test-e.ro

CAIET ELEV

Lecția 13

# Recapitulare și aplicații integrate — prelucrări numerice

*Nimic nou de învățat astăzi — doar legăm între ele uneltele din lecțiile 10–13. Fiecare secțiune rezolvă un algoritm cu nume propriu, rulat chiar acum de cineva: cardul bancar, IBAN-ul, cheile de criptare, raportul „16:9" al ecranului tău și cifrul lui Cezar. Aceleași **%**, **//**, **while** și **ord()**, puse la treabă în lumea reală.*

**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. Cifrele (inclusiv resturile la 2) se parcurg mereu cu buclă și cu `%` / `//` — niciodată prin transformare în șir. Fără liste, fără `def`. Regula de aur: **rezultatul îl calculezi tu, înainte de a rula.** Verificarea automată rămâne pe platformă.

**CUPRINS**


1 Ne reamintim toate uneltele .....
2 Algoritmul lui Luhn · cardul bancar .....
3 Rădăcina digitală · trei metode .....
4 Schema lui Horner · IBAN mod 97 .....
5 Diviziunea succesivă · numere prime .....
6 Raportul de aspect 16:9 · Euclid .....
7 Cifrul lui Cezar și bitul de paritate .....

SECȚIUNEA 1 · REACTUALIZARE GENERALĂ

# 1 Ne reamintim *toate uneltele*

Lecția de azi nu predă nimic nou — leagă între ele lucruri deja învățate. Reamintim cele patru unelte pe care le vom folosi: **extragerea cifrelor** (`c = n % 10`, apoi `n = n // 10`, într-un `while n > 0`), **divizibilitatea** (`n % d == 0` înseamnă „`d` îl împarte exact pe `n`"), **algoritmul lui Euclid** (înlocuim perechea `(a, b)` cu `(b, a % b)` până când al doilea devine 0) și **codul ASCII** (`ord()` dă codul unei litere, `chr()` face drumul invers).

> **CELE PATRU UNELTE**
>
> `n % 10` ultima cifră · `n // 10` restul numărului · `n % d == 0` „d divide n" · perechea Euclid `(a, b) → (b, a % b)` · `ord("A") = 65`. Cu ele se rezolvă toate cele șase aplicații de mai jos.

## A EXERCIȚIUL A · GRILĂ

### *Uneltele din ultimele lecții*

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce afișează `print(2718 // 100 % 10)`? 7 · 271 · 8 · 1
2. Care este cel mai mare divizor comun al numerelor 48 și 180? 6 · 24 · 12 · 36
3. Ce valoare are `ord("F")` în Python? 1 · 70 · 97 · 26
4. Ce înseamnă că `n % d == 0` este adevărată?

   `d îl împarte exact pe n` · `n < d mereu` · `n e prim` · `d e ultima cifră a lui n`

**HINT** *`2718 // 100 = 27`, apoi `27 % 10 = 7`. CMMDC(48, 180) = 12. `ord("F") = 70`.*

## B EXERCIȚIUL B · DRY-RUN

### *Algoritmul lui Euclid pentru* CMMDC(48, 36)

DRY-RUN

La fiecare pas calculează `r = a % b`, apoi noua pereche devine `(b, r)`. Continuă până când `b` ajunge 0 — atunci `a` este CMMDC-ul. Primul rând e exemplu.

```
a = 48
b = 36
while b != 0:
    r = a % b
```

```
    a = b
    b = r
print(a)
```

| a | b | r = a % b | noua pereche (b, r) |
|---|---|---|---|
| 48 | 36 | 12 | (36, 12) |
| 36 | 12 | | |
| | | | |

→ Programul afișează (CMMDC): _____

**HINT** *36 % 12 = 0 → perechea devine (12, 0), bucla se oprește, se afișează* `a = 12`.

C

EXERCIȚIUL C · SCRIE CODUL

## *Reîncălzire: suma cifrelor*

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Afișează suma cifrelor lui. Ex.: pentru `n = 1234` → 1 + 2 + 3 + 4 = `10`.

**HINT** *cât timp* `n > 0`: `s = s + n % 10`, *apoi* `n = n // 10`.

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

## *Câte cifre are numărul?*

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Afișează câte cifre are. Ex.: `1234` → 4; `7` → 1.

**HINT** *numeri de câte ori poți face `n = n // 10` până când `n` devine 0.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

## *Ultima cifră și restul numărului*

SCRIERE COD

Se citește un număr natural cu cel puțin două cifre. Afișează pe primul rând ultima cifră, iar pe al doilea numărul rămas după eliminarea ei. Ex.: `1234` → `4`, apoi `123`.

