INFORMATICĂ · PYTHON · CLASA A IX-A · ORGANIZAREA CONCEPTUALĂ A DATELOR · LECȚIA 14

Modelul liniar: listă, stivă, coadă, listă de frecvențe

Durată: 150 min Nivel: mate-info · Python Înainte: lecțiile py9-02, py9-09

CE VEI ȘTI DUPĂ LECȚIE

- Recunoști o listă (model liniar) și explici ce înseamnă primul, ultimul, predecesorul și succesorul unui element.
- Deosebești accesul direct de accesul secvențial și stiva (LIFO) de coadă (FIFO), cu exemple din viața reală.
- Construiești și interpretezi o listă de frecvențe.
- Alegi între parcurgerea fără memorare și parcurgerea cu memorare, după ce cere problema.

1. Lista: date așezate „la rând”

Gândește-te la mașinile oprite la semafor pe o stradă cu sens unic. Există o **primă** mașină și o **ultimă** mașină, iar fiecare dintre celelalte are **exact o mașină în fața ei** (predecesor) și **exact una în spatele ei** (succesor). Asta este o **listă**: un șir de elemente așezate liniar, în care contează ordinea.

În Python, modelul se scrie direct cu `list`. Pozițiile (indicii) încep de la **0**.

```
masini = ["Dacia", "Ford", "BMW", "Opel", "Skoda"]
print(masini[0], masini[-1], len(masini))     # Dacia Skoda 5
i = 2
print("înainte de", masini[i], "e", masini[i - 1], "; după e", masini[i + 1])
# înainte de BMW e Ford ; după e Opel
```

Noțiune	În listă	În Python, pentru lista `a` cu `n` elemente
primul element	nu are predecesor	`a[0]`
ultimul element	nu are succesor	`a[n - 1]` sau `a[-1]`
predecesorul lui `a[i]`	există pentru `i ≥ 1`	`a[i - 1]`
succesorul lui `a[i]`	există pentru `i ≤ n − 2`	`a[i + 1]`
lista vidă	niciun element	`[]`, `len(a) == 0`

Citirea unei liste de pe un rând o știi deja: `a = list(map(int, input().split()))`.

2. Acces direct și acces secvențial

Diferența dintre modelele liniare este **cum ajungi la un element**.

- **Acces direct**: ajungi imediat la elementul de pe o poziție dată. Profesorul îl strigă pe al cincilea elev din catalog fără să-i citească pe primii patru: `catalog[4]`.

- **Acces secvențial**: ca să ajungi la un element, treci pe rând prin cele dinaintea lui. Așa cauți o melodie pe o casetă sau un nume pe o listă nesortată.

```
catalog = ["Ana", "Bogdan", "Carla", "Dan", "Elena", "Filip"]
print(catalog[4])                  # Elena: acces direct, un singur pas
```

Lista Python permite acces direct, dar multe prelucrări sunt totuși **parcurgeri secvențiale**: verifici fiecare element, de la primul la ultimul. Exemplu: câte perechi de zile **consecutive** au avut aceeași temperatură?

```
temperaturi = [3, 5, 5, 2, 7, 7, 7, 1]
c = 0
for i in range(len(temperaturi) - 1):      # i merge până la penultimul
    if temperaturi[i] == temperaturi[i + 1]:
        c += 1
print(c)                                   # 3: (5,5), (7,7), (7,7)
```

ATENȚIE — GREȘELI FRECVENTE

La perechile consecutive, `i` merge până la `n - 2`, adică `range(len(a) - 1)`. Cu `range(len(a))`, ultimul pas cere `a[n]`, care nu există: `IndexError`. Testează mereu și o pereche aflată **la capătul** listei.

3. Stiva: ultimul venit, primul servit (LIFO)

O **stivă** de farfurii: pui farfuria nouă **deasupra** și tot de deasupra o iei. Elementul scos este mereu **ultimul adăugat**: LIFO (*Last In, First Out*). Ai acces doar la un capăt, numit **vârf**.

