test-e.ro

CAIET ELEV

Lecția 12

Transformarea unui număr dintr-o bază în alta și codul ASCII

*Un număr rămâne același; doar „haina" lui — baza de numerație — se schimbă. Împarți repetat la 2 și citești resturile pentru a trece în baza 2; aduni puteri ale lui 2 pentru drumul invers. Aceeași idee de „cod" se aplică și literelor: fiecare are un cod ASCII, iar **ord()** și **chr()** te duc dintr-o parte în alta.*

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 (resturile la 2, cifrele binare) se parcurg mereu cu buclă și cu `%` / `//` — niciodată prin transformare în șir. Nou în această lecție: `ord()` și `chr()`. Regula de aur: **rezultatul îl calculezi tu, înainte de a rula.** Verificarea automată rămâne pe platformă.

CUPRINS

1 Ne reamintim: %, // și while .....
2 Din baza 2 în baza 10 .....
3 Codul ASCII: ord() .....
4 De la cod la caracter: chr() .....
5 Poziția unei litere în alfabet .....
6 Câți biți de 1 are un număr .....
7 Verifică-ți codul .....

SECȚIUNEA 1 · REACTUALIZARE

1 Ne reamintim %, // și while

Lecția de azi se sprijină pe unelte vechi. `n % 2` dă restul împărțirii la 2 (0 dacă `n` e par, 1 dacă e impar), iar `n // 2` dă câtul întreg — practic „jumătatea" numărului, rotunjită în jos. Dacă împărțim un număr la 2 din nou și din nou, mai devreme sau mai târziu ajunge la 0. Bucla `while n > 0` ne lasă să repetăm o acțiune atâta timp cât numărul e mai mare ca zero.

> **CELE DOUĂ UNELTE DE BAZĂ**
>
> `a % b` îți dă **restul** împărțirii; `a // b` îți dă **câtul întreg**. Injumătățirea repetată cu `// 2` este motorul întregii lecții: din ea obținem cifrele binare ale unui număr.

A EXERCIȚIUL A · DRY-RUN

De câte ori împărțim 13 la 2 până ajunge la 0

DRY-RUN

Programul împarte `n = 13` la 2 până devine 0 și numără împărțirile în `c`. La fiecare pas completează noua valoare `n // 2` și cât devine `c` după acel pas. Primul rând e exemplu.

```
1  n = 13
2  c = 0
3  while n > 0:
4      n = n // 2
5      c = c + 1
6  print(c)
```

n (la intrare)	n // 2	c (după pas)
13	6	1
6		
3		
1		

→ Programul afișează: _____

HINT *13 → 6 → 3 → 1 → 0, deci 4 împărțiri:* `c` *ajunge la 4.*

B

EXERCIȚIUL B · GRILĂ

Rest, cât întreg și bucla while

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce afișează `print(84 % 9)`? (3) (9) (12) (0)
2. Ce afișează `print(84 // 9)`? (3) (9) (12) (0,33)
3. După `n = 13 // 2`, ce valoare are `n`? (6) (7) (6,5) (1)
4. Cât timp continuă bucla `while n > 0:` să se execute?

   (`cât timp n > 0`) (`cât timp n == 0`) (`exact de 10 ori`) (`niciodată`)

HINT *`84 % 9 = 3` (restul), `84 // 9 = 9` (câtul întreg), `13 // 2 = 6`. Bucla merge cât timp condiția e adevărată.*

C

EXERCIȚIUL C · SCRIE CODUL

Numără împărțirile la 2

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Împarte-l repetat la 2 (împărțire întreagă) până când devine 0 și afișează de câte ori a fost necesar să îl împarți. Ex.: pentru `n = 8` → `4` (8 → 4 → 2 → 1 → 0).

