INFORMATICĂ · PYTHON · CLASA A IX-A · PRINCIPII DE ELABORARE A UNUI PROGRAM · LECȚIA 1

Gândire computațională și etapele unui program

Durată: 100 min Nivel: mate-info · Python

CE VEI ȘTI DUPĂ LECȚIE

- Parcurgi pe o problemă scurtă toate etapele: analiză, proiectare, implementare, testare și depanare.
- Reprezinți același algoritm în blocuri grafice, în pseudocod și în Python și explici avantajele și limitele fiecărei forme.
- Explici diferența dintre un limbaj cu interpretor și unul cu compilator.
- Rulezi primul program Python, atât în modul interactiv (>>>), cât și dintr-un fișier .py.

1. Gândire computațională: cum privește un informatician o problemă

Programarea nu începe la tastatură. Începe cu felul în care **gândești problema**, astfel încât s-o poată rezolva un calculator. Asta numim **gândire computațională**. Are patru componente pe care le folosești deja, poate fără să le dai un nume.

Componenta	Ce înseamnă	Exemplu: organizezi o excursie
descompunere	spargi problema în subprobleme	transport, cazare, buget
recunoașterea tiparelor	observi ce seamănă cu ceva rezolvat	„câte autobuze?” e ca „câte cutii?”
abstractizare	păstrezi doar datele care contează	numărul elevilor, nu numele lor
algoritm	scrii pașii exacți, fără ambiguități	pașii de calcul de mai jos

2. Etapele elaborării unui program

Lucrăm pe o problemă reală: **la o excursie merg `n` elevi, iar un autobuz are 50 de locuri. De câte autobuze este nevoie?**

Etapa	Întrebarea ta	Ce produci
1. Analiză	Ce primesc? Ce obțin? Ce condiții sunt?	intrare: `n` natural; ieșire: numărul de autobuze; un autobuz parțial plin contează
2. Proiectare	Ce pași fac?	algoritmul: câtul `n / 50`, plus 1 dacă rămâne rest
3. Implementare	Cum scriu pașii în limbaj?	programul Python
4. Testare	Merge pe toate cazurile?	tabel de teste: intrare, așteptat, obținut

Etapa	Întrebarea ta	Ce produci
5. Depanare	Unde e greșeala?	programul corectat și **retestat**

Etapele nu sunt o linie dreaptă, ci un **ciclu**: o eroare găsită la testare te poate întoarce la proiectare sau chiar la analiză. Diagrama lor (după testare te întorci la etapa în care e greșeala):

```
analiză → proiectare → implementare → testare → gata
   ↑           ↑             ↑           |
   └───────────┴─────────────┴───────────┘ greșeală găsită
   înțelegere  algoritm      cod (depanare)
```

Datele unui algoritm. În analiză împarți datele în trei feluri: **de intrare** (le primește algoritmul), **de ieșire** (rezultatul cerut) și **de manevră** (variabile ajutătoare, create doar pe parcurs). La suma cifrelor lui `n`, `n` este de intrare, `s` de ieșire, iar `c` (o copie a lui `n`, din care „tăiem" cifre) este de manevră:

```python
n = int(input())      # intrare
c = n                 # manevră: n rămâne neschimbat
s = 0                 # ieșire
while c > 0:
    s += c % 10
    c //= 10
print(s)              # pentru 4821 afișează 15
```

Datele de manevră nu apar în enunț, dar fără ele algoritmul nu merge sau strică datele de intrare.

