Laboratorul 3 - Explicatii

Problema #1 laboratorul #3

Stiind ca `l = [1, 2, 3, 4, 5, 6, 7, 8, 9]`, care este outputul urmatoarelor comenzi:

a) `print(l[7:5])`

b) `print(l[0:1000])`

c) `print(l[-4:-1])`

d) `print(l[-4:0])`

e) `print(l[-4:0:-1])`

f) `print(l[-4:])`

g) `print(l[-1:-4])`

h) `print(l[::])`

i) `print(l[::100])`

j) `print(l[::-100])`

Despre indexarea listelor

Pentru lista `l = [1, 2, 3, 4, 5, 6, 7, 8, 9]`, iata cum vor fi indexate elementele (pe randul din mijloc sunt elementele lui l, sus sunt idecsii obisnuiti de la 0 la 8 si sub sunt cei inversi):

$$\begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 \end{array}$$

Poate parea ca Python admite doi indecsi in acelasi timp pentru acelasi element, dar, behind the scenes, indecsii negativi sunt convertiti la indecsi pozitivi.

Observatie: In explicatia de mai jos despre feliere (slicing), folosim doar indecsi pozitivi.

Despre feliere (Slicing)

Listele dispun de ceea ce se cheama *slicing*.

Cum o pizza poate fi feliata si putem extrage felii mai mari sau mai mici din ea, la fel putem face si cu listele din Python.

Tineti minte sintaxa pentru slicing (e similar cu ce scriem la `range()`):

- `lista_mea[start:stop:step]`
- Daca start sau stop sunt omisi, Python seteaza niste valori implicite pentru ele, in functie de semnul lui step.
- `stop` este **exclusiv**, nu inclusiv (vor fi luate toate obiectele de la indexul i, unde i pleaca de la start si se aduna step cat timp `i < stop`, in cazul in care `step > 0` sau cat timp `i > stop`, cand `step < 0`)
- cand `start` nu e specificat, el e:
  - `0`, daca `step > 0`
  - `len(lista) - 1`, daca `step < 0`
- cand `stop` nu e specificat, el e:
  - `len(lista)`, daca `step > 0` (atentie: e exclusiv, deci ultimul element luat va fi la indexul len(lista)-1)
  - `-1`, daca `step < 0` (exclusiv, deci ultimul element luat va fi la indexul 0)
- `step` este implicit 1 (cand nu e specificata alta valoare)

Exemple de slicing:

```
l_6 = ["rinocer", "hipopotam", 32, None, "ana are mere", 8, "Gigel"]
print(l_6[0:3]) # primele 3 elemente
print(l_6[:3])  # tot primele 3 elemente
                # daca omitem primul numar, el va fi 0 implicit

print(l_6[4:5])
print(l_6[-3:])     # elementele de la -3 pana la -1, adica ultimele 3 elemente

# Asa cum am zis, un element dintr-o lista are simultan 2 indecsi:
# cel negativ si cel pozitiv
# Slice-ul urmator incepe de la pozitia 0 si se termina la pozitia -1 exclusiv
# Dar pozitia -1 stim ca e ultimul element din lista, deci va lua toate elementele
# mai putin ultimul
print(f"l_6[0:-1]:", l_6[0:-1])
```

```
['rinocer', 'hipopotam', 32]
['rinocer', 'hipopotam', 32]
['ana are mere']
['ana are mere', 8, 'Gigel']
l_6[0:-1]: ['rinocer', 'hipopotam', 32, None, 'ana are mere', 8]
```

Rezolvarea problemei #1 laboratorul #3

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]

print("l[7:5]", l[7:5])          # incepe de la elementul de pe poz. 7 si se termina la
 cel de pe poz 5-1 = 4, dar pasul este +1 (default)
                                 # deci va fi un slice gol

print("l[0:1000]", l[0:1000])    # incepe de la elem. de pe poz. 0 si se termina la ele
m. de pe poz. 1000, care este mult peste
                                 # ultima pozitie din lista, ceea ce inseamna ca o sa f
ie luate toate elementele din lista (si atat!)

print("l[-4:-1]", l[-4:-1])      # incepe de la poz. -4 si se termina la poz -1 exclusi
v deci la poz. dinaintea pozitiei -1,
```