HINT *pornește cu `c = 0` și, cât timp `n > 0`, fă `n = n // 2` și `c = c + 1`.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · DIN BAZA 2 ÎN BAZA 10

2 Din baza 2 în *baza 10*

Un număr binar se „citește" în baza 10 ca sumă de puteri ale lui 2. Îi luăm cifrele de la dreapta la stânga și le înmulțim cu puteri din ce în ce mai mari ale lui 2: cifra din dreapta valorează 1, următoarea 2, apoi 4, 8, 16… Pentru `1101` avem 1·1 + 0·2 + 1·4 + 1·8 = **13**. În program extragem cifrele cu `% 10`, le eliminăm cu `// 10` și adunăm cifra înmulțită cu puterea curentă a lui 2, putere pe care o dublăm la fiecare pas.

> **UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ**
>
> Text, imagini, muzică — totul e memorat de calculator ca șiruri de biți. Adresele IP (ex. `192.168.0.1`) sunt de fapt numere binare scrise pe bucăți în baza 10, iar codurile QR codează informația tot ca tipar de biți.

A EXERCIȚIUL A · GRILĂ

Cât valorează în baza 10?

GRILĂ

Încercuiește răspunsul corect.

1. Cât valorează în baza 10 numărul binar `101`? (5) (3) (101) (6)
2. Ce „poziție" (putere a lui 2) are a treia cifră binară de la dreapta? (4) (2) (3) (8)
3. Cât valorează în baza 10 numărul binar `1000`? (8) (4) (1000) (16)

HINT *`101 = 1·1 + 0·2 + 1·4 = 5`; pozițiile sunt 1, 2, 4, 8…; `1000 = 8`.*

B EXERCIȚIUL B · COMPLETEAZĂ

Cifrele bazei 2 și conversia

COMPLETARE

Completează propoziția, apoi transformă fiecare număr binar în baza 10 (prima e model).

→ În baza 2 se folosesc doar cifrele 0 și _____ .

NUMĂR BINAR	VALOAREA ÎN BAZA 10
`110`	`6`
`11`	_____
`1010`	_____
`1101`	_____

HINT *doar cifra 1 lipsește.* `11 = 3`, `1010 = 10`, `1101 = 13`.

C

EXERCIȚIUL C · ORDONARE

Ordonează programul: baza 2 → baza 10

ORDONARE

Se dă algoritmul în pseudocod care transformă un număr binar `b` în valoarea lui zecimală. Numerotează instrucțiunile Python de mai jos (scrie 1–9 în căsuță) în ordinea corectă, respectând pseudocodul.

```
citește b
rez ← 0
p ← 1
cât timp b > 0 execută
    c ← ultima cifră a lui b
    rez ← rez + c * p
    p ← p * 2
    elimină ultima cifră a lui b
afișează rez
```

- ☐ `    p = p * 2`
- ☐ `rez = 0`
- ☐ `    c = b % 10`
- ☐ `print(rez)`
- ☐ `b = int(input())`
- ☐ `    rez = rez + c * p`
- ☐ `p = 1`
- ☐ `    b = b // 10`
- ☐ `while b > 0:`

HINT *inițializările (`rez`, `p`) vin înainte de buclă; în interior ordinea e: iei cifra, aduni `c*p`, dublezi `p`, elimini cifra.*

SECȚIUNEA 3 · CODUL ASCII

3 Litere care sunt de fapt *numere*

În memoria calculatorului nu există „litera A", ci un număr care o reprezintă: codul ei ASCII. Literele mari sunt consecutive: `A` = 65, `B` = 66, …, `Z` = 90. Literele mici încep mai sus: `a` = 97, …, `z` = 122. Chiar și cifra-caracter `0` are cod: 48. În Python, `ord(caracter)` întoarce codul caracterului. Fiindcă `ord("A") = 65` și `ord("a") = 97`, diferența dintre o literă mică și corespondenta ei mare este mereu 32.

> **UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ**
>
> Când apeși o tastă, tastatura trimite codul ASCII al literei, nu litera. Sortarea alfabetică a numelor compară aceste coduri, iar când un site cere „o literă mare" în parolă, verifică dacă există un caracter cu codul între 65 și 90.

