test-e.ro

CAIET ELEV

*Lecția 11*

# *Descompunerea în factori primi și* algoritmul lui Euclid

*Orice număr se scrie unic ca produs de numere prime — încerci divizorii de la 2 în sus, iar primul găsit e mereu prim. A doua idee mare:* ***algoritmul lui Euclid*** *pentru CMMDC, cu scăderi și cu împărțiri.*

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

**NUMELE ELEVULUI** .....

**CUM FOLOSEȘTI CAIETUL**

Rezolvă exercițiile pe hârtie: urmărești execuția pas cu pas, prezici ce afișează un program și scrii singur soluțiile. Testul de bază rămâne `n % d == 0`, cu buclă (fără liste, fără funcții gata făcute). Regula de aur: **rezultatul îl calculezi tu, înainte de a rula.** Verificarea automată rămâne pe platformă.

**CUPRINS**


1 Ce știm despre divizori .....
2 Cel mai mic factor prim .....
3 Descompunerea completă .....
4 Factori primi distincți .....
5 Euclid · două variante .....
6 Numere prime între ele .....
7 Verifică-ți codul .....

SECȚIUNEA 1 · REACTUALIZARE

# 1 Ce știm despre *divizori*

Descompunerea în factori primi se sprijină pe divizori. Un număr `d` divide `n` dacă `n % d == 0` — restul împărțirii este `0`. Operatorul `%` dă restul, iar `//` dă câtul întreg. Ne încălzim numărând și enumerând divizorii, cu unealta cunoscută: bucla `while d <= n`.

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

## A EXERCIȚIUL A · DRY-RUN

### *Numărarea divizorilor lui 12*

DRY-RUN

Programul numără divizorii lui `n = 12` într-un contor `c`. Pentru fiecare `d` de la 1 la 12, completează restul `12 % d`, scrie **DA/NU** dacă e divizor și cât devine `c` după acest pas. Pornim cu `c = 0`. Primul rând e exemplu.

```
n = 12
c = 0
d = 1
while d <= n:
    if n % d == 0:
        c = c + 1
    d = d + 1
print(c)
```

| d | 12 % d | divizor? | c după pas |
|---|---|---|---|
| 1 | 0 | DA | 1 |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| … | | | |
| 12 | | | |

→ Programul afișează: _____ Divizorii lui 12 sunt: _____

**HINT** *divizorii lui 12 sunt 1, 2, 3, 4, 6, 12 — deci* `c` *ajunge la 6. Printre ei apar atât factorii primi (2, 3), cât și multiplii lor.*

B

EXERCIȚIUL B · SCRIE CODUL

## *Afișează divizorii ≥ 2 ai lui n*

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Afișează toți divizorii lui `n` mai mari sau egali cu 2, **câte unul pe linie**. Ex.: pentru `n = 12` se afișează 2, 3, 4, 6, 12 (fiecare pe câte o linie).

**HINT** *parcurge candidații* `d` *de la 2 la* `n` *și afișează-l pe fiecare la care* `n % d == 0`.

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · CONCEPT-CHEIE

# 2 Cel mai mic *factor prim*

Dacă parcurgi candidații de la `2` în sus, **primul divizor găsit este automat prim** — toți factorii mai mici au fost deja excluși. Aceasta e cheia descompunerii: primul lucru pe care îl scoți dintr-un număr e mereu un număr prim.

> **DE CE E PRIMUL DIVIZOR MEREU PRIM?**
>
> Dacă primul divizor `d ≥ 2` ar fi compus, ar avea el însuși un divizor `d'` mai mic (dar tot `≥ 2`), care ar divide și `n` — deci l-am fi întâlnit **înainte**. Contradicție. Așadar primul divizor găsit nu poate fi compus.

