test-e.ro

CAIET ELEV

Lecția 6

Gândire computațională și reprezentarea algoritmilor

*Un pas „de sus" peste tot ce ai învățat: cum **gândești** o problemă înainte de a scrie cod. Cei patru piloni ai gândirii computaționale, cele cinci etape ale unui program, datele unei probleme, cele trei reprezentări (schemă · pseudocod · Python) și algoritmul lui **Euclid**.*

DISCIPLINA Informatică **CLASA** a IX-a · mate-info **LIMBAJ** Python 3

NUMELE ELEVULUI .....

CUM FOLOSEȘTI CAIETUL

Acest caiet adună exercițiile scrise din Lecția 6. Rezolvă-le pe hârtie — descoperi ce problemă rezolvă un algoritm, potrivești situații cu pilonii gândirii, ordonezi etape, clasifici datele, desenezi scheme logice, urmărești algoritmul lui Euclid și scrii programe. Folosește ce s-a predat până acum (lecțiile 1–5): structura **secvențială**, **decizia** (**if · elif · else**), repetiția (**while**, **for**) și operațiile cu cifre (**% 10**, **// 10**). **Fără** funcții (**def/return**): modularitatea se marchează prin **comentarii**. Spațiile punctate și liniate sunt pentru scrisul tău; verificarea automată rămâne pe platformă.

CUPRINS

1 Recapitulare: gândește codul .....
2 Gândirea computațională .....
3 Etapele elaborării unui program .....
4 Datele unei probleme .....
5 Trei reprezentări ale unui algoritm .....
6 Gândirea modulară · Euclid .....
7 Verifică-ți codul .....

SECȚIUNEA 1 · RECAPITULARE

1 Ce știm deja: *gândește codul*

În lecțiile 1–5 ai învățat cele trei structuri de control — `secvențială`, `de decizie` (`if…elif…else`) și `repetitivă` (`while`, `for`) — și operațiile cu cifrele unui număr (`% 10`, `// 10`). Recapitularea de azi nu-ți cere să reciți reguli, ci **să gândești cod**: descoperi ce problemă rezolvă un algoritm, depistezi unde greșește raționamentul cuiva și scrii singur o soluție.

A EXERCIȚIUL A · RECAPITULARE

Bazele, pe scurt

GRILĂ COMPLETARE

Încercuiește `True` / `False` sau completează pe linia punctată.

1. Structura de **decizie** alege între variante în funcție de o condiție.

True False

2. Ultima cifră a lui `n` se obține cu operația _____

3. `n // 10` elimină ultima cifră a lui `n` . True False

4. O cifră `c` este impară dacă `c % 2` este egal cu _____

B EXERCIȚIUL B · GĂSEȘTE ÎNTREBAREA

Ce problemă rezolvă codul?

GRILĂ

De obicei pornești de la o problemă și scrii codul. Aici mergi **invers**: citești algoritmul și descoperi ce problemă rezolvă. Bifează enunțul corect. Atenție la capcane — unele variante seamănă, dar codul face altceva.

```
citește n
s ← 0
cât timp n > 0 execută
    s ← s + n % 10
    n ← n // 10
scrie s
```

_____ *a)* Afișează numărul de cifre ale lui `n` .

_____ *b)* Afișează suma cifrelor lui `n` .

_____ *c)* Afișează numărul `n` scris cu cifrele în ordine inversă.

HINT *execută-l în minte pentru* `n = 43`*: ce se adună la* `s` *la fiecare pas?*

C

EXERCIȚIUL C · REPARĂ RAȚIONAMENTUL

Unde a greșit colegul?