**HINT** *ultima cifră e `n % 10`; restul numărului e `n // 10`.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · ALGORITMUL LUI LUHN

# 2 Cum știe un site că ai greșit o cifră din *card*

Când tastezi greșit un număr de card, site-ul îți spune „număr invalid" instantaneu — fără să întrebe banca. Reamintim tiparul de parcurgere: `c = n % 10` ia ultima cifră, `n = n // 10` o îndepărtează, iar `while n > 0` repetă. **Algoritmul lui Luhn** parcurge cifrele de la dreapta la stânga și **dublează fiecare a doua cifră** (pozițiile pare, cu un contor `poz` care pornește de la 1). Dacă dublarea trece de 9, se scade 9 (14 − 9 = 5). Adună tot; cardul e valid doar dacă suma se împarte exact la 10.

> **CINE ÎL RULEAZĂ**
>
> Fiecare card Visa, Mastercard și Maestro are ultima cifră calculată astfel. Codul din pagina magazinului rulează Luhn **înainte** de a trimite ceva — de aceea apare „număr invalid" instant, fără internet. Același algoritm verifică și codurile IMEI ale telefoanelor.

A

EXERCIȚIUL A · GRILĂ

## *Regulile algoritmului lui Luhn*

GRILĂ

Încercuiește răspunsul corect.

1. Ce facem dacă dublarea unei cifre dă 14?

`scădem 9 → 5` `păstrăm 14` `păstrăm doar 1` `ignorăm cifra`

2. Un card trece testul Luhn dacă suma finală:

`e prime` `se împarte exact la 10` `= ultima cifră` `se împarte la 9`

3. Cu ce expresie eliminăm ultima cifră a lui `n`? `n % 10` `n // 10` `n % 2` `n - 10`

**HINT** *dublare peste 9 → scade 9; suma validă se împarte exact la 10; ultima cifră se elimină cu* `// 10`.

B

EXERCIȚIUL B · DRY-RUN

## *Luhn pe numărul 1380*

DRY-RUN

Aplică Luhn de la dreapta la stânga. Poziția și cifra sunt date; tu calculezi **valoarea** cu care intră cifra în sumă (dublată pe pozițiile pare și, dacă trece de 9, minus 9) și **suma** de până acolo. Primul rând e exemplu.

| poz | cifra c | dublăm? | valoare | suma |
|---|---|---|---|---|
| 1 | 0 | nu | 0 | 0 |
| 2 | 8 | DA | | |
| 3 | 3 | nu | | |
| 4 | 1 | DA | | |

→ Suma finală: _____ · 1380 trece testul? DA NU

**HINT** *poz 2: 8·2 = 16, 16 − 9 = 7; poz 4: 1·2 = 2. Suma = 0 + 7 + 3 + 2 = 12; 12 nu se împarte la 10 ⇒ NU.*

C

EXERCIȚIUL C · SCRIE CODUL

## *Validarea unui card cu Luhn*

SCRIERE COD

Se citește un număr natural `n` — un număr de card. Afișează pe același rând numărul citit, un spațiu, apoi `DA` dacă trece testul lui Luhn sau `NU` altfel. Ex.: pentru `1380` se afișează `1380 NU`.

**HINT** *păstrează numărul într-o a doua variabilă înainte de buclă (parcurgerea îl distruge). Ține un contor `poz` de la 1; dublează când `poz % 2 == 0`; la final testează `suma % 10 == 0`.*

SOLUȚIA TA · PYTHON

★

PERFORMANȚĂ · SCRIE CODUL

### *Generează cifra de control Luhn*

SCRIERE COD

Se citește un număr natural `n` reprezentând primele cifre ale unui card, **fără ultima cifră (cifra de control)**. Afișează cifra care trebuie adăugată la sfârșit ca numărul complet să treacă testul lui Luhn. Ex.: pentru `7992739871` → `3`.

**HINT** *calculează suma Luhn știind că prima cifră adăugată va sta pe poziția 1 (deci contorul pornește de la 2); cifra de control este* `(10 - suma % 10) % 10`.