A

EXERCIȚIUL A · DRY-RUN

## *Primul divizor al lui 15*

DRY-RUN

Algoritmul se oprește la **primul** divizor găsit — de aceea folosește variabila `gasit` și condiția `gasit == 0` în buclă. Pentru `n = 15`, completează restul și coloana `gasit` după fiecare pas. Primul rând e exemplu.

```
1  n = 15
2  d = 2
3  gasit = 0
4  while d <= n and gasit == 0:
5      if n % d == 0:
6          gasit = d
7      d = d + 1
8  print(gasit)
```

| d | 15 % d | divizor? | gasit după pas |
|---|---|---|---|
| 2 | 1 | NU | 0 |
| 3 | | | |

→ Bucla se oprește când _____ , iar programul afișează: _____

**HINT** *la `d = 3` avem `15 % 3 = 0`, deci `gasit` devine 3. Condiția `gasit == 0` devine falsă, bucla se oprește și se afișează `3`.*

B

EXERCIȚIUL B · SCRIE CODUL

## *Cel mai mic factor prim*

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 2`). Afișează cel mai mic divizor al lui `n` mai mare ca 1 (adică cel mai mic factor prim al său). Ex.: pentru `n = 15` → `3`; pentru `n = 14` → `2`.

**HINT** *pornește cu `d = 2` și oprește-te la primul `d` pentru care `n % d == 0`. Acela e mereu prim.*

SOLUȚIA TA · PYTHON

C

EXERCIȚIUL C · CLASIFICARE

## *Prim sau compus?*

ASOCIERE

Un număr **prim** are exact doi divizori (1 și el însuși); un număr **compus** are și alți divizori. Încercuiește categoria corectă pe fiecare rând.

| NUMĂRUL | ESTE... |
|---|---|
| 17 | prim compus |
| 21 | prim compus |
| 23 | prim compus |
| 33 | prim compus |
| 49 | prim compus |

**HINT** *atenție la 21 (`3 × 7`), 33 (`3 × 11`) și 49 (`7 × 7`) — par prime, dar au divizori proprii. 17 și 23 sunt prime.*

SECȚIUNEA 3 · ALGORITM

# 3 Descompunerea *completă*

Scoți fiecare factor prim de câte ori intră (`while n % d == 0`), apoi treci la următorul candidat. Numărul se împarte succesiv până când valoarea sa devine `1`. Partea mai grea e **bucla-în-buclă**: `while`-ul interior epuizează un factor, `while`-ul exterior trece la următorul.

```
n = int(input())
d = 2
while d <= n:
    while n % d == 0:
        print(d)
        n = n // d
    d = d + 1
```

**OPTIMIZARE ȘI DOMENIU**

La căutarea factorilor e suficient să testezi `d` cât timp `d*d <= n`; ce rămâne peste `√n`, dacă `n` încă nu a devenit 1, este el însuși prim. Pentru `n = 0` și `n = 1` nu vorbim de factori primi — de aceea cerem `n ≥ 2`.

A

EXERCIȚIUL A · DRY-RUN

## *Descompunerea lui 63*

DRY-RUN

Urmărește codul de mai sus pentru `n = 63`. De fiecare dată când `while`-ul interior găsește un divizor, completează factorul **afișat** și noua valoare a lui `n` după împărțire. Primul rând e exemplu.

| d | n curent | afișează? | n după n // d |
|---|---|---|---|
| 2 | 63 | — | 63 |
| 3 | 63 | 3 | 21 |
| 3 | 21 | | |
| … | | | |
| 7 | | | 1 |

→ Descompunerea completă: **63 =** _____

**HINT** *2 nu divide 63. La* `d = 3`*:* `63 → 21 → 7` *(afișezi 3 de două ori). La* `d = 7`*:* `7 → 1`*. Deci* `63 = 3·3·7`*.*

B

EXERCIȚIUL B · COMPLETEAZĂ

## *Descompuneri în factori primi*

COMPLETARE

Scrie descompunerea fiecărui număr ca produs de factori primi (cu factori repetați, în ordine crescătoare). Prima e dată ca model.

| NUMĂRUL | DESCOMPUNEREA ÎN FACTORI PRIMI |
|---|---|
| `60` | `2·2·3·5` |
| `36` | _____ |
| `100` | _____ |
| `45` | _____ |
| `72` | _____ |

**HINT** *împarte mereu la cel mai mic factor prim posibil:* `36 = 2·2·3·3`*,* `100 = 2·2·5·5`*,* `45 = 3·3·5`*,* `72 = 2·2·2·3·3`*.*

C

EXERCIȚIUL C · SCRIE CODUL

## *Afișează factorii primi ai lui n*

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 2`). Afișează factorii primi ai lui `n`, câte unul pe linie, repetând un factor de câte ori apare. Ex.: pentru `n = 12` se afișează 2, 2, 3.

