test-e.ro

CAIET ELEV

Lecția 17

Clasa list din Python. Operatori și metode

*Până acum lista a fost un desen pe caiet. De astăzi este un **obiect real**, care se poate modifica pe loc: învățăm operatorii ei (**in**, +, *, comparațiile) și metodele care o schimbă — și deosebim, în sfârșit, o **copie** de un simplu **al doilea nume**.*

DISCIPLINA Informatică **CLASA** a IX-a · mate-info **LIMBAJ** Python 3

NUMELE ELEVULUI .....

CUM FOLOSEȘTI CAIETUL

Rezolvă exercițiile pe hârtie: urmărești execuția pas cu pas, prezici ce afișează un program și scrii singur soluțiile. Noutățile pe care le poți folosi de acum: `for x in lista`, metodele clasei `list` (`append`, `insert`, `remove`, `pop`, `clear`, `extend`, `reverse`, `sort`, `count`, `index`, `copy`), funcția `sorted()` și slicing-ul `lista[i:j]`. **Fără** `def`, fără `import`, fără `min` / `max` / `sum`. Cifrele unui număr se parcurg tot cu `while` și `%`, `//`. Regula de aur: **rezultatul îl calculezi tu, înainte de a rula.**

CUPRINS

1 Ne reamintim lista ca model conceptual .....
2 De la model la obiectul real .....
3 Operatorii clasei list .....
4 Metodele care șterg .....
5 Sortarea: pe loc sau într-o listă nouă .....
6 Alias, copie și slicing .....
7 Cod ghidat · cinci probleme .....

SECȚIUNEA 1 · REACTUALIZARE L15–L16

1 Ne reamintim lista ca *model conceptual*

La lecțiile 15 și 16 am privit lista ca **model conceptual**: o colecție **liniară** (elementele stau unul după altul), **ordonată** (ordinea contează și o stabilim noi) și cu **poziții** numerotate. Poziția se numește **indice** și pornește de la `0`. Ordonată nu înseamnă „sortată crescător": înseamnă doar că fiecare element are un loc al lui, care nu se schimbă de la sine.

Cele trei unelte de bază. `lista[i]` dă elementul de pe poziția `i`; `len(lista)` dă numărul de elemente; iar parcurgerea cu indice se scrie astfel:

```
lista = [4, 7, 10, 3]
i = 0
while i < len(lista):
    print(lista[i])
    i = i + 1
```

ATENȚIE LA ULTIMUL INDICE

Într-o listă cu `len(lista)` elemente, indicele ultimului element este `len(lista) - 1`, nu `len(lista)`. Bucla se oprește exact când `i` ajunge la `len(lista)` — dacă am cere `lista[len(lista)]`, am ieși din listă și programul s-ar opri cu eroare.

A

EXERCIȚIUL A · GRILĂ

Ce ne amintim despre liste

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Care este indicele **primului** element dintr-o listă? `0` `1` `len(lista)` `depinde de listă`
2. Pentru `lista = [4, 5, 6]`, ce afișează `print(lista[len(lista) - 1])`?

   `4` `5` `6` `eroare de indice`
3. Ce valoare are `len([5, 8, 2])`? `2` `3` `8` `15`
4. Ce înseamnă că o listă este o colecție **ordonată**?

   `elementele își păstrează ordinea în care le-am pus` `elementele sunt așezate crescător`

   `toate elementele sunt numere` `elementele nu se mai pot schimba`

HINT *„ordonată" se referă la existența unei ordini stabile, nu la sortare; iar* `len()` *numără elemente, nu le adună.*

B

EXERCIȚIUL B · DRY-RUN

Câte valori pare are lista?