SOLUȚIA TA · PYTHON

## 3 SECȚIUNEA 3 · RĂDĂCINA DIGITALĂ

## Aceeași cifră de control, în *trei feluri*

Aduni cifrele unui număr; dacă rezultatul are încă două cifre, aduni iar — până rămâi cu o singură cifră. Aceea e **rădăcina digitală**. Sunt trei căi. **Metoda 1** traduce definiția: două bucle una în alta, aduni toate cifrele cât timp rezultatul e ≥ 10. **Metoda 2** face un singur pas mic de fiecare dată: `n = n // 10 + n % 10`; la fiecare pas numărul scade cu un multiplu de 9, deci restul la 9 nu se schimbă. **Metoda 3** folosește direct `n % 9` (proba lui 9), într-un singur pas — cu grijă la cazul `n = 0`.

**IDEEA ÎN TREI RÂNDURI DE COD**

```
while n >= 10:      # metoda 2
    n = n // 10 + n % 10
print(n)
```

A

EXERCIȚIUL A · GRILĂ

## *Rădăcina digitală și cele trei metode*

GRILĂ

Încercuiește răspunsul corect.

1. Care este rădăcina digitală a numărului 99999? `45` `9` `5` `99999`

2. La metoda 2, când înlocuim 1234 cu 127, cu cât a scăzut numărul?

`cu 1107 (123·9)` `cu 10` `cu 4` `nu a scăzut`

3. De ce metoda cu `n % 9` tratează separat `n = 0`?

`0 % 9 = 0, iar regula ar da greșit 9` `0 % 9 dă eroare` `0 n-are cifre` `formula merge doar peste 10`

4. Care metodă face același număr de pași oricât de mare e numărul?

`suma cifrelor repetată` `n // 10 + n % 10` `formula cu % 9` `toate la fel`

**HINT** *9+9+9+9+9 = 45 → 4+5 = 9. Metoda 3 (formula) e singura care nu depinde de mărimea numărului.*

B

EXERCIȚIUL B · DRY-RUN

## *Metoda 2 pe numărul 1234*

DRY-RUN

La fiecare pas `n` devine `n // 10 + n % 10`. Completează cele două părți și noua valoare, până când `n` rămâne cu o singură cifră. Primul rând e exemplu.

```
1  n = 1234
2  while n > 9:
3      n = n // 10 + n % 10
4  print(n)
```

| n | n // 10 | n % 10 | n nou = suma lor |
|---|---|---|---|
| 1234 | 123 | 4 | 127 |
| 127 | | | |
| 19 | | | |
| 10 | | | |

→ Rădăcina digitală (n final): _____ · nr. de pași: _____

**HINT** *127 → 19, 19 → 10, 10 → 1. Bucla se oprește la 1, după 4 pași. Formula: 1234 % 9 = 1.*

C

EXERCIȚIUL C · COMPLETEAZĂ

## Cele trei metode

COMPLETARE

Completează golurile folosind cuvintele din rezervă (două cuvinte rămân nefolosite).

`10` `9` `0` `2` `100`

Rădăcina digitală se poate calcula adunând repetat cifrele până rezultatul scade sub _____ ; metoda scurtă înlocuiește numărul cu `n // 10 + n % 10`, ceea ce îl micșorează de fiecare dată cu un multiplu de _____ ; iar metoda cea mai rapidă folosește direct restul împărțirii la 9, tratând separat cazul în care numărul este _____ .

**HINT** *ordinea: 10 → 9 → 0.*

D

EXERCIȚIUL D · SCRIE CODUL

## Rădăcina digitală prin formulă

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 0`). Afișează rădăcina lui digitală folosind **metoda 3** (formula bazată pe restul la 9), nu bucla — și tratează corect cazul `n = 0`. Ex.: `1234` → `1`.

**HINT** *dacă `n == 0` → 0; altfel dacă `n % 9 == 0` → 9; altfel → `n % 9`.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

### *Câți pași face metoda 2?*

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 0`). Afișează de câte ori se execută pasul `n = n // 10 + n % 10` până când `n` rămâne cu o singură cifră (numărul de pași, nu rădăcina). Ex.: `1234` → `4`; `7` → `0`.

**HINT** *ține un contor pe care îl crești la fiecare trecere prin bucla* `while n > 9`.

SOLUȚIA TA · PYTHON

## 4 SECȚIUNEA 4 · SCHEMA LUI HORNER

## Cum verifică banca un IBAN, *cifră cu cifră*

Un IBAN are peste 20 de caractere; transformat în număr, e prea mare pentru o variabilă obișnuită. Și totuși banca îi calculează restul la 97 instantaneu. Standardul ISO 7064 (MOD 97-10) spune: IBAN-ul e valid dacă, ca număr, dă **restul 1 la împărțirea cu 97**. **Schema lui Horner** nu construiește

niciodată numărul întreg — îl citește cifră cu cifră, de la stânga la dreapta, și ține doar restul de până acum: `rest = (rest * 10 + cifra) % 97`. La fiecare pas `rest` rămâne sub 97, oricât de lung ar fi IBAN-ul.