În Python, vârful stivei este **sfârșitul listei**: `append(x)` pune, `pop()` scoate, `s[-1]` doar se uită.

```
sarcini = []
sarcini.append("teme la mate")
sarcini.append("sună bunica")
sarcini.append("udă florile")
print(sarcini.pop())      # udă florile   (ultima adăugată)
print(sarcini.pop())      # sună bunica
print(sarcini)            # ['teme la mate']
print(sarcini[-1])        # teme la mate  (vârful, fără să-l scoatem)
```

Unde apare: butonul **Anulează** (*undo*) dintr-un editor, butonul **Înapoi** din browser, farfuriile, cărțile puse una peste alta.

4. Coada: primul venit, primul servit (FIFO)

La **coada** de la casa de bilete a cinematografului, oamenii intră **la sfârșit** și sunt serviți **de la început**. Primul sosit pleacă primul: FIFO (*First In, First Out*).

În Python: `append(x)` adaugă la sfârșit, `pop(0)` scoate de la început. Aceleași trei sarcini, puse într-o coadă, ies în **altă ordine**:

```python
coada = []
coada.append("teme la mate")
coada.append("sună bunica")
coada.append("udă florile")
print(coada.pop(0))       # teme la mate  (prima adăugată)
print(coada.pop(0))       # sună bunica
print(coada)              # ['udă florile']
```

	Stivă	Coadă
regula	LIFO	FIFO
adaugi	la vârf: `s.append(x)`	la sfârșit: `q.append(x)`
scoți	de la vârf: `s.pop()`	de la început: `q.pop(0)`
exemple	undo, farfurii, „Înapoi”	apeluri la un call-center, imprimantă, rândul la ghișeu

DE ȘTIUT

Stiva și coada nu sunt tipuri noi de date, ci **reguli de acces** puse peste o listă. Tu respecți regula: la o stivă nu iei elemente din mijloc. Pentru cozi foarte lungi, Python are `collections.deque`, mai rapid la scoaterea din față; pentru problemele noastre, `pop(0)` este suficient.

`pop()` pe o listă vidă dă eroare: `IndexError: pop from empty list`. Înainte să scoți, verifici `if s:` (o listă nevidă este „adevărată”).

5. Lista de frecvențe: numeri, nu memorezi

Un magazin cere clienților o notă de la 1 la 5 stele. Nu ne interesează **cine** a dat ce notă, ci **de câte ori** apare fiecare notă. Pentru asta folosim o **listă de frecvențe**: poziția este **valoarea**, iar elementul este **numărul de apariții**.

```python
raspunsuri = [5, 4, 5, 3, 5, 1, 4, 5, 2, 4, 5, 3]
fr = [0] * 6                  # pozițiile 0..5; poziția 0 nu se folosește
for r in raspunsuri:
    fr[r] += 1                # încă o apariție a valorii r
print(fr)                     # [0, 1, 1, 2, 3, 5]
print(fr.index(max(fr)))      # 5: nota cea mai frecventă
```

Citire: `fr[4] = 3` înseamnă „trei clienți au dat 4 stele”. Suma frecvențelor (`sum(fr) = 12`) este numărul de răspunsuri. Aceeași idee centralizează voturile la alegeri sau numără erorile zilnice de fiecare tip.

LA BAC

Lista de frecvențe funcționează când valorile sunt **numere întregi dintr-un interval mic și cunoscut** (cifre, note 1–10, stele 1–5, coduri 1–8). Dimensiunea ei este `valoarea maximă + 1`. Pentru valori de până la un miliard nu are sens: ai avea nevoie de un miliard de contoare.