DRY-RUN

Urmărește bucla pas cu pas. Coloanele `i` și `lista[i]` îți sunt date; tu completezi doar coloana `c` — valoarea contorului **după** ce s-a executat pasul de pe rândul respectiv. Primul rând e completat ca exemplu.

```
1 lista = [4, 7, 10, 3]
2 i = 0
3 c = 0
4 while i < len(lista):
5     if lista[i] % 2 == 0:
6         c = c + 1
7     i = i + 1
8 print(c)
```

i (dat)	lista[i] (dat)	c după pas
0	4	1
1	7	
2	10	
3	3	
4	–	

→ Ce afișează programul? _____

HINT *contorul crește numai când elementul este par; la ultimul rând condiția buclei este deja falsă, deci* `c` *nu se mai schimbă.*

SPRIJIN · SCRIE CODUL

Câte elemente am citit?

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând. Construiește lista acestor numere și afișează câte elemente are ea, folosind funcția corespunzătoare. Ex.: pentru `n = 3` și 5, 8, 2 se afișează `3`.

HINT *rezultatul este întotdeauna egal cu `n` — scopul exercițiului este să exersezi construirea listei element cu element, nu calculul în sine.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

Primul și ultimul element

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând. Afișează pe același rând primul element al listei, un spațiu, apoi ultimul element. Ex.: pentru `n = 3` și 5, 8, 2 se afișează `5 2`.

HINT *ultimul element are indicele `len(lista) - 1`, nu `len(lista)`. Când lista are un singur element, primul și ultimul sunt același element, deci se afișează aceeași valoare de două ori.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · MUTABILITATE, APPEND, FOR X IN LISTA

2 De la model la *obiectul real*

Până acum, orice valoare cu care am lucrat era **imutabilă**: dacă `n` era 5 și scriam `n = n + 1`, nu schimbam cifra 5, ci puneam în `n` o valoare nouă. Lista se comportă altfel — ea este **mutabilă**, adică se poate modifica **pe loc**, păstrându-și identitatea.

APPEND(V)

Adaugă valoarea `v` la **capătul din dreapta** al listei. Nu produce o listă nouă; schimbă lista pe care o apelezi. Se poate porni și de la lista **vidă**, scrisă `[]` — perfect valabilă, cu zero elemente.

```
lista = []
lista.append(5)
lista.append(8)
lista.append(2)
print(lista)
print(len(lista))
```

După cele trei apeluri, `lista` este `[5, 8, 2]`, iar `len(lista)` este 3. A doua noutate a lecției este o formă nouă a lui `for`, care ne dă direct **valorile**, fără indice:

```
lista = [5, 8, 2]
s = 0
for x in lista:
    s = s + x
print(s)
```

Aici `x` nu este o poziție, ci ia pe rând valorile 5, 8, 2. Rezultatul afișat este 15. **Alege forma după ceea ce îți trebuie:** `for x in lista` când ai nevoie doar de valoare, `for i in range(len(lista))` când ai nevoie și de poziție. Amândouă parcurg toată lista, exact o dată fiecare element.

> **ATENȚIE**
>
> `append()` adaugă **un singur** element. Dacă îi dai ca argument o listă, o pune întreagă ca element unic: pe `[1, 2]`, apelul `append([3, 4])` dă `[1, 2, [3, 4]]` — o listă cu **3** elemente, nu cu 4.

A

EXERCIȚIUL A · COMPLETEAZĂ

Definiția clasei list

COMPLETARE

Completează golurile folosind cuvintele din rezervă (două cuvinte rămân nefolosite).

`ordonată` `poziție` `mutabilă` `append` `imutabilă` `sorted`

O listă din Python este o colecție _____ , în care fiecare element are o _____ , și este _____ , adică se poate modifica pe loc. Metoda care adaugă o valoare la capătul din dreapta al listei se numește _____ .

HINT *cele două cuvinte nefolosite descriu exact opusul listei și, respectiv, o funcție care nu modifică nimic.*

B

EXERCIȚIUL B · SCRIE CODUL

Câte elemente și cu ce sumă?

