Lecția 21

Subprograme predefinite Python: operații matematice și colecții

Python vine cu o trusă de subprograme deja scrise, numite predefinite: conversiile de tip (int, float, str), operațiile matematice (abs, round și, din modulul math, sqrt, floor, ceil) și cele patru subprograme pentru colecții (len, min, max, sum), cu care media aritmetică devine o singură expresie.

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, completezi, asociezi și scrii singur soluțiile, apoi le verifici pe platformă. Ce ai voie să folosești: secvențial, if/elif/else, while, for cu range, %, //, ord/chr, liste cu indexare și len, metodele listei, def/return și, nou la această lecție, import math și min/max/sum. Fără felii de listă, fără fișiere, fără clase. Cifrele unui număr se parcurg tot cu while și %/// — nicio conversie la șir. Fără diacritice în codul Python. Rezultatele zecimale se afișează prin round(x, 2).

CUPRINS

1 Uneltele scrise de tine .....
2 Subprograme pe care nu trebuie să le scrii .....
3 O bibliotecă pe care o aduci în program .....
4 len, min, max, sum .....
5 Care rotunjire, în ce situație .....
6 Ce se întâmplă la margini .....
7 Cod ghidat · cinci probleme .....

SECȚIUNEA 1 · CE ȘTIM DEJA DESPRE SUBPROGRAME

1 Unelte scrise de *tine*

La lecțiile trecute ai învățat să-ți construiești singur uneltele: un antet cu `def`, un corp, parametri care primesc valori la apel și un `return` care trimite rezultatul înapoi. Înainte de a merge mai departe, reactualizăm exact acest mecanism — plus ce știm despre liste și despre numărul de pași al unui algoritm.

Un subprogram are trei părți pe care le recunoști dintr-o privire: **antetul** (`def nume(parametri):`), **corpul** (instrucțiunile indentate) și **valoarea întoarsă** (`return`). Numele scrise în antet sunt *parametri formali*; valorile trimise la apel sunt *parametri actuali*. Variabilele create în corp sunt **locale**: se nasc la apel și dispar când subprogramul se termină, iar programul principal nu le vede.

Aceeași idee, în sintaxă reală:

```
1  def bonus(puncte, prag):
2      if puncte > prag:
3          return puncte - prag
4      return 0
6  print(bonus(30, 25))
```

DE REȚINUT

`return` face două lucruri deodată: trimite valoarea înapoi **și** oprește imediat execuția subprogramului. Instrucțiunile scrise după un `return` care s-a executat nu mai rulează niciodată.

A

EXERCIȚIUL A · GRILĂ

Recapitulare: subprograme, liste, eficiență

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce face, de fapt, instrucțiunea `return` dintr-un subprogram?

   - `Creează o variabilă globală, vizibilă din orice punct al programului`
   - `Trimite o valoare înapoi către locul din care a fost apelat subprogramul`
   - `Afișează valoarea pe ecran, exact la fel ca instrucțiunea print`
   - `Oprește execuția întregului program, nu doar pe cea a subprogramului`

2. Ce fel de variabile sunt cele create în corpul unui subprogram?

- Constante, cărora nu li se mai poate schimba valoarea după creare
- Parametri formali, care primesc valori doar în momentul apelului
- Variabile globale, vizibile din orice punct al programului principal
- Variabile locale, care există doar cât timp se execută subprogramul

3. Pentru `l = [4, 9, 2, 7]`, ce valoare are `l[1]`?

- Valoarea 9, pentru că indicii unei liste încep de la 0
- Valoarea 4, pentru că indicii unei liste încep de la 1
- Eroare, pentru că lista are doar patru elemente
- Valoarea 2, pentru că se numără de la sfârșitul listei

4. De ce testul de primalitate cu `while d * d <= n` e mai rapid decât cel cu `while d <= n`?

- Pentru că folosește înmulțire, iar înmulțirea e mai rapidă decât împărțirea
- Pentru că nu mai are nevoie de nicio instrucțiune `if` în interiorul buclei
- Pentru că se oprește la primul divizor găsit, spre deosebire de cealaltă variantă
- Pentru că face aproximativ √n pași în loc de n pași

B

EXERCIȚIUL B · DRY-RUN

Câți bonuși se adună?

