test-e.ro

CAIET ELEV

Lecția 15

Modelul conceptual liniar – listă. Caracteristici și tipuri de acces

Cinci note încap în cinci variabile. Cinci sute, nu. Astăzi învățăm modelul care rezolvă asta: o singură denumire pentru o colecție ordonată, în care fiecare element are poziția lui. Și, mai important, învățăm ***când*** *merită să păstrezi datele și când e destul să le vezi o singură dată.*

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. Din clasa `list` folosim doar indexarea `v[i]`, `len(v)` și crearea listei cu `[0] * n` — fără `append`, `in`, `sort` sau alte metode (acelea vin la L17). Fără `def`. Regula de aur: **rezultatul îl calculezi tu, înainte de a rula.** Verificarea automată rămâne pe platformă.

CUPRINS

1 Ce știm deja: valori, cifre și parcurgeri .....
2 Lista ca model conceptual .....
3 Acces direct și acces secvențial .....
4 Căutarea unei valori .....
5 Parcurgerea fără memorare .....
6 Când lista devine necesară .....
7 Cod ghidat · șapte probleme .....

SECȚIUNEA 1 · REACTUALIZARE L9–L14

1 Ce știm *deja*

Înainte de a organiza multe valori la un loc, ne reamintim cum lucrăm cu una singură: cum îi extragem cifrele, cum numărăm și cum acumulăm un rezultat pas cu pas. Tiparul de bază e cel știut: `c = n % 10` ia ultima cifră, `n = n // 10` o îndepărtează, iar `while n > 0` repetă. Al doilea tipar este **acumularea**: o variabilă pornește de la o valoare neutră (de obicei 0) și se actualizează la fiecare pas.

```
c = 0
while n > 0:
    c = c + 1
    n = n // 10
print(c)
```

Observă ceva care va deveni important astăzi: după bucla de mai sus `n` **a ajuns 0** — valoarea inițială s-a pierdut. Am folosit cifrele o singură dată, în ordinea în care au fost extrase, și nu le mai putem reciti. Dacă am avea nevoie de ele a doua oară, ar fi trebuit să le fi **păstrat** undeva.

> **ATENȚIE · CAPCANA N = 0**
>
> Bucla `while n > 0` **nu rulează niciodată** pentru `n = 0`, deci un contor de cifre ar rămâne 0, deși numărul 0 se scrie cu o cifră. Când enunțul permite `n = 0`, tratează cazul explicit înainte de buclă.

A EXERCIȚIUL A · GRILĂ

Recapitulare L9–L14

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce valoare are `2506 // 100 % 10`? `2` `250` `5` `0`
2. După `n = 407` urmat de `n = n // 10`, ce valoare are `n`? `40` `7` `407` `4`
3. Câte cifre are 52080 și de câte ori se execută bucla `while n > 0: n = n // 10` pentru el?

   `4 cifre, 4 execuții` `5 cifre, 5 execuții` `5 cifre, 4 execuții` `6 cifre, 6 execuții`
4. Cum se testează corect, la această disciplină, dacă cifra `c` este pară?

   `c % 2 == 1` `c / 2 == 0` `c % 2 != 0` `c % 2 == 0`

HINT *`2506 // 100 = 25`, apoi `25 % 10 = 5`. Numărul de execuții ale buclei este egal cu numărul de cifre. `% 2 != 0` testează* **impar** *.*

B

EXERCIȚIUL B · DRY-RUN

Cum se extrag cifrele dintr-un număr

DRY-RUN

Urmărește bucla care numără cifrele lui `n = 6408`. Coloanele `n` și `c` de la începutul fiecărui rând sunt date; tu completezi cifra extrasă și valoarea următoare a lui `n`. Ultimul rând arată starea în care bucla se oprește.

```
1  n = 6408
2  c = 0
3  while n > 0:
4      c = c + 1
5      n = n // 10
6  print(c)
```

n (dat)	c (dat)	cifra = n % 10	n // 10
6408	0	—	—
6408	1		
640	2		
64	3		
6	4		
0	4	—	—

→ Programul afișează: _____

HINT *pe ultimul rând `n` este 0, condiția `n > 0` e falsă, bucla se oprește și se afișează `c = 4`.*