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând. Construiește lista folosind metoda `append()`, apoi afișează pe același rând numărul de elemente, un spațiu, și suma lor. Parcurge lista cu forma `for x in lista`. Ex.: pentru `n = 3` și 5, 8, 2 se afișează `3 15`.

HINT *suma se inițializează cu 0 înainte de buclă. Dacă toate elementele sunt zero, suma rămâne 0 — și este un rezultat corect, nu o eroare.*

SOLUȚIA TA · PYTHON

PERFORMANȚĂ · SCRIE CODUL

Aceeași valoare la ambele capete

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând, și în final o valoare `v`. Adaugă `v` atât la începutul, cât și la sfârșitul listei, apoi afișează toate elementele listei rezultate pe un singur rând, separate prin câte un spațiu. Ex.: pentru `n = 3`, numerele 5, 8, 2 și `v = 9` se afișează `9 5 8 2 9`.

HINT *`append()` adaugă numai la dreapta; pentru capătul din stânga ai nevoie de `insert(0, v)`, care împinge celelalte elemente cu o poziție la dreapta. Lista finală are `n + 2` elemente.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 3 · OPERATORII CLASEI LIST

3 Apartenență, concatenare, *multiplicare*

Apartenența se testează cu `in` și `not in`, iar rezultatul este `True` sau `False`: `3 in [1, 2, 3]` este `True`, iar `5 not in [1, 2, 3]` este tot `True`. Folosirea tipică este direct în condiție:

```
lista = [10, 20, 30]
v = 20
if v in lista:
    print("gasit")
else:
    print("absent")
```

Concatenarea cu `+` lipește două liste și produce o listă nouă: `[10, 20, 30] + [40]` dă `[10, 20, 30, 40]`. **Multiplicarea** cu `*` repetă conținutul: `[1, 2] * 3` dă `[1, 2, 1, 2, 1, 2]`. Amândouă lasă listele originale neatinse. **Relaționarea** (`==`, `<`, `>`) compară element cu element, începând din stânga: `[1, 2] < [1, 3]` este `True`, pentru că primele elemente sunt egale, dar 2 < 3.

DOUĂ METODE CARE DOAR ÎNTREABĂ LISTA

count(v) numără de câte ori apare valoarea `v`: pe `[7, 3, 7, 1]`, `count(7)` dă 2, iar pentru o valoare absentă dă 0 — nu eroare. **index(v)** dă poziția **primei** apariții: pe aceeași listă, `index(7)` dă 0, nu 2. Dacă valoarea lipsește, programul se oprește cu eroare — de aceea `index()` se apelează după ce ai verificat cu `in` că valoarea există.

ATENȚIE · CAZUL K = 0

Când repeți o listă de `k` ori, rezultatul are `len(lista) * k` elemente. Dacă însă `k` este 0, atunci `lista * 0` produce lista vidă `[]`, cu zero elemente — corect matematic, dar ușor de trecut cu vederea când `k` se citește de la tastatură.

A

EXERCIȚIUL A · GRILĂ

Ce fac operatorii pe liste

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce afișează `print([1, 2] * 3)`? `[1, 2, 1, 2, 1, 2]` `[3, 6]` `[1, 2, 3]` `eroare`
2. Pentru `lista = [1, 3, 4, 7]`, ce valoare are `lista.index(7)`? `1` `2` `3` `7`
3. Pentru `lista = [8, 3, 8, 1, 8]`, ce valoare are `lista.count(8)`? `0` `1` `3` `4`
4. Ce afișează `print([10, 20, 30] + [40])`?

`[10, 20, 30, 40]` `[50, 60, 70]` `100` `[10, 20, 70]`

HINT `*` *repetă lista întreagă,* `+` *o lipește la capăt;* `index()` *întoarce o* **poziție** *(numărată de la 0), iar* `count()` *un* **număr de apariții** *.*

B

EXERCIȚIUL B · SCRIE CODUL

Este valoarea în listă?

