Laboratorul 4 - Explicatii

Problema #1 laboratorul #4

1. Care este outputul următoarelor comenzi:

a) `print([n ** 2 for n in (0, 1, 2, 3)])`

b) `print((n ** 2 for n in (0, 1, 2, 3) if n > 0))`

c) `print({n ** 2 for n in range(-3, 3)})`

Răspundeți, justificați și apoi rulați pentru a vă verifica răspunsul.

```
#a)
print([n ** 2 for n in (0,1,2,3)])

"""
Se creeaza o lista cu patratele numerelor din tuplul (0,1,2,3)

"""

#b)
print((n ** 2 for n in (0,1,2,3) if n > 0))

"""
Atunci cand avem paranteze rotunde expresia nu mai este o lista, ci un fel de
expresie care memoreaza modul cum sa genereze numerele 0,1,4,9. Aceasta expresie
'generator' nu memoreaza elementele 0,1,4,9, ci doar stie cum sa le obtina.
Astfel, nu putem accesa elementele 0,1,4,9, fiindca ele nu exista.
Cand printam, este afisat faptul ca acela este un obiect de tip generator
si ii vedem si adresa din memorie (0x7f9a1cb28520).
Datorita acestui fapt expresia generator este mai eficienta din punct de vedere al me
moriei.

Ca sa accesam efectiv elementele, avem doua optiuni:
    1. CONVERTIM GENERATORUL INTR-O LISTA

        print(list(n ** 2 for n in (0,1,2,3) if n > 0))

    2. ITERAM MANUAL

        for x in (n ** 2 for n in (0,1,2,3) if n > 0):
            print(x)
"""


#c)
print({n ** 2 for n in range(-3,3)})


"""
Citim pe romaneste:

Printeaza <vad acolada deci e vorba de comprehensiunea unui set> setul
format din n patrat pentru fiecare n de la -3 la 2 (ne oprim inainte
```

```
de capatul din dreapta).

BINE DE STIUT: (eliminarea duplicatelor dintr-o lista)
Acest lucru se poate face prin simpla instructiune ls = set(ls)
"""

ls = [1, 1, 1, 2, 3, 3, 4, 4, 0, 9, 9]
ls = set(ls)
print(ls)
# daca dorim sa afisam elementele setului fara virgule si paranteze avem
# instructiunea print(*ls)
print(*ls)
```

```
[0, 1, 4, 9]
<generator object <genexpr> at 0x7f9a1cb0edc0>
{0, 9, 4, 1}
{0, 1, 2, 3, 4, 9}
0 1 2 3 4 9
```

Problema #2 laboratorul #4

Care este outputul următoarelor comenzi:

a) `print({1, 2, 3} * 2)`

b) `print({1, 2, 3} * 0)`

c) `print({1, 2, 3} * -2)`

d) `print({1, 2, 3} + {4, 5, 6})`

e) `print({1, 2, 3} + {0, 1, 2})`

Răspundeți, justificați și apoi rulați pentru a vă verifica răspunsul.

```
#a)
print({1, 2, 3} * 2)
#b)
print({1, 2, 3} * 0)
#c)
print({1, 2, 3} * -2)
#d)
print({1, 2, 3} + {4, 5, 6})
#e)
print({1, 2, 3} + {0, 1, 2})
"""
Niciuna dintre operatiile de mai sus nu este suportata, deoarece '*', repetarea
si '+' concatenarea nu sunt definite pentru seturi.
Repetarea nu are sens, deoarece elementele unei multimi sunt unice
Concatenarea nu are sens, deoarece elementele unei multimi nu au o ordine.
"""
```

```
unsupported operand type(s) for *: 'set' and 'int'
```

Problema #3 laboratorul #4

Care este diferența dintre copierea simplă și copierea profundă a mulțimilor? Dar pentru tupluri?

