returneaza restul
iteratorului'''
    itertools.dropwhile(lambda x: x < 10, itertools.count())  # 10,
11, 12, ...
    itertools.dropwhile(lambda x: x % 2 == 0, itertools.count())  # 1,
2, 3, ...

    ''' itertools.compress(data, selectors) -> primeste 2 iteratori si
returneaza doar elementele din primul iterator pentru care elementul
corespunzator din al doilea este True'''
    itertools.compress([1, 2, 3, 4, 5], [True, True, False, False,
True])  # 1, 2, 5

    # Functii combinatorice
    ''' itertools.combinations(iterable, r) -> returneaza un iterator
care da toate tuplele de lungime r combinatii ale elementelor din
iterator'''
    itertools.combinations([1, 2, 3, 4], 2)  # combinari de 4 luate
cate 2
    '''(1, 2), (1, 3), (1, 4),
       (2, 3), (2, 4),
       (3, 4)'''
    itertools.combinations([1, 2, 3, 4], 3)  # combinari de 4 luate
cate 3
    '''(1, 2, 3), (1, 2, 4), (1, 3, 4),
       (2, 3, 4)'''
```

```
    ''' itertools.combinations_with_replacement(iterable, r) -> spre
deosebire de functia itertools.combinations. elementele se pot repeta
in aceeasi tupla'''
    itertools.combinations_with_replacement([1, 2, 3, 4], 2)
    '''(1, 1), (1, 2), (1, 3), (1, 4),
       (2, 2), (2, 3), (2, 4),
       (3, 3), (3, 4),
       (4, 4)'''

    ''' itertools.permutations(iterable, r=None) -> returneaza toate
aranjamentele de lungime r'''
    itertools.permutations([1, 2, 3, 4], 2)  # aranjamente de 4 luate
cate 2
    '''(1, 2), (1, 3), (1, 4),
       (2, 1), (2, 3), (2, 4),
       (3, 1), (3, 2), (3, 4),
       (4, 1), (4, 2), (4, 3)'''
    itertools.permutations(([1, 2, 3]))
    # (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)

    # Gruparea elementelor
    City = namedtuple('City', ['city_name', 'country_code',
'city_zip_code'])
    city_list = [City(city_name='Iasi', country_code='RO',
city_zip_code=100),
                 City('Bucuresti', 'RO', 200),
                 City('Cluj', 'RO', 300),
                 City('Paris', 'FR', 1100),
                 City('Lyon', 'FR', 1200),
                 City('Copenhagen', 'DK', 2100)]

    ''' itertools.groupby(iter, key_func=None) -> colecteaza toate
elementele consecutive din iterator care au aceeasi cheie si
returneaza un stream de tuple cu 2 elemente: cheia si un iterator cu
elementele corespunzatoare cheii'''
    itertools.groupby(city_list, lambda city: city.country_code)
    # echivalent cu:
    itertools.groupby(city_list, lambda city: city[1])
    ''' ('RO', iterator-1), unde iterator-1 este:
    City(city_name='Iasi', country_code='RO', city_zip_code=100)
    City(city_name='Bucuresti', country_code='RO', city_zip_code=200)
    City(city_name='Cluj', country_code='RO', city_zip_code=300)
        ('FR', iterator-2), unde iterator-2 este:
    City(city_name='Paris', country_code='FR', city_zip_code=1100)
    City(city_name='Lyon', country_code='FR', city_zip_code=1200)
        ('DK', iterator-3), unde iterator-3 este:
    City(city_name='Copenhagen', country_code='DK', ity_zip_code=2100)
'''
```

Modulul more_itertools

Instalare dependențe:

```
sudo pip3 install more-itertools
```

Codul sursă (**vezi comentariile din cod!**):

```
import more_itertools
import datetime
from pprint import pprint
from decimal import Decimal

''' more_itertools.make_decorator() si more_itertools.map_except()'''
mapper_except =
more_itertools.make_decorator(more_itertools.map_except,
                                                result_index=1)