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând, iar în final o valoare `v`. Dacă `v` se află în listă, afișează pe același rând: `v`, un spațiu, textul `DA`, un spațiu, de câte ori apare `v`, un spațiu, și poziția primei apariții. Dacă `v` nu se află în listă, afișează `v`, un spațiu, textul `NU`, un spațiu, 0, un spațiu și -1. Ex.: pentru `n = 5`, numerele 3, 7, 3, 9, 3 și `v = 3` se afișează `3 DA 3 0`.

HINT `index(v)` *dă eroare dacă valoarea lipsește. Testează întâi apartenența cu* `in` *și doar pe ramura în care valoarea există apelează* `count()` *și* `index()`*.*

SOLUȚIA TA · PYTHON

C

EXERCIȚIUL C · SCRIE CODUL

Repetă lista de k ori

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând, iar în final un număr natural `k` (`k >= 0`). Construiește lista obținută prin repetarea de `k` ori a listei citite. Afișează pe primul rând numărul de elemente al listei rezultate, iar pe rândul următor elementele ei, separate prin câte un spațiu. Ex.: pentru `n = 2`, numerele 1, 2 și `k = 3` se afișează 6, apoi `1 2 1 2 1 2`.

HINT *pentru `k = 0` lista rezultată este vidă: se afișează 0 pe primul rând, iar al doilea rând rămâne gol — bucla de afișare pur și simplu nu rulează.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 4 · REMOVE, POP, CLEAR

4 Ștergerea: după valoare sau *după poziție*

Ștergerea dintr-o listă se face în trei feluri, iar diferența dintre primele două este sursa cea mai frecventă de confuzie: `remove()` vrea să știe **ce valoare** să șteargă, `pop()` vrea să știe **de unde**.

REMOVE(V) · POP(P) · CLEAR()

remove(v) șterge **prima apariție** a valorii `v`; pe `[3, 7, 3, 9]` lasă `[7, 3, 9]`. Dacă valoarea nu există, programul se oprește cu eroare. **pop(p)** șterge elementul de pe **poziția** `p` și, spre deosebire de `remove`, **returnează** valoarea ștearsă; fără argument, scoate ultimul element. **clear()** golește lista: ea există în continuare, dar `len(lista)` devine 0.

```
lista = [3, 7, 3, 9]
lista.remove(3)
print(lista)
sters = lista.pop()
print(sters)
print(lista)
lista.clear()
print(len(lista))
```

Afișează, pe rând: `[7, 3, 9]`, apoi 9, apoi `[7, 3]`, apoi 0.

ATENȚIE · POZIȚIILE SE RENUMEROTEAZĂ

După fiecare ștergere, elementul care era pe poziția 1 ajunge pe 0, cel de pe 2 ajunge pe 1 și așa mai departe. De aceea, când vrei să ștergi *toate* aparițiile unei valori, nu poți parcurge lista cu un indice care crește liniștit — mai sigur este să repeți `remove(v)` cât timp `v in lista`.

A

EXERCIȚIUL A · DRY-RUN

Efectul fiecărei metode, pas cu pas

DRY-RUN

Coloana **Operație** îți este dată; tu completezi conținutul listei după operație (elementele separate prin spațiu, fără paranteze; scrie `(goală)` dacă nu are elemente) și valoarea `len`. Primul rând e completat ca exemplu.

```
lista = [5, 8, 5]
lista.append(2)
lista.insert(1, 9)
lista.remove(5)
```

```
lista.pop()
lista.pop(0)
print(lista)
```

operație (dată)	lista după operație	len
start: [5, 8, 5]	5 8 5	3
append(2)		
insert(1, 9)		
remove(5)		
pop()		
pop(0)		

HINT *`insert(1, 9)` pune 9 pe poziția 1 și împinge restul la dreapta; `remove(5)` șterge doar* **primul** *5; `pop()` ia de la sfârșit, `pop(0)` de la început.*

B

EXERCIȚIUL B · SCRIE CODUL

Șterge prima apariție

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând, iar în final o valoare `v`. Dacă `v` se află în listă, șterge prima ei apariție și afișează textul `STERS`, un spațiu, și numărul de elemente rămase. Dacă `v` nu se află în listă, afișează textul `ABSENT`, un spațiu, și numărul de elemente. Ex.: pentru `n = 4`, numerele 3, 7, 3, 9 și `v = 3` se afișează `STERS 3`.