**HINT** *folosește bucla-în-buclă din teorie:* `while d <= n` *pe afară,* `while n % d == 0` *înăuntru (afișezi* `d` *și faci* `n = n // d`*).*

SOLUȚIA TA · PYTHON

## 4 SECȚIUNEA 4 · VARIAȚIE

# Factori primi *distincți*

Uneori nu ne interesează factorii repetați, ci **câte numere prime diferite** apar: `12 = 2²·3` are doi factori primi distincți. După ce numărăm un factor, îl scoatem complet (`while n % d == 0`) înainte de a merge mai departe, ca să nu-l numărăm de mai multe ori.

A

EXERCIȚIUL A · NUMĂRĂ CULORILE

### *Câte numere prime diferite?*

COMPLETARE

Fiecare factor prim din descompunere ar primi o culoare; factorii identici — aceeași culoare. Scrie descompunerea și numără **câte culori distincte** apar. Prima e model.

| NUMĂRUL | DESCOMPUNEREA | FACTORI DISTINCȚI |
|---|---|---|
| `12` | `2·2·3` | `2` |
| `60` | | |
| `16` | | |
| `30` | | |

**HINT** *60 = 2·2·3·5 → 3 distincți; 16 = 2·2·2·2 → 1 distinct; 30 = 2·3·5 → 3 distincți.*

B

EXERCIȚIUL B · SCRIE CODUL

## *Câți factori primi distincți*

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 2`). Afișează câți factori primi **distincți** are `n`. Ex.: pentru `n = 12` → `2` (factorii sunt 2 și 3).

**HINT** *de fiecare dată când găsești un factor nou, adaugă 1 la contor și apoi scoate-l* **complet** *cu* `while n % d == 0`*, ca să nu-l numeri iar.*

SOLUȚIA TA · PYTHON

C

EXERCIȚIUL C · PREZICE PAȘII

### *Cursa: până la n vs până la √n*

COMPLETARE

Aceeași numărare de divizori, două bucle. Metoda lentă verifică `d` până la `n`; cea rapidă doar cât timp `d*d <= n`. Pentru fiecare număr, scrie câți pași face fiecare.

| NUMĂRUL N | PAȘI PÂNĂ LA N | PAȘI PÂNĂ LA √N |
|---|---|---|
| `36` | _____ | _____ |
| `100` | _____ | _____ |
| `400` | _____ | _____ |

**HINT** *metoda lentă face `n` pași; cea rapidă face partea întreagă a lui `√n`: la 36 → 6, la 100 → 10, la 400 → 20 pași.*

## 5 SECȚIUNEA 5 · ALGORITMUL LUI EUCLID

# Euclid: *scăderi și împărțiri*

Același algoritm pentru **cel mai mare divizor comun** (CMMDC), în două forme. Prima scade repetat numărul mic din cel mare, până devin egale. A doua înlocuiește perechea `(a, b)` cu `(b, a % b)` până când `b` devine 0. Ambele ajung la același CMMDC — dar a doua e mult mai rapidă. Din CMMDC obții imediat `CMMMC = a·b // CMMDC`.

CU SCĂDERI · MAI LENTĂ

```
a = int(input())
b = int(input())
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
print(a)
```

CU ÎMPĂRȚIRI · MAI RAPIDĂ

```
a = int(input())
b = int(input())
while b != 0:
  r = a % b
  a = b
  b = r
print(a)
```

**FĂRĂ ATRIBUIRI SIMULTANE**

La varianta cu împărțiri folosește o variabilă temporară `r` pentru rest. Ordinea contează: întâi `r = a % b`, apoi `a = b`, apoi `b = r`. Dacă schimbi ordinea, pierzi o valoare de care mai ai nevoie.

