INFORMATICĂ · PYTHON · CLASA A IX-A · MEMORAREA DATELOR ȘI ORGANIZAREA CODULUI · LECȚIA 15

# Clasa list din Python

Durată: 150 min  Nivel: mate-info · Python  Înainte: lecțiile py9-14

**CE VEI ȘTI DUPĂ LECȚIE**

- Folosești operatorii listelor: acces și modificare cu [], in / not in, +, *, comparații.
- Alegi metoda potrivită (append, insert, pop, remove, index, count, extend, reverse, clear, copy, sort) și știi ce întoarce fiecare.
- Explici diferența dintre b = a și b = a.copy() și dintre a.sort() și sorted(a).
- Extragi o porțiune de listă cu slicing (a[i:j], a[::-1]).

## 1. O listă este un obiect

În lecția py9-14 ai folosit liste ca model: stivă, coadă, frecvențe. Acum ne uităm la **instrumentul** Python. O listă este un **obiect al clasei predefinite** `list` : are date (elementele) și **metode** (operații pe care le apelezi cu punct: `note.append(8)` ).

```
note = [9, 7, 10]
print(type(note), len(note))              # <class 'list'> 3
print(list(range(1, 6)), list("abc"), list())
# [1, 2, 3, 4, 5] ['a', 'b', 'c'] []
```

| Caracteristică | Ce înseamnă | Față de tabloul din C++ |
|---|---|---|
| ordonată | elementele au poziții `0, 1, …, n – 1` | la fel |
| mutabilă | poți modifica, adăuga, șterge elemente pe loc | la fel la modificare |
| dinamică | crește și scade singură | `int a[100]` are mărime fixă |
| eterogenă | poate amesteca tipuri: `[3, "trei", 3.0]` | toate elementele au același tip |

**SFAT**

Faptul că o listă poate amesteca tipuri nu înseamnă că e o idee bună. La probleme ții o listă **de un singur fel** (numere, sau nume), altfel `sort()` , `sum()` și comparațiile dau erori.

## 2. Operatorii listelor

```
note = [9, 7, 10, 8]
note[1] = 8                 # modificare pe loc
print(note, note[-1])       # [9, 8, 10, 8] 8
print(10 in note, 5 not in note)          # True True
d1 = ["lapte", "pâine"]
d2 = ["ouă"]
print(d1 + d2, d2 * 3)      # ['lapte', 'pâine', 'ouă'] ['ouă', 'ouă', 'ouă']
```

| Operator | Rol | Exemplu | Rezultat |
|---|---|---|---|
| `a[i]` | acces / modificare | `[9, 8, 10][-1]` | `10` |
| `in`, `not in` | apartenență | `5 in [1, 5]` | `True` |
| `+` | concatenare (listă nouă) | `[1, 2] + [3]` | `[1, 2, 3]` |
| `*` | multiplicare (listă nouă) | `[0] * 4` | `[0, 0, 0, 0]` |
| `==`, `!=` | aceleași elemente, în aceeași ordine | `[1, 2] == [2, 1]` | `False` |
| `<`, `>`, `<=`, `>=` | comparație **lexicografică** | `[1, 2, 9] < [1, 3]` | `True` |

Comparația lexicografică merge ca în dicționar: compari primul element; dacă sunt egale, al doilea, și tot așa. Prima diferență decide: `[2] > [1, 99, 99]` este `True`. Dacă o listă este începutul celeilalte, cea mai scurtă e mai mică: `[1, 2] < [1, 2, 0]`.

> **ATENȚIE — GREȘELI FRECVENTE**
>
> `note[4]` pe o listă cu 4 elemente dă `IndexError: list index out of range`. Indicii valizi sunt `0 … len(a) - 1` și, din spate, `-1 … -len(a)`.