C

EXERCIȚIUL C · DRY-RUN

Suma numerelor pare dintr-un șir

DRY-RUN

Programul citește 5 numere și adună doar pe cele pare. Coloanele `i` și `x` sunt date; tu completezi restul, verdictul (DA / NU) și valoarea sumei după acel pas. Primul rând arată starea dinaintea buclei.

```
n = 5
s = 0
for i in range(n):
    x = int(input())
    if x % 2 == 0:
        s = s + x
print(s)
```

i	x	x % 2	este pară?	s
–	–	–	–	0
0	4			
1	7			
2	10			
3	3			
4	8			

→ Suma finală: _____

HINT *suma crește doar când* `x % 2` *este 0. Final: 4 + 10 + 8.*

D

EXERCIȚIUL D · DRY-RUN

Câte numere impare sunt în șir

DRY-RUN

Același tipar, alt scop: numărăm câte valori sunt impare. Completează restul, verdictul (DA / NU) și contorul după fiecare pas.

```
n = 5
c = 0
for i in range(n):
    x = int(input())
    if x % 2 != 0:
        c = c + 1
print(c)
```

i	x	x % 2	este impară?	c
–	–	–	–	0
0	9			
1	4			
2	7			
3	7			
4	2			

→ Contorul final: _____

HINT *valoarea 7 apare de două ori și se numără de fiecare dată. Contorul final este 3.*

SPRIJIN · SCRIE CODUL

Câte cifre are numărul?

SCRIERE COD

Se citește un număr natural `n`. Afișează câte cifre are numărul citit. Ex.: pentru `n = 407` se afișează `3`.

HINT *pentru `n = 0` bucla nu pornește deloc, dar 0 se scrie cu o cifră — tratează cazul separat, înainte de buclă.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

Suma cifrelor pare

SCRIERE COD

Se citește un număr natural `n`. Afișează suma cifrelor pare ale numărului citit. Ex.: pentru `1234` cifrele pare sunt 2 și 4, deci se afișează `6`.

HINT *extrage cifra cu `c = n % 10` și adaug-o la sumă doar dacă `c % 2 == 0`. Cifra 0 este pară, dar nu schimbă suma.*

SOLUȚIA TA · PYTHON

2 SECȚIUNEA 2 · LISTA CA MODEL

Lista ca *model conceptual*

Pentru media a trei note, `n1`, `n2`, `n3` sunt suficiente. Pentru media unei clase de treizeci de elevi, metoda se prăbușește — nu pentru că e grea, ci pentru că **numărul de variabile ar trebui scris în cod**, iar codul nu știe dinainte câți elevi sunt. **Lista** rezolvă acest lucru ca **model**: o singură denumire, `v`, pentru o **colecție ordonată** de elemente, fiecare cu **poziția** lui.

„Ordonată" aici **nu înseamnă sortată**. Înseamnă că fiecare element are un loc bine stabilit: al treilea element rămâne al treilea. Lista `[15, 4, 23, 8, 42]` este ordonată în acest sens, deși valorile nu sunt crescătoare.

```
1  v = [15, 4, 23, 8, 42]
2  print(v[0])
3  print(v[2])
```

```
4  print(len(v))
```

Codul de mai sus afișează 15, apoi 23, apoi 5. Numerotarea pozițiilor **începe de la 0**: într-o listă cu 5 elemente indicii valizi sunt 0, 1, 2, 3, 4 — deci ultimul element este `v[len(v) - 1]`. Când nu știm valorile, dar știm câte sunt, creăm lista goală și o completăm prin indexare cu `v = [0] * n`, apoi `v[i] = int(input())` într-o buclă `for i in range(n)`.

DE REȚINUT

La L15 lista este un **model conceptual** — un mod de a gândi organizarea datelor. Din clasa `list` folosim **strict două elemente**, ca suport de reprezentare: indexarea `v[i]` și crearea unei liste de dimensiune dată, `[0] * n`. Tot restul (`in`, `+`, `append`, `sort`, `index` …) se studiază la **L17**.

ATENȚIE

`v[len(v)]` **nu** este ultimul element — este o **eroare**. Pentru `v = [15, 4, 23, 8, 42]`, `len(v)` este 5, iar `v[5]` produce `IndexError`, fiindcă indicii merg doar până la 4. Ultimul element se scrie `v[len(v) - 1]`. Este greșeala numărul 1 la prima întâlnire cu indexarea.

