INFORMATICĂ · PYTHON · CLASA A IX-A · PRINCIPII DE ELABORARE A UNUI PROGRAM · LECȚIA 2

De la pseudocod (sau C++) la Python

Durată: 100 min Nivel: mate-info · Python Înainte: lecțiile py9-01

CE VEI ȘTI DUPĂ LECȚIE

- Traduci în Python orice algoritm cu citire, afișare, decizii și repetiții pe care îl știi din gimnaziu.
- Folosești corect input(), conversiile int()/float(), print() și f-string-urile.
- Recunoști cele 5 diferențe care îi încurcă pe cei care vin din C++: indentarea, range(), //, % la negative și tipul text al lui input().

1. Ce rămâne la fel: algoritmul

În gimnaziu ai învățat să gândești un algoritm: date de intrare, pași, decizii, repetiții, rezultat. **Partea asta nu se schimbă.** În Python se schimbă doar felul în care o scrii. Tabelul de mai jos este „dicționarul" lecției: păstrează-l la îndemână.

Ce vrei să faci	Pseudocod	C++	Python
citești un număr	`citește x`	`cin >> x;`	`x = int(input())`
afișezi	`scrie x`	`cout << x;`	`print(x)`
atribui	`x ← x + 1`	`x = x + 1;`	`x = x + 1` sau `x += 1`
decizie	`dacă x > 0 atunci … altfel …`	`if (x > 0) {…} else {…}`	`if x > 0:` … `else:` …
și / sau / nu	`și`, `sau`, `nu`	`&&`, `||`, `!`	`and`, `or`, `not`
cât timp	`cât timp n > 0 execută`	`while (n > 0) {…}`	`while n > 0:`
pentru	`pentru i ← 1, n execută`	`for (i = 1; i <= n; i++)`	`for i in range(1, n + 1):`
câtul întreg, restul	`[a / b]`, `a % b`	`a / b`, `a % b` (la `int`)	`a // b`, `a % b`
comentariu		`// text`	`# text`

2. Primul program: input() citește TEXT

```
a = int(input())       # citește un rând și îl transformă în număr întreg
b = int(input())
print(a + b)
```

> **ATENȚIE — GREȘELI FRECVENTE**
>
> `input()` întoarce mereu un **șir de caractere**. Fără `int(...)`, `a + b` lipește textele: `"2" + "3"` dă `"23"`, nu `5`. Pentru numere reale folosești `float(input())`.

Mai multe numere pe același rând (de exemplu `12 7`):

```
a, b = map(int, input().split())   # split() taie rândul la spații, map(int, …) le convertește
```

Citirea în stil concurs / pbinfo. Datele vin în formate fixe, iar programul trebuie să le citească exact așa. Cele patru tipare de mai jos acoperă aproape toate enunțurile:

Formatul datelor	Cod
un număr pe rând	`n = int(input())`
valorile pe un singur rând (numărul lor nu se citește)	`v = list(map(int, input().split()))`
pe primul rând `n`, pe al doilea cele `n` valori	`n = int(input())` ↵ `v = list(map(int, input().split()))`
număr necunoscut de valori, pe oricâte rânduri	`import sys` ↵ `v = list(map(int, sys.stdin.read().split()))`

Ultimul tipar este „citirea până la sfârșitul datelor” (EOF): `sys.stdin.read()` ia **tot** ce a primit programul, iar `split()` îl taie la spații și la rânduri noi. Pentru intrarea `1 2 3`, `4 5`, `6` (pe trei rânduri) obții lista `[1, 2, 3, 4, 5, 6]`. Varianta rând cu rând: `for linie in sys.stdin:`. Mai există și `while True:` cu `input()` într-un `try … except EOFError: break`, dar `try` îl explicăm abia în lecția py9-13, deci deocamdată folosește `sys.stdin`.

La **ieșire**, evaluatorul compară textul exact, deci contează spațiile și rândurile noi: `print(*v)` afișează valorile separate prin spațiu, pe un rând; `print(*v, sep="\n")`, câte una pe rând; `print(s, end="")` nu trece la rândul următor. Nu afișa mesaje de tipul `"Introdu n:"`, pentru că nu fac parte din ieșirea cerută.

3. Indentarea este sintaxă

În Python nu există acolade. **Un bloc este format din rândurile indentate la fel** (de regulă 4 spații), după un rând care se termină cu `:`.

```
x = int(input())
if x > 0:
    print("pozitiv")
elif x == 0:
    print("zero")
else:
    print("negativ")
print("gata")          # neindentat: se execută mereu
```

`elif` înseamnă „altfel, dacă”. Poți avea oricâte `elif`-uri. Python acceptă și comparații înlănțuite, ca la matematică: `10 <= x <= 20` este corect în Python, dar **greșit în C++**.