DRY-RUN

Urmărește apelurile subprogramului bonus, rând cu rând. Coloanele i și puncte îți sunt date. Tu completezi valoarea întoarsă de fiecare apel și totalul de după acel apel — ambele numerice.

```
1  def bonus(puncte, prag):
2      if puncte > prag:
3          return puncte - prag
4      return 0
5  total = 0
6  i = 1
7  while i <= 4:
8      total = total + bonus(i * 10, 25)
9      i = i + 1
10 print(total)
```

i (dat)	puncte = i * 10 (dat)	bonus(puncte, 25) (completeaza)	total după apel (completeaza)
1	10	0	0
2	20		
3	30		
4	40		
5 — condiția i <= 4 e falsă, bucla se oprește	—	—	20

HINT *Pentru fiecare rând întreabă-te întâi dacă puncte depășește pragul 25: dacă nu, apelul întoarce 0 și totalul rămâne neschimbat.*

C

EXERCIȚIUL C · SCRIE CODUL

Suma elementelor unei liste — exercițiu de sprijin

SCRIERE COD

Se citește un număr natural n, apoi n numere întregi, fiecare pe câte o linie. Scrie un subprogram suma_lista(l) care primește o listă și întoarce suma elementelor ei, calculată cu o buclă. Afișează valoarea întoarsă de subprogram. Ex.: pentru n = 3 și numerele 5, 7, 9, subprogramul întoarce 5 + 7 + 9 = 21, deci se afișează 21.

HINT *Aduni tu, cu bucla, element cu element — exact ca la lecțiile despre liste.*

SOLUȚIA TA · PYTHON

D

EXERCIȚIUL D · SCRIE CODUL

Distanța până la zero — exercițiu de sprijin

SCRIERE COD

Se citește un număr întreg x. Scrie un subprogram distanta(x) care întoarce distanța de la x până la 0, adică valoarea lui fără semn. Afișează valoarea întoarsă de subprogram. Ex.: pentru x = -7 se afișează 7. Pentru x = 12 se afișează 12.

HINT **Atenție.** *Subprogramul trebuie să întoarcă o valoare pe fiecare drum posibil: și când x e negativ, și când nu este. Dacă uiți o ramură, apelul întoarce altceva decât te aștepți.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · TRUSA GATA FĂCUTĂ: CONVERSII, ABS, ROUND

2 Subprograme pe care nu trebuie să le *scrii*

Python vine cu zeci de subprograme deja scrise și verificate, numite *predefinite*. Nu le definești tu — le apelezi direct, după exact aceleași reguli ca la propriile tale subprograme. Începem cu cele pe care le foloseai deja, fără să le numești: conversiile de tip, `abs` și `round`.

Un **subprogram predefinit** e un subprogram scris de creatorii limbajului și disponibil în orice program, fără nicio pregătire. Îl apelezi la fel: nume, paranteze, argumente, valoare întoarsă.

Primul grup sunt **conversiile de tip**. `int(x)` transformă în număr întreg, `float(x)` în număr real, `str(x)` în text. Le foloseai de la prima lecție, doar că suprapuse: în `int(input())` se execută întâi `input()`, care întoarce **întotdeauna text**, apoi `int()`, care transformă textul în număr.

```
varsta = int(input())
pret = float(input())
mesaj = "Ai " + str(varsta) + " ani"
```

Al doilea grup: `abs(x)` întoarce valoarea fără semn (`abs(-8)` dă `8`), iar `round(x)` rotunjește la cel mai apropiat întreg. `round` primește și un al doilea argument, numărul de zecimale păstrate: `round(3.14159, 2)` dă `3.14`.

ATENȚIE

`int(7.9)` dă `7`, nu `8` — conversia **taie** partea zecimală, nu rotunjește. Când vrei rotunjire, ceri explicit `round(7.9)`, care dă `8`. Și încă o capcană: `round(2.5)` dă `2`, nu 3, pentru că la exact jumătate Python rotunjește către numărul par.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Orice formular online face conversii: câmpul în care scrii vârsta ajunge la server ca text și trebuie transformat în număr înainte de orice comparație. Aplicațiile de fitness afișează `round(distanta, 2)` km în loc de 5,283746 km. Iar `abs` apare peste tot unde contează *cât de mult* diferă două valori, nu în ce direcție: cu câte grade s-a abătut temperatura frigiderului de la cea setată, cu câți lei diferă bonul de suma din cont.