Argumentați. (*cititi informatiile de mai jos si, in lumina acestora, incercati sa raspundeti singuri la intrebare*)

Despre copierea colectiilor (copiere simpla vs deepcopy)

Hai sa luam o lista (`lista1`) si sa incercam sa ii facem o copie simpla numita `lista2` careia sa ii adaugam ceva la final. Ce se intampla si de ce?

```
lista1 = [[1, 2, "Gigel"], None, 76]
print(f"lista1 before: {lista1}")

lista2 = lista1.copy()

print(f"lista2 before: {lista2}")

lista2[0].append("Militaru")

print(f"lista1 after: {lista1}")
print(f"lista2 after: {lista2}")
```

```
lista1 before: [[1, 2, 'Gigel'], None, 76]
lista2 before: [[1, 2, 'Gigel'], None, 76]
lista1 after: [[1, 2, 'Gigel', 'Militaru'], None, 76]
lista2 after: [[1, 2, 'Gigel', 'Militaru'], None, 76]
```

Ce am adaugat la `lista2` a ajuns in `lista1` . Motivul pentru asta este ca o lista nu contine obiectele in sine care "se vad pe ecran", ci **referinte** la ele.

Referinta la un obiect este de fapt adresa din memorie a acelui obiect. Daca printr-o referinta eu accesez obiectul si il schimb si apoi incerc sa il mai accesez si printr-o a doua referinta pe care o aveam creata, observ ca obiectul s-a schimbat.

Analogie: adresele din memorie sunt precum adresele caselor de pe o strada. Un obiect este o casa, cu o adresa. Daca un om are adresa unei case si merge si pune un brad de Craciun in acea casa, toti ceilalti oameni care vor accesa casa vor vedea schimbarea.

Copierea: Cand copiam acea lista, de fapt cream un "container" nou, dar ... (vezi mai jos):

```
l1 = [1, 2, 3, [4, 5]]
l2 = l1.copy()
print(id(l1) != id(l2))       # l1 si l2 nu sunt acelasi obiect (nu sunt acelasi con-
tainer)
```

```
True
```

... dar copiam fix aceleasi referinte la obiectele vechi. Din acest motiv, daca programatorul foloseste `l2` ca sa ii bage bradu-n casa listei, `l1` o sa aiba si ea brad in casa.

La fel, daca stergem din `l1` , `l2` va primi si el stergerea (spoiler, asta e o minciuna partiala):

```
print(f"l1 before: {l1}")
print(f"l2 before: {l2}")

del l1[0]

print(f"l1 after: {l1}")
print(f"l2 after: {l2}")
```

```
l1 before: [1, 2, 3, [4, 5]]
l2 before: [1, 2, 3, [4, 5]]
```

```
l1 after: [2, 3, [4, 5]]
l2 after: [1, 2, 3, [4, 5]]
```

```python
# Hai sa incercam sa stergem/modificam altceva din l1
l1 = [1, 2, 3, [4, 5]]
l2 = l1.copy()

print(f"l1 before: {l1}")
print(f"l2 before: {l2}")

del l1[3][0]

print(f"l1 after: {l1}")
print(f"l2 after: {l2}")
```

```
l1 before: [1, 2, 3, [4, 5]]
l2 before: [1, 2, 3, [4, 5]]
l1 after: [1, 2, 3, [5]]
l2 after: [1, 2, 3, [5]]
```

Se pare ca daca am sters un element care e numar intreg din `l1`, aceasta stergere nu a aparut si in `l2`, dar daca am sters un element din lista continuta (incuibata) in `l1`, schimbarea "s-a propagat" in `l2`.

Care-i diferenta dintre cele doua situatii?

In prima situatie, stergem un element **imutabil** (nr. intreg - int), iar in a doua stergem un element **mutabil** (lista).

Asadar, ajungem la concluzia:

Morala

Daca aveti o colectie ce contine lucruri **mutabile** (i.e. liste, set-uri, dictionare), folositi `deepcopy`.