A

EXERCIȚIUL A · DRY-RUN

## *Euclid cu scăderi pe (12, 8)*

DRY-RUN

Urmărește varianta cu scăderi pentru `a = 12`, `b = 8`. La fiecare pas, scrie care număr e mai mare și noile valori ale lui `a` și `b`. Primul rând e exemplu. Bucla se oprește când `a == b`.

| pas | a | b | a > b? | a, b după pas |
|---|---|---|---|---|
| start | 12 | 8 | DA | a=4, b=8 |
| 1 | 4 | 8 | | |
| 2 | | | | |

→ CMMDC(12, 8) = _____

**HINT** *12,8 → 4,8 → 4,4. Când devin egale, valoarea comună (4) este CMMDC.*

B

EXERCIȚIUL B · DRY-RUN

## *Euclid cu împărțiri pe (48, 36)*

DRY-RUN

Urmărește varianta cu împărțiri pentru `a = 48`, `b = 36`. La fiecare pas calculează `r = a % b`, apoi mută `a = b` și `b = r`. Primul rând e exemplu. Bucla se oprește când `b == 0`.

| pas | a | b | r = a % b | a, b după pas |
|---|---|---|---|---|
| 1 | 48 | 36 | 12 | a=36, b=12 |
| 2 | 36 | 12 | | |
| 3 | | | | |

→ CMMDC(48, 36) = _____ · În câți pași? metoda cu scăderi ar face mult mai mulți: `adevărat` `fals`

**HINT** *48,36 → r=12 → 36,12 → r=0 → 12,0. Bucla se oprește, `a = 12` este CMMDC. Doar 2 pași față de multe scăderi.*

C

EXERCIȚIUL C · SCRIE CODUL

### *CMMDC prin împărțiri repetate*

SCRIERE COD

Se citesc două numere naturale `a` și `b` (cu `a, b ≥ 1`), pe câte o linie. Afișează CMMDC folosind împărțiri repetate. Folosește o variabilă temporară `r`; nu folosi atribuiri simultane. Ex.: pentru `a = 48, b = 36` → `12`.

**HINT** *repetă cât timp `b != 0`: calculează `r = a % b`, apoi `a = b` și `b = r`. La final afișezi `a`.*

SOLUȚIA TA · PYTHON

## 6 SECȚIUNEA 6 · APLICAȚIE

## Numere *prime între ele*

Două numere sunt **prime între ele** (relativ prime) dacă singurul lor divizor comun este 1, adică `CMMDC(a, b) == 1`. Nu trebuie să fie ele însele prime: `8 = 2·2·2` și `9 = 3·3` nu au niciun factor comun, deci sunt prime între ele. Acesta e și testul care spune când o fracție e deja ireductibilă.

A

EXERCIȚIUL A · DRY-RUN

## *Sunt 8 și 9 prime între ele?*

DRY-RUN

Rulează Euclid cu împărțiri pe `a = 8`, `b = 9`. La fiecare pas completează `r = a % b` și noile valori. Primul rând e exemplu. Când `b` ajunge la 0, valoarea din `a` este CMMDC.

| pas | a | b | r = a % b | a, b după pas |
|---|---|---|---|---|
| 1 | 8 | 9 | 8 | a=9, b=8 |
| 2 | 9 | 8 | | |
| 3 | | | | |
| 4 | | | | |

→ CMMDC(8, 9) = _____ , deci 8 și 9 `sunt prime între ele` `nu sunt prime între ele`

**HINT** *8,9 → 9,8 → 8,1 → 1,0. CMMDC = 1, deci sunt prime între ele — deși niciunul nu e prim.*

B

EXERCIȚIUL B · PREZICE

## *Sunt prime între ele?*

GRILĂ

Pentru fiecare pereche, decide dacă numerele sunt prime între ele (CMMDC = 1). Încercuiește răspunsul corect.