## 3. Bucăți de listă: slicing

`a[i:j]` este o **listă nouă** cu elementele de la poziția `i` **până la** `j`, **fără** `j` (ca la `range`).

```
a = [10, 20, 30, 40, 50, 60]
print(a[1:4], a[:2], a[3:])       # [20, 30, 40] [10, 20] [40, 50, 60]
print(a[-2:], a[::2], a[::-1])    # [50, 60] [10, 30, 50] [60, 50, 40, 30, 20, 10]
```

`a[:2]` = primele 2, `a[-2:]` = ultimele 2, `a[::2]` = din 2 în 2, `a[::-1]` = inversată. Slicing-ul nu dă niciodată `IndexError`: `a[4:100]` este pur și simplu `[50, 60]`.

## 4. Metodele clasei list

Exemplu: lista participanților la un eveniment.

```python
p = ["Ana", "Bogdan", "Carla"]
p.append("Dan")              # ['Ana', 'Bogdan', 'Carla', 'Dan']
p.insert(1, "Elena")         # ['Ana', 'Elena', 'Bogdan', 'Carla', 'Dan']
x = p.pop(2)                 # x = 'Bogdan', p = ['Ana', 'Elena', 'Carla', 'Dan']
print(p.pop(), p)            # Dan ['Ana', 'Elena', 'Carla']
print(p.index("Carla"))      # 2
p.append("Ana")
print(p.count("Ana"))        # 2
p.remove("Ana")              # șterge PRIMA apariție: ['Elena', 'Carla', 'Ana']
p.extend(["Filip", "Gelu"]) # ['Elena', 'Carla', 'Ana', 'Filip', 'Gelu']
p.reverse()                  # ['Gelu', 'Filip', 'Ana', 'Carla', 'Elena']
```

| Metodă | Efect asupra listei | Întoarce | Capcană |
|---|---|---|---|
| `append(x)` | adaugă `x` la sfârșit | `None` | `append([1, 2])` adaugă **o** listă, nu două elemente |
| `extend(b)` | adaugă toate elementele lui `b` | `None` | ca `a += b` |
| `insert(i, x)` | pune `x` pe poziția `i`, restul se mută la dreapta | `None` | `i` mai mare decât lungimea = la sfârșit |
| `pop()` / `pop(i)` | scoate ultimul / elementul de pe poziția `i` | elementul scos | listă vidă sau `i` invalid: `IndexError` |
| `remove(x)` | șterge **prima** apariție a lui `x` | `None` | `x` lipsă: `ValueError` |
| `index(x)` | — | prima poziție a lui `x` | `x` lipsă: `ValueError` |
| `count(x)` | — | de câte ori apare `x` | `0` dacă lipsește (fără eroare) |
| `reverse()` | inversează ordinea pe loc | `None` | nu sortează! |
| `sort()` | sortează pe loc | `None` | vezi secțiunea 5 |
| `copy()` | — | o listă nouă, cu aceleași elemente | vezi secțiunea 6 |
| `clear()` | golește lista | `None` | lista rămâne, dar e `[]` |

**LA BAC**

Înainte de `index(x)` sau `remove(x)`, verifici `if x in a:`. Altfel, pentru o valoare care lipsește, programul se oprește cu `ValueError` (la `p.index("Zoe")`: `'Zoe' is not in list`).

## 5. sort() sau sorted()?

```
preturi = [45, 12, 99, 30]
r = preturi.sort()
print(r, preturi)              # None [12, 30, 45, 99]
preturi.sort(reverse=True)
print(preturi)                 # [99, 45, 30, 12]

v = [45, 12, 99, 30]
s = sorted(v)
print(v, s)                    # [45, 12, 99, 30] [12, 30, 45, 99]
```

| | `a.sort()` | `sorted(a)` |
|---|---|---|
| ce este | metodă a listei | funcție predefinită |
| modifică `a`? | **da**, pe loc | nu |
| întoarce | `None` | o listă nouă, sortată |
| când îl folosești | nu mai ai nevoie de ordinea inițială | vrei și ordinea inițială, și cea sortată |

Ambele acceptă `reverse=True` (descrescător) și `key=` (criteriul): `sorted(["pere", "kiwi", "banane"], key=len)` sortează după lungime. Metodele de sortare „de mână” le faci în lecția py9-17.

