INFORMATICĂ · PYTHON · CLASA A IX-A · MEMORAREA DATELOR ȘI ORGANIZAREA CODULUI (INTENSIV) · LECȚIA I2

Clasa list, în profunzime, și tablourile din C++

Durată: 200 min Nivel: intensiv · Python + C++ Înainte: lecțiile py9-09, py9-14, py9-15, py9-17

CE VEI ȘTI DUPĂ LECȚIE

- Prezici efectul metodelor extend(), clear(), remove(), sort(), reverse() și copy() atunci când aceeași listă are mai multe nume.
- Sortezi după un criteriu (key=) și explici de ce sort(reverse=True) nu este întotdeauna același lucru cu sort() urmat de reverse().
- Declari în C++ un tablou unidimensional potrivit problemei (tip, dimensiune, nume) și îl prelucrezi cu operatorul [ ] și for.
- Compari tabloul din C++ cu lista din Python: dimensiune, tip, copiere, verificarea indicilor.

1. extend() și clear(): același obiect, alt conținut

Tabelul cu toate metodele și diferența `b = a` / `b = a.copy()` le ai în lecția py9-15. Aici ne uităm la **efecte** care nu se văd la prima vedere. Întrebarea-cheie, la fiecare metodă: *modifică lista existentă sau creează alta?*

`extend()` primește orice colecție parcurgibilă (listă, `range`, text) și îi adaugă elementele **unul câte unul**.

```
cos = ["lapte", "pâine"]
cos.extend(["ouă", "unt"])        # ['lapte', 'pâine', 'ouă', 'unt']
cos.append(["mere", "pere"])      # o SINGURĂ listă în plus: len(cos) devine 5
litere = []
litere.extend("ion")              # ['i', 'o', 'n']: un text se parcurge literă cu literă
nr = [1, 2]
nr.extend(range(3, 6))            # [1, 2, 3, 4, 5]
a = [1, 2]
a.extend(a)                       # [1, 2, 1, 2]
```

`a += b` se comportă ca `a.extend(b)` (pe loc, acceptă și `+= "ab"`), dar `a = a + "ab"` dă `TypeError`: `+` lipește doar listă cu listă.

`clear()` golește obiectul; `a = []` doar mută numele. Diferența apare când lista are încă un nume:

```
cos = ["lapte", "ouă"]
comanda = cos                     # același coș, alt nume
cos.clear()
print(cos, comanda)               # [] []            ← comanda s-a pierdut!

cos = ["lapte", "ouă"]
comanda = cos
cos = []                          # cos indică o listă NOUĂ, goală
print(cos, comanda)               # [] ['lapte', 'ouă']
```

La fel într-o funcție: `def goleste(l): l.clear()` golește lista apelantului, pe când `l = []` în funcție nu schimbă nimic în afara ei (vezi py9-09, secțiunea 3).

> **SFAT**
>
> Regula de aur: **metodele** (`extend` , `clear` , `remove` , `sort` , `reverse`) lucrează pe obiect și întorc `None` ; **operatorii** `+` , `*` , slicing-ul și `sorted()` construiesc o listă nouă. Singura metodă din lecție care întoarce o listă este `copy()` .

2. remove(): ștergeri controlate

`remove(x)` șterge **prima** apariție. Pentru alte cerințe combini metodele:

Cerința	Soluție	De ce merge
șterge **toate** aparițiile lui 0	`for _ in range(a.count(0)): a.remove(0)`	numărul de ștergeri e fixat **înainte**
la fel, fără să modifici lista	construiești `b` : `for e in a: if e != 0: b.append(e)`	o singură parcurgere
șterge **ultima** apariție	`a.reverse()` , `a.remove(x)` , `a.reverse()`	ultima devine prima
la fel, cu poziția	`i = len(a) - 1 - a[::-1].index(x)` , apoi `a.pop(i)`	poziția din listă inversată, „întoarsă”

```
p = ["Ana", "Dan", "Ana", "Ion"]
p.reverse(); p.remove("Ana"); p.reverse()
print(p)                          # ['Ana', 'Dan', 'Ion']
```

> **ATENȚIE — GREȘELI FRECVENTE**
>
> `remove()` caută valoarea de la început, deci fiecare apel parcurge lista. `while 0 in a: a.remove(0)` este corect, dar pe o listă cu multe zerouri face foarte multe parcurgeri (și `in` , și `remove` caută). Pe liste mari, varianta cu lista nouă `b` este de departe cea mai rapidă.