6. Parcurgere cu sau fără memorare

Întrebarea-cheie: **am nevoie de valori după ce le-am citit?**

Fără memorare. Verifici facturile lunare: pentru fiecare citești suma și dacă e plătită (1) sau nu (0), și numeri facturile restante. Fiecare factură se prelucrează imediat și apoi poate fi „uitată".

```
n = int(input())
restante = total = 0
for _ in range(n):
    suma, platit = map(int, input().split())
    if platit == 0:
        restante += 1
        total += suma
print(restante, total)        # pentru 4 / 120 0 / 85 1 / 230 0 / 60 1: 2 350
```

Cu memorare. Vrei notele **peste media clasei**. Media o afli abia după ce ai văzut **toate** notele, deci trebuie să le păstrezi și să le parcurgi a doua oară.

```
note = [7, 9, 5, 10, 8, 6]
media = sum(note) / len(note)
peste = []
for x in note:
    if x > media:
        peste.append(x)
print(f"{media:.2f}", peste)  # 7.50 [9, 10, 8]
```

Cerința	Memorare?	De ce
suma, numărul, maximul unor valori	nu	se actualizează la fiecare valoare
valorile peste medie	da	media se știe abia la final
zilele cu numărul minim de clienți	da	minimul se știe abia la final, apoi cauți zilele
valorile în ordine inversă	da	ultima citită se afișează prima
perechi consecutive cu o proprietate	nu (ajunge valoarea anterioară)	compari doar cu precedenta

7. Cum alegi modelul

Situația	Model	Argument
cereri la suportul tehnic	coadă	cine a sunat primul este servit primul (corect față de clienți)
„anulează ultima acțiune"	stivă	se anulează mereu cea mai recentă
catalogul clasei	listă cu acces direct	vrei imediat elevul de pe poziția k

Situația	Model	Argument
câte voturi a primit fiecare candidat (1–8)	listă de frecvențe	interval mic de coduri, contează doar numărul
registru de intrări din care numeri vizitatorii de sâmbătă	parcurgere fără memorare	o singură trecere, un contor

Testele tipice pentru toate: **lista vidă**, **un singur element**, elementul căutat **la început** și **la sfârșit**.

8. Verifică-te

1. La semafor sunt 6 mașini. Câte dintre ele au și predecesor, și succesor?
2. Ce conține `s` la final? `s = []`, `s.append(1)`, `s.append(2)`, `s.pop()`, `s.append(3)`.
3. Ce valoare întoarce al doilea `pop(0)` și ce rămâne în `q`? `q = [1]`, `q.append(2)`, `q.pop(0)`, `q.append(3)`, `q.append(4)`, `q.pop(0)`.
4. Vrei frecvențele notelor de la 1 la 10. Ce lungime are lista `fr` și ce înseamnă `fr[7] == 3`?
5. Care cerință are nevoie de memorare: (a) cea mai mare temperatură dintr-o lună; (b) zilele în care temperatura a depășit media lunii?
6. Butonul „Înapoi" din browser: stivă sau coadă? Dar ordinea în care imprimanta tipărește documentele trimise?
7. Ce se întâmplă la `[].pop()` și cum previi situația?

Răspunsuri

1. **4** (toate, în afară de prima și ultima).
2. `[1, 3]`.
3. Întoarce `2`; în `q` rămâne `[3, 4]`.
4. Lungime **11** (pozițiile 0–10, poziția 0 nefolosită); trei elevi au luat nota 7.
5. (b): media se cunoaște abia după toate valorile. La (a) ajunge un maxim actualizat pe parcurs.
6. „Înapoi": **stivă** (revii la ultima pagină vizitată). Imprimanta: **coadă**.
7. `IndexError: pop from empty list`. Verifici înainte: `if s: s.pop()`.

Probleme de lucru în clasă (pbinfo)

Le rezolvăm împreună în ședință, în ordinea de mai jos.

- [ ] #486 **minmax0** — ușoară
  https://www.pbinfo.ro/probleme/486/minmax0
- [ ] #487 **numarare2** — ușoară
  https://www.pbinfo.ro/probleme/487/numarare2
- [ ] #187 **ciffrecv** — ușoară
  https://www.pbinfo.ro/probleme/187/ciffrecv

Temă pentru acasă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] #553 **pozminmax** — ușoară
  https://www.pbinfo.ro/probleme/553/pozminmax
- [ ] #264 **maxcif** — ușoară
  https://www.pbinfo.ro/probleme/264/maxcif
- [ ] #267 **unice** — ușoară
  https://www.pbinfo.ro/probleme/267/unice