DRY-RUN

Un coleg a executat „în minte" algoritmul de mai jos pentru `n = 472` (numără cifrele lui `n`) și a completat tabelul — dar **rezultatul lui e greșit**. Recompletează tu tabelul corect și încercuiește rândul la care apare prima eroare de raționament.

```
citește n
nr ← 0
cât timp n > 0 execută
    nr ← nr + 1
    n ← n // 10
scrie nr
```

Pas (trecere prin buclă)	n (colegul)	nr (colegul)	Corect? scrie tu
1	47	1	
2	4	2	
3	0	4	

HINT *la fiecare trecere,* `nr` *crește cu exact 1. Câte treceri face bucla pentru un număr de 3 cifre?*

D

SCRIE SINGUR · CU WHILE

Oglinditul din cifrele impare

SCRIERE COD

Se citește un număr natural `n`. Construiește și afișează numărul format **doar din cifrele impare** ale lui `n`, luate de la ultima spre prima (oglindit). Dacă `n` nu are nicio cifră impară, afișează `0`. Exemplu: `n = 13245` → `531`.

HINT *parcurge cifrele cu* `% 10` *și* `// 10`*. Când o cifră e impară (*`cifra % 2 != 0`*), adaug-o la dreapta rezultatului:* `rezultat = rezultat * 10 + cifra`*.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · CONCEPTE

2 Gândirea *computațională*

Înainte de a scrie cod, un programator **gândește** problema. Gândirea computațională înseamnă să o abordezi în patru moduri complementare. Le urmărim pe un fir concret: **organizarea unui mic turneu sportiv la clasă**.

1 · DESCOMPUNERE

Împarți în subprobleme

Spargi problema mare în bucăți mici, ușor de rezolvat separat.

Turneu → înscrierea echipelor · programarea meciurilor · calculul clasamentului.

2 · RECUNOAȘTEREA TIPARELOR

Ce se repetă?

Observi ce se repetă, ca să tratezi cazurile la fel.

Fiecare meci se punctează la fel: 3p victorie, 1p egal, 0p înfrângere.

3 · ABSTRACTIZARE

Păstrezi doar ce contează

Ignori detaliile nesemnificative pentru problemă.

Pentru clasament contează scorul, nu culoarea tricourilor.

4 · ALGORITMIZARE

Pașii, în ordine

Descrii pașii precișii care duc de la date la rezultat.

„Pentru fiecare meci, adună punctele; apoi ordonează echipele."

CUM LUCREAZĂ ÎMPREUNĂ

Cei patru piloni nu se folosesc pe rând, ci împreună. **Descompunerea** îți dă subproblemele, **tiparele** îți arată ce se repetă, **abstractizarea** curăță zgomotul datelor nesemnificative, iar **algoritmul** pune totul în pași executabili.

A

EXERCIȚIUL A · POTRIVIRE

Ce pilon folosești aici?

ASOCIERE

Pentru fiecare situație, scrie pe linia punctată pilonul potrivit: **D** (descompunere), **T** (recunoașterea tiparelor), **A** (abstractizare) sau **AI** (algoritmizare).

1. Împarți „gătitul cinei" în: cumpărături · pregătire · fierbere · servire.
2. Observi că fiecare rând al unui tabel se prelucrează la fel. _____
3. Pentru un bilet de tren păstrezi doar plecarea și destinația, ignori culoarea vagonului.
4. Scrii pas cu pas rețeta prin care afli media a trei note. _____

B

EXERCIȚIUL B · APLICĂ

Descompune o problemă a ta

COMPLETARE

Problema: **„afișează media notelor unui elev și mențiunea promovat / nepromovat"**. Descompune-o în cel puțin trei subprobleme (pași mari), apoi numește **o dată** nesemnificativă pe care abstractizarea o poate ignora.

SUBPROBLEMELE (DESCOMPUNERE)

→ Detaliu ignorat prin abstractizare:

SECȚIUNEA 3 · CONCEPTE

3 Etapele *elaborării* unui program

Un program bun nu se scrie dintr-o suflare. Trece prin cinci etape, mereu în aceeași ordine. Le urmărim pe o problemă simplă: **calculează media unui elev la o disciplină**.