3. sort() pe bune: texte, criterii, stabilitate

Textele se compară după codurile caracterelor: toate majusculele (A = 65 … Z = 90) sunt înaintea minusculelor (a = 97), iar literele cu diacritice au coduri mari (Ș = 536).

```
nume = ["ana", "Zoe", "Ștefan", "Bogdan", "Emil"]
print(sorted(nume))                 # ['Bogdan', 'Emil', 'Zoe', 'ana', 'Ștefan']
print(sorted(nume, key=str.lower))  # ['ana', 'Bogdan', 'Emil', 'Zoe', 'Ștefan']
```

`key=str.lower` compară variantele cu litere mici, dar **păstrează** textele originale. „Ștefan” rămâne la final: ordinea alfabetică românească reală cere instrumente pe care nu le folosim aici. O listă cu tipuri amestecate (`[3, "3"]`) nu se poate sorta: `TypeError` .

`key=` **cu funcția ta.** Lista elevilor are perechi `[nume, notă]`. Vrem clasamentul după notă, descrescător, iar la note egale în ordine alfabetică.

```
elevi = [["Radu", 9], ["Ana", 10], ["Ioana", 9], ["Bogdan", 10], ["Cezar", 8]]
def nota(e):
    return e[1]
elevi.sort()                          # pasul 1: după nume (criteriul secundar)
elevi.sort(key=nota, reverse=True)    # pasul 2: după notă (criteriul principal)
print(elevi)
# [['Ana', 10], ['Bogdan', 10], ['Ioana', 9], ['Radu', 9], ['Cezar', 8]]
```

Merge pentru că `sort()` este **stabil**: elementele cu aceeași cheie își păstrează ordinea dinaintea sortării. Deci sortezi întâi după criteriul **secundar**, apoi după cel **principal**.

4. Lista de prețuri și catalogul: sort, reverse, copy

Evidența vânzărilor. Prețurile se înregistrează în ordinea vânzării. Două rapoarte, două metode diferite:

```
vanzari = [45, 12, 99, 30, 12]
ultimele = vanzari.copy(); ultimele.reverse()
print(ultimele)                   # [12, 30, 99, 12, 45]  ← cele mai recente primele
scumpe = sorted(vanzari, reverse=True)
print(scumpe)                     # [99, 45, 30, 12, 12]  ← cele mai scumpe primele
print(vanzari)                    # [45, 12, 99, 30, 12]  ← evidența rămâne intactă
```

`reverse()` răspunde la „în ce ordine s-au întâmplat”, `sort()` la „cât de mari sunt”. Pe numere simple, `sort()` urmat de `reverse()` dă același rezultat ca `sort(reverse=True)`. **Cu `key=` nu mai este adevărat**, când există chei egale:

```
prod = [["pix", 5], ["caiet", 8], ["riglă", 5], ["gumă", 3]]
def pret(e): return e[1]
a1 = sorted(prod, key=pret, reverse=True)
a2 = sorted(prod, key=pret); a2.reverse()
print(a1)   # [['caiet', 8], ['pix', 5], ['riglă', 5], ['gumă', 3]]
print(a2)   # [['caiet', 8], ['riglă', 5], ['pix', 5], ['gumă', 3]]
```

`reverse=True` păstrează ordinea inițială a egalilor (sortarea rămâne stabilă); `reverse()` de după îi inversează și pe ei.

Catalogul cu liste paralele. `nume[i]` și `note[i]` aparțin aceluiași elev. Dacă sortezi `note` pe loc, legătura se rupe:

```
nume = ["Ana", "Bogdan", "Carla", "Dan", "Elena"]
note = [7, 10, 5, 9, 8]
clasament = note                  # GREȘIT: alias, nu copie
clasament.sort(reverse=True)
print(note)                       # [10, 9, 8, 7, 5]
print(nume[note.index(10)])       # Ana  ← fals: 10 l-a luat Bogdan
```

Corect: `clasament = note.copy()` (sau `sorted(note, reverse=True)`). Atunci `note` rămâne `[7, 10, 5, 9, 8]` și poți răspunde la întrebări care cer **ambele** ordini: locul lui Dan este

`clasament.index(note[nume.index("Dan")]) + 1`, adică `2`, iar mediana este `clasament[len(clasament) // 2]`, adică `8`.

> **DE ȘTIUT**
>
> `copy()` este o copie **superficială**: copiază lista, nu și listele din interiorul ei (vezi fișa fp9-15, P22). Pentru `elevi` (listă de perechi), `elevi.copy()` îți dă altă ordine a acelorași perechi: sortarea copiei e sigură, dar `copie[0][1] = 4` schimbă nota și în original.