A

EXERCIȚIUL A · GRILĂ

Conversii, abs și round

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce tip de date întoarce `input()`?

- Număr întreg (`int`), dacă utilizatorul a tastat doar cifre
- Număr real (`float`), pentru că acceptă și valori cu zecimale
- Text (`str`), indiferent ce a tastat utilizatorul
- Tipul se decide automat, în funcție de ce anume s-a tastat

2. Ce afișează `print(round(2.5))`?

- Eroare, pentru că `round` cere obligatoriu numărul de zecimale
- 2.5, pentru că `round` fără al doilea argument nu schimbă nimic
- 3, pentru că la exact jumătate rotunjirea se face întotdeauna în sus
- 2, pentru că la exact jumătate rotunjirea merge către numărul par

3. Ce se obține prin `int("12") + int("30")`?

- Eroare, pentru că în Python nu se pot aduna două texte între ele
- Numărul 42, pentru că ambele texte au fost convertite în numere
- Numărul 42.0, pentru că orice conversie produce un număr real
- Textul "1230", pentru că valorile rămân șiruri de caractere lipite

B

EXERCIȚIUL B · ASOCIERE

Apelul și rezultatul lui

ASOCIERE

Scrie în căsuța fiecărui apel de subprogram predefinit litera rezultatului pe care îl produce. Ai grijă la diferența dintre „taie zecimalele” și „rotunjește”. Fiecare literă se folosește o singură dată.

☐	`abs(-8)`
☐	`round(4.62, 1)`
☐	`int(7.9)`
☐	`str(25) + "3"`

A "253" · B 7 · C 4.6 · D 8

C

EXERCIȚIUL C · SCRIE CODUL

Cu cât s-a abătut temperatura

SCRIERE COD

Se citesc două numere reale, fiecare pe câte o linie: temperatura măsurată într-o cameră frigorifică și temperatura setată pe termostat. Scrie un subprogram abatere(a, b) care întoarce abaterea dintre ele, adică diferența fără semn. Afișează valoarea întoarsă, rotunjită la două zecimale. Ex.: pentru 12.5 și 8.25, diferența e 4.25, deci se afișează 4.25. Pentru -3.5 și 4.25, diferența fără semn e 7.75.

HINT *Nu contează care temperatură e mai mare — abaterea e o distanță, deci întotdeauna pozitivă sau zero.*

HINT **Atenție.** *Citirea se face cu float(input()), nu cu int(input()): temperaturile au zecimale, iar int le-ar tăia înainte de orice calcul.*

SOLUȚIA TA · PYTHON

3 SECȚIUNEA 3 · MODULUL MATH: RĂDĂCINĂ, PODEA, TAVAN

O bibliotecă pe care o aduci în *program*

Nu toate subprogramele predefinite sunt disponibile din start. Cele matematice mai serioase — rădăcina pătrată, rotunjirea în jos, rotunjirea în sus — stau într-o bibliotecă separată, numită modulul `math`. O aduci în program cu o singură linie și apoi o folosești cu notația cu punct.

Ca să folosești modulul `math`, îl anunți la începutul programului:

```
import math
print(math.sqrt(9))
print(math.floor(4.7))
print(math.ceil(4.2))
```

Notația cu punct se citește „`sqrt` din `math`” și e obligatorie: fără prefixul `math.`, numele nu e cunoscut.

RĂDĂCINA PĂTRATĂ — MATH.SQRT(X)

Întoarce **întotdeauna un număr real**, chiar și când rezultatul e rotund: `math.sqrt(9)` dă `3.0`, nu `3`. Argumentul trebuie să fie ≥ 0 — pe un număr negativ, programul se oprește cu eroare.

PODEAUA — MATH.FLOOR(X)

Coboară la cel mai apropiat întreg mai mic sau egal: `math.floor(4.7)` dă `4`, iar `math.floor(-4.2)` dă `-5`. Rezultatul e un întreg, nu un număr real.

TAVANUL — MATH.CEIL(X)

Urcă la cel mai apropiat întreg mai mare sau egal: `math.ceil(4.2)` dă `5`, iar `math.ceil(-4.8)` dă `-4`.