```
                                # adica la pozitia -2

print("l[-4:0]", l[-4:0])       # inseamna "ia al 4-lea element din coada si opreste-
te inainte de primul element, step = +1"
                                # cum stop < start, clar lista rezultata e vida

print("l[-4:0:-1]", l[-4:0:-1]) # inseamna "ia tot de la dreapta la stanga (deoarece s
tep = -1) incepand de la
                                # poz. -4 pana chiar inainte de pozitia 0 (=> pana la
poz. 1)

print("l[-4:]", l[-4:])         # de la -4 la ultimul index (len(l)-1) => 6, 7, 8, 9

print("l[-1:-4]", l[-1:-4])     # de la poz. -1 ia-
o spre dreapta (deoarece step = +1) pana la pozitia -4
                                # cum stop = -4 < start = -1 => lista goala []

print("l[::]", l[::])           # de la 0 la ultimul index (= len(l)-1) => toata lista
print("l[::100]", l[::100])     # de la 0 la ultimul index cu step = 100 => doar primul
 element: 1
print("l[::-100]", l[::-100])   # step < 0 => o luam de la dreapta la stanga, deci prim
a poz. luata va fi -1 (prima din dreapta);
                                # step = -100 => o luam de la dreapta la stanga cu cate
 100 de pozitii
                                # "incepe de la -1, ia-
o spre stanga pana la ultima poz. din lista"
```

```
l[7:5] []
l[0:1000] [1, 2, 3, 4, 5, 6, 7, 8, 9]
l[-4:-1] [6, 7, 8]
l[-4:0] []
l[-4:0:-1] [6, 5, 4, 3, 2]
l[-4:] [6, 7, 8, 9]
l[-1:-4] []
l[::] [1, 2, 3, 4, 5, 6, 7, 8, 9]
l[::100] [1]
l[::-100] [9]
```

Problema #2 laboratorul #3

Stiind ca `l = [1, 2, 3]`, care este outputul urmatorului fragment de cod?:

```
l[len(l):] = [4, 5, 6]
print(l)

l[:0] = [-3, -2, -1, 0]
print(l)
```

```
l = [1, 2, 3]

l[len(l):] = [4, 5, 6]
# len(l) e 3
# l[3:] inseamna "selecteaza toate elementele de la indexul 3 pana la sfarsit"
# Cum lista are doar indecsii 0, 1 si 2, slice-ul nostru va fi gol -> []
```

```
# El se va situa la finalul listei
# Acolo (la finalul listei) inseram elementele din lista [4, 5, 6]
# Deci vom avea l == [1, 2, 3, 4, 5, 6]
print(l)

l[:0] = [-3, -2, -1, 0]
# l[:0] inseamna "selecteaza toate elementele de la indexul 0 pana la indexul
# 0 exclusiv"
# Cum nu exista elemente in acest range, slice-ul selectat va fi gol -> []
# Si se va situa la inceputul listei
# Acolo, inseram elementele din lista [-3, -2, -1, 0]
print(l)
```

```
[1, 2, 3, 4, 5, 6]
[-3, -2, -1, 0, 1, 2, 3, 4, 5, 6]
```

```
# Bonus!
# Iata cum inseram elementele din lista [5, 6, 7, 8] in lista l, inlocuindu-le
# pe cele de la indecsii 0:3 -> adica indecsii 0, 1, 2
l = [1, 2, 3, 4]
l[0:3] = [5, 6, 7, 8]
print(l)
```

```
[5, 6, 7, 8, 4]
```

Problema #3 laboratorul #3

Fie `l` o lista oarecare. Completati codul `l[_:_:-1]` in locurile marcate cu "_" cu expresii (nevide) potrivite astfel incat rezultatul rularii acestuia sa fie identic cu cel al rularii codului `l[::-1]`.

Rezolvarea problemei #3 laboratorul #3

Vrem aceeasi functionalitate ca `l[::-1]`.

Am zis mai sus ca daca `step < 0`, implicit vom avea:

- elementele vor fi luate invers (de la dreapta la stanga)
- `start = len(lista) - 1 (= -1 pe indexare negativa)` (prima poz. din dreapta)
- `stop = -1` exclusiv deci `0` inclusiv

```
# Solutia problemei #3 laboratorul #3
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]

print(f"l[::-1] == {l[::-1]}")
print(f"l[len(l)-1::-1]: {l[len(l)-1::-1]}")
```

```
l[::-1] == [9, 8, 7, 6, 5, 4, 3, 2, 1]
l[len(l)-1::-1]: [9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Problema #4 laboratorul #3

