test-e.ro

CAIET ELEV

Lecția 10

Divizori, divizibilitate și numere perfecte

*Firul lecției de azi: un singur test — restul împărțirii, **n % d == 0** — deschide totul. Cu el afli dacă un număr e divizibil cu altul, îi enumeri toți divizorii, treci de la o căutare până la **n** la una mult mai rapidă până la **√n** și recunoști numerele perfecte și numerele prime.*

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

NUMELE ELEVULUI .....

CUM FOLOSEȘTI CAIETUL

Acest caiet adună exercițiile scrise din Lecția 10. Rezolvă-le pe hârtie — repari cod, trasezi execuția pas cu pas prin tabele de urmărire, prezici ce afișează un program, ordonezi liniile unui algoritm, completezi operatori lipsă și scrii singur soluțiile. Testul de bază al întregii lecții este mereu același: `n % d == 0` înseamnă „`d` este divizor al lui `n`". Parcurgi candidații cu buclă (`while` sau `for`), fără funcții gata făcute. Regula de aur: **rezultatul îl calculezi tu, înainte de a rula programul.** Verificarea automată a codului rămâne pe platformă.

CUPRINS

1 Recapitulare prin depanare .....
2 Divizor și multiplu .....
3 Enumerarea divizorilor .....
4 Cât de repede? · eficiență .....
5 Numere perfecte .....
6 Numere prime .....
7 Verifică-ți codul .....

SECȚIUNEA 1 · RECAPITULARE

1 Recapitulare prin *depanare*

În Lecția 10 ai lucrat cu cifrele unui număr: ultima cifră cu `n % 10`, numărul rămas după eliminarea ei cu `n // 10`. Astăzi folosești aceleași unelte — restul (`%`) și câtul (`//`) — dar cu alt scop: să afli dacă un număr se împarte exact la altul. Ne încălzim reparând cod și rescriind un algoritm cunoscut.

> **CELE DOUĂ UNELTE DE BAZĂ**
>
> `a % b` îți dă **restul** împărțirii lui `a` la `b`; `a // b` îți dă **câtul** întreg. Când restul este `0`, împărțirea „iese exact" — iar aceasta este cheia întregii lecții.

A

EXERCIȚIUL A · REPARĂ CODUL · 2 GREȘELI

Câtul cifrelor unui număr de două cifre

CORECTARE

Programul citește un număr de **două cifre** și ar trebui să afișeze **câtul împărțirii întregi dintre cifra unităților și cifra zecilor** (unitățile împărțite la zeci). Are **2 greșeli**. Fă un dry-run în minte, încercuiește liniile greșite și rescrie-le corect.

```
1  n = int(input())
2  u = n // 10
3  z = n % 10
4  print(u // z)
```

→ Liniile greșite: _____ corectate: _____

HINT *cifra unităților se obține cu `% 10`, iar cifra zecilor cu `// 10` — în cod sunt inversate. La un număr de două cifre, cifra zecilor e mereu între 1 și 9, deci nu riscăm împărțire la zero.*

B

EXERCIȚIUL B · REPARĂ CODUL · 1 GREȘEALĂ

Numerele de la 1 la n

CORECTARE

Programul ar trebui să afișeze toate numerele de la 1 la `n` pe același rând, separate prin spațiu. Are **o greșeală** care îl face să nu se termine niciodată. Încercuiește problema și scrie linia lipsă.

```
1  n = int(input())
2  i = 1
3  while i <= n:
```

```
4       print(i, end=" ")
```

→ Linia pe care o adaugi (și unde): _____

HINT *afișarea pe același rând e deja rezolvată prin* `end=" "`*. Ce anume îl face pe* `i` *să crească de la o repetare la alta? Fără acel pas,* `i` *rămâne* `1` *la infinit.*

C

EXERCIȚIUL C · SCRIE SINGUR

Câte cifre are un număr

SCRIERE COD

Se citește un număr natural `n`. Afișează **câte cifre are**. Ex.: pentru `n = 4072` programul afișează `4`.

HINT *„taie" repetat ultima cifră cu* `// 10` *și numără câte tăieturi faci până ajungi la* `0`*.*

SOLUȚIA TA · PYTHON

2 SECȚIUNEA 2 · CONCEPTE

Divizor și *multiplu*

Spunem că `d` este **divizor** al lui `n` dacă `n` se împarte exact la `d`, adică fără rest: `n % d == 0`. Când condiția e adevărată, `d` este divizor al lui `n`, iar `n` este **multiplu** al lui `d` — aceeași relație privită din două părți. De exemplu `12 % 3` este `0`, deci 3 e divizor al lui 12; dar `12 % 5` este `2`, deci 5 nu e divizor.