4. Repetiții

`while` se scrie exact ca în pseudocod. Exemplu: suma cifrelor.

```python
n = int(input())
s = 0
while n > 0:
    s += n % 10
    n //= 10
print(s)              # pentru 9081 afișează 18
```

`for` parcurge valorile produse de `range()`. **Capătul din dreapta nu este inclus.**

`range(...)`	valori
`range(5)`	0, 1, 2, 3, 4
`range(2, 10, 3)`	2, 5, 8
`range(10, 0, -3)`	10, 7, 4, 1
`range(5, 5)`	nicio valoare

Ca să parcurgi `1, 2, …, n`, scrii `range(1, n + 1)`. `break` și `continue` funcționează ca în C++.

5. Operatori: trei diferențe față de C++

Expresie	Python	C++ (la `int`)	De reținut
`10 / 4`	`2.5`	`2`	`/` dă mereu rezultat real
`10 // 4`	`2`	(nu există)	câtul întreg
`-7 // 2`	`-4`	`-3` (la `-7 / 2`)	Python rotunjește **în jos**
`-7 % 2`	`1`	`-1`	în Python restul are semnul împărțitorului
`2 ** 100`	`1267650600228229401496703205376`	depășire	întregii nu au limită

La BAC și la probleme lucrezi de obicei cu numere naturale, deci diferența la negative apare rar. Totuși, ține-o minte.

6. Afișarea frumoasă

```python
x = 7
print(1, 2, 3, sep="-", end="!\n")     # 1-2-3!
print(f"{x} la pătrat = {x ** 2}")     # 7 la pătrat = 49
print(f"{2/3:.2f}")                    # 0.67 (două zecimale)
```

`sep` este separatorul dintre valori (implicit un spațiu), iar `end` este ce se afișează la final (implicit un rând nou).

7. Capcanele celor care vin din C++

Ce scrii	Ce se întâmplă	Corect
`x = input()` și apoi `x + 1`	eroare: text + număr	`x = int(input())`
`for i in range(1, n):`	ultima valoare este `n - 1`	`range(1, n + 1)`
`if x > 0` (fără `:`)	eroare de sintaxă	`if x > 0:`
amesteci tab-uri cu spații	`IndentationError`	folosește doar 4 spații
`i++`	eroare	`i += 1`
`x = 7 / 2` când vrei câtul	`3.5`	`7 // 2`

DE ȘTIUT

PEP 8 este ghidul de stil al limbajului Python. Programul merge și fără el, dar codul îngrijit se citește (și se depanează) mai ușor. Regulile de început:

- **nume**: litere mici și `_` pentru variabile și funcții (`suma_cifrelor`, `nr_pare`), nu `SumaCifrelor`; nume care spun ce conțin, nu `a1`, `x2` (excepție: `i`, `j` ca index în bucle și `n`, `a`, `b` în probleme matematice scurte);
- **spații**: în jurul operatorilor și după virgulă (`a = b + 1`, `f(x, y)`), dar nu în interiorul parantezelor (`f(x, y)`, nu `f( x,y )`) și nu înainte de `:`;
- **indentare**: exact 4 spații pe nivel, niciodată tab-uri amestecate cu spații;
- **un rând, o instrucțiune**: fără `;` și fără `if x > 0: print(x)` pe același rând; rândurile se țin sub aproximativ 80 de caractere; două rânduri goale între funcții.

8. Verifică-te

1. Ce afișează `print("5" + "5", int("5") + 5)`?
2. Câte valori produce `range(3, 20, 4)` și care sunt ele?
3. Scrie în Python: `pentru i ← n, 1, −1 execută scrie i`.
4. Ce afișează `print(-9 // 4, -9 % 4)`?
5. Rescrie fără `and`: `x >= 1 and x <= 9`.
6. Ce este greșit în programul următor?

```
n = int(input())
for i in range(n):
print(i)
```

Răspunsuri

1. `55 10`.
2. 5 valori: 3, 7, 11, 15, 19.
3. `for i in range(n, 0, -1): print(i)`.
4. `-3 3`: `-9 // 4 = -3` (−2,25 rotunjit în jos), iar `-3 · 4 + 3 = -9`.

5. `1 <= x <= 9`.
6. `print(i)` nu este indentat, deci nu face parte din `for`: Python dă `IndentationError`.

> **SFAT**
>
> Pe pbinfo, la trimiterea unei soluții, alege limbajul **Python**. Problemele de mai jos le-ai mai văzut poate în C++: rescrie-le în Python și compară cele două variante.

Probleme de lucru în clasă (pbinfo)

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

- [ ] `#939` **sum00** — ușoară
  https://www.pbinfo.ro/probleme/939/sum00
- [ ] `#105` **max2** — ușoară
  https://www.pbinfo.ro/probleme/105/max2
- [ ] `#51` **suma** — ușoară
  https://www.pbinfo.ro/probleme/51/suma

Temă pentru acasă (pbinfo)

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

- [ ] `#109` **paritate** — ușoară
  https://www.pbinfo.ro/probleme/109/paritate
- [ ] `#2` **suma-n** — ușoară
  https://www.pbinfo.ro/probleme/2/suma-n
- [ ] `#10` **suma-cifrelor** — ușoară
  https://www.pbinfo.ro/probleme/10/suma-cifrelor