HINT *`remove(v)` se oprește cu eroare dacă valoarea lipsește. Verifică apartenența cu `in` înainte de a apela metoda.*

SOLUȚIA TA · PYTHON

C

EXERCIȚIUL C · SCRIE CODUL

Scoate elementul de pe poziția p

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând, iar în final o poziție `p` (`0 <= p <= n - 1`). Șterge elementul de pe poziția `p` și afișează pe primul rând valoarea ștearsă, un spațiu, și numărul de elemente rămase. Pe rândul următor afișează elementele rămase, separate prin câte un spațiu. Ex.: pentru `n = 4`, numerele 10, 20, 30, 40 și `p = 1` se afișează `20 3`, apoi `10 30 40`.

HINT `pop(p)` *face două lucruri deodată: modifică lista și returnează valoarea scoasă — păstrează-o într-o variabilă. Când lista avea un singur element, al doilea rând rămâne gol.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · SORT() VS SORTED()

5 Sortarea: pe loc sau *într-o listă nouă*?

Aici se vede cel mai clar ce înseamnă că lista este **mutabilă**. Două instrumente fac aproape același lucru, dar unul schimbă lista originală, iar celălalt o lasă neatinsă.

SORT() · SORTED() · REVERSE()

sort() rearanjează elementele **chiar în lista** pe care o apelezi, crescător, și returnează `None`. **sorted()** nu este o metodă, ci o **funcție**: primește lista și produce o **listă nouă**, sortată, lăsând originalul neschimbat. **reverse()** întoarce ordinea elementelor, tot pe loc — **nu** sortează descrescător; pentru ordine descrescătoare sortezi întâi crescător, apoi inversezi.

```
b = [5, 1, 4]
b.sort()
print(b)
c = [5, 1, 4]
d = sorted(c)
print(c)
print(d)
```

Afișează `[1, 4, 5]`, apoi `[5, 1, 4]`, apoi `[1, 4, 5]`.

ATENȚIE · GREȘEALA CLASICĂ

`lista = lista.sort()` distruge lista! Metoda returnează `None`, deci variabila ajunge să conțină `None`, iar orice apel următor, de exemplu `len(lista)`, se oprește cu eroare. Scrie fie `lista.sort()` (fără atribuire), fie `copie = sorted(lista)`.

A

EXERCIȚIUL A · ASOCIERE

Ce returnează și ce modifică fiecare

ASOCIERE

Scrie în căsuță litera efectului potrivit pentru fiecare apel. Fiecare efect se folosește o singură dată.

☐ `b.sort()`

☐ `sorted(c)`

`lista.reverse()`

`lista.pop()`

A `Inversează ordinea pe loc, fără să sorteze`

B `Returnează valoarea scoasă și scurtează lista cu un element`

C `Returnează None și rearanjează chiar lista b`

D `Returnează o listă nouă, sortată, iar c rămâne neschimbată`

HINT *întreabă-te de fiecare dată două lucruri separat: ce returnează apelul și dacă schimbă lista pe care a fost apelat.*

B

EXERCIȚIUL B · SCRIE CODUL

Sortează, apoi inversează

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând. Afișează pe primul rând elementele listei sortate crescător, separate prin câte un spațiu, iar pe al doilea rând aceleași elemente în ordine inversă (adică descrescător). Ex.: pentru `n = 4` și 30, 10, 20, 0 primul rând este `0 10 20 30`, iar al doilea `30 20 10 0`.