A EXERCIȚIUL A · GRILĂ

Coduri ASCII

GRILĂ

Încercuiește răspunsul corect.

1. Ce afișează `print(ord("A"))`? (65) (1) (97) (66)
2. Ce afișează `print(ord("a"))`? (97) (65) (26) (1)
3. Cu cât e mai mare codul unei litere mici față de corespondenta ei mare?

(cu 32) (cu 26) (cu 1) (cu 65)

HINT *`ord("A") = 65`, `ord("a") = 97`, iar 97 − 65 = 32.*

B EXERCIȚIUL B · SCRIE CODUL

Codul unei litere

SCRIERE COD

Se citește o singură literă. Afișează codul ei ASCII. Ex.: pentru litera `A` → `65`.

HINT *citește litera cu `input()` și afișează `ord(litera)`.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 4 · DE LA COD LA CARACTER

4 De la cod la caracter: *chr()*

Funcția `chr()` este perechea lui `ord()`: primește un cod ASCII și întoarce caracterul corespunzător. `chr(65)` este `A`, `chr(90)` este `Z`, `chr(97)` este `a`, iar `chr(48)` este cifra-caracter `0`. Cu `ord()` și `chr()` împreună putem trece oricând dintr-o parte în alta: dintr-o literă aflăm codul, dintr-un cod refacem litera.

> **UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ**
>
> Perechea `ord()` / `chr()` este inima criptării simple (cifrul Cezar): aplicațiile „mută" literele pe coduri și le refac cu `chr()` la destinație. Tot așa transformă un editor un text din MAJUSCULE în litere mici — adună 32 la fiecare cod și reface caracterul.

A

EXERCIȚIUL A · GRILĂ

Din cod la literă

GRILĂ

Încercuiește răspunsul corect.

1. Ce afișează `print(chr(66))`? `B` `A` `66` `b`
2. Ce afișează `print(chr(99))`? `c` `A` `97` `z`
3. Ce literă are codul ASCII 90? `Z` `A` `z` `9`

HINT *`chr(66) = B`, `chr(99) = c` (litere mici de la 97), iar codul 90 este `Z`.*

B

EXERCIȚIUL B · COMPLETEAZĂ

Completează definiția: ord și chr

COMPLETARE

Completează golurile cu `ord`, `chr` sau `codul`.

→ Funcția _____ transformă o literă în _____ ei ASCII, iar funcția _____ transformă un cod înapoi în literă.

HINT *ordinea corectă: `ord` → `codul` → `chr`.*

C

EXERCIȚIUL C · SCRIE CODUL

Din cod la literă

SCRIERE COD

Se citește un număr `cod` (un cod ASCII valid). Afișează caracterul corespunzător. Ex.: pentru `cod = 65` → `A`.

HINT *citește codul cu* `int(input())` *și afișează* `chr(cod)`.

SOLUȚIA TA · PYTHON

5 SECȚIUNEA 5 · POZIȚIA ÎN ALFABET

Poziția unei litere în *alfabet*

Fiindcă literele mari sunt consecutive, poziția unei litere în alfabet se află scăzând codul lui `A`: `ord(litera) - ord("A") + 1`. Pentru `A` obținem 65 − 65 + 1 = 1, pentru `B` 66 − 65 + 1 = 2, pentru `Z` 90 − 65 + 1 = 26. Acest tipar (scade codul de referință) este cheia cifrurilor simple.

> **UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ**
>
> Cifrurile de tip Cezar mută fiecare literă cu un număr fix de poziții, iar poziția în alfabet e primul lucru de care ai nevoie. La fel, gruparea cuvintelor pe litera inițială (ca într-o agendă cu despărțitoare A–Z).

A

EXERCIȚIUL A · GRILĂ

Poziția în alfabet

GRILĂ

Încercuiește răspunsul corect.