Daca colectia voastra contine doar lucruri **imutabile**, puteti copia simplu.

```python
import copy

lista1 = [[1, 2, "Gigel"], None, 76]
lista2 = copy.deepcopy(lista1)

print(f"lista1 before: {lista1}")
print(f"lista2 before: {lista2}")

lista2[0].append("Militaru")

print(f"lista1 after: {lista1}")
print(f"lista2 after: {lista2}")
```

```
lista1 before: [[1, 2, 'Gigel'], None, 76]
lista2 before: [[1, 2, 'Gigel'], None, 76]
lista1 after: [[1, 2, 'Gigel'], None, 76]
lista2 after: [[1, 2, 'Gigel', 'Militaru'], None, 76]
```

Problema #4 laboratorul #4

Se citeste n, apoi un sir de n numere intregi. Sa se afiseze:

a) Cate perechi de numere identice se pot forma extragand numere din sir

Exemplu:

Input:

```
11
3
6
4
3
2
6
3
4
5
3
6
```

Output:

```
4
```

b) Modificati cerinta a) pentru a afisa perechile

Exemplu:

Input:

```
11
3
6
4
3
2
6
3
4
5
3
6
```

Output:

```
(3,3), (3,3), (4,4), (6,6)
```

```
# a)
n = int(input("n: "))
frecvente = dict()

for _ in range(n):
  nr = int(input("nr: "))
  frecvente[nr] = frecvente.get(nr, 0) + 1

total = 0
for nr, frecventa in frecvente.items():
  if frecventa >= 2:
    total += frecventa // 2

print(total)
n: 11
nr: 3
```

```
nr: 6
nr: 4
nr: 3
nr: 2
nr: 6
nr: 3
nr: 4
nr: 5
nr: 3
nr: 6
4
```

```python
#b)
n = int(input("n: "))
frecvente = dict()

for _ in range(n):
  nr = int(input("nr: "))
  frecvente[nr] = frecvente.get(nr, 0) + 1

for nr, frecventa in frecvente.items():
  nr_perechi_posibile = frecventa // 2
  for _ in range(nr_perechi_posibile):
    print(f"({nr}, {nr})", end=" ")
```

```
n: 11
nr: 3
nr: 6
nr: 4
nr: 3
nr: 2
nr: 6
nr: 3
nr: 4
nr: 5
nr: 3
nr: 6
(3, 3) (3, 3) (6, 6) (4, 4)
```

Problema #5 laboratorul #4

Institutia “Carpe diem” are n angajati (n citit de la tastatura). Institutia are program de lucru intre orele a, b (citite de la tastatura). Pentru fiecare angajat se citeste (de la tastatatura) ora la care are o activitate in institutie. Orice activitatea dureaza exact o ora. Seful doreste sa organizeze o sedinta (de o ora) in timpul programului de lucru, care sa nu se suprapuna cu nicio activitate din institutie (astfel incat sa poata participa toti angajatii). Care sunt orele disponibile (daca exista) dintre care seful poate alege sa organizeze sedinta?

Exemplu:

Input:

```
n = 7 (angajati)
a = 9
b = 17
Orele la care incep activitatile:
12
10
14
```

```
14
10
15
9
```

```
n = int(input("n: "))
a = int(input("a: "))
b = int(input("b: "))
ore = set()

for _ in range(n):
  ore.add(int(input()))

for ora in range(a, b):
  if ora not in ore:
    print(ora, end=" ")
```

```
n: 7
a: 9
b: 17
12
10
14
14
10
15
9
11 13 16
```

Problema #6 laboratorul #4

Se citește n, apoi două șiruri având câte n numere întregi (valorile se pot repeta). Să se afișeze:

a) valorile comune celor două șiruri;

b) valorile comune celor două șiruri (cu tot cu repetiții);

Exemplu: pentru `n = 8` și șirurile `s1: 1, 2, 2, 3, 4, 4, 4, 5` și `s2: 2, 2, 2, 3, 4, 4, 5, 5` se va afișa: 2, 2, 3, 4, 4, 5;

c) valorile care apar în plus în primul șir față de al doilea (se iau în considerare repetițiile)