HINT *nu scrie `lista = lista.sort()` — apeleaz-o simplu, `lista.sort()`, fără atribuire. Apoi `lista.reverse()` lucrează pe lista deja sortată.*

SOLUȚIA TA · PYTHON

C

EXERCIȚIUL C · SCRIE CODUL

sorted() nu atinge originalul

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând. Construiește cu `sorted()` o listă nouă, sortată crescător. Afișează pe primul rând elementele listei originale, în ordinea în care au fost citite, iar pe al doilea rând elementele listei sortate. Ex.: pentru `n = 4` și 30, 10, 20, 0 primul rând este `30 10 20 0`, iar al doilea `0 10 20 30`.

HINT *dacă ai folosi `lista.sort()` în loc de `sorted(lista)`, cele două rânduri ar ieși identice — exercițiul verifică exact această diferență.*

SOLUȚIA TA · PYTHON

6 SECȚIUNEA 6 · ALIAS, COPY(), EXTEND(), SLICING

Două nume pentru *aceeași listă*

Numele unei variabile-listă nu conține lista, ci **arată spre ea**, ca o etichetă lipită pe o cutie. De aici decurge un comportament care surprinde:

```python
lista = [1, 2]
alias = lista
copie = lista.copy()
lista.append(3)
print(alias)
print(copie)
```

Afișează `[1, 2, 3]`, apoi `[1, 2]`.

> **ALIAS = LISTA · COPIE = LISTA.COPY()**
>
> **Atribuirea directă** nu creează nimic nou: cele două nume ajung să arate spre **aceeași** listă din memorie, deci orice modificare se vede prin ambele nume. **copy()** creează o **listă nouă**, cu aceleași elemente; cele două liste sunt de acum independente.

`extend(alta)` adaugă la sfârșitul listei **toate elementele** altei liste, unul câte unul. Se confundă ușor cu `append()`, dar rezultatele diferă net: pornind de la `[1, 2]`, `extend([3, 4])` dă `[1, 2, 3, 4]` — patru elemente; `append([3, 4])` dă `[1, 2, [3, 4]]` — trei elemente, ultimul fiind o listă întreagă.

Ne rămâne o notație nouă, care nu e nici operator, nici metodă: **slicing-ul**. Până acum `lista[2]` dădea **un singur** element; slicing-ul selectează o **felie**:

```python
lista = [10, 20, 30, 40]
print(lista[1:3])
print(lista[:2])
print(lista[2:])
print(lista[:])
```

Afișează, pe rând: `[20, 30]`, `[10, 20]`, `[30, 40]`, `[10, 20, 30, 40]`. `lista[i:j]` ia elementele de la `i` **inclusiv** până la `j` **exclusiv**, deci felia are cel mult `j - i` elemente. Capetele pot fi omise, iar `lista[:]` selectează toate elementele — de aceea este a treia cale de a copia o listă. Există și un **pas**: pe `[10, 20, 30, 40, 50]`, expresia `lista[0:5:2]` dă `[10, 30, 50]`.

> **ATENȚIE**
>
> Slicing-ul **nu** dă eroare când capătul din dreapta trece de sfârșitul listei — se oprește pur și simplu la ultimul element existent. De aceea nu calcula lungimea feliei prin scăderea `j - i`, ci află-o cu `len()` aplicat pe felia obținută.

A

EXERCIȚIUL A · ORDONARE

Cum verifici dacă ai alias sau copie

ORDONARE

Numerotează pașii de la 1 la 6, în ordinea corectă, ca să obții procedura prin care afli dacă două nume arată spre aceeași listă.