A

EXERCIȚIUL A · SCRIE CODUL

Marginile unei liste

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi, fiecare de pe câte un rând. Afișează pe același rând, separate prin câte un spațiu, numărul de elemente citite, primul element și ultimul element. Ex.: pentru `n = 4` și 11, 5, 30, 8 se afișează `4 11 8`.

HINT *ultimul element se obține cu `v[len(v) - 1]`. Dacă scrii `v[len(v)]`, programul se oprește cu eroare.*

SOLUȚIA TA · PYTHON

B

EXERCIȚIUL B · COMPLETEAZĂ

Definiția listei

COMPLETARE

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

`ordonată` `sortată` `poziție` `0` `1`

O listă este o colecție _____ de elemente, în care fiecare element are o _____ bine stabilită, iar numerotarea pozițiilor începe de la _____ .

HINT *atenție la diferența dintre „ordonată" și „sortată" — doar una dintre ele descrie o listă oarecare.*

3 SECȚIUNEA 3 · TIPURI DE ACCES

Acces direct și acces *secvențial*

Faptul că elementele unei liste au poziții numerotate ne dă ceva foarte valoros: **accesul direct**. Scriem `v[3]` și ajungem la al patrulea element **într-un singur pas**, indiferent cât de lungă e lista — calculatorul nu „caută", pentru că poziția se transformă direct într-o adresă în memorie. Opusul este **accesul secvențial**: ca să ajungi la al patrulea element, treci obligatoriu prin primele trei. Un **raft cu locuri numerotate** dă acces direct; o **casetă audio** dă acces secvențial.

```
1  v = [15, 4, 23, 8, 42]
2  for i in range(len(v)):
3      print(i, v[i])
```

DE REȚINUT

`range(len(v))` produce exact indicii valizi: 0, 1, 2, 3, 4 — tiparul standard de parcurgere la această lecție. Accesul direct nu e „mai bun" în mod absolut: e mai bun **când știi poziția**. Dacă întrebarea e „care e cel mai mare element?", poziția nu ajută cu nimic — trebuie oricum să treci prin toate.

A

EXERCIȚIUL A · DRY-RUN

Indicii unei parcurgeri

DRY-RUN

Urmărește parcurgerea listei `v = [15, 4, 23, 8, 42]`. Coloana `i` este dată; tu completezi valoarea `v[i]` și răspunzi cu DA / NU la întrebarea „mai există un indice de parcurs?". Ultimul rând, cu `i = 5`, nu este o iterare reală: arată de ce bucla se încheie acolo.

```
v = [15, 4, 23, 8, 42]
for i in range(len(v)):
    print(i, v[i])
```

i (dat)	v[i]	mai există un indice de parcurs?
0	15	DA
1		
2		
3		
4		
5	—	NU

HINT *indicii valizi merg de la 0 la `len(v) - 1`, adică de la 0 la 4. La o buclă `for`, Python nu verifică o condiție de forma `i < len(v)` — iteratorul pur și simplu nu mai are ce livra.*

B

EXERCIȚIUL B · ASOCIERE

Ce fel de acces?

ASOCIERE

Scrie în căsuță litera tipului de acces potrivit pentru fiecare situație.

- ☐ `v[7]` într-o listă cu 20 de elemente
- ☐ Găsirea celui mai mare element dintr-o listă
- ☐ Deschiderea manualului la pagina 112
- ☐ Citirea a 5 numere de la tastatură, unul câte unul

A Acces secvențial — parcurg toate elementele

B Acces direct — ajung într-un pas

C Acces secvențial — nu pot sări peste

D Acces direct — poziția e cunoscută

HINT *o singură întrebare: știu dinainte poziția elementului care mă interesează? Dacă da, e acces direct.*

C

EXERCIȚIUL C · SCRIE CODUL

Câte elemente sunt egale cu primul

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi, fiecare de pe câte un rând. Afișează câte elemente ale listei sunt egale cu primul element. Ex.: pentru `n = 5` și 5, 7, 5, 1, 5 se afișează `3`.