Exemplu: pentru `n = 8` și șirurile `s1: 1, 2, 2, 2, 1, 4, 4, 4` și `s2: 2, 3, 4, 4, 5, 5, 7, 8` se va afișa: 1, 1, 2, 2, 4.

```
# a)
n = int(input("n: "))
s1 = []
s2 = []

for _ in range(n):
  s1.append(int(input("nr s1: ")))

for _ in range(n):
  s2.append(int(input("nr s2: ")))
```

```python
print(set(s1) & set(s2))        # intersectie de multimi -
 dar se printeaza cu acolade

print(*sorted(set(s1) & set(s2)))     # * e folosit pt. unpacking.
                                      # e usor de inteles ce face daca dai run
                                      # la cod fara * si apoi cu * si vezi ce se schi
mba
```

```
n: 8
nr s1: 1
nr s1: 2
nr s1: 2
nr s1: 3
nr s1: 4
nr s1: 4
nr s1: 4
nr s1: 5
nr s2: 2
nr s2: 2
nr s2: 2
nr s2: 3
nr s2: 4
nr s2: 4
nr s2: 5
nr s2: 5
{2, 3, 4, 5}
2 3 4 5
```

```python
# b)
n = int(input("n: "))
s1 = []
s2 = []

for _ in range(n):
  s1.append(int(input("nr: ")))

for _ in range(n):
  s2.append(int(input("nr: ")))

for elem in s1:                       # parcurgem elementele din s1
  if elem in s2:                      # daca elementul actual e si in s2
    print(elem, end=" ")              # il pritez direct de data asta (nu mai fac un s3
)
    i = s2.index(elem)                # aflu indicele primei aparitii a lui elem in s2
    del s2[i]                         # iar apoi sterg elementul egal cu elem din s2 po
zitia i
```

```
n: 8
nr: 1
nr: 2
nr: 2
nr: 3
nr: 4
nr: 4
nr: 4
nr: 5
nr: 2
nr: 2
nr: 2
```

```
nr: 3
nr: 4
nr: 4
nr: 5
nr: 5
2 2 3 4 4 5
```

```python
# c)
n = int(input("n: "))
s1 = []
s2 = []
s3 = []

for _ in range(n):
  s1.append(int(input("nr s1: ")))

for _ in range(n):
  s2.append(int(input("nr s2: ")))

for nr in s1:
  if nr in s2:
    s2.remove(nr)            # sterg nr din s2 daca e si in s1 si in s2
  else:
    s3.append(nr)            # daca nr e doar in s1 il pun in s3 (la final,
                             # acolo se va regasi rezultatul cautat)

print(*sorted(s3))
```

```
n: 8
nr s1: 1
nr s1: 2
nr s1: 2
nr s1: 2
nr s1: 1
nr s1: 4
nr s1: 4
nr s1: 4
nr s2: 2
nr s2: 3
nr s2: 4
nr s2: 4
nr s2: 5
nr s2: 5
nr s2: 7
nr s2: 8
1 1 2 2 4
```

Problema #7 laboratorul #4

Sa se citeasca un numar natural n, apoi n valori numere intregi ce se vor memora intr-o lista l. Scrieți un program care să extragă duplicatele din lista l într-o alta, numită `dubluri`, lasand lista initiala cu valori unice. Nu ne intereseaza ordinea elementelor din liste.

Exemplu:

pt. `l = [2, 3, 2, 2, 2, 4, 4]` vom avea

`dubluri = [2, 2, 2, 4]`

iar continutul listei l va fi

```
l = [2, 3, 4]
```

```
n = int(input("n: "))
l = []
numere_gasite = set()
unice = []
dubluri = []

for _ in range(n):
  l.append(int(input("nr: ")))

for nr in l:
  if nr not in numere_gasite:
    numere_gasite.add(nr)
    unice.append(nr)
  else:
    dubluri.append(nr)

l = unice

print(dubluri)
print(l)
```

```
n: 7
nr: 2
nr: 3
nr: 2
nr: 2
nr: 2
nr: 4
nr: 4
[2, 2, 2, 4]
[2, 3, 4]
```

Problema #8 laboratorul #4

Se citește un șir s de la tastatură. Să se afișeze literele mici ale alfabetului englez care nu apar în șirul s.