ATENȚIE

Pe numere negative, `math.floor` și `int` **nu** dau același rezultat: `math.floor(-4.2)` dă `-5` (coboară), iar `int(-4.2)` dă `-4` (taie zecimalele). Pe numere pozitive coincid, ceea ce face capcana și mai ușor de ratat.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Aplicațiile de hărți calculează distanța în linie dreaptă dintre două puncte cu exact `math.sqrt` din teorema lui Pitagora. Magazinele online folosesc `math.ceil` ca să afle câte colete sunt necesare — 17 produse în cutii de câte 6 înseamnă 3 cutii, nu 2,83. Iar topografii calculează aria unui teren neregulat descompunându-l în triunghiuri și aplicând formula lui Heron, care conține o rădăcină pătrată.

A

EXERCIȚIUL A · GRILĂ

Rădăcină, podea, tavan

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce afișează `print(math.sqrt(9))`?

- `3.0 doar dacă scrii math.sqrt(9.0) cu argument real la apel`
- `Eroare, pentru că sqrt acceptă doar argumente cu zecimale`
- `3.0, pentru că math.sqrt întoarce întotdeauna un număr real`
- `3, pentru că 9 e pătrat perfect, deci rezultatul iese întreg`

2. Cât este `math.floor(-4.2)`?

- `-5, pentru că rotunjirea în jos merge spre numere tot mai mici`
- `-4, pentru că se șterge pur și simplu tot ce e după punct`
- `4, pentru că rotunjirea în jos elimină și semnul minus`
- `-4.0, pentru că rezultatul păstrează tipul real al argumentului`

3. Ce se întâmplă dacă folosești `math.sqrt(16)` fără linia `import math`?

- `Funcționează, dar rezultatul e întotdeauna rotunjit la un întreg`
- `Se importă automat, pentru că math e folosit în multe programe`
- `Funcționează, pentru că sqrt face parte din limbajul de bază`
- `Programul se oprește cu eroare, pentru că numele math nu e cunoscut`

B

EXERCIȚIUL B · DRY-RUN

Latura pătratului cu două metode

DRY-RUN

Bucla caută, prin încercări, cel mai mare număr al cărui pătrat nu depășește 50. Coloana lat îți e dată. Tu completezi pătratul (numeric) și verdictul condiției (text: DA sau NU). Ultimul rând e cel pe care bucla se oprește.

```
import math
n = 50
lat = 1
while lat * lat <= n:
    lat = lat + 1
lat = lat - 1
print(lat, math.floor(math.sqrt(n)))
```

lat (dat)	lat * lat (completeaza)	lat * lat <= 50? (completeaza: DA / NU)
1	1	DA
2		
3		
4		
5		
6		
7		
8		

HINT *Calculează pe rând lat * lat și compară-l cu 50: cât timp rezultatul e mai mic sau egal cu 50, condiția e adevărată și bucla merge mai departe.*

C

EXERCIȚIUL C · SCRIE CODUL

Aria unui teren triunghiular

SCRIERE COD

Se citesc trei numere naturale a, b și c, fiecare pe câte o linie — lungimile în metri ale laturilor unui teren triunghiular. Scrie un subprogram aria(a, b, c) care întoarce aria terenului, calculată cu formula lui Heron: se ia semiperimetrul p = (a + b + c) / 2, iar aria este rădăcina pătrată din p · (p − a) · (p − b) · (p − c). Afișează valoarea întoarsă, rotunjită la două zecimale. Ex.: pentru a = 3, b = 4, c = 5: p = 6, iar aria = √(6 · 3 · 2 · 1) = √36 = 6.0, deci se afișează 6.0.

HINT *Formula lui Heron e folosită de topografi tocmai pentru că nu are nevoie de înălțime — se măsoară doar cele trei laturi, cu ruleta.*

HINT **Atenție.** *Semiperimetrul se calculează cu /, nu cu //: pentru un perimetru impar, împărțirea întreagă ar pierde jumătatea de metru și aria ar ieși greșită.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 4 · PATRU SUBPROGRAME PENTRU O LISTĂ ÎNTREAGĂ

4 len, min, max, *sum*

Până acum, ca să afli cel mai mare element al unei liste scriai o buclă cu un „campion” care se schimba pe parcurs. Python are deja aceste patru unelte scrise: câte elemente, cel mai mic, cel mai mare, suma tuturor. Media aritmetică devine, dintr-o buclă, o singură expresie.