1. Ce poziție are `G` (cu `ord("G") - ord("A") + 1`)? 7 6 71 8
2. Cât este `ord("D") - ord("A") + 1`? 4 0 65 66
3. Ce poziție în alfabet are litera `Z`? 26 25 90 27

HINT `G` *e a 7-a,* `D` *a 4-a,* `Z` *a 26-a literă.*

B

EXERCIȚIUL B · COMPLETEAZĂ

Formula poziției

COMPLETARE

Completează golul.

→ Poziția unei litere mari în alfabet se calculează cu `ord(litera) - ord("A") +` _____ .

HINT *adaugi 1, ca poziția lui* `A` *să iasă 1, nu 0.*

C

EXERCIȚIUL C · SCRIE CODUL

A câta literă din alfabet?

SCRIERE COD

Se citește o singură literă mare (A–Z). Afișează poziția ei în alfabet (A = 1, …, Z = 26). Ex.: pentru `G` → `7`.

HINT *afișează* `ord(litera) - ord("A") + 1`.

SOLUȚIA TA · PYTHON

EXPLOREAZĂ · CIFRUL CEZAR

Roata alfabetului

COMPLETARE

„Mutarea unei litere" este de fapt o adunare pe coduri ASCII. Cu deplasarea `k = 3`, fiecare literă clară devine litera aflată cu 3 poziții mai departe în alfabet. Completează litera cifrată (prima e model).

LITERĂ CLARĂ	CIFRATĂ (K = 3)
`A`	`D`
`B`	_____
`H`	_____
`M`	_____

HINT *A → D, B → E, H → K, M → P (fiecare mutată cu 3 poziții).*

SECȚIUNEA 6 · BIȚI DE 1

6 Numărarea *biților de 1*

Combinăm conversia în baza 2 cu un contor. Parcurgem cifrele binare ale numărului cu `n % 2` (ultima cifră binară) și `n // 2` (restul numărului), dar în loc să construim reprezentarea, doar **numărăm** de câte ori cifra binară este 1 — testând `n % 2 != 0`. Pentru `13 = 1101` sunt trei biți de 1; pentru `255 = 11111111` sunt opt. Numerele cu exact un singur bit de 1 sunt puterile lui 2.

> **UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ**
>
> Numărarea biților de 1 („popcount") apare la verificarea erorilor de transmisie (bitul de paritate) și la „setările pe steaguri" din jocuri, unde fiecare bit aprins e o opțiune activă. Recunoașterea puterilor lui 2 e peste tot: 1 KB = 1024, rezoluții, mărimi de fișier.

EXERCIȚIUL A · GRILĂ

A *Biți de 1*

GRILĂ

Încercuiește răspunsul corect.

1. Câți biți de 1 are numărul 13 în baza 2? `3` `2` `4` `13`
2. Câți biți de 1 are numărul 255 în baza 2? `8` `4` `255` `1`
3. Cum testăm corect că restul `c = n % 2` este cifra binară 1?

   `if c != 0:` `if c == 2:` `if c > 1:` `if c == 10:`

HINT *`13 = 1101` → 3 biți; `255 = 11111111` → 8 biți; restul la 2 e fie 0, fie 1, deci testul e `c != 0`.*

EXERCIȚIUL B · DRY-RUN

B *Numără biții de 1 ai lui 13*

DRY-RUN

Urmărește codul pentru `n = 13`. La fiecare pas completează cifra binară `bit = n % 2`, noua valoare `n // 2` și cât devine `c`. Primul rând e exemplu.

```
1  n = 13
2  c = 0
3  while n > 0:
4      bit = n % 2
5      if bit != 0:
```

```
6           c = c + 1
7       n = n // 2
8   print(c)
```

n (la intrare)	bit = n%2	n // 2	c (după pas)
13	1	6	1
6			
3			
1			

→ Programul afișează: _____

HINT *biții lui 13 = 1101 sunt 1, 0, 1, 1, deci* `c` *ajunge la 3.*

C

EXERCIȚIUL C · SCRIE CODUL

Numărul de biți de 1

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Afișează câți biți de 1 are reprezentarea lui binară. Ex.: pentru `n = 13` → `3`.