5. Primul tablou în C++: declarare

În C++ fiecare variabilă are un **tip declarat**, iar un **tablou unidimensional** este un șir de elemente de **același tip**, cu **dimensiune fixă**, stabilită la declarare. Programul minim, alături de Python:

Python	C++
(nimic)	`#include <iostream>` și `using namespace std;`
codul se scrie direct	codul stă în `int main() { … return 0; }`
`a, b = map(int, input().split())`	`int a, b;` și `cin >> a >> b;`
`print(a + b)`	`cout << a + b << "\n";`
blocuri prin indentare, fără `;`	blocuri între `{ }`, fiecare instrucțiune cu `;`

Stocul unui magazin cu 100 de produse: produsul `i` are `stoc[i]` bucăți.

```cpp
int stoc[100] = {0};     // tip   nume[dimensiune] = {valori inițiale};
stoc[0] = 25;            // primul produs
stoc[99] = 4;            // ultimul produs: indicele 99, NU 100
stoc[0] -= 3;
cout << stoc[0] << " " << stoc[1] << " " << stoc[99];   // 22 0 4
```

Alegerea	Aici	De ce
tipul	`int`	bucățile sunt numere întregi; pentru kWh sau prețuri: `double`
dimensiunea	`100`	o constantă cunoscută la compilare; nu se mai schimbă
numele	`stoc`	spune ce reprezintă fiecare element (nu `v`, `a`, `x`)
indicii	`0 … 99`	mereu de la 0 la dimensiune − 1

Câtă memorie ocupă un tablou? Operatorul `sizeof` dă dimensiunea în **octeți**. Pe calculatoarele obișnuite, un element ocupă: `char` 1 octet, `int` 4, `double` 8, `long long` 8. Un tablou ocupă **dimensiunea · octeții unui element**:

```cpp
int stoc[100] = {0};
cout << sizeof(stoc) << " " << sizeof(stoc[0]);          // 400 4
cout << sizeof(stoc) / sizeof(stoc[0]);                  // 100 (câte elemente are)
```

Deci `int stoc[100]` ocupă 400 de octeți, iar `double t[7]` ocupă 56. Tipul alege și **domeniul valorilor**: un `int` ajunge cam până la 2 miliarde (`INT_MAX` , din `<climits>`), un `long long` mult mai sus. Dacă sumele pot depăși limita unui tip, alegi tipul mai mare, chiar dacă ocupă mai mult. Valorile de mai sus sunt cele ale unui compilator uzual (g++); la alte sisteme pot diferi, de aceea se verifică cu `sizeof` .

Inițializarea: `int t[5] = {4, 7};` dă `4 7 0 0 0` (restul se completează cu 0). Un tablou **local fără inițializare** conține valori întâmplătoare; un tablou **global** (declarat înainte de `main`) pornește cu toate elementele 0. Când `n` vine de la tastatură, declari tabloul la dimensiunea maximă din enunț (`n ≤ 100` → `int a[100];`) și folosești doar primele `n` poziții.

> **LA BAC**
>
> La Bac, enunțurile numerotează adesea elementele de la 1 la `n` . Atunci declari `int a[101];` și parcurgi `for (int i = 1; i <= n; i++)` , lăsând `a[0]` nefolosit. Important este să alegi **o** convenție și să o respecți în tot programul.

6. Operatorul [ ] și parcurgerile clasice

`temp[2]` este **elementul** de pe poziția 2, adică temperatura din **a treia zi**. Indicele leagă poziția de valoare; cu el citești și modifici. Parcurgerile le știi din Python (py9-14); se schimbă doar scrierea.

```cpp
#include <iostream>
using namespace std;

int main() {
    int temp[7] = {12, 15, 9, 18, 18, 11, 14};   // temperaturile unei săptămâni
    cout << temp[2] << "\n";       // a treia zi: 9
    temp[2] = 10;                  // corectăm valoarea zilei a treia
    int suma = 0, imax = 0;
    for (int i = 0; i < 7; i++) {
        suma += temp[i];
        if (temp[i] > temp[imax])
            imax = i;
    }
    cout << suma << " " << imax + 1 << "\n";   // 98 4
    return 0;
}
```

Ziua afișată este `imax + 1` (zilele se numără de la 1, indicii de la 0). Cu `>` găsești **prima** zi cu maximul (ziua 4); cu `>=` ai găsi **ultima** (ziua 5).

Operația	Python	C++
citire `n` , apoi `n` valori	`a = list(map(int, input().split()))`	`cin >> n;` `for (int i = 0; i < n; i++) cin >> a[i];`
afișare în ordine inversă	`for x in a[::-1]: print(x, end=" ")`	`for (int i = n - 1; i >= 0; i--) cout << a[i] << " ";`
suma	`sum(a)`	`int s = 0;` + `s += a[i]` în `for`

Operația	Python	C++
maximul și poziția lui	`max(a)`, `a.index(max(a))`	`imax = 0;` + `if (a[i] > a[imax]) imax = i;`
aparițiile lui `x`	`a.count(x)`	`if (a[i] == x) ap++;` în `for`

În C++ nu există `sum`, `max` pe tablou, `count` sau `a[-1]`: le scrii tu, cu `for`. Exact algoritmii din gimnaziu.