CEI CINCI PAȘI

1. **Analiză** — înțeleg ce se cere și ce date am. *„Am notele; vreau media aritmetică."*
2. **Proiectare** — gândesc pașii (pseudocod / schemă), fără cod încă. *„Citesc notele, le adun, împart la câte sunt."*
3. **Implementare** — scriu efectiv programul în Python. *Traduc pseudocodul în input(), calcule, print().*
4. **Testare** — verific pe cazuri, inclusiv cazuri-limită. *„Ce iese pentru note egale? Dar pentru o singură notă?"*
5. **Depanare** — caut și corectez erorile descoperite. *„Media iese greșit — am împărțit la numărul greșit de note?"*

A

EXERCIȚIUL A · ORDONARE

Ordonează etapele

ORDONARE

Numerotează etapele de la **1** la **5** în ordinea corectă a elaborării unui program. O etichetă este o **capcană** (nu e o etapă reală) — taie-o.

- Testare
- Analiză
- Implementare
- Depanare

Proiectare

Publicitate ← *capcană?*

B

EXERCIȚIUL B · ORDONARE

Un algoritm din viața ta

ORDONARE

Orice succesiune ordonată de pași este un algoritm. Mai jos sunt, amestecați, pașii pentru **trimiterea unei teme online pe platforma școlii**. Numerotează-i de la **1** la **5**. Un pas e o **capcană** — taie-l.

Apeși butonul „Trimite"

Te autentifici pe platformă

Atașezi fișierul temei

Deschizi tema (cerința) primită

Verifici mesajul de confirmare

Oprești calculatorul colegului ← *capcană?*

C

EXERCIȚIUL C · GĂSEȘTE GREȘEALA

Ordinea contează

DRY-RUN

Programul **ar trebui** să calculeze suma `1 + 2 + … + n`, dar dă un rezultat greșit. Nu are erori de scriere — problema e că **două instrucțiuni din corpul buclei sunt în ordine greșită**. Încercuiește cele două linii care ar trebui să-și schimbe locul.

```
n = int(input())
s = 0
i = 1
while i <= n:
    i = i + 1
    s = s + i
print(s)
```

HINT *fă un dry-run pentru `n = 3`: ce valoare are `i` chiar când o aduni la `s`? Ce ar trebui adunat, de fapt?*

CUM ARATĂ CORPUL BUCLEI CORECTAT

SECȚIUNEA 4 · CONCEPTE

4 Datele unei probleme: *intrare · ieșire · intermediară*

Orice problemă lucrează cu trei feluri de date. Le recunoști punând trei întrebări:

CELE TREI FELURI DE DATE

1. **Date de intrare** — ce **primește** programul (de obicei de la tastatură). Întrebarea: „ce știu de la început?"
2. **Date de ieșire** — ce **produce** programul ca rezultat. Întrebarea: „ce vreau să aflu / să afișez?"
3. **Date intermediare** — variabile **de lucru**, folosite temporar, care nu sunt nici intrare, nici ieșire. Întrebarea: „de ce ajutor am nevoie pe drum?"

EXEMPLU · BONUL DE CASĂ

Prețul și cantitatea fiecărui produs sunt **intrare**; totalul de plată este **ieșire**; subtotalul calculat pentru fiecare produs, folosit doar ca să ajungi la total, este o **dată intermediară**. La „demontarea" unui număr, `cifra = n % 10` e tot o dată intermediară — nu o citești, nu o afișezi, doar te ajută.

A

EXERCIȚIUL A · CLASIFICARE

Ce fel de dată este?

GRILĂ

Pentru fiecare problemă, bifează felul fiecărei variabile: **In** (intrare), **Ie** (ieșire) sau **It** (intermediară).

PROBLEMĂ / VARIABILĂ	ROLUL VARIABILEI	ALEGE
Ecuația de gradul 2 *a·x² + b·x + c = 0, rădăcini reale*	`a`, `b`, `c` (coeficienții)	In Ie It
	`delta` (discriminantul)	In Ie It
	`x1`, `x2` (rădăcinile)	In Ie It
Suma cifrelor *suma cifrelor unui număr n*	`n` (numărul citit)	In Ie It
	`cifra = n % 10`	In Ie It

B

EXERCIȚIUL B · IDENTIFICARE

Bancnotele pentru o sumă

COMPLETARE

Problema: **se citește o sumă în lei; se află câte bancnote de 100, 50 și 10 trebuie date și cât rămâne**. Scrie fiecare variabilă în coloana potrivită.