@mapper_except(float, ValueError, TypeError)
def read_file(f):
    # Citeste text care contine si numere
    # ...
    return ['1.5', '6', 'not-important', '11', '1.23E-7', 'remove-me',
'25', 'trash']
# list(read_file("file.txt")) -> [1.5, 6.0, 11.0, 1.23e-07, 25.0]


if __name__ == '__main__':
    ''' more_itertools.divide(n, iterable) -> imparte un iterator in n
    sub-iteratori'''
    [list(iterator) for iterator in more_itertools.divide(3,
list(range(8)))]
    # [[0, 1, 2], [3, 4, 5], [6, 7]]

    ''' more_itertools.partition(func, iterable) ->'''
    files = ["file0.jpg", "file1.exe", "file2.gif", "file3.txt",
"file4.bin"]
    ALLOWED_EXTENSIONS = ('jpg', 'jpeg', 'gif', 'bmp', 'png')
    is_allowed = lambda file: file.split('.')[1] in ALLOWED_EXTENSIONS
    forbidden, allowed = more_itertools.partition(is_allowed, files)
    print(list(allowed), list(forbidden))

    ''' more_itertools.flatten(list_of_lists) -> despacheteaza listele
intr-una singura'''
    print(list(more_itertools.flatten([[1, 2], [3, 4], [5, 6]])))  #
[1, 2, 3, 4, 5, 6]

    ''' more_itertools.collapse(iterable) -> similar cu flatten, dar
permite despachetarea mai multor niveluri de imbricari'''
    tree = [40, [25, [10, 3, 17], [32, 30, 38]], [78, 50, 93]]
    print(list(more_itertools.collapse(tree)))

    ''' more_itertools.split_at(iterable, predicate) -> similar cu
split, dar pentru colectii'''
    lines = ['abc', 'def', None, 'ghi', None, 'jkl', 'mno']
    print(list(more_itertools.split_at(lines, lambda item: item is
None)))
```

```
    ''' more_itertools.map_reduce(iterable, key_func, value_func,
reduce_func)'''
    data = 'This sentence has words of various lengths in it, both
short ones and long ones'.split()
    key_func = lambda word: len(word)
    pprint(more_itertools.map_reduce(data, key_func))
    '''defaultdict(None,
            {2: ['of', 'in'],
             3: ['has', 'it,', 'and'],
             4: ['This', 'both', 'ones', 'long', 'ones'],
             5: ['words', 'short'],
             7: ['various', 'lengths'],
             8: ['sentence']})'''
    value_func = lambda word: 1
    pprint(more_itertools.map_reduce(data, key_func, value_func))
    '''defaultdict(None,
            {2: [1, 1],
             3: [1, 1, 1],
             4: [1, 1, 1, 1, 1],
             5: [1, 1],
             7: [1, 1],
             8: [1]})'''
    reduce_func = sum
    pprint(more_itertools.map_reduce(data, key_func, value_func,
reduce_func))
    # defaultdict(None, {4: 5, 8: 1, 3: 3, 5: 2, 2: 2, 7: 2})

    ''' more_itertools.sort_together() -> sortare dupa coloane'''
    cols = [
        ("John", "Ben", "Andy", "Mary"),  # Name
        ("1994-02-06", "1985-04-01", "2000-06-25", "1998-03-14"),  #
Date of birth
        ("2020-05-14", "2019-05-15", "2020-05-16", "2018-08-17")  #
updated at
    ]
    pprint(more_itertools.sort_together(cols, key_list=(1, 2)))

    ''' more_itertools.filter_except()'''
    data = ['1.5', '6', 'not-important', '11', '1.23E-7', 'remove-me',
'25', 'trash']
    print(list(map(float, more_itertools.filter_except(float, data,
TypeError, ValueError))))
    '''se filtreaza elementele din iterator, incercand convertirea
acestora la float
    prin functia float(item), eliminand elementele pentru care se
genereaza exceptie'''
    #  [1.5, 6.0, 11.0, 1.23e-07, 25.0]

