test-e.ro

CAIET ELEV

Lecția 7

Elaborarea testelor · validare și depanare

Firul lecției de azi: ***programul tău poate să mintă****. Rulează frumos, nu dă nicio eroare roșie — și totuși afișează un rezultat greșit. Înveți să-l prinzi: cele trei feluri de greșeli, de ce testăm, anatomia unui test bun, vânătoarea de cazuri-limită, depanarea „de mână" (dry-run) și scrisul de cod care* ***trece testele****.*

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

NUMELE ELEVULUI .....

CUM FOLOSEȘTI CAIETUL

Acest caiet adună exercițiile scrise din Lecția 7. Rezolvă-le pe hârtie — repari cod care „minte", recunoști felul fiecărei erori, prezici ce afișează un program, construiești perechi **intrare → rezultat așteptat**, clasifici cazuri-limită, trasezi execuția pas cu pas și scrii programe care rezistă la teste. Folosește doar ce s-a predat până acum (lecțiile 1–6). Regula de aur: **rezultatul așteptat îl calculezi tu, înainte de a rula programul.** Verificarea automată rămâne pe platformă.

CUPRINS

1 Recapitulare prin depanare .....
2 Eroarea care tace .....
3 Anatomia unui test .....
4 Vânătoarea de cazuri-limită .....
5 Depanare „de mână" (dry-run) .....
6 Testează complet · anul bisect .....
7 Scrie cod care trece testele .....
8 Evaluare formativă .....

SECȚIUNEA 1 · RECAPITULARE

1 Recapitulare prin *depanare*

Programul rulează — dar oare spune adevărul? Înainte de a repara ceva, trebuie să știi **ce fel** de greșeală vânezi. Sunt trei:

SINTAXĂ

Nu pornește

Python nu înțelege ce ai scris (lipsă de `:` sau o paranteză). Se oprește imediat, cu mesaj roșu.

LA EXECUȚIE

Pornește, apoi crapă

Se oprește **în timpul rulării** la ceva imposibil: variabilă neexistentă (`NameError`), împărțire la zero (`ZeroDivisionError`).

LOGICĂ

Rulează, dar minte

Merge până la capăt și afișează un rezultat… **greșit**. Nimeni nu te avertizează — cea mai periculoasă.

A

EXERCIȚIUL A · REPARĂ CODUL

Trei programe care mint

DRY-RUN CORECTARE

Fiecare program **rulează fără eroare**, dar afișează un rezultat greșit (eroare **logică**). Fă un dry-run în minte, **încercuiește linia cu bug-ul**, scrie numărul ei și rescrie linia corectată.

Programul 1 — ar trebui să afișeze suma `1 + 2 + … + n`. Pentru `n = 3` corect este `6`.

```
1  n = int(input())
2  s = 1
3  i = 1
4  while i <= n:
5      s = s + i
6      i = i + 1
7  print(s)
```

→ Linia greșită: _____ corectată: _____

Programul 2 — ar trebui să afișeze **câte** numere pare sunt de la 1 la n. Pentru `n = 6` corect este `3`.

```
1  n = int(input())
2  c = 0
3  i = 1
4  while i <= n:
```

```
    if i % 2 == 1:
        c = c + 1
    i = i + 1
print(c)
```

→ Linia greșită: _____ corectată: _____

Programul 3 — ar trebui să afișeze cel mai mare divizor al lui `n` mai mic decât `n` (cu `n > 1`). Pentru `n = 12` corect este `6`.

```
n = int(input())
d = n - 1
while d >= 1:
    if n % d == 0:
        print(d)
        break
    d = d + 1
```

→ Linia greșită: _____ corectată: _____

HINT *la fiecare program întreabă-te: cu ce valoare pornește contorul/acumulatorul și în ce direcție se mișcă? O buclă care nu-și schimbă variabila spre condiția de oprire rulează la nesfârșit.*

B

EXERCIȚIUL B · QUIZ

Țipă sau tace?

GRILĂ

Pentru fiecare situație, încercuiește felul greșelii: **S** = sintaxă (nu pornește), **E** = la execuție (pornește, apoi crapă), **L** = logică (rulează, dar greșește).

1. Ai scris `print(x)` dar nu ai creat niciodată variabila `x` → „NameError". (S) (E) (L)
2. Media a două note adună notele, dar împarte la **3** în loc de 2. Rulează și afișează un număr. (S) (E) (L)
3. Ai uitat `:` la finalul liniei `if n > 0` → „SyntaxError". (S) (E) (L)
4. `print(a / b)` cu `b = 0` → programul se oprește la acea linie cu „ZeroDivisionError". (S) (E) (L)
5. Verifici dacă un număr e par cu `n % 2 == 1`. Rulează perfect, dar răspunde mereu pe dos. (S) (E) (L)

SECȚIUNEA 2 · DE CE TESTĂM

2 Eroarea care *tace*

Cel mai periculos moment e când rulezi programul pe exemplul din enunț, iese exact cât trebuia — și te oprești. „Merge!" Dar a mers **pe acel exemplu**. A testa înseamnă să încerci **tu, dinadins**, valori pe care programul ar putea să nu le prevadă.

A EXERCIȚIUL A · PREZICE IEȘIREA

Ghicești ce afișează?

GRILĂ

Citește cu atenție fiecare cod, execută-l în minte pe intrarea dată și încercuiește ce afișează. Apoi scrie pe scurt **de ce** — unde e capcana.

MEDIE.PY — `a = 4`, `b = 6`

```
a = int(input())
b = int(input())
# media a două numere
m = a + b / 2
print(m)
```

→ Afișează: `5.0` `7.0` `10.0`

De ce? _____

NUMARARE.PY — `n = 5` (ar