DIVIZORI IMPROPRII ȘI PROPRII

Orice număr natural `n > 0` are doi divizori evidenți: pe `1` (`n % 1` e mereu 0) și pe el însuși (`n % n` e 0). Aceștia se numesc divizori **improprii**. Ceilalți, dacă există, sunt divizori **proprii**. Tot ce urmează în lecție se sprijină pe testul `n % d == 0`.

A

EXERCIȚIUL A · SCRIE CODUL

Test de divizibilitate

SCRIERE COD

Se citesc două numere naturale: `n` și `k` (cu `k` nenul). Afișează `DA` dacă `n` este divizibil cu `k`, altfel `NU`. Ex.: `n = 12, k = 4` → `DA`; `n = 12, k = 5` → `NU`.

HINT *verifică restul `n % k`. Dacă este `0`, atunci `k` este divizor al lui `n`.*

SOLUȚIA TA · PYTHON

B

EXERCIȚIUL B · SITA DIVIZORILOR

Găsește divizorii lui 12

GRILĂ

Numărul este `n = 12`. Mai jos ai toate numerele de la 1 la 12. **Încercuiește-le pe cele care sunt divizori ai lui 12** (numerele la care 12 se împarte fără rest).

1	2	3	4	5	6	7	8	9	10	11	12

→ Câți divizori are 12 în total? _____ Ce observi despre perechile lor (1×12, 2×6, …)?

HINT *divizorii vin în perechi: fiecare divizor mic are un partener mai mare, iar cele două se află de o parte și de alta a lui `√12 ≈ 3,46`.*

C

EXERCIȚIUL C · CLASIFICARE

Divizor propriu sau impropriu?

ASOCIERE

Pentru numărul `n = 10`, clasifică fiecare divizor: **impropriu** (chiar `1` sau chiar `n`) sau **propriu** (orice altul). Divizorii lui 10 sunt 1, 2, 5, 10. Încercuiește varianta corectă pe fiecare rând.

DIVIZORUL LUI 10	ESTE…
`1`	`propriu` `impropriu`
`2`	`propriu` `impropriu`
`5`	`propriu` `impropriu`
`10`	`propriu` `impropriu`

HINT *doar `1` și `n` (adică 10) sunt improprii; 2 și 5 sunt divizori proprii.*

3 SECȚIUNEA 3 · ALGORITM

Enumerarea *divizorilor*

În Secțiunea 2 ai verificat, unul câte unul, dacă un număr e divizor. Dar cum îi găsim **pe toți**? Parcurgem fiecare candidat de la `1` la `n` și îl reținem pe cel care trece testul `n % d == 0`. În pseudocod și în Python:

PSEUDOCOD

```
citește n
pentru d ← 1, n execută
  dacă n mod d = 0 atunci
    scrie d
```

PYTHON

```python
n = int(input())
for d in range(1, n + 1):
  if n % d == 0:
    print(d)
```

DE CE PORNIM DE LA 1, NU DE LA 0?

`n % 0` produce o eroare de execuție (`ZeroDivisionError`) — nu poți împărți la zero. Aceasta e o eroare de tip **runtime**: programul pornește, dar se oprește brusc. (Diferită de una **sintactică**, unde codul nici nu pornește, și de una **logică**, unde rulează dar dă rezultat greșit.)

A

EXERCIȚIUL A · DRY-RUN

Urmărește execuția pentru n = 6

DRY-RUN

Pentru fiecare valoare a lui `d` de la 1 la 6, completează restul `6 % d` și scrie **DA** dacă `d` este divizor, **NU** dacă nu. Primul rând e dat ca exemplu.

d	6 % d	divizor?
1	0	DA
2		
3		
4		
5		
6		

→ Divizorii lui 6 sunt: _____

B

EXERCIȚIUL B · PREZICE IEȘIREA

Ce afișează programul pentru n = 8?