    ''' more_itertools.numeric_range(start, stop, step) -> similar cu
range, dar functioneaza si cu alte tipuri de date (nu doar int)'''
    print(list(more_itertools.numeric_range(Decimal(1.5),
Decimal(3.6), Decimal(0.5))))
    # [Decimal('1.5'), Decimal('2.0'), Decimal('2.5'), Decimal('3.0'),
Decimal('3.5')]
    start = datetime.datetime(2020, 5, 15)
```

```
    stop = datetime.datetime(2020, 5, 23)
    step = datetime.timedelta(days=2)
    print(list(more_itertools.numeric_range(start, stop, step)))
    '''[datetime.datetime(2020, 5, 15, 0, 0),
    datetime.datetime(2020, 5, 17, 0, 0),
    datetime.datetime(2020, 5, 19, 0, 0),
    datetime.datetime(2020, 5, 21, 0, 0)]'''

    ''' more_itertools.roundrobin(*iterables)'''
    list(more_itertools.roundrobin('ABC', 'D', 'EF'))  # [A, D, E, B,
F, C]
```

Modulul functools

Vezi comentariile din cod!

```
import functools
import operator

def log(message, subsystem):
    print("{}: {}".format(subsystem, message))

class Cell:
    def __init__(self):
        self._alive =  False

    @property
    def alive(self):
        return self._alive

    def set_state(self, state):
        self._alive = bool(state)

    set_alive = functools.partialmethod(set_state, True)
    set_dead = functools.partialmethod(set_state, True)

if __name__ == '__main__':
    ''' functools.partial(func, *args, **kwargs) -> returneaza un nou
obiect partial care se va comporta ca o functie cand va fi apelat.
Functia permite adaugarea unor argumente cu o valoare implicita'''
    server_log = functools.partial(log, subsystem='server')
    server_log("whatever")

    ''' functools.partialmethod(func, *args, **kwargs) -> similar cu
functia partial, doar ca e specifica pentru metodele dintr-o clasa'''
    cell = Cell()
    cell.set_alive()
    print(cell.alive)  # @property -> nu mai trebuie apelata (e
proprietate)

    ''' functools.reduce(func, iter, [initial_value]) -> efectueaza
cumulativ o operatie pe toate elementele din iterabil (nu poate fi
aplicat pe iteratori infiniti)
    Argumentul func este o functie care primeste doua elemente si
calculeaza func(A, B)'''
    functools.reduce(operator.concat, ['A', 'BC', 'DEF', 'G'])  #
```

```
ABCDEFG
    # exemplu echivalent cu: ''.join(['A', 'BC', 'DEF', 'G'])
    functools.reduce(operator.mul, [1, 2, 3], 1)  # 6 (factorial)
    functools.reduce(operator.add, [1, 2, 3, 4], 0)  # 10
```

Modulul PyFunctional (similar cu procesările din Kotlin)

Instalare dependențe:

```
sudo pip3 install pyfunctional
```

Codul sursă:

```
from functional import seq
from collections import namedtuple

Transaction = namedtuple('Transaction', ['reason', 'amount'])
transactions = [
    Transaction(reason='github', amount=7),
    Transaction(reason='food', amount=10),
    Transaction(reason='coffee', amount=5),
    Transaction(reason='netflix', amount=5),
    Transaction(reason='food', amount=5),
    Transaction(reason='youtube', amount=25),
    Transaction(reason='food', amount=10),
    Transaction(reason='amazon', amount=200),
    Transaction(reason='paycheck', amount=-1000)
]

if __name__ == '__main__':
    seq(1, 2, 3, 4)\
        .map(lambda x: x * 2)\
        .filter(lambda x: x > 4)\
        .reduce(lambda x, y: x + y)  # 14
    # sau fara \
    (seq(1, 2, 3, 4)
        .map(lambda x: x * 2)
        .filter(lambda x: x > 4)
        .reduce(lambda x, y: x + y))  # 14