Care este outputul urmatoarelor comenzi?:

a) `print([1, 2, 3] * 1)`

b) `print([1, 2, 3] * 0)`

c) `print([1, 2, 3] * -1)`

Rezolvarea problemei #4 laboratorul #3

```
# Solutia problemei #4 laboratorul #3
print([1, 2, 3] * 1)
print([1, 2, 3] * 0)    # face fix ce ne asteptam sa faca
print([1, 2, 3] * -1)   # Python trateaza inmultirea listelor cu numere negative la fel
 ca
                        # inmultirea lor cu 0 => lista goala []
```

```
[1, 2, 3]
[]
[]
```

Problema #5 laboratorul #3

Care dintre codurile urmatoare au ca efect inserarea jumatatii numarului dupa orice numar par dintr-o lista `l`, fara a folosi liste auxiliare? Justificati, scrieti ce se va afisa pentru urmatoarele liste `l = [1, 14, 18, 23]`, `l = [1, 14, 18, 23, 6, 24]`, `l = [1, 24, 18, 23, 12]` si apoi rulati pentru a verifica raspunsurile.

a)

```
for i in range(len(l)):
  if l[i] % 2 == 0:
    l.insert(i + 1, l[i] // 2)
    i = i + 1
  i = i + 1
print(l)
```

b)

```
i = 0
while i < len(l):
  if l[i] % 2 == 0:
    l.insert(i + 1, l[i] // 2)
  i = i + 1
print(l)
```

c)

```
i = 0
while i < len(l):
  if l[i] % 2 == 0:
    l.insert(i + 1, l[i] // 2)
    i = i + 1
  i = i + 1
print(l)
```

d)

```
for el in list(l):
  if el % 2 == 0:
    i = l.index(el)
    l.insert(i + 1, l[i] // 2)
print(l)
```

Rezolvarea problemei #5 laboratorul #3

```
# Solutia problemei #5 laboratorul #3

# a)
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in range(len(l)):             # problema: range(len(l)) este calculat o singura data, la inceput
  if l[i] % 2 == 0:                 # de aceea, dupa ce facem inserari, i-ul va fi tot blocat in range-ul
    l.insert(i + 1, l[i] // 2)      # de la 0 la 8 si nu vom obtine ceea ce dorim
    i = i + 1                       # degeaba adunam 1 la i. el e cam prostovan si stie doar sa se miste
  i = i + 1                         # in range-ul specificat la inceput. nu ne lasa sa il mutam noi cum vrem
print(l)

# b)
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
i = 0
while i < len(l):                   # aici while recalculeaza len(l) la fiecare pas, cea ce e un upgrade fata de a)
  if l[i] % 2 == 0:
    l.insert(i + 1, l[i] // 2)      # dupa inserare, i nu este incrementat decat o data, deci la urmatoarea iterare
  i = i + 1                         # noul element inserat va fi tratat ca un element al listei si, daca e par, vom
print(l)                            # insera si jumatatea lui si tot asa (ex: daca ajungem pe elementul cu valoarea 8,
                                    # inseram 4, apoi 2, apoi 1) - gresit!

# c) (CORECT)
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
i = 0
while i < len(l):                   # luam fiecare index din lista
  if l[i] % 2 == 0:                 # daca elementul aflat la indexul i e par
    l.insert(i + 1, l[i] // 2)      # inseram jumatatea lui pe urmatoarea pozitie
    i = i + 1                       # am adaugat un nou element listei, deci incrementam i
  i = i + 1                         # incrementam i ca parte a while loop-ului pentru trece la urmatorul element din lista
print(l)

# d)
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for el in list(l):              # list(l) creeaza o copie a listei l, deci e gresit din start (vezi cerinta)
```

```
  if el % 2 == 0:
    i = l.index(el)
    l.insert(i + 1, l[i] // 2)
print(l)
```

```
[1, 2, 1, 3, 4, 2, 1, 5, 6, 3, 7, 8, 9]
[1, 2, 1, 3, 4, 2, 1, 5, 6, 3, 7, 8, 4, 2, 1, 9]
[1, 2, 1, 3, 4, 2, 5, 6, 3, 7, 8, 4, 9]
[1, 2, 1, 3, 4, 2, 5, 6, 3, 7, 8, 4, 9]
```

Problema #6 laboratorul #3

Fie lista `l_1 = [1, 2, 240, 120, 2, 3, 18, 12, 22, 28, 1004]`. Sa se stearga:

a. toate valorile pare

b. toate valorile prime

Rezolvarea problemei #6 laboratorul #3

Observatie: Nu ni se cere sa lucram pe caz general, cu orice lista, ci cu lista `l_1` data. Cu toate astea, am inclus un program care sterge valorile pare pentru orice lista de numere intregi.