COMPLETARE

Scrie toate numerele afișate, în ordine, separate prin spațiu.

```
n = int(input())
for d in range(1, n + 1):
    if n % d == 0:
        print(d, end=" ")
```

→ Afișează: _____

HINT *parcurgi 1, 2, …, 8 și afișezi doar numerele la care `8 % d` este `0`.*

C

EXERCIȚIUL C · CONSTRUIEȘTE ALGORITMUL

Pune liniile în ordinea corectă

ORDONARE

Liniile de mai jos afișează toți divizorii unui număr `n`. Numerotează-le de la **1** la **4** în căsuțele din stânga, respectând indentarea. **Atenție:** una dintre linii este greșită și **nu se folosește** — lasă căsuța ei goală.

```
        print(d)
n = int(input())
    if n // d == 0:
    if n % d == 0:
for d in range(1, n + 1):
```

HINT *întâi citirea lui `n`, apoi bucla `for`, apoi testul* **corect** *de divizibilitate și afișarea. Linia cu `n // d == 0` este capcana — câtul, nu restul.*

4 SECȚIUNEA 4 · EFICIENȚĂ

Cât de *repede*?

Metoda din Secțiunea 3 funcționează, dar dacă `n` e un milion, face un milion de verificări. Putem mai repede. Divizorii vin în **perechi**: la 12 avem `1×12`, `2×6`, `3×4`. Cele două numere ale unei perechi stau de o parte și de alta a lui `√n`. Deci căutăm doar până la `√n`, iar la fiecare divizor `d` luăm și perechea `n // d`.

METODA LENTĂ · PÂNĂ LA N

```
d = 1
while d <= n:
  if n % d == 0:
    ...
  d = d + 1
```

La un milion → un milion de verificări.

METODA RAPIDĂ · PÂNĂ LA √N

```
d = 1
while d * d <= n:
  if n % d == 0:
    print(d)
    if d != n // d:
      print(n // d)
  d = d + 1
```

La un milion → doar ~o mie de verificări.

CAPCANA PĂTRATULUI PERFECT

La pătrate perfecte (ex. `36 = 6×6`) divizorul `6` este propriul său partener. Condiția `if d != n // d` îl împiedică să fie afișat de două ori.

A

EXERCIȚIUL A · DRY-RUN

Urmărește perechile pentru n = 18

DRY-RUN

`√18 ≈ 4,24`, deci bucla merge cât timp `d * d <= 18` (adică `d` de la 1 la 4). Pentru fiecare `d`, completează restul `18 % d`, dacă e divizor și care e perechea `18 // d` (scrie `–` dacă nu e divizor). Primul rând e exemplu.

d	18 % d	divizor?	perechea 18 // d
1	0	DA	18
2			
3			
4			

→ Toți divizorii lui 18 sunt: _____

B

EXERCIȚIUL B · PREZICE NUMĂRUL DE PAȘI

Câți pași face metoda rapidă?

COMPLETARE

Metoda rapidă verifică valorile lui `d` cât timp `d * d <= n` — adică până la partea întreagă a rădăcinii lui `n`. Pentru fiecare număr, scrie câți pași face bucla.

NUMĂRUL N	CÂȚI PAȘI (VALORI ALE LUI D)
`16`	_____
`50`	_____
`100`	_____
`1000`	_____

HINT *numărul de pași este partea întreagă a lui `√n`: cel mai mare `d` cu `d * d <= n`. Pentru 1000, `31 × 31 = 961 <= 1000`, dar `32 × 32 = 1024 > 1000`.*

SECȚIUNEA 5 · APLICAȚIE

5 Numere *perfecte*

Un număr natural este **perfect** dacă este egal cu suma divizorilor săi **mai mici decât el** (toți divizorii, în afară de el însuși, inclusiv 1). La `6`: `1 + 2 + 3 = 6`. La `28`: `1 + 2 + 4 + 7 + 14 = 28`. Numerele perfecte sunt rare: după 6 și 28 urmează 496, apoi 8128.

CUM VERIFICĂM ÎN PYTHON

Parcurgem divizorii mai mici decât `n` (de la 1 la `n-1`), îi adunăm și comparăm suma cu `n`. Bucla merge `while d < n` (nu `<=`): îl oprim înainte de `n`, fiindcă pe `n` însuși nu îl adunăm.

ATENȚIE LA CAZUL N = 0

Pentru `n = 0`, bucla `while d < n` nu rulează, `suma` rămâne `0`, iar `0 == 0` ar spune fals că 0 e perfect. Dar cel mai mic număr perfect este 6 — un exemplu de **eroare logică**. Când codul poate primi 0, adăugăm condiția `n > 0`.