> **CINE ÎL RULEAZĂ**
>
> Fiecare bancă, la fiecare transfer, înainte de a trimite banii. Dacă tastezi greșit o cifră din IBAN, aplicația te oprește pe loc — nu pentru că știe al cui e contul, ci pentru că restul la 97 nu mai iese 1.

A

EXERCIȚIUL A · GRILĂ

## *Schema lui Horner și restul la 97*

GRILĂ

Încercuiește răspunsul corect.

1. De ce nu construim numărul IBAN întreg înainte de a-l împărți la 97?

   `ar fi prea mare; ținem doar restul` `modulo nu merge pe numere mari`
   `IBAN conține litere` `restul iese din ultima cifră`

2. Care este formula lui Horner pentru rest?

   `rest = rest + cifra % 97` `rest = (rest * 10 + cifra) % 97`
   `rest = (rest + 10*cifra) % 97` `rest = cifra % 97`

3. Un IBAN e valid dacă restul împărțirii la 97 este: `0` `1` `97` `nr. de cifre`

**HINT** *ținem doar restul mic la fiecare pas (Horner); numărul e valid când restul final e exact 1.*

B

EXERCIȚIUL B · DRY-RUN

## *Horner mod 97 pe numărul 195*

DRY-RUN

Citim cifrele de la stânga la dreapta. La fiecare pas: `rest·10 + cifra`, apoi restul la 97. Completează cele două coloane. Primul rând e exemplu.

```
1  rest = 0
2  rest = (rest * 10 + 1) % 97
3  rest = (rest * 10 + 9) % 97
4  rest = (rest * 10 + 5) % 97
5  print(rest)
```

| cifra | rest vechi | rest·10 + cifra | rest nou = … % 97 |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
| 9 | 1 | | |
| 5 | 19 | | |

→ Restul final: _____ · e IBAN valid? DA NU

**HINT** *1·10 + 9 = 19 (19 % 97 = 19); 19·10 + 5 = 195, iar 195 % 97 = 1 → DA (195 = 2·97 + 1).*

C

EXERCIȚIUL C · SCRIE CODUL

## *Verificarea unui IBAN cu Horner*

SCRIERE COD

Se citește un număr natural `n` — forma numerică a unui IBAN. Afișează `DA` dacă restul împărțirii la 97 este exact 1, altfel `NU`. Cerință: folosește Horner cifră cu cifră, de la stânga la dreapta.

**HINT** *Horner cere cifrele de la stânga, dar `% 10` le dă de la dreapta. Nu oglindi numărul (pierzi zerourile finale). Află întâi puterea lui 10 a primei cifre, apoi coboară cu ea extrăgând cifra cu `n // p % 10`.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · DIVIZIUNEA SUCCESIVĂ

# 5 De ce criptografia are nevoie de *numere prime*

Parola ta ajunge la server criptată cu chei construite din numere prime foarte mari. Ca să le găsești, întâi trebuie să răspunzi la o întrebare simplă: e prim acest număr? Un **număr prim** are exact doi divizori. Metoda directă e **diviziunea succesivă**: încerci pe rând fiecare candidat `d` și verifici `n % d == 0`. Trucul de la L11: dacă `n` are un divizor mai mare decât rădăcina, are obligatoriu și unul mai mic (perechea lui) — deci e destul să testezi cât timp `d * d <= n`.

> **CINE ÎL RULEAZĂ**
>
> Când browserul deschide o conexiune securizată (lacătul din bară), se generează și se testează numere prime mari. Securitatea RSA se sprijină pe asimetria asta: e ușor să înmulțești două prime, dar foarte greu să refaci drumul invers. **Ciurul lui Eratostene** ar fi mai rapid, dar are nevoie de liste — îl scriem la lecția 16.

## A EXERCIȚIUL A · GRILĂ

### *Primalitate și diviziune succesivă*

GRILĂ

Încercuiește răspunsul corect.