INTRARE	IEȘIRE	INTERMEDIARĂ

HINT *„suma" o citești (intrare); „câte bancnote de fiecare fel" e rezultatul (ieșire); „cât a mai rămas de plătit după fiecare tip" te ajută pe parcurs (intermediară).*

5 SECȚIUNEA 5 · CONCEPTE

Trei feluri de a *reprezenta* același algoritm

Un algoritm poate fi descris în trei moduri — **schemă logică**, **pseudocod** și **cod Python** — aceeași idee, trei limbaje. Schema se citește ușor, pseudocodul e independent de limbaj, iar Python rulează efectiv. Le legăm pe o problemă simplă: **minimul a trei numere** `a`, `b`, `c`.

PSEUDOCOD

```
citește a, b, c
m ← a
dacă b < m atunci
    m ← b
dacă c < m atunci
    m ← c
scrie m
```

PYTHON

```
a = int(input())
b = int(input())
c = int(input())
m = a
if b < m:
    m = b
if c < m:
    m = c
print(m)
```

A

EXERCIȚIUL A · CONCEPEREA SCHEMEI

Schema minimului a trei numere

SCHEMĂ LOGICĂ

Desenează schema logică a pseudocodului de mai sus. Reține formele: **oval** = START/STOP, **paralelogram** = citire/afișare, **dreptunghi** = calcul/atribuire, **romb** = decizie. Fiecare romb are **două ieșiri**: ramura **DA** și ramura **NU**.

DESENEAZĂ SCHEMA AICI · NU UITA CELE DOUĂ ROMBURI ȘI RAMURILE DA / NU

B

EXERCIȚIUL B · POTRIVIRE

Pseudocod ↔ Python

ASOCIERE

Fiecare linie de pseudocod (stânga) are un echivalent Python (dreapta). Trage o linie între perechile care se potrivesc.

PSEUDOCOD		PYTHON	
1.	`citește a`	*a)*	`print(m)`
2.	`m ← a`	*b)*	`m = a`
3.	`dacă b < m atunci`	*c)*	`a = int(input())`
4.	`scrie m`	*d)*	`if b < m:`

→ Perechi: 1– ——— 2– ——— 3– ——— 4– ———

C

EXERCIȚIUL C · COMPLETEAZĂ COD

Completează programul

COMPLETARE

Programul află minimul a trei numere, dar îi lipsesc **patru piese** (spațiile marcate). Scrie ce trebuie să apară în fiecare.

```
a = int(input())
b = int(input())
c = int(input())
m = ____________________
if b < m ____________________
    m = ____________________
if c < m:
    m = c
print(            )
      ____________________
```

CELE PATRU PIESE, ÎN ORDINE

6 SECȚIUNEA 6 · CONCEPTE

Gândirea *modulară* și algoritmul lui Euclid

A gândi **modular** înseamnă a rezolva o problemă împărțind-o în pași clari, fiecare cu rolul lui. La clasa a IX-a nu folosim încă funcții (`def`) — marcăm pașii cu **comentarii**. Exemplu: „simplifică o fracție" = (1) află cel mai mare divizor comun (CMMDC) al numărătorului și numitorului; (2) împarte ambii la el.

ALGORITMUL LUI EUCLID (SCĂDERI REPETATE)

Ideea: **cât timp cele două numere sunt diferite, scade-l pe cel mic din cel mare**. Când devin egale, acea valoare este CMMDC-ul.

```
citește a, b
cât timp a ≠ b execută
    dacă a > b atunci
        a ← a - b
    altfel
        b ← b - a
scrie a
```

Un pic de istorie: algoritmul e descris de matematicianul grec **Euclid** în *Elemente* (sec. al III-lea î.Hr.) — unul dintre cei mai vechi algoritmi cunoscuți, folosit și azi aproape neschimbat.