7. Fără plasă de siguranță: limitele tabloului

```cpp
int t[7] = {12, 15, 9, 18, 18, 11, 14};
int s = 0;
for (int i = 0; i <= 7; i++)      // BUG: i ajunge la 7
    s += t[i];
```

Python ar opri programul cu `IndexError: list index out of range`. **C++ nu verifică indicii**: `t[7]` citește memoria de după tablou. Programul poate afișa o sumă greșită, poate „merge" pe un calculator și pica pe altul sau se poate opri brusc. Se numește **comportament nedefinit**. Pentru acest cod, `g++ -Wall -O2` dă un avertisment (`iteration 7 invokes undefined behavior`), iar `g++ -fsanitize=address` oprește programul cu `stack-buffer-overflow`. Citește avertismentele! Și testează mereu cazurile limită: `n = 1`, `n` maxim, elementul căutat pe prima și pe ultima poziție.

	`list` din Python	tablou din C++
dimensiunea	crește și scade (`append`, `pop`)	fixă, aleasă la declarare
tipul elementelor	oricare, chiar amestecate	unul singur, declarat
lungimea	`len(a)`	o știi tu, într-o variabilă `n`
index invalid	`IndexError`, programul se oprește	nicio verificare: comportament nedefinit
indici negativi	`a[-1]` = ultimul	interzis (tot comportament nedefinit)
`b = a`	alias: același obiect	eroare de compilare (`invalid array assignment`); copiezi cu `for`
metode	`sort`, `count`, `index`, …	niciuna; scrii algoritmii
viteza	mai lent	foarte rapid, elementele stau unul lângă altul în memorie

8. Verifică-te

1. Ce conține `a` după `a = [5]`, `a.extend("12")`, `a.extend([12])`?
2. `x = [1, 2]`, `y = x`, apoi `x = []`. Ce conține `y`? Dar dacă în loc de `x = []` scriai `x.clear()`?
3. Ce afișează `print(sorted(["mac", "Măr", "Mere"]))`? (Ajutor: `ord("ă") = 259`.)

4. Lista `[["A", 2], ["B", 1], ["C", 2]]` se sortează după al doilea element cu `reverse=True`. Ce ordine obții? Dar cu `sort` crescător urmat de `reverse()`?
5. Declară tabloul pentru consumul lunar de energie (în kWh, cu zecimale) al celor 24 de apartamente dintr-un bloc. Care este indicele ultimului apartament?
6. În `for (int i = 1; i <= n; i++) cin >> a[i];` cu `int a[100];` și `n = 100`, ce se întâmplă?
7. De ce `int b[7]; b = temp;` nu se compilează, iar în Python `b = temp` „merge"? Ce face fiecare limbaj, de fapt?

Răspunsuri

1. `[5, '1', '2', 12]`: textul se desface în caractere, lista adaugă un element.
2. `[1, 2]`: `x = []` doar mută numele `x`. Cu `x.clear()`, `y` ar fi `[]` (aceeași listă).
3. `['Mere', 'Măr', 'mac']`: majusculele sunt primele, iar `e` (101) < `ă` (259).
4. `[['A', 2], ['C', 2], ['B', 1]]`; a doua variantă dă `[['C', 2], ['A', 2], ['B', 1]]`.
5. `double consum[24];`, ultimul indice este `23`.
6. `a[100]` nu există (indicii sunt 0…99): comportament nedefinit. Corect: `int a[101];` sau indici de la 0.
7. În C++ tablourile nu se pot atribui; copierea se face element cu element. În Python `b = temp` nu copiază nimic, doar dă un al doilea nume aceleiași liste.

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
- [ ] **#491** **suma2** — ușoară
  https://www.pbinfo.ro/probleme/491/suma2
- [ ] **#493** **constr** — ușoară
  https://www.pbinfo.ro/probleme/493/constr

Temă pentru acasă (pbinfo)

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

- [ ] **#487** **numarare2** — ușoară
  https://www.pbinfo.ro/probleme/487/numarare2
- [ ] **#553** **pozminmax** — ușoară
  https://www.pbinfo.ro/probleme/553/pozminmax
- [ ] **#161** **inlocuire** — ușoară
  https://www.pbinfo.ro/probleme/161/inlocuire