- [ ] Modifică prima listă, de exemplu cu `lista.append(3)`
- [ ] Dacă lungimea a crescut, aveai un alias; dacă a rămas la fel, aveai o copie
- [ ] Pornește de la o listă cunoscută, de exemplu `lista = [1, 2]`
- [ ] Compară din nou lungimea celui de-al doilea nume cu cea reținută
- [ ] Reține câte elemente are al doilea nume înainte de orice modificare
- [ ] Creează al doilea nume, fie prin `alias = lista`, fie prin `copie = lista.copy()`

HINT *orice procedură de verificare are aceeași formă: pregătești situația, măsori* **înainte** *, provoci o schimbare, măsori din nou și abia apoi tragi concluzia.*

B

EXERCIȚIUL B · SCRIE CODUL

Alias sau copie?

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând. Creează o copie a listei cu `copy()` și un alias prin atribuire directă. Adaugă apoi valoarea 0 la sfârșitul listei originale și afișează pe același rând, separate prin câte un spațiu, numărul de elemente al copiei, al aliasului și al listei originale. Ex.: pentru `n = 3` și 1, 2, 3 se afișează `3 4 4`.

HINT *aliasul și originalul afișează mereu aceeași valoare — sunt același obiect. Copia rămâne mereu cu o unitate mai puțin, adică exact* `n`.

SOLUȚIA TA · PYTHON

C

EXERCIȚIUL C · SCRIE CODUL

Lipește a doua listă la prima

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi (prima listă), apoi un număr natural `m` (`m >= 1`) și `m` numere întregi (a doua listă), toate câte unul pe rând. Adaugă la prima listă toate elementele celei de-a doua, folosind metoda `extend()`. Afișează pe primul rând numărul de elemente al listei rezultate, iar pe al doilea rând elementele ei, separate prin câte un spațiu. Ex.: pentru prima listă 1, 2 și a doua 3, 4, 5 se afișează 5, apoi `1 2 3 4 5`.

HINT *nu folosi `append()` aici: ar adăuga a doua listă ca un singur element, iar lungimea ar ieși `n + 1` în loc de `n + m`.*

SOLUȚIA TA · PYTHON

D

EXERCIȚIUL D · SCRIE CODUL

Taie o felie din listă

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând, iar în final două poziții `a` și `b` (`0 <= a <= b`). Extrage cu slicing felia care începe la poziția `a` și se termină înainte de poziția `b` sau la sfârșitul listei, dacă `b > n`. Afișează pe primul rând numărul de elemente al feliei, iar pe al doilea rând elementele ei, separate prin câte un spațiu. Ex.: pentru `n = 5`, numerele 10, 20, 30, 40, 50 și `a = 1`, `b = 3` se afișează 2, apoi `20 30`.

HINT *poziția `b` este exclusă, iar slicing-ul nu dă eroare la depășire — de aceea lungimea feliei se află cu `len()`, nu prin `b - a`. Când `a` și `b` sunt egale, felia este vidă.*

SOLUȚIA TA · PYTHON

PERFORMANȚĂ · SCRIE CODUL

Copie prin lista[:], fără să atingi originalul

SCRIERE COD

Se citește un număr natural `n` (`n >= 2`), apoi `n` numere întregi, fiecare de pe câte un rând. Creează o copie a listei folosind notația `lista[:]`, apoi șterge ultimul element din copie. Afișează pe primul rând numărul de elemente al listei originale, un spațiu, și numărul de elemente al copiei. Pe al doilea rând afișează elementele listei originale, separate prin câte un spațiu. Ex.: pentru `n = 3` și 5, 8, 2 se afișează `3 2`, apoi `5 8 2`.

HINT *dacă ai scrie `copie = lista` în loc de `copie = lista[:]`, ștergerea din copie ar scurta și originalul, iar primul rând ar afișa `2 2` în loc de `3 2`.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 7 · COD GHIDAT

7 Scrie cod care *trece testele*

Cinci probleme, în ordine crescătoare de dificultate. Scrie soluția pe hârtie, urmărește-o mental pe exemplul dat și abia apoi tastează-o pe platformă, unde e verificată automat pe 5 cazuri de test.