A

EXERCIȚIUL A · DRY-RUN

Urmărește-l pe Euclid pas cu pas

DRY-RUN

Execută algoritmul de mai sus pentru `a = 20`, `b = 12`. Completează tabelul cu valorile lui `a` și `b` **după** fiecare scădere, până devin egale. Adaugă rânduri dacă ai nevoie.

Pas	a	b	Ce scădere ai făcut?
start	20	12	—
1			
2			
3			
4			

→ CMMDC(20, 12) = _____

B

EXERCIȚIUL B · COMPLETEAZĂ COD

Completează algoritmul lui Euclid

COMPLETARE

Programul calculează CMMDC prin scăderi repetate, dar îi lipsesc piesele marcate. Scrie ce trebuie să apară în fiecare spațiu.

```
a = int(input())
b = int(input())
while a ..................... b:
```

```
    if a > b:
        a = ....................
    else:
        b = ....................
print(                  )
      ....................
```

CELE PATRU PIESE, ÎN ORDINE

C

EXERCIȚIUL C · ÎNTREBAREA-CHEIE

Ce reprezintă valoarea rămasă?

GRILĂ

Când barele (numerele) devin egale, bara rămasă e cea mai mare „cărămidă" cu care măsori exact ambele numere de la început. Bifează ce reprezintă ea.

- *a)* `Suma celor două numere.`
- *b)* `Cel mai mare divizor comun al celor două numere.`
- *c)* `Cel mai mic multiplu comun al celor două numere.`

HINT *ambele numere se împart* **exact** *(fără rest) în cărămizi de mărimea celei rămase — asta e definiția unui divizor comun.*

7

SECȚIUNEA 7 · VERIFICĂ-ȚI CODUL

Verifică-ți *codul*

Scrie soluția pe hârtie folosind ce ai învățat (decizie, `while`, `for`, operații cu cifre), apoi testeaz-o pe platformă. Identifică întâi datele problemei (intrare · ieșire · intermediară) și gândește-te la cazurile-limită.

1

PROBLEMA 1

Câte cifre pare are numărul?

SCRIERE COD

Se citește un număr natural `n`. Afișează **câte** dintre cifrele lui sunt pare (0, 2, 4, 6, 8). Exemplu: `n = 2468` → `4`. Atenție la `n = 0`: are o singură cifră, iar 0 este par → răspunsul e `1`.

SOLUȚIA TA · PYTHON

2

PROBLEMA 2

Cifra maximă a unui număr de 3 cifre

SCRIERE COD

Se citește un număr natural `n` de **exact 3 cifre** (`100 ≤ n ≤ 999`). Afișează **cea mai mare cifră** a lui. Exemplu: `n = 528` → `8`.

HINT *desprinde cele trei cifre cu `% 10` și `// 10`, apoi compară-le două câte două ținând maximul într-o variabilă.*

SOLUȚIA TA · PYTHON

3 PROBLEMA 3

Simplificarea unei fracții

SCRIERE COD

Se citesc numărătorul `a` și numitorul `b` ai unei fracții (naturale nenule). Afișează **fracția simplificată** în forma `a/b` (fără spații), împărțind ambii termeni la CMMDC-ul lor. Exemplu: `a = 12`, `b = 8` → `3/2`.

HINT *gândește* **modular** *, cu comentarii:* `# 1) aflu CMMDC(a, b) cu Euclid`*, apoi* `# 2) împart a și b la el`*. Afișezi cu* `print(a // d, "/", b // d, sep="")`*.*

SOLUȚIA TA · PYTHON

4 PROBLEMA 4 · PROVOCARE

CMMDC a trei numere

SCRIERE COD

Se citesc trei numere naturale nenule `a`, `b`, `c`. Afișează cel mai mare divizor comun al tuturor trei. Exemplu: `12`, `8`, `20` → `4`.

HINT *folosește proprietatea* `CMMDC(a, b, c) = CMMDC(CMMDC(a, b), c)`*: afli întâi CMMDC-ul primelor două cu Euclid, apoi al rezultatului cu* `c`*.*

SOLUȚIA TA · PYTHON