**ATENȚIE — GREȘELI FRECVENTE**

`a = a.sort()` distruge lista: `sort()` întoarce `None`, iar `a` devine `None`. Scrii fie `a.sort()`, fie `a = sorted(a)`. La fel, `reverse()` doar inversează: `[45, 12, 99, 30]` devine `[30, 99, 12, 45]`, nu o listă descrescătoare.

## 6. Atribuire sau copie?

`b = a` **nu copiază** lista. Dă încă un nume **aceleiași** liste (*aliasing*). Orice modificare făcută prin `b` se vede și prin `a`.

```
a = [7, 8, 9]
b = a
b.append(10)
print(a, b, a is b)          # [7, 8, 9, 10] [7, 8, 9, 10] True

c = a.copy()                 # sau c = a[:]  sau c = list(a)
c[0] = 0
print(a, c, a is c)          # [7, 8, 9, 10] [0, 8, 9, 10] False
```

`is` verifică dacă două nume arată spre **același obiect**; `==` verifică dacă au **aceleași elemente**. Același efect apare la funcții: o funcție care primește o listă și o modifică (`lista[i] += 1`) schimbă lista apelantului (vezi lecția despre subprograme). Dacă vrei să păstrezi originalul, trimiți o copie.

> **DE ȘTIUT**
>
> În C++, `vector<int> b = a;` copiază. În Python, `b = a` nu copiază. Este cea mai frecventă capcană de tranziție la liste.

## 7. Verifică-te

1. Ce afișează `print([3, 1] + [2] * 2)`?
2. Ce este `[5, 1] < [4, 9, 9]`: `True` sau `False`? De ce?
3. Ce afișează `a = [1, 2, 3, 2]`, `a.remove(2)`, `print(a, a.count(2))`?
4. Ce conține `a` după `a = [4, 6]`, `a.insert(0, 5)`, `a.append(a.pop(1))`?
5. Pentru `a = [10, 20, 30, 40]`, ce dă `a[1:3]` și ce dă `a[::-1][0]`?
6. `x = [2, 1]`, `y = x`, `y.sort()`. Ce conține `x`?
7. De ce `t = [3, 1, 2].sort()` și apoi `print(t[0])` dă eroare?

### Răspunsuri

1. `[3, 1, 2, 2]`.
2. `False`: decide primul element, iar `5 > 4`.
3. `[1, 3, 2] 1` (se șterge doar prima apariție).
4. `[5, 6, 4]`: după `insert` lista e `[5, 4, 6]`, `pop(1)` scoate `4`, care se adaugă la sfârșit.
5. `[20, 30]` și `40`.
6. `[1, 2]`: `y` și `x` sunt aceeași listă.
7. `sort()` întoarce `None`, deci `t` este `None`, iar `None[0]` dă `TypeError`. Corect: `t = sorted([3, 1, 2])`.

## Probleme de lucru în clasă (pbinfo)

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

- [ ] **#1452** **stergere-element** — ușoară
  https://www.pbinfo.ro/probleme/1452/stergere-element
- [ ] **#158** **inserare** — ușoară
  https://www.pbinfo.ro/probleme/158/inserare
- [ ] **#488** **afisare** — ușoară
  https://www.pbinfo.ro/probleme/488/afisare

## Temă pentru acasă (pbinfo)

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

- [ ] **#500** **verificareordonat** — ușoară
  https://www.pbinfo.ro/probleme/500/verificareordonat

- [ ] #505 **identice** — ușoară
  https://www.pbinfo.ro/probleme/505/identice
- [ ] #162 **permcirc** — medie
  https://www.pbinfo.ro/probleme/162/permcirc

---