Cele patru subprograme predefinite pentru colecții primesc o listă și întorc o singură valoare:

```
n = [8, 6, 9, 10]
print(len(n))
print(min(n))
print(max(n))
print(sum(n))
```

Rezultatele sunt, în ordine: `4`, `6`, `10`, `33`. Media se obține direct: `sum(n) / len(n)` dă `8.25`.

DE REȚINUT

`sum` întoarce un întreg dacă toate elementele sunt întregi, dar `sum(l) / len(l)` întoarce **întotdeauna** un număr real, pentru că `/` produce mereu un rezultat real. De aceea media se afișează cu `round(media, 2)`.

ATENȚIE

Pe o listă goală, `sum([])` întoarce liniștit `0`, dar `min([])` și `max([])` **opresc programul cu eroare** — nu au ce compara. Când lista poate fi goală, verifici întâi `len(l) > 0`.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Aplicația de note a școlii calculează media unei materii exact așa: `sum(note) / len(note)`. Cronometrele de la competițiile de atletism afișează cel mai bun timp al seriei cu `min`, pentru că la alergare minimul câștigă. Iar aplicația de pe telefon care îți spune „ai făcut 8 214 pași azi” adună cu `sum` valorile înregistrate în fiecare oră.

A

EXERCIȚIUL A · COMPLETARE

Fiecare subprogram la rolul lui

COMPLETARE

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

`sum` `min` `abs` `max` `len`

Subprogramul _____ întoarce numărul de elemente ale unei liste, _____ întoarce cel mai mic element, iar _____ adună toate elementele într-o singură valoare.

B

EXERCIȚIUL B · SCRIE CODUL

Performanța sportivilor

SCRIERE COD

Se citește un număr natural n (n ≥ 1), apoi n numere reale, fiecare pe câte o linie — timpii, în secunde, obținuți de sportivii unei serii. Afișează pe trei linii separate: cel mai bun timp (cel mai mic), cel mai slab timp (cel mai mare) și media timpilor rotunjită la două zecimale. Folosește subprogramele predefinite pentru colecții. Ex.: pentru n = 3 și timpii 10.2, 9.85, 11.4 se afișează, pe trei linii: 9.85, apoi 11.4, apoi 10.48.

HINT *La alergare „cel mai bun” înseamnă minimul, nu maximul — e o inversare pe care programatorii o greșesc des la prima scriere.*

HINT **Atenție.** *Timpii se citesc cu float(input()); media se afișează rotunjită, dar minimul și maximul se afișează așa cum sunt.*

SOLUȚIA TA · PYTHON

C

EXERCIȚIUL C · SCRIE CODUL

Ce notă îți mai trebuie — exercițiu de performanță

SCRIERE COD

Se citește un număr natural n, apoi n note întregi obținute deja de un elev, fiecare pe câte o linie, apoi un număr întreg M — media pe care elevul vrea s-o atingă. Afișează cea mai mică notă întreagă pe care trebuie s-o ia la următoarea lucrare pentru ca media celor n + 1 note să fie cel puțin M. Dacă nici nota 10 nu ar fi suficientă, afișează IMPOSIBIL. Dacă orice notă ajunge, afișează 1 (nota minimă). Ex.: pentru notele 7, 8, 9 și M = 8: suma celor 4 note trebuie să fie cel puțin 8 · 4 = 32, iar acum suma e 24, deci mai trebuie 8. Se afișează 8.

HINT *Aici nu mai calculezi media, ci pornești de la media dorită și afli ce lipsește — e algoritmul întors pe dos.*

HINT **Atenție.** *Rezultatul poate ieși mai mic decât 1 (dacă elevul stă deja foarte bine) sau mai mare decât 10 (dacă nu mai are cum să recupereze). Ambele situații trebuie tratate explicit.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · TREI ROTUNJIRI, TREI REZULTATE DIFERITE

5 Care rotunjire, în ce *situație*

`round`, `math.floor` și `math.ceil` par să facă același lucru, dar dau rezultate diferite pentru aceeași valoare — iar diferența costă bani. În această secțiune înveți să alegi rotunjirea potrivită și să argumentezi alegerea.

Pentru valoarea 4,3 cele trei subprograme dau: `round(4.3)` → `4`, `math.floor(4.3)` → `4`, `math.ceil(4.3)` → `5`. Pentru 4,7: `5`, `4`, `5`. Diferența apare exact acolo unde contează.