1. De ce ne oprim când `d * d > n`?

   `un divizor mare are pereche mică, deja testată` `peste rădăcină nu-s niciodată divizori` `bucla are pași limitați` `după rădăcină toți divizorii sunt primi`

2. Numărul 91 este prim?

   `da, e impar` `nu, 91 = 7 · 13` `da, nu se împarte la 2,3,5` `nu, e par`

3. Câți divizori are un număr prim? `exact doi: 1 și el` `exact unul` `niciunul` `mereu trei`

**HINT** *91 = 7 · 13, deci nu e prim; un prim are exact doi divizori: 1 și el însuși.*

## B EXPLOREAZĂ · CURSA

### *Testez toți divizorii vs. mă opresc la rădăcină*

COMPLETARE

Pentru un număr prim, metoda naivă testează toți candidații de la 2 la `n - 1`, iar cea isteață se oprește când `d * d > n`. Completează câte verificări face fiecare metodă (candidați `d = 2, 3, …`). Primul rând e exemplu.

| N (PRIM) | TESTEZ TOȚI (D PÂNĂ LA N−1) | MĂ OPRESC LA RĂDĂCINĂ (D·D ≤ N) |
|---|---|---|
| `23` | `21` | `3` (d = 2, 3, 4) |
| `49` | _____ | _____ |
| `97` | _____ | _____ |

**HINT** *49: toți → 47 (d = 2..48), rădăcină → 6 (d = 2..7, fiindcă 7·7 = 49 ≤ 49). 97: toți → 95, rădăcină → 8 (d = 2..9, 9·9 = 81 ≤ 97).*

C

EXERCIȚIUL C · SCRIE CODUL

## *E bun ca număr prim pentru o cheie?*

SCRIERE COD

Se citește un număr natural `n`. Afișează pe același rând numărul citit, un spațiu, apoi `DA` dacă `n` este prim sau `NU` altfel. Cerință: diviziune succesivă, oprire când `d * d > n`. Ex.: `91` → `91 NU`; `61` → `61 DA`.

**HINT** *numerele sub 2 nu sunt prime; pornește cu* `d = 2` *și, cât timp* `d * d <= n`*, dacă* `n % d == 0` *nu e prim.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 6 · RAPORTUL DE ASPECT

# 6 De unde vine „16:9"

Ecranul tău are 1920 pe 1080 de pixeli. De ce toată lumea spune despre el că e „16:9"? Pentru că cineva a rulat **algoritmul lui Euclid**: înlocuim repetat perechea `(a, b)` cu `(b, a % b)` până când al doilea devine 0. **Raportul de aspect** e rezoluția redusă la forma cea mai simplă. Pentru 1920 × 1080, CMMDC-ul e 120, iar `1920 // 120 = 16` și `1080 // 120 = 9` — de aici „16:9". Pentru 640 × 480 iese 4:3.

> **CINE ÎL RULEAZĂ**
>
> Playerele video, ca să știe unde să pună benzile negre; YouTube, când alege rezoluția; telefonul, când încadrează o poză. Formatul 16:9 pe care îl vezi peste tot e literalmente rezultatul acestui algoritm pe 1920 și 1080.

A

EXERCIȚIUL A · ASOCIERE

## *Rezoluție → raport de aspect*

ASOCIERE

Scrie în căsuță litera raportului corect pentru fiecare rezoluție. Împarte ambele numere la CMMDC-ul lor.

| | |
|---|---|
| ☐ | `1920 × 1080` |
| ☐ | `640 × 480` |
| ☐ | `2560 × 1600` |
| ☐ | `1280 × 1024` |

A 16:9 · B 4:3 · C 8:5 · D 5:4

**HINT** *1920×1080 → A · 640×480 → B · 2560×1600 → C · 1280×1024 → D.*

B

EXERCIȚIUL B · DRY-RUN

## *Euclid pe 1920 și 1080*

DRY-RUN

Calculează `r = a % b` la fiecare pas; noua pereche devine `(b, r)`. Continuă până când `b` ajunge 0. Primul rând e exemplu.

```
1  a = 1920
```

```
b = 1080
while b != 0:
    r = a % b
    a = b
    b = r
print(a)
```

| a | b | r = a % b | noua pereche (b, r) |
|---|---|---|---|
| 1920 | 1080 | 840 | (1080, 840) |
| 1080 | 840 | | |
| 840 | 240 | | |
| 240 | 120 | | |

→ CMMDC: _____ · raportul 1920 // CMMDC : 1080 // CMMDC = _____

**HINT** *1080%840=240, 840%240=120, 240%120=0 → CMMDC = 120; 1920//120 = 16, 1080//120 = 9 → 16:9.*

C

EXERCIȚIUL C · SCRIE CODUL

## *Din rezoluție în „16:9"*

SCRIERE COD

Se citesc două numere naturale `a` și `b` (lățimea și înălțimea în pixeli). Afișează raportul de aspect: pe primul rând lățimea redusă, pe al doilea înălțimea redusă (fiecare împărțită la CMMDC). Ex.: `1920`, `1080` → `16` apoi `9`.