HINT *compară fiecare `v[i]` cu `v[0]`. Primul element se compară și cu el însuși, deci rezultatul este cel puțin 1.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 4 · CĂUTAREA

4 Când cunoști valoarea și cauți *poziția*

Până acum am cerut mereu „elementul de pe poziția `k`". În practică, întrebarea vine adesea invers: **știm valoarea și vrem poziția**. „Pe ce loc în clasament e echipa noastră?", „la ce poziție apare nota 10?". Aici accesul direct nu mai ajută: nu putem sări la elementul căutat, pentru că tocmai asta întrebăm. Singura soluție este **parcurgerea**.

```
x = int(input())
p = -1
for i in range(len(v)):
    if v[i] == x and p == -1:
        p = i
print(p)
```

Două detalii de proiectare. Primul: `p` pornește de la **-1**, o valoare care **nu poate fi un indice valid** — așa deosebim „am găsit pe poziția 0" de „nu am găsit deloc". Al doilea: condiția `p == -1` face ca `p` să fie fixat doar la **prima** potrivire; fără ea, o valoare care apare de mai multe ori ar suprascrie `p` până la ultima apariție.

VALOARE SANTINELĂ

Valoarea `-1` folosită ca „încă nu am găsit" se numește **valoare santinelă**: aleasă special pentru că nu se poate confunda cu un rezultat valid. Alegerea nu e întâmplătoare — indicii sunt mereu ≥ 0, deci -1 e liber.

ATENȚIE

Nu confunda **poziția** cu **valoarea**. Pentru `v = [15, 4, 23, 8, 42]`, valoarea 23 se află pe poziția 2, iar valoarea de pe poziția 23 nu există. Când citești un enunț, întreabă-te dacă ți se cere un indice (`i`) sau un element (`v[i]`).

A

EXERCIȚIUL A · ORDONARE

Pașii unei căutări

ORDONARE

Numerotează etapele (scrie 1–6 în căsuță) în ordinea corectă a algoritmului care găsește **ultima** apariție a valorii căutate.

- ☐ Parcurge lista de la primul până la ultimul indice.
- ☐ Afișează poziția reținută.

- [ ] Citește numărul de elemente și completează lista.
- [ ] La fiecare element egal cu `x`, reține indicele lui, suprascriind valoarea reținută anterior.
- [ ] Inițializează poziția rezultat cu `-1`.
- [ ] Citește valoarea căutată `x`.

HINT *cele două citiri și inițializarea cu -1 stau înaintea parcurgerii, reținerea indicelui este pasul dinlăuntrul ei, iar afișarea vine ultima.*

B

EXERCIȚIUL B · SCRIE CODUL

Ultima apariție a unei valori

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi fiecare de pe câte un rând, iar la final o valoare `x`. Afișează poziția pe care apare **ultima** dată valoarea `x` în listă sau `-1` dacă nu apare deloc. Pozițiile se numerotează de la 0. Ex.: `n = 5`, numerele 23, 4, 23, 8, 42 și `x = 23` → `2`.

HINT *pornește variabila care reține poziția de la -1 și, de data asta,* **nu** *pune condiția `p == -1` — vrei ultima potrivire, deci lași suprascrierea să se întâmple.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · FĂRĂ MEMORARE

5 Când *nu* ai nevoie de listă

O întrebare care merită pusă înainte de a scrie orice cod: **chiar am nevoie să păstrez datele?** De multe ori, nu. Dacă vrem suma a 1000 de numere, nu ne trebuie o listă cu 1000 de elemente — ne trebuie **o singură variabilă**, actualizată la fiecare număr citit. Aceasta este **parcurgerea fără memorare**: fiecare valoare e citită, prelucrată pe loc și abandonată.

```
1  n = int(input())
2  s = 0
3  mx = 0
4  for i in range(n):
5      x = int(input())
6      s = s + x
7      if i == 0 or x > mx:
8          mx = x
9  print(s, mx)
```

Programul calculează **și** suma, **și** maximul, deși nu reține nicio valoare. Observă condiția `i == 0 or x > mx`: la primul element maximul se inițializează chiar cu acea valoare — fără asta, un maxim pornit de la 0 ar da rezultate greșite pe date negative. Avantajul e real: memoria **nu crește** cu numărul de valori. Prețul e la fel de real: datele **se pierd**, exact ca la bucla pe cifre din S1.

> **REGULA DE DECIZIE**
>
> Dacă răspunsul se poate **actualizа pas cu pas**, pe măsură ce vezi fiecare valoare, atunci nu ai nevoie de memorare. Suma, produsul, maximul, minimul, un contor — toate se calculează așa.