Vom folosi **metoda .pop(index)** pentru liste, care sterge elementul de la indexul specificat (si il returneaza -- mai vorbim la functii despre asta).

```
# Solutia problemei #6 laboratorul #3
# a.
l_1 = [1, 2, 240, 120, 2, 3, 18, 12, 22, 28, 1004]
l_1.pop(-1)       # stergem 1004 (par)
l_1.pop(-1)       # stergem 28 (par)
l_1.pop(-1)       # stergem 22 (par)
l_1.pop(-1)       # stergem 12 (par)
l_1.pop(-1)       # stergem 18 (par)
l_1.pop(-2)       # stergem 2 (par)
l_1.pop(-2)       # stergem 120 (par)
l_1.pop(-2)       # stergem 240 (par)
l_1.pop(-2)       # stergem 2 (par)
print(l_1)

# b.
l_1 = [1, 2, 240, 120, 2, 3, 18, 12, 22, 28, 1004]
l_1.pop(5)       # stergem 3 (prim)
print(l_1)
```

```
[1, 3]
[1, 2, 240, 120, 2, 18, 12, 22, 28, 1004]
```

```
# Solutia pentru o lista oarecare
# Solutia 1
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
i = 0
while i < len(l):       # len(l) se calculeaza la fiecare iterare
  if l[i] % 2 == 0:
    l.pop(i)            # aici nu incrementam i, pentru ca dupa stergere,
```

```
  else:                 # elementele se muta la stanga
    i += 1              # doar daca valoarea nu e para incrementam i
print(l)

# Solutia 2
# aici range-ul e calculat doar o data, la prima iterare (reamintim)
for i in range(len(l) -
 1, -1, -1):      # de la ultimul element la primul (-1 exclusiv => 0 inclusiv)
  if l[i] % 2 == 0:
    del l[i]                # del reprezinta alta metoda de a sterge lucruri
print(l)

# Incercati sa va ganditi de ce nu ar fi mers sa parcurgem lista de la stanga la dreapt
a
# in for-ul de mai sus si sa nu schimbam nimic
#
#
#
#
#

#(hint: s-ar fi dereglat indecsii dupa stergere)
[1, 3, 5, 7, 9]
[1, 3, 5, 7, 9]
```

Problema #7 laboratorul #3

Se citeste n, apoi n numere. Sa se rearanjeze numerele si sa se memoreze intr-o lista astfel incat toate valorile nule sa fie la finalul acesteia.

Rezolvarea problemei #7 laboratorul #3

```
# Solutia problemei #7 laboratorul #3
n = int(input("n: "))
numere = []
for _ in range(n):
  nr = int(input("introdu numar: "))
  numere.append(nr)

print(f"lista initiala: {numere}")

# Facem o lista care include doar valorile nenule din lista initiala,
# folosind comprehensiune (list comprehension). Face exact ce "zice" ca face:
# [x for x in numere if x != 0]
# expresia de mai sus e intre paranteze patrate, deci va crea o lista noua, cu
# fiecare x din numere, dar numai daca x != 0
nenule = [x for x in numere if x != 0]

# nenule = [x for x in numere if x != 0] e echivalent cu:
# nenule = []
# for x in numere:
#   if x != 0:
```

```
#     nenule.append(x)

nr_zerouri = n - len(nenule)

rezultat = nenule + [0] * nr_zerouri       # operatorul +, cand e aplicat listelor, le concateneaza
                                           # [0] * nr_zerouri ne da o lista cu nr_zerouri zerouri :)
                                           # asadar, expresia asta pune atatea zerouri cate specifica nr_zerouri la finalul listei nenule

print(f"lista noua: {rezultat}")
```

```
n: 6
introdu numar: 0
introdu numar: 1
introdu numar: 9
introdu numar: 0
introdu numar: 0
introdu numar: 5
lista initiala: [0, 1, 9, 0, 0, 5]
lista noua: [1, 9, 5, 0, 0, 0]
```

Problema #8 laboratorul #3

Care este efectul executiei secventei de cod?:

```
import math
L = []               # initializam o lista goala L
x = int(input())     # luam un numar intreg ca input si il retinem in x
for n in range(2, x + 1):     # luam toate numerele naturale de la 2 la x (inclusiv)
  for factor in L:            # pt. fiecare numar (factor) din L
    if n % factor == 0 and factor <= math.sqrt(x):    # daca n se imparte la factor si factor <= radical din x
      break                   # oprim bucla for
  else:                   # else-ul e parte din "for else" si se executa daca bucla for a ajuns la final (s-au parcurs toate elementele din lista L)
    L.append(n)           # adaugam pe n la finalul listei (concatenam)
print(L)
```