Exemplu:

Input:

Cateva cuvinte mostenite probabil din limba daca: abur, aidoma, amurg, aprig, balta, brusture, codru, copac, descurca, gutui, mazare etc.

Output: f h j k q w x y

```
# Metoda 1
s = input("s: ")
litere_mici = "qwertyuiopasdfghjklzxcvbnm"

litere_negasite = []

for litera in litere_mici:
```

```
  if litera not in s:
    litere_negasite.append(litera)

print(" ".join(sorted(litere_negasite)))
```

```
s: Cateva cuvinte mostenite probabil din limba daca: abur, aidoma, amurg, aprig, balta
f h j k q w x y
```

Metoda 2 (ord, chr)

Introducem doua functii noi (veti aprofunda la string-uri):

- `ord(c)` : returneaza numarul intreg care reprezinta caracterul c (vezi tabel ASCII pe net)
- `chr(c)` : face operatia inversa, adica ii dam un numar intreg si ne da caracterul corespunzator

Exemplu:

```
print(ord('a'))
print(ord('b'))
print(ord('c'))
print("...")
print(ord('z'))

print("-" * 40)

print(chr(97))
print(chr(98))

print("-" * 40)

print(ord('X'))
print(chr(ord('X')))
```

```
97
98
99
...
122
----------------------------------------
a
b
----------------------------------------
88
X
```

```
# Metoda 2 (in cerinta din laborator scrie ca puteti folosi ord si chr aici)
s = input("s: ")
gasit_sau_nu = [0] * 26

for litera in s:
  if litera.islower():
    gasit_sau_nu[ord(litera) - ord('a')] = 1

for i in range(len(gasit_sau_nu)):
  if gasit_sau_nu[i] == 0:
    print(chr(ord('a') + i), end=" ")
```

```
s: Cateva cuvinte mostenite probabil din limba daca: abur, aidoma, amurg, aprig, balta
f h j k q w x y
```

Problema #9 laboratorul #4

Scrieți un program care:

a) realizeaza un tabel de frecvente pentru literele mici dintr-un text introdus de la tastatura.

Exemplu:

Input:

`ana are mere`

Output:

```
e : 3
n : 1
a : 3
m : 1
r : 2
```

b) rezolvă cerința a) cu tabelul ordonat descrescător după numărul de apariții (nu este necesar parametrul `key` al metodei `sort` – vom detalia in cursurile urmatoare, la subiectul Functii).

Exemplu:

Input: `ana are mere`

Output:

```
e : 3
a : 3
r : 2
n : 1
m : 1
```

```
# a) - rezolvare propusa chiar de doamna profesoara
text = input("Dati textul \n")
l = list(text)

litere_mici = {chr(i) for i in range(ord('a'), ord('z') + 1)}
litere_text = set(text) & litere_mici     # pentru a ignora alte caractere diferite d
e litere

"""
Vom retine frecventele in aceasta variabila care este un dictionar
 (cheile sunt literele mici ale textului si valo-
rile sunt numarul de aparitii al fiecarei litere)

"""
frecvente = {}

for el in litere_text:
    frecvente[el] = l.count(el)

for el in frecvente:
    print(el, ":", frecvente[el])
Dati textul
ana are mere
e : 3
```

```
m : 1
r : 2
n : 1
a : 3
```

```python
# b) - rezolvare propusa chiar de doamna profesoara
text = input("Dati textul \n")
l = list(text)

litere_mici = {chr(i) for i in range(ord('a'), ord('z') + 1)}
litere_text = set(text) & litere_mici

frecvente = {}
lista = []

for el in litere_text:
    lista.append((l.count(el), el))

"""
Cum lista este o lista de tupluri, sort va sorta intai dupa primul element din fiecar
e tuplu,
in ordine descrescatoare (reverse = True) si, ca "tie-
breaker", daca doua elemente au primul
element egal, sorteaza tot descrescator dupa al doilea element (care e un caracter, d
eci va sorta
invers alfabetic).
"""

lista.sort(reverse = True)

for el in lista:
    print(el[1], ":", el[0])
```

```
Dati textul
ana
a : 2
n : 1
```