A

EXERCIȚIUL A · DRY-RUN

Este 6 perfect?

DRY-RUN

Pentru fiecare `d` de la 1 la 5, scrie restul, dacă e divizor și cât devine **suma acumulată** după acest pas. Pornim cu `suma = 0`. Primul rând e exemplu.

d	6 % d	divizor?	suma după pas
1	0	DA	1
2			
3			
4			
5			

→ Suma finală este _____ , deci 6 `este perfect` `nu este perfect`

B

EXERCIȚIUL B · CLASIFICARE

Perfect, deficient sau abundent?

ASOCIERE

Pentru fiecare număr, calculează suma divizorilor mai mici decât el și încercuiește categoria: **perfect** (suma = numărul), **deficient** (suma < numărul), **abundent** (suma > numărul).

NUMĂRUL	SUMA DIVIZORILOR PROPRII	CATEGORIA
8	_____	perfect · deficient · abundent
12	_____	perfect · deficient · abundent
28	_____	perfect · deficient · abundent

HINT *la 8:* `1+2+4 = 7 < 8` *(deficient); la 12:* `1+2+3+4+6 = 16 > 12` *(abundent). Categoriile deficient/abundent sunt „extindere · opțional".*

C

EXERCIȚIUL C · COMPLETEAZĂ CODUL

Verificarea numărului perfect

COMPLETARE

Completează programul care verifică dacă un număr natural `n` (cu `n ≥ 1`) este perfect. Trei operatori lipsesc — scrie-i în spațiile marcate.

```
n = int(input())
suma = 0
d = 1
while d _____ n:          # (1) condiția de oprire
    if n % d == 0:
        suma = suma _____ d   # (2) acumularea sumei
    d = d + 1
if suma _____ n:          # (3) comparația finală
    print("perfect")
else:
    print("nu este perfect")
```

1. operator de comparație (oprire înainte de n) **2.** operație aritmetică **3.** operator de comparație (egalitate)

HINT *(1) bucla merge cât timp* `d` *e strict mai mic decât* `n`*; (2) aduni cifra la sumă; (3) e perfect dacă suma este* **egală** *cu* `n`*.*

SECȚIUNEA 6 · APLICAȚIE

6 Numere *prime*

Un număr natural `n` (cu `n ≥ 2`) se numește **prim** dacă are exact **doi divizori**: pe `1` și pe el însuși. Dacă are mai mulți, se numește **compus**. De exemplu `7` e prim (divizorii 1 și 7), dar `12` e compus (1, 2, 3, 4, 6, 12). Numărul `1` are un singur divizor, deci nu este nici prim, nici compus.

NUMĂRĂ DIVIZORII ȘI VERIFICĂ DACĂ SUNT 2

Un număr e prim exact când are **exact 2 divizori**. Deci numărăm divizorii într-un contor `nrdiv` și comparăm cu 2 la final. Metoda tratează corect și cazul `n = 1` (`nrdiv` va fi 1, nu 2 → nu e prim), fără verificare specială.

A

EXERCIȚIUL A · PREZICE

Recunoaște numerele prime

GRILĂ

Pentru fiecare număr, decide dacă este prim. Încercuiește răspunsul corect.

1. Este `13` prim? DA NU
2. Este `21` prim? DA NU
3. Este `1` prim? DA NU
4. Este `29` prim? DA NU
5. Este `51` prim? DA NU

HINT *atenție la 21 (`3 × 7`) și la 51 (`3 × 17`) — par prime, dar au divizori proprii. 1 nu este prim.*

B

EXERCIȚIUL B · DRY-RUN

Numărăm divizorii lui 9

DRY-RUN

Pentru fiecare `d` de la 1 la 9, scrie restul, dacă e divizor și cât devine contorul `nrdiv` după acest pas. Pornim cu `nrdiv = 0`. Primul rând e exemplu.

d	9 % d	divizor?	nrdiv după pas
1	0	DA	1
2			
3			
…			
9			

→ `nrdiv` final este _____ , deci 9 `este prim` `nu este prim`

HINT *divizorii lui 9 sunt 1, 3, 9 → `nrdiv = 3`. Cum nu e 2, numărul 9 nu este prim (e compus).*