Observatii inainte de rezolvare:

- denumirea variabilei `factor` e sugestiva
- conditia `factor <= math.sqrt(x)` miroase a divizori (deoarece divizorii unui numar se cauta eficient pana la radical din acel numar)
- daca conditia `if` are loc, bucla `for` se opreste
- nu am dat raspunsul la cerinta problemei. Lasam asta pentru cititor (daca nu reusiti sa va dati seama, incercati pentru mai multe input-uri si observati comportamentul programului; pentru intrebari, contactati-ne pe discord sau whatsapp)

```
import math
L = []
x = int(input())
```

```
for n in range(2, x + 1):
  for factor in L:
    if n % factor == 0 and factor <= math.sqrt(x):
      break
  else:
    L.append(n)
print(L)
20
[2, 3, 5, 7, 11, 13, 17, 19]
```

Problema #9 laboratorul #3

Fie listele `l_1, l_2` si `n, m, k <= n * m` numere naturale citite. Se citesc `n, m` valori intregi ce se memoreaza in lista `l_1`, respectiv `l_2`. Sa se afiseze primele `k` perechi (l_1, l_2), $l_1 \in$ `l_1`, $l_2 \in$ `l_2` ce au suma minima.

```
n = int(input("n: "))
m = int(input("m: "))
k = int(input("k: "))
l1 = [int(x) for x in input().split()]
l2 = [int(x) for x in input().split()]

perechi = []
for x in l1:
  for y in l2:
    perechi.append((x + y, x, y))

# sortam dupa suma (x+y)
# daca suma e egala pt. doua perechi (x+y = a+b), sorteaza automat dupa
# primul element (adica compara x, a) si daca si ele sunt egale, sorteaza automat dupa
# al doilea element (adica compara y, b)
perechi.sort()

for i in range(k):
  _, a, b = perechi[i]
  print(f"({a}, {b})")
n: 5
m: 6
k: 4
5 2 19 0 -4
59 -200 -900 -91 12 22
(-4, -900)
(0, -900)
(2, -900)
(5, -900)
```

Problema #10 laboratorul #3

Fie o lista ce contine liste si tupluri incuibate (ex: `l = [1, [2, 3], [4, 5, (6, (7, 8, 9))], 10]`. Sa se transforme intr-o lista simpla fara a folosi liste suplimentare:

a) ignorand ordinea elementelor

b) pastrand ordinea elementelor

```
# a)
l = [1, [2, 3], [4, 5, (6, (7, 8, 9))], 10]

while True:          # ruleaza pana cand lista devine complet "plata"
  for element in l:       # cauta in lista de la stanga la dreapta, pana la primul elem
ent de tip tuple sau list
    if type(element) in [list, tuple]:     # daca elementul 'element' este de tip tuple
 sau list,
      l.remove(element)                    # stergem element din lista
      l.extend(element)                    # concatenam elementele din 'element' in lis
ta l (adaugam la finalul listei, "despachetand" element)
      break                                # oprim for-
ul si sarim la urmatoarea iteratie a buclei din exterior (adica a while-ului)
  else:                  # else e parte din instructiunea for..else si se executa cand
for-ul itereaza complet prin toata lista fara sa ajunga la vreun break
    break                # daca am mers prin toate elementele listei l si nu am gasit n
imic de tip list sau tuple, ne oprim, deoarece lista e "plata". gata!
print(l)

# b)
l = [1, [2, 3], [4, 5, (6, (7, 8, 9))], 10]

while True:                # ruleaza pana cand lista devine complet "plata"
  for i in range(len(l)):     # cauta in lista de la stanga la dreapta
    if type(l[i]) in [list, tuple]:      # daca elementul 'element' este de tip tuple s
au list,
      l[i:i+1] = l[i]                    # inlocuim slice-
ul format din unicul element de pe pozitia i cu ceea ce e continut in l[i]
      break                              # break aici inseamna ca oprim bucla for si sa
rim la urmatoarea iteratie a buclei din afara, adica a while-ului
  else:                                  # else e parte din instructiunea for..else si
se executa cand for-ul itereaza complet prin tot range-ul fara sa ajunga la vreun break
      break                           # daca am mers prin toate elementele listei l si n
u am gasit nimic de tip list sau tuple, ne oprim, deoarece lista e "plata". gata!
print(l)
```

```
[1, 10, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```