**HINT** *calculează CMMDC(a, b) cu Euclid pe copii ale lui a și b, apoi afișează* `a // g` *și* `b // g`.

SOLUȚIA TA · PYTHON

SECȚIUNEA 7 · CEZAR ȘI BITUL DE PARITATE

# 7 Litere care devin *numere*

Calculatorul nu știe ce e litera „A" — știe doar numărul 65, codul ei ASCII. Reamintim: `ord("A") = 65`, `ord("Z") = 90`, `chr(65) = A`; poziția unei litere în alfabet e `ord(litera) - ord("A")`. **Cifrul lui Cezar** deplasează fiecare literă cu un număr fix de poziții (A cu 3 → D); ca să nu ieșim din alfabet la sfârșit, folosim `% 26` (Z deplasat cu 1 revine la A). **Bitul de paritate** numără câți biți de 1 are numărul în baza 2 și spune dacă numărul lor e par sau impar.

> **CINE LE RULEAZĂ**
>
> Cifrul lui Cezar e strămoșul criptografiei; varianta lui **ROT13** (deplasare cu 13) ascunde și azi spoilerele pe forumuri. Bitul de paritate a fost folosit în fiecare modem și în memoria RAM cu corecție de erori — dacă un bit se strică pe drum, numărătoarea nu mai iese și receptorul știe că datele-s corupte.

## A EXERCIȚIUL A · GRILĂ

### *ASCII, Cezar și paritate*

GRILĂ

Încercuiește răspunsul corect.

1. Ce afișează `print(chr(66))`? `66` `B` `A` `b`

2. De ce folosim `% 26` în cifrul lui Cezar?

   `ca să revenim la început după Z` `alfabetul are 26 litere mici` `ord dă valori sub 26` `deplasarea nu trece de 26`

3. 13 în baza 2 este 1101. Care e bitul lui de paritate (1 dacă nr. de biți de 1 e impar)? `0` `1` `13` `4`

**HINT** *`chr(66) = B`; `% 26` ține litera în alfabet; 1101 are trei biți de 1 (impar) → paritate 1.*

B

EXERCIȚIUL B · ORDONARE

## *Pașii cifrului lui Cezar*

ORDONARE

Numerotează etapele (scrie 1–5 în căsuță) în ordinea în care se execută.

- ☐ Aduni deplasarea k la poziție.
- ☐ Afli codul ASCII al literei cu `ord(litera)`.
- ☐ Aduni la loc `ord("A")` și transformi înapoi în literă cu `chr`.
- ☐ Scazi `ord("A")` ca să obții poziția literei în alfabet (0 pentru A).
- ☐ Aplici `% 26` ca să rămâi în alfabet după Z.

**HINT** *ordinea: ord → scazi ord("A") → aduni k → % 26 → aduni ord("A") și chr.*

C

EXERCIȚIUL C · SCRIE CODUL

## *Cifrul lui Cezar*

SCRIERE COD

Se citește o literă mare (A–Z) și un număr natural `k` (deplasarea). Afișează litera obținută prin cifrul lui Cezar, cu revenire la A după Z. Ex.: litera `A` și `k = 3` → `D`; `Z` și `k = 1` → `A`.

**HINT** *poziția e `ord(litera) - ord("A")`; adaugi k, iei `% 26`, apoi `chr(poz + ord("A"))`.*

SOLUȚIA TA · PYTHON

D

EXERCIȚIUL D · SCRIE CODUL

## *Bitul de paritate*

SCRIERE COD

Se citește un număr natural `n`, scris în baza 10. Afișează bitul de paritate: `1` dacă numărul de biți egali cu 1 din reprezentarea binară a lui `n` este impar, altfel `0`. Ex.: `n = 13` (binar 1101, trei biți de 1) → `1`.

**HINT** *treci numărul în binar cu împărțiri repetate la 2: cât timp `n > 0`, dacă `n % 2 == 1` crește contorul, apoi `n = n // 2`. La final afișează `1` dacă `contor % 2 != 0`.*

SOLUȚIA TA · PYTHON