HINT *cât timp* `n > 0`*: dacă* `n % 2 != 0` *crește contorul, apoi* `n = n // 2`.

SOLUȚIA TA · PYTHON

EXPLOREAZĂ · BAZA 2 ↔ BAZA 10

Comutatoarele binare

COMPLETARE

Fiecare bit aprins (scrie 1) adaugă valoarea de deasupra lui. Aprinde biții care compun litera `A` (cod ASCII 65) — scrie 1 sau 0 în fiecare căsuță — apoi verifică suma.

128	64	32	16	8	4	2	1

→ Suma pozițiilor aprinse (baza 10): _____ · litera ASCII: _____

HINT *65 = 64 + 1, deci aprinzi biții de la 64 și de la 1 (`01000001`). Suma = 65, litera = `A`.*

7 SECȚIUNEA 7 · COD GHIDAT

Verifică-ți *codul*

Acum scrii tu programele. Fiecare problemă are 5 cazuri de test alese special. Parcurge cifrele doar cu buclă (`while`) și cu `%` / `//` — fără liste, 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

Baza 10 → baza 2

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Afișează reprezentarea lui în baza 2. Ex.: pentru `n = 6` → `110`.

HINT *împarte repetat la 2 și adună cifra `n % 2` înmulțită cu puterea curentă a lui 10 (`p = 1, 10, 100…`), ca resturile să apară în ordinea corectă.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.2 · NIVEL B

2 Baza 2 → baza 10

SCRIERE COD

Se citește un număr `b` scris cu cifre 0 și 1. Afișează valoarea lui în baza 10. Ex.: pentru `b = 101` → `5`.

HINT *ia cifrele cu `b % 10`, adună `c * p` și dublează `p` (1, 2, 4, 8…) la fiecare pas — exact algoritmul din S2.*

SOLUȚIA TA · PYTHON

3

PROBLEMA 7.3 · NIVEL C

Din majusculă în minusculă

SCRIERE COD

Se citește o singură literă mare (A–Z). Afișează litera mică corespunzătoare, folosind codul ASCII (litera mică are codul cu 32 mai mare). Ex.: pentru `A` → `a`.

HINT *calculează* `ord(litera) + 32` *și transformă rezultatul înapoi în caracter cu* `chr()`.

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 conversia în baza 2

CORECTARE

Codul de mai jos ar trebui să afișeze reprezentarea în baza 2 a unui număr `n ≥ 1`, dar are o greșeală: dublează `p` (ca și cum ar reconstrui numărul), deci pentru `n = 13` afișează greșit **13** în loc de **1101**. Găsește greșeala și rescrie programul ca să afișeze corect reprezentarea binară.

```python
n = int(input())
rez = 0
p = 1
while n > 0:
    c = n % 2
    rez = rez + c * p
    p = p * 2
    n = n // 2
print(rez)
```

→ Ce e greșit și cum repari: _____

HINT *ca cifrele binare să apară pe pozițiile zecimale corecte (unități, zeci, sute…),* `p` *trebuie înmulțit cu* **10** *, nu cu 2:* `p = p * 10`*.*

CODUL REPARAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Este putere a lui 2?

SCRIERE COD

Se citește un număr natural `n` (cu `n ≥ 1`). Afișează `DA` dacă `n` este o putere a lui 2 (adică are exact un bit de 1 în baza 2) și `NU` altfel. Ex.: `n = 8` → `DA` (8 = 1000); `n = 6` → `NU` (6 = 110).

HINT *numără biții de 1 ca în S6; la final afișează* `DA` *dacă numărul de biți este exact 1, altfel* `NU`.

SOLUȚIA TA · PYTHON