A

EXERCIȚIUL A · COMPLETEAZĂ

Când NU ai nevoie de listă

COMPLETARE

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

`fără` `cu` `nu` `dublu` `se pierd` `rămân`

Parcurgerea _____ memorare prelucrează fiecare valoare pe loc și nu o mai reține, deci memoria folosită _____ crește cu numărul de valori; în schimb, datele _____ după ce bucla s-a terminat.

HINT *primul gol descrie tipul parcurgerii, al doilea neagă creșterea memoriei, al treilea spune ce se întâmplă cu datele la final.*

B

EXERCIȚIUL B · SCRIE CODUL

Suma și maximul dintr-o singură parcurgere

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi, fiecare de pe câte un rând. Afișează pe același rând, separate printr-un spațiu, suma numerelor citite și cel mai mare dintre ele. Cerință: **fără** a memora numerele într-o listă. Ex.: `n = 4` și 3, 9, 2, 6 → `20 9`.

HINT *inițializează maximul cu prima valoare citită, nu cu 0 — numerele pot fi negative, iar un maxim pornit de la 0 ar rămâne 0.*

SOLUȚIA TA · PYTHON

6 SECȚIUNEA 6 · CU MEMORARE

Când lista devine *necesară*

Suma și maximul nu cer memorare. Dar iată o întrebare aparent înrudită: **câte numere sunt mai mari decât media?** Încearcă să o rezolvi dintr-o singură parcurgere și te blochezi imediat: ca să compari primul număr cu media, ai avea nevoie de medie — dar media se află abia după ce ai citit **toate** numerele. Când o afli, primul număr a trecut de mult. Aceasta este situația care **impune** memorarea: avem nevoie de **două parcurgeri**.

```python
n = int(input())
v = [0] * n
s = 0
for i in range(n):
    v[i] = int(input())
    s = s + v[i]
c = 0
for i in range(len(v)):
    if v[i] * n > s:
        c = c + 1
print(c)
```

Prima buclă citește **și** acumulează suma. A doua reia lista de la capăt — lucru posibil **doar pentru că am păstrat-o**. Observă trucul din comparație: în loc de `v[i] > s / n` scriem `v[i] * n > s`, înmulțim ambii membri cu `n` și rămânem în numere întregi.

> **REGULA DE DECIZIE, COMPLETATĂ**
>
> Memorarea e **necesară** când răspunsul depinde de valori pe care nu le-ai văzut încă în momentul prelucrării, sau când trebuie să treci de mai multe ori prin date. Media, „câte sunt peste medie", „lista e simetrică?" — toate cer memorare.

A

EXERCIȚIUL A · DRY-RUN

Numărarea elementelor peste medie

DRY-RUN

Urmărește a doua buclă pentru `v = [3, 9, 2, 6]`, cu `n = 4` și `s = 20`. Coloanele `i` și `v[i]` sunt date; tu completezi produsul, verdictul comparației cu `s` (DA / NU) și contorul după acel pas. Primul rând e exemplu.

i	v[i]	v[i] * n	> 20?	c
0	3	12	NU	0
1	9			
2	2			
3	6			

→ Contorul final: _____

HINT *compară `v[i] * n` cu `s`, nu `v[i]` cu `s`. Media este 20 / 4 = 5, iar 9 și 6 o depășesc.*

B

EXERCIȚIUL B · SCRIE CODUL

Câte elemente sunt peste medie

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere naturale, fiecare de pe câte un rând. Afișează câte dintre numerele citite sunt strict mai mari decât media lor aritmetică. Ex.: `n = 4` și 3, 9, 2, 6 → `2`.

HINT *compară cu `v[i] * n > s`, nu cu `v[i] > s / n` — a doua variantă introduce numere cu virgulă și poate da rezultate greșite.*

SOLUȚIA TA · PYTHON

★

PERFORMANȚĂ · SCRIE CODUL

Lista este simetrică?

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi, fiecare de pe câte un rând. Afișează pe același rând numărul `n`, un spațiu, apoi `DA` dacă lista este simetrică (se citește la fel de la stânga la dreapta și invers) sau `NU` altfel. Ex.: `n = 5` și 1, 2, 3, 2, 1 → `5 DA`.

HINT *elementul simetric al celui de pe poziția `i` se află pe poziția `len(v) - 1 - i`, nu pe `len(v) - i`. O listă cu un singur element este simetrică.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 7 · COD GHIDAT

7 Scrie cod care *trece testele*

Șapte 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. Toate folosesc doar `[0] * n`, `v[i]` și `len(v)`.