Depanare, pe viu. Prima variantă scrisă de un elev a fost `print(n // 50)`. Iată testele:

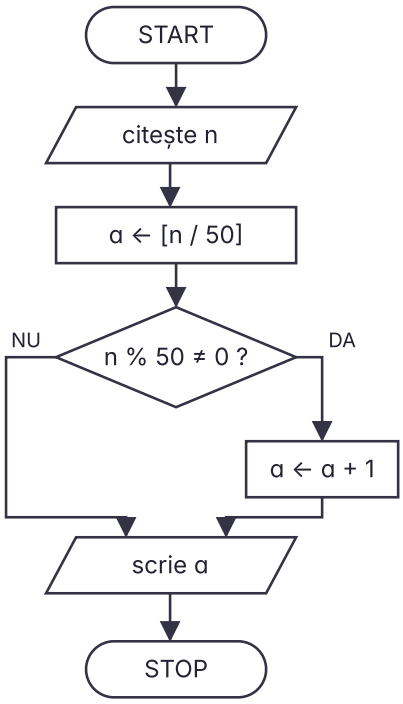
`n`	așteptat	`n // 50`	corect?
100	2	2	da
51	2	1	**nu**
1	1	0	**nu**
0	0	0	da

Cu un singur test (`100`) programul părea corect. Testele **51** și **1** arată că lipsește cazul cu rest. Eroarea nu ține de Python, ci de **proiectare**: în analiză s-a pierdut observația despre autobuzul pe jumătate gol.

> **SFAT**
>
> Alege testele înainte să scrii codul: un caz obișnuit, cazurile de la limită (`50`, `51`, `1`, `0`) și, dacă se poate, o intrare greșită. Criteriile de construire a testelor le facem pe larg în lecția py9-03.

3. Trei moduri de a reprezenta același algoritm

Algoritmul corect pentru autobuze, scris în trei forme:




```
citește n
a ← [n / 50]
dacă n % 50 ≠ 0 atunci
    a ← a + 1
scrie a
```

```python
n = int(input())      # numărul de elevi
a = n // 50           # autobuze pline
if n % 50 != 0:       # au rămas elevi?
    a += 1
print(a)
```

Reprezentare	Avantaje	Limite
blocuri grafice (schemă logică)	se vede imediat drumul execuției; bună pentru decizii și bucle mici	ocupă mult loc; greu de desenat și de modificat la algoritmi mari; nu se poate rula
pseudocod	aproape de limba română; te concentrezi pe idee, nu pe reguli de scriere; formatul folosit la Bac	nu există o formă unică; nu se poate rula, deci nu se poate testa automat
limbaj de programare (Python, C++)	se rulează și se testează; aceeași descriere e înțeleasă de om și de calculator	reguli stricte de scriere; o virgulă greșită oprește programul

LA BAC

La Bac primești algoritmi în **pseudocod** și trebuie să-i urmărești, să-i completezi sau să-i scrii în limbajul studiat. Exersează traducerea în ambele sensuri: pseudocod → Python și Python → pseudocod.

4. Interpretor sau compilator?

Calculatorul nu înțelege direct textul programului. Un alt program îl traduce: un **compilator** sau un **interpretor**.

	Compilator (de exemplu C++)	Interpretor (de exemplu Python)
cum lucrează	traduce **tot** programul într-un fișier executabil, apoi îl rulezi	citește și execută programul **instrucțiune cu instrucțiune**

	Compilator (de exemplu C++)	Interpretor (de exemplu Python)
când afli de erorile de scriere	înainte de rulare, pe toate deodată	erorile de sintaxă tot la pornire; celelalte abia când se ajunge la rândul greșit
viteză	de obicei mai rapid	de obicei mai lent
comoditate	compilezi la fiecare modificare	rulezi imediat; poți încerca o singură instrucțiune (`>>>`)

Granița nu e absolută (Python își traduce întâi programul într-o formă intermediară, *bytecode*), dar pentru tine contează ideea: în Python scrii și rulezi direct.

5. Primul contact cu Python

Ai nevoie de un mediu de lucru: **Thonny** (cel mai simplu la început), **IDLE** (vine cu Python), **PyCharm** sau **Visual Studio Code**. Fără instalare: OnlineGDB sau Programiz, iar **Python Tutor** îți arată pas cu pas valorile variabilelor.