1. `8` și `9` DA NU
2. `12` și `18` DA NU
3. `14` și `15` DA NU
4. `21` și `14` DA NU
5. `25` și `9` DA NU

**HINT** *12 și 18 au factorul comun 6 (NU); 21 și 14 au factorul comun 7 (NU). 8-9, 14-15 și 25-9 nu au factori comuni (DA).*

C

EXERCIȚIUL C · SCRIE CODUL

## *Prime între ele?*

SCRIERE COD

Se citesc `a` și `b` (cu `a, b ≥ 1`), pe câte o linie. Afișează `1` dacă sunt prime între ele (CMMDC = 1), altfel `0`. Ex.: `a = 8, b = 9` → `1`; `a = 12, b = 18` → `0`.

**HINT** *calculează întâi CMMDC cu Euclid (împărțiri), apoi verifică dacă rezultatul este egal cu 1.*

SOLUȚIA TA · PYTHON

## 7 SECȚIUNEA 7 · COD GHIDAT

# Verifică-ți *codul*

Acum scrii tu programele. Fiecare problemă are 5 cazuri de test alese special. Parcurge candidații cu buclă (`while` sau `for`), fără liste și 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 de reparare și de provocare. Verificarea automată rămâne pe platformă.

### 1 PROBLEMA 7.1 · NIVEL A

## CMMDC (*Euclid cu împărțiri*)

SCRIERE COD

Se citesc `a` și `b` (cu `a, b ≥ 1`), pe câte o linie. Afișează cel mai mare divizor comun folosind algoritmul lui Euclid cu împărțiri. Ex.: `a = 12, b = 18` → `6`.

**HINT** *repetă* `while b != 0`: `r = a % b`, `a = b`, `b = r`. *La final afișezi* `a`.

SOLUȚIA TA · PYTHON

PROBLEMA 7.2 · NIVEL B

## 2 CMMMC — *cel mai mic multiplu comun*

SCRIERE COD

Se citesc `a` și `b` (cu `a, b ≥ 1`), pe câte o linie. Afișează cel mai mic multiplu comun, folosind `CMMMC = a·b // CMMDC(a, b)`. Ex.: `a = 4, b = 6` → `12`.

**HINT** *calculează întâi CMMDC cu Euclid, apoi `a * b // cmmdc`. Folosește `//` (împărțire întreagă), nu `/`.*

SOLUȚIA TA · PYTHON

3

PROBLEMA 7.3 · NIVEL C

## *Cel mai mare factor prim*

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 2`). Afișează cel mai mare factor prim al lui `n`. Ex.: pentru `n = 60` → `5` (deoarece `60 = 2²·3·5`).

**HINT** *descompune ca în S3: scoate pe rând fiecare factor prim. Reține ultimul factor scos — acela este cel mai mare.*

SOLUȚIA TA · PYTHON

**REPARARE ȘI PROVOCARE**

Următoarele două probleme sunt diferite: una are un cod greșit pe care îl repari, cealaltă e o provocare de extindere.

PROBLEMA 7.4 · REPARĂ CODUL

## *Repară: Euclid cu scăderi*

CORECTARE

Programul ar trebui să afle `CMMDC(a, b)` prin scăderi repetate, dar bucla folosește condiția greșită `while a > b` și scade doar într-o singură direcție — deci dă rezultat greșit când `b > a`. Rescrie-l ca să scadă corect în ambele direcții, până când numerele devin egale.

```
a = int(input())
b = int(input())
while a > b:
    a = a - b
print(a)
```

→ Ce e greșit și cum repari:

**HINT** *condiția trebuie să fie `while a != b`, iar înăuntru un `if a > b: a = a - b` / `else: b = b - a`.*

CODUL REPARAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

# *Factorul prim cu puterea cea mai mică*

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 2`). Afișează factorul prim care apare la **cea mai mică putere**, împreună cu acea putere, pe aceeași linie, separate prin spațiu. Dacă mai mulți factori au aceeași putere minimă, afișează-l pe **cel mai mare** dintre ei. Ex.: pentru `n = 60 = 2²·3·5` → `5 1`.

**HINT** *pentru fiecare factor prim, numără de câte ori intră (puterea lui). Reține factorul cu puterea minimă; la egalitate, păstrează-l pe cel mai mare (candidații cresc, deci ultimul cu putere minimă egală e cel mai mare).*

SOLUȚIA TA · PYTHON