C

EXERCIȚIUL C · COMPLETEAZĂ CODUL

Verificarea primalității cu while

COMPLETARE

Completează cele **3 spații libere** pentru a verifica dacă un număr natural `n` (cu `n ≥ 1`) este prim, numărând divizorii cu `while`.

```
n = int(input())
nrdiv = _____          # (1) pornirea contorului
d = 1
while d _____ n:          # (2) condiția buclei
    if n % d == 0:
        nrdiv = nrdiv + 1
    d = d + 1
if nrdiv _____ 2:      # (3) testul de primalitate
    print("prim")
else:
    print("nu este prim")
```

1. valoarea de pornire a contorului **2.** operator de comparație (până la n inclusiv) **3.** operator de egalitate

HINT *(1) contorul pornește de la `0`; (2) parcurgem toți candidații până la `n` inclusiv, deci `<=`; (3) e prim dacă `nrdiv == 2`.*

SECȚIUNEA 7 · COD GHIDAT

7 Verifică-ți *codul*

Acum scrii tu programele. Fiecare problemă are 5 cazuri de test alese special. Parcurge candidații cu buclă (`while` sau `for`), fără funcții gata făcute. Afișează **doar** ce se cere, pe o singură linie, fără text în plus. Primele trei probleme intră în progres (prag 4 din 5); ultimele două sunt opționale. Verificarea automată rămâne pe platformă.

1

PROBLEMA 7.1 · NIVEL A · STRUCTURĂ SIMPLĂ

Cel mai mare divizor propriu

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 2`). Afișează **cel mai mare divizor propriu** al lui `n` — cel mai mare divizor din intervalul `[2, n//2]`. Dacă `n` este prim, afișează `0`. Ex.: `12` → `6`; `7` → `0`.

HINT *cel mai mare divizor propriu nu poate fi mai mare decât `n // 2`. Caută de sus în jos, pornind de la `n // 2`.*

SOLUȚIA TA · PYTHON

2

PROBLEMA 7.2 · NIVEL B · ALGORITM EFICIENT

Suma tuturor divizorilor

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Afișează **suma tuturor divizorilor** lui `n` (inclusiv 1 și `n`), folosind **metoda eficientă**: căutare până la `√n`, cu perechi. Ex.: `6` → `12`; `12` → `28`.

HINT *folosește `while d * d <= n`. Când găsești un divizor `d`, adaugă atât `d`, cât și perechea `n // d` — dar la pătrate perfecte (`d == n // d`) adaugă-l o singură dată.*

SOLUȚIA TA · PYTHON

3

PROBLEMA 7.3 · NIVEL C · ALGORITM EFICIENT

Câte numere prime până la n

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Afișează **câte numere prime** există în intervalul de la 2 la `n` inclusiv. Verificarea primalității fiecărui număr trebuie să folosească **algoritmul eficient**: căutarea divizorilor merge doar până la `√k` (`d * d <= k`). Ex.: `10` → `4`; `1` → `0`.

HINT *ai nevoie de două bucle. Cea exterioară parcurge `k` de la 2 la `n`. Cea interioară verifică dacă `k` e prim, căutând un divizor cât timp `d * d <= k`.*

SOLUȚIA TA · PYTHON

EXERCIȚII DIFERENȚIATE · OPȚIONALE

Următoarele două nu contează la deblocarea evaluării. Una e o recapitulare de remediere, cealaltă o provocare de extindere.

↺

REMEDIERE · OPȚIONAL

Câți divizori are n

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Afișează **câți divizori** are `n` în total. Ex.: `12` → `6`; `7` → `2`.

HINT *parcurge toate numerele de la 1 la `n` și numără-le pe cele care împart exact (`n % d == 0`).*

SOLUȚIA TA · PYTHON

EXTINDERE · OPȚIONAL

Numere prietene

SCRIERE COD

Două numere sunt **prietene** dacă suma divizorilor mai mici decât fiecare este egală cu celălalt număr (ex.: 220 și 284). Se citesc `a` și `b`. Afișează `DA` dacă sunt prietene, altfel `NU`. Suma divizorilor se calculează cu **algoritmul eficient** (până la `√n`, cu perechi). Ex.: `220, 284` → `DA`; `12, 15` → `NU`.

HINT *pentru fiecare număr, adună `d` și perechea `n // d` cu `while d * d <= n` (fără `n` înșuși, fără dublare la pătrate). Sunt prietene dacă suma lui `a` dă `b`, suma lui `b` dă `a`, și `a ≠ b`.*

SOLUȚIA TA · PYTHON