ROTUNJIREA LA CEL MAI APROPIAT — ROUND

Se folosește când rezultatul trebuie să fie cât mai apropiat de valoarea reală, în ambele direcții: valoarea unei facturi, o medie, o temperatură afișată.

ROTUNJIREA ÎN JOS — MATH.FLOOR

Se folosește când depășirea e interzisă: câte produse întregi poți cumpăra cu suma din buzunar, câte porții întregi ies dintr-o cantitate de ingrediente.

ROTUNJIREA ÎN SUS — MATH.CEIL

Se folosește când o fracțiune de unitate consumă totuși o unitate întreagă: câte cutii, câte autobuze, câte pagini, câte bancnote.

ATENȚIE

`round` are două comportamente care surprind. La exact jumătate rotunjește **către numărul par**: `round(0.5)` dă `0`, `round(1.5)` dă `2`, `round(2.5)` dă tot `2`. Iar `round(2.675, 2)` dă `2.67`, nu 2.68, pentru că 2,675 nu se poate scrie exact în binar. Regula practică pentru bani: lucrează în bani (numere întregi) ori de câte ori poți, nu în lei cu zecimale.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Casele de marcat rotunjesc totalul cu `round` la doi bani, iar diferența de un ban înmulțită cu milioane de bonuri e motivul pentru care regula e scrisă în lege. Firmele de curierat calculează numărul de colete cu `ceil` — un colet pe jumătate plin ocupă tot un loc în dubă. Iar un magazin care afișează „câte bucăți poți lua cu 50 de lei” folosește `floor`, pentru că nu-ți poate vinde 3,7 bucăți.

A

EXERCIȚIUL A · ORDONARE

Totalul de plată al unei facturi

ORDONARE

Numerotează de la 1 la 5 pașii prin care un program calculează totalul de plată al unei facturi cu TVA. Ai grijă unde intervine rotunjirea. Pașii îți apar amestecați.

- [ ] Citește prețul fără TVA și cota de TVA, în procente
- [ ] Rotunjește totalul la două zecimale, cu round
- [ ] Calculează valoarea TVA, înmulțind prețul cu cota și împărțind la 100
- [ ] Afișează totalul rotunjit, ca sumă de plată
- [ ] Adună valoarea TVA la prețul fără TVA, obținând totalul exact

B

EXERCIȚIUL B · SCRIE CODUL

Aceeași factură, trei rotunjiri

SCRIERE COD

Se citește un număr real p — prețul unui produs fără TVA, în lei — și un număr întreg c — cota de TVA, în procente. Scrie un subprogram total_cu_tva(pret, cota) care întoarce valoarea p + p · c / 100. Afișează pe trei linii separate: totalul rotunjit la două zecimale cu round, apoi același total rotunjit în jos la lei întregi cu math.floor, apoi rotunjit în sus cu math.ceil. Ex.: pentru p = 12.5 și c = 9: totalul exact e 13,625, deci se afișează 13.62 (round), apoi 13 (floor), apoi 14 (ceil).

HINT *Prima linie e ce apare pe factură, a treia e ce ți-ar cere un magazin care nu dă rest sub un leu.*

HINT **Atenție.** *math.floor și math.ceil se aplică pe totalul nerotunjit, nu pe valoarea deja trecută prin round — altfel a doua rotunjire ar porni de la o valoare modificată.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 6 · TESTAREA SUBPROGRAMELOR: CAZURILE LIMITĂ

6 Ce se întâmplă la *margini*

Un subprogram care merge pe valorile obișnuite poate ceda exact la margini: la zero, la valori negative, pe o listă goală sau pe o ramură pe care ai uitat `return`. Testarea nu e o formalitate de la final — e partea din muncă în care descoperi ce ai scris greșit.

Un **test** înseamnă o pereche: o intrare concretă și rezultatul pe care îl aștepți. Un **caz limită** e o intrare aflată la marginea domeniului: zero, cea mai mică valoare admisă, o valoare negativă, o listă cu un singur element sau fără niciunul.

Cel mai neplăcut caz limită e cel în care subprogramul nu întâlnește niciun `return`:

```
def semn(x):
    if x > 0:
        return 1
    if x < 0:
        return -1
print(semn(0))
```

ATENȚIE

Programul de mai sus afișează `None`. Un subprogram care ajunge la sfârșitul corpului fără să execute vreun `return` întoarce automat valoarea specială `None` — nu zero, nu eroare. `None` se plimbă apoi tăcut prin program și produce o eroare mult mai târziu, într-un loc aparent fără legătură. De aceea fiecare ramură posibilă trebuie să se termine cu un `return`.