    # sintaxa inspirata din Spark:
    food_cost = seq(transactions) \
        .filter(lambda x: x.reason == 'food') \
        .map(lambda x: x.amount).sum()

    # sintaxa inspirata din LINQ
    food_cost = seq(transactions)\
        .where(lambda x: x.reason == 'food')\
        .select(lambda x: x.amount).sum()

    words = 'I dont want to believe I want to know'.split(' ')
    seq(words).map(lambda word: (word, 1)).reduce_by_key(lambda x, y:
x + y)
    '''[('dont', 1), ('I', 2), ('to', 2), ('know', 1), ('want', 2),
('believe', 1)]'''
```

Aplicații și teme

Aplicații de laborator:

1. Pornind de la exemplul cu decorator care simulează programarea orientată pe aspecte din laboratorul 9, să se creeze o extensie la clasa *int* care să determine dacă numărul este prim sau nu. Exercițiul este unul academic, funcția decorată din clasa *int* nefiind implementată.

2. Pornind de la exemplul de extensii din laboratorul curent, să se creeze o extensie la clasa *str* care să convertească un string la PascalCase.

3. Să se implementeze funcția *zip* cu ajutorul funcției *map.*

4. Utilizând *generator comprehension* şi funcția *filter* să se determine toate pătratele perfecte (care sunt numere pare) de la 0 până la n, unde n este citit de la tastatură.

5. Utilizând secvențe (PyFunctional), să se elimine caracterele duplicate dintr-un string citit de la tastatură (`aaaabbbcc` devine `abc`).

6. Utilizând secvențe (PyFunctional), să se micșoreze un string citit de la tastatură astfel: `aaaabbbccd` devine `a4b3c2d`. Dacă caracterul apare o singură dată, nu se va adăuga count-ul.

7. Să se implementeze în două moduri (o dată cu funcția *partial*, o dată cu *starmap*) o funcție care primeşte un iterator şi o constantă k şi returnează rezultatul aplicării funcției `f(x, k) = x * x + k` peste fiecare element din iterator.

8. Pornind de la exemplul de automat din cursul 13, să se implementeze un automat cu 4 stări în care să se proceseze cu expresii lambda o listă de numere întregi trimisă ca argument în constructor astfel:
   - lista să rămână în starea0 de prelucrare cât timp mai are elemente care nu sunt prime
   - lista să rămână în starea1 de prelucrare cât timp mai are elemente pare
   - lista să rămână în starea2 de prelucrare cât timp mai are elemente > 50
   - lista ajunge în starea de STOP unde va fi afişată.

Tema pe acasă:

1. Să se reia exemplul de utilizare a secvențelor şi funcțiilor de prelucrare a datelor din laboratorul 12 şi să se implementeze în Python cu ajutorul modulului *PyFunctional*.

2. Pornind de la exemplul cu *map_reduce* din modulul *more_itertools,* să se implementeze un algoritm de sortare. Funcția de mapare va extrage cuvintele dintr-un text (poate fi memorat şi într-o variabilă) şi va returna perechi de forma `<prima_literă_din_cuvânt, cuvântul>`, iar funcția de reducere va reuni perechile generate de funcția de mapare în perechi de forma `<literă, [cuvânt0, cuvânt1, ...]>`.

3. Utilizând programarea funcțională, să se efectueze următoarele operații pe lista [1, 21, 75, 39, 7, 2, 35, 3, 31, 7, 8]:
   - Eliminarea oricărui număr < 5 --> [21, 75, 39, 7, 35, 31, 7, 8]
   - Gruparea numerelor în perechi --> [(21, 75), (39, 7), (35, 31), (7, 8)]
   - Multiplicarea numerelor din perechi --> [1575, 273, 1085, 56]

- Sumarea rezultatelor --> 1575 + 273 + 1085 + 56 = 2989

4. Să se reia problema 8 cu automatul şi să se implementeze utilizând doar expresii lambda, funcții *map / reduce / filter*.