Modul interactiv (`>>>` , numit și *shell* sau *consolă*): scrii o instrucțiune, apeși Enter, vezi imediat rezultatul. E perfect pentru încercări rapide.

```
>>> 2 + 3
5
>>> 2 ** 100
1267650600228229401496703205376
>>> print("Salut, Python!")
Salut, Python!
```

Modul program (script): scrii instrucțiunile într-un fișier, de exemplu `autobuze.py` , îl salvezi și îl rulezi (în Thonny: butonul verde *Run* sau tasta F5). Programul așteaptă să introduci `n` , apoi afișează rezultatul. Pentru `51` afișează `2` .

Prima eroare. Python face diferența între litere mari și mici:

```
>>> Print("Salut")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    Print("Salut")
    ^^^^^
NameError: name 'Print' is not defined. Did you mean: 'print'?
```

Citește mesajul de eroare până la capăt: îți spune **rândul**, **tipul** erorii și, de multe ori, chiar soluția. Tipurile de erori (de sintaxă, de execuție, de logică) le facem în lecția py9-03.

6. Cum comunică un program cu utilizatorul

Programul primește date și dă rezultate. Programa de clasa a IX-a cere trei moduri de comunicare; aici doar le prezentăm, pe fiecare îl lucrăm separat mai târziu.

Mod	Cum arată	Când e potrivit	În Python
consolă (interfață text)	rânduri de text: scrii, programul răspunde	exerciții, probleme pbinfo, programe rapide	`input()`, `print()` (lecția py9-02)
interfață grafică	ferestre, butoane, etichete, casete text	aplicații pentru oricine, nu doar pentru programatori	biblioteca **Tkinter**
fișiere	datele sunt citite dintr-un fișier și scrise în altul	date multe, date care trebuie păstrate după închiderea programului	`open()`, `read()`, `write()`, `close()`

Același algoritm (de exemplu, cel cu autobuzele) poate fi folosit cu oricare dintre ele. Se schimbă doar „ambalajul": de unde vine `n` și unde apare rezultatul.

7. Verifică-te

1. În ce etapă stabilești că `n` este un număr natural și că rezultatul este tot un număr natural?
2. Un program trece testul din enunț. Înseamnă că este corect? Dă un exemplu din lecție.
3. Numește un avantaj al schemei logice față de pseudocod și un dezavantaj.
4. De ce poți încerca în Python o singură instrucțiune (`>>> 2 + 3`), dar în C++ nu, în mod obișnuit?
5. Ce afișează programul `autobuze.py` pentru `n = 150`? Dar pentru `n = 149`?
6. O aplicație de catalog trebuie să păstreze notele și după ce e închisă. Ce mod de comunicare din secțiunea 6 este necesar?

Răspunsuri

1. La **analiză** (datele de intrare și de ieșire și condițiile lor).
2. Nu. Varianta `print(n // 50)` trece testul `100`, dar greșește pentru `51` și pentru `1`.
3. Avantaj: se vede imediat drumul execuției. Dezavantaj: ocupă mult loc și e greu de modificat la algoritmi mari.
4. Python are **interpretor**, care execută instrucțiunile pe rând, deci poate executa și una singură. C++ se **compilează** ca program întreg înainte de rulare.
5. `3` pentru 150 (exact 3 autobuze pline) și tot `3` pentru 149 (2 pline și unul parțial).
6. **Fișierele** (datele scrise în fișier rămân după închiderea programului).

Probleme de lucru în clasă (pbinfo)

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

- [ ] `#941` **urare** — ușoară
  https://www.pbinfo.ro/probleme/941/urare
- [ ] `#103` **curte** — ușoară
  https://www.pbinfo.ro/probleme/103/curte

Temă pentru acasă (pbinfo)

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

- [ ] #182 **cub** — ușoară
  https://www.pbinfo.ro/probleme/182/cub
- [ ] #1260 **asii** — ușoară
  https://www.pbinfo.ro/probleme/1260/asii