DE REȚINUT

Nici subprogramele predefinite nu se comportă la fel la margini: `sum([])` întoarce `0`, dar `min([])` și `max([])` opresc programul cu eroare. Iar `int("abc")` se oprește cu eroare, pentru că textul nu reprezintă un număr.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Aplicațiile bancare sunt testate obligatoriu pe suma zero și pe sume negative, pentru că un transfer de −100 lei ar însemna furt. Programele de salarii se testează pe „brut zero” (angajat aflat în concediu fără plată) și pe luna cu 28 de zile. Iar orice formular de pe internet se testează cu câmpul gol, pentru că cineva **sigur** va apăsa „Trimite” fără să scrie nimic.

A

EXERCIȚIUL A · GRILĂ

Cazurile de la margine

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1.

Un subprogram `semn(x)` întoarce 1 dacă x > 0 și −1 dacă x < 0, dar nu are nicio altă instrucțiune `return`. Ce afișează `print(semn(0))`?

- Eroare de execuție, pentru că lipsește ramura pentru valoarea zero
- `0`, pentru că subprogramul întoarce automat valoarea primită ca parametru
- `None`, pentru că execuția ajunge la sfârșit fără să întâlnească `return`
- Nimic, pentru că un subprogram fără `return` nu produce nicio ieșire

2. Ce se întâmplă la apelul `min([])`, pe o listă goală?

- Întoarce lista goală, pentru că nu are altceva ce să întoarcă
- Programul se oprește cu eroare, pentru că nu există elemente de comparat
- Întoarce 0, pentru că zero e valoarea neutră a oricărei comparații
- Întoarce `None`, la fel ca un subprogram care nu execută niciun `return`

3. De ce setul de teste al unui subprogram trebuie să conțină și cazul zero?

- Pentru că Python cere cel puțin un test cu valoarea zero la fiecare subprogram
- Pentru că zero e cel mai mic număr natural, deci se testează primul dintre toate
- Pentru că multe formule se comportă altfel exact la zero — împărțiri, bucle care nu pornesc
- Pentru că fără el subprogramul nu poate fi apelat niciodată cu valori negative

B

EXERCIȚIUL B · SCRIE CODUL

Salariul net, testat la margini

SCRIERE COD

Se citește un număr întreg b — salariul brut lunar, în lei. Scrie un subprogram salariu_net(brut) care întoarce salariul net, calculat astfel: dacă brutul este zero sau negativ, subprogramul întoarce 0; altfel se reține CAS = brut · 25 // 100, CASS = brut · 10 // 100, se obține baza = brut − CAS − CASS, se calculează impozitul = baza · 10 // 100, iar netul este baza − impozit. Afișează valoarea întoarsă. Ex.: pentru brut = 4000: CAS = 1000, CASS = 400, baza = 2600, impozit = 260, deci netul e 2340 și se afișează 2340.

HINT *Toate reținerile se calculează cu împărțire întreagă (//), pentru că statul lucrează în lei întregi la fiecare treaptă.*

HINT **Atenție.** *Ramura pentru brut ≤ 0 nu e o formalitate: fără ea, un brut negativ ar produce un „net” negativ, iar subprogramul ar întoarce o valoare fără sens în loc să se apere.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 7 · COD GHIDAT

7 Scrie cod care *trece testele*

Cinci probleme. Scrie soluția pe hârtie, urmărește-o mental pe exemplul dat și abia apoi tastează-o pe platformă, unde e verificată automat pe 5 cazuri de test.

1

PROBLEMA 7.1 · COD GHIDAT

Distanța dintre două puncte

SCRIERE COD

Se citesc patru numere întregi, fiecare pe câte o linie: x1, y1, x2, y2 — coordonatele în metri ale două puncte de pe harta unui parc. Scrie un subprogram distanta(x1, y1, x2, y2) care întoarce distanța în linie dreaptă dintre cele două puncte, folosind teorema lui Pitagora. Afișează valoarea întoarsă, rotunjită la două zecimale. Ex.: pentru (0, 0) și (3, 4): distanța = $\sqrt{(3^2 + 4^2)} = \sqrt{25} = 5.0$, deci se afișează 5.0.