PROBLEMA 7.1 · COD GHIDAT

1 *Câte valori depășesc primul element*

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând. Afișează pe același rând primul element al listei, un spațiu, apoi câte elemente din listă sunt **strict** mai mari decât el. Ex.: pentru `n = 5` și 10, 3, 17, 10, 20 se afișează `10 2`.

HINT *comparația este strictă: un element egal cu primul nu se numără. Primul element se compară și cu el însuși, dar `x > prim` este fals pentru el, deci nu strică rezultatul.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.2 · COD GHIDAT

2 *Lista inversă, construită element cu element*

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând. Construiește o listă nouă, care conține aceleași elemente în ordine inversă, folosind `insert(0, x)` — **fără** să apelezi metoda `reverse()`. Afișează elementele listei noi pe un singur rând, separate prin câte un spațiu. Ex.: pentru `n = 4` și 1, 2, 3, 0 se afișează `0 3 2 1`.

HINT *parcurgi lista de la stânga la dreapta și pui fiecare element* **în fața** *celor deja adăugate; ultimul element citit ajunge astfel primul.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.3 · COD GHIDAT

3 Elimină toate aparițiile

SCRIERE COD

Se citește un număr natural `n` (`n >= 1`), apoi `n` numere întregi, fiecare de pe câte un rând, iar în final o valoare `v`. Șterge din listă toate aparițiile valorii `v`. Afișează pe primul rând numărul de elemente rămase, iar pe al doilea rând elementele rămase, separate prin câte un spațiu. Ex.: pentru `n = 5`, numerele 3, 7, 3, 9, 3 și `v = 3` se afișează 2, apoi `7 9`.

HINT *`remove(v)` șterge o singură apariție per apel, iar după fiecare ștergere pozițiile se renumerotează. Cea mai sigură formă este `while v in lista:`. Dacă se șterg toate elementele, al doilea rând rămâne gol.*

SOLUȚIA TA · PYTHON

4

PROBLEMA 7.4 · REPARĂ CODUL

De ce se strică programul acesta?

SCRIERE COD

Un coleg a vrut să obțină o listă sortată fără să strice lista originală și a scris codul de mai jos. Programul nu afișează niciun rezultat: se oprește cu mesajul `TypeError: object of type 'NoneType' has no len()`. Găsește cauza, corectează programul și afișează pe primul rând numărul de elemente al listei sortate, iar pe al doilea rând elementele ei sortate crescător, separate prin câte un spațiu. Ex.: pentru `n = 3` și 30, 10, 20 se afișează 3, apoi `10 20 30`.

```
1  n = int(input())
2  lista = []
3  i = 0
4  while i < n:
5      x = int(input())
6      lista.append(x)
7      i = i + 1
8  copie = lista.sort()
9  print(len(copie))
```

HINT *eroarea confirmă chiar afirmația din S5: `sort()` returnează `None`. Nu înlocui pur și simplu `copie` cu `lista` după apel — ai obține rezultatul corect, dar ai fi sortat originalul, adică exact ce colegul voia să evite.*

PROGRAMUL CORECTAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Lista de frecvențe a cifrelor

SCRIERE COD

Se citește un număr natural `n` (`n >= 0`), scris în baza 10. Construiește lista de frecvențe a cifrelor lui: o listă cu 10 poziții, în care elementul de pe poziția `c` arată de câte ori apare cifra `c`. Afișează, câte una pe rând, doar cifrele care apar efectiv: mai întâi cifra, un spațiu, apoi de câte ori apare, în ordine crescătoare a cifrei. Ex.: pentru `n = 1223` se afișează, pe trei rânduri, `1 1`, `2 2`, `3 1`.

HINT *cifrele se extrag tot cu `while`, prin `n % 10` și `n // 10` — nu prin transformarea numărului în șir. Cazul `n = 0` este special: bucla nu rulează niciodată, deci tratează-l separat și afișează `0 1`.*

SOLUȚIA TA · PYTHON