1

PROBLEMA 7.1 · COD GHIDAT

Al k-lea și ultimul

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi fiecare de pe câte un rând, iar la final un număr natural `k`. Afișează pe același rând, separate printr-un spațiu, al `k`-lea element al listei (numărând de la 1) și ultimul element. Ex.: `n = 5`, numerele 15, 4, 23, 8, 42 și `k = 3` → `23 42`.

HINT *al `k`-lea element numărând de la 1 se află pe indicele `k - 1`, pentru că indexarea începe de la 0.*

SOLUȚIA TA · PYTHON

2

PROBLEMA 7.2 · COD GHIDAT

Maximul și poziția lui

SCRIERE COD

Se citește un număr natural nenul `n`, apoi `n` numere naturale nenule, fiecare de pe câte un rând. Afișează pe același rând, separate printr-un spațiu, cel mai mare element al listei și poziția pe care apare prima dată (pozițiile se numerotează de la 0). Ex.: 15, 4, 23, 8, 42 → `42 4`.

HINT *dacă valoarea maximă apare de mai multe ori se cere prima poziție — folosește comparația strictă* `>`*, nu* `>=`.

SOLUȚIA TA · PYTHON

PROBLEMA 7.3 · COD GHIDAT

3 Câte elemente sunt mai mari decât vecinul din stânga

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi. Afișează câte elemente ale listei sunt strict mai mari decât elementul aflat imediat înaintea lor. Primul element nu se numără. Ex.: 3, 7, 2, 8, 8 → `2`.

HINT *parcurgerea începe de la indicele 1, nu de la 0 — altfel* `v[i - 1]` *ar însemna* `v[-1]`*, care în Python nu dă eroare, dar înseamnă ultimul element.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.4 · REPARĂ CODUL

4 Suma primelor k elemente

SCRIERE COD

Programul de mai jos ar trebui să citească `n`, apoi `n` numere întregi, apoi un număr natural `k`, și să afișeze suma primelor `k` elemente. Programul este greșit: pentru `n = 5`, numerele 10, 20, 30, 40, 50 și `k = 3`, el afișează 50 în loc de 60. Găsește greșeala și scrie programul corect.

```python
n = int(input())
v = [0] * n
for i in range(n):
    v[i] = int(input())
k = int(input())
s = 0
for i in range(1, k):
    s = s + v[i]
print(s)
```

HINT *greșeala e în intervalul de parcurgere: `range(1, k)` sare peste elementul de pe poziția 0 și se oprește cu unul prea devreme. Primele `k` elemente au indicii de la 0 la `k - 1`.*

PROGRAMUL CORECTAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Câte perechi de elemente au aceeași paritate

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi. Afișează câte perechi de elemente au aceeași paritate (fie amândouă pare, fie amândouă impare). Ordinea nu contează, iar un element nu formează pereche cu el însuși. Ex.: 3, 7, 2, 8, 5 → `4` perechi: (3,7), (3,5), (7,5) și (2,8).

HINT *două bucle imbricate:* `for i in range(0, len(v) - 1)` *și* `for j in range(i + 1, len(v))` *— așa fiecare pereche e luată o singură dată. Aceeași paritate:* `v[i] % 2 == v[j] % 2`.

SOLUȚIA TA · PYTHON

PROBLEMA 7.6 · REMEDIERE

Primul element negativ

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi. Afișează poziția primului element negativ din listă sau `-1` dacă lista nu conține niciun element negativ. Pozițiile se numerotează de la 0. Ex.: 3, -7, 2, -1, 8 → `1`.

HINT *pornește poziția de la -1 și fixează-o o singură dată: folosește condiția* `p == -1` *alături de testul de negativitate, ca în programul din S4.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.7 · EXCELENȚĂ

Câte elemente sunt mai mari decât toate cele dinaintea lor

SCRIERE COD

Se citește un număr natural `n`, apoi `n` numere întregi. Afișează câte elemente ale listei sunt **strict** mai mari decât **toate** elementele aflate înaintea lor. Primul element se numără întotdeauna. Ex.: 3, 7, 2, 8, 8 → `3` (3, 7 și primul 8).

HINT *problema are două soluții corecte, cu costuri diferite: una cu bucle imbricate, alta cu o singură parcurgere, care reține maximul de până acum. Încearcă-le pe amândouă și compară cât lucru face fiecare.*

SOLUȚIA TA · PYTHON