HINT *Exact acest calcul îl face aplicația de hărți de pe telefon când îți arată „la 250 m de tine”.*

HINT **Atenție.** *Diferențele pot fi negative (x2 - x1 poate da −3), dar ridicarea la pătrat le face oricum pozitive — nu ai nevoie de abs aici.*

SOLUȚIA TA · PYTHON

2

PROBLEMA 7.2 · COD GHIDAT

Bugetul lunii

SCRIERE COD

Se citește un număr natural n (n ≥ 1), apoi n numere reale, fiecare pe câte o linie — cheltuielile din luna curentă, în lei. Afișează pe trei linii separate: totalul cheltuit, rotunjit la două zecimale; cea mai mare cheltuială; numărul de cheltuieli care depășesc strict media lunii. Ex.: pentru cheltuielile 120.5, 89, 230.25 și 45: totalul e 484.75, cea mai mare e 230.25, media e 121.1875, iar o singură cheltuială (230.25) o depășește, deci se afișează 484.75, apoi 230.25, apoi 1.

HINT *Media se calculează, dar nu se afișează — e doar pragul față de care se compară.*

HINT **Atenție.** *Comparația e strictă: o cheltuială exact egală cu media nu se numără.*

SOLUȚIA TA · PYTHON

3

PROBLEMA 7.3 · COD GHIDAT

Câte cutii de transport

SCRIERE COD

Se citesc două numere naturale, fiecare pe câte o linie: n — numărul de produse de expediat — și c — câte produse încap într-o cutie (c ≥ 1). Scrie un subprogram cutii(n, c) care întoarce numărul minim de cutii necesare. Afișează valoarea întoarsă. Ex.: pentru n = 17 produse și cutii de câte c = 6: 17 / 6 = 2,83, dar produsele rămase au și ele nevoie de o cutie, deci se afișează 3.

HINT *Firmele de curierat calculează exact așa numărul de colete facturate.*

HINT **Atenție.** *Rezultatul se obține cu împărțire reală (/) urmată de math.ceil, nu cu împărțire întreagă — // ar da 2 și ar lăsa cinci produse pe rampă.*

SOLUȚIA TA · PYTHON

4 PROBLEMA 7.4 · REPARĂ CODUL

Codul care afișează None

SCRIERE COD

Codul de mai jos ar trebui să citească două numere întregi și să afișeze pe cel mai mare dintre ele. Pentru intrarea 3 și 8 el afișează însă None, iar pentru 8 și 3 afișează corect 8. Găsește cauza și repară subprogramul, fără să schimbi partea de citire și afișare. Ex.: pentru a = 3 și b = 8, subprogramul intră în corp, condiția 3 > 8 e falsă, nu se execută niciun return, deci apelul întoarce None și se afișează None.

Cod stricat:

```
1  def maxim_doi(a, b):
2      if a > b:
3          return a
4  a = int(input())
5  b = int(input())
6
```

```
7  print(maxim_doi(a, b))
```

HINT **Atenție.** *Ieșirea greșită nu e o eroare de execuție și nu e un număr greșit — e cuvântul None. Exact asta face un return lipsă pe una dintre ramuri, cazul descris în secțiunea despre testare.*

PROGRAMUL CORECTAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Bonul de casă complet

SCRIERE COD

Se citește un număr natural n (n ≥ 1), apoi n numere reale — prețurile produselor de pe bon, în lei, fiecare pe câte o linie — apoi un număr întreg r, procentul de reducere aplicat întregului bon. Afișează pe patru linii separate: totalul înainte de reducere, valoarea reducerii și suma finală de plată, toate rotunjite la două zecimale, apoi numărul minim de bancnote de 50 de lei cu care se poate acoperi suma de plată. Ex.: pentru prețurile 25.5, 40 și 12.25 și o reducere de 10%: totalul e 77.75, reducerea e 7.78 (afișată rotunjit), plata e 69.97, iar bancnotele necesare sunt 2, pentru că 69,97 / 50 = 1,39 și o bancnotă începută tot se dă întreagă.

HINT *Numărul de bancnote se calculează pe suma nerotunjită de plată; rotunjirea de pe ecran e doar pentru afișare.*

HINT **Atenție.** *Reducerea și plata se calculează din totalul exact, nu din totalul deja rotunjit — altfel cele trei linii n-ar mai fi coerente între ele.*

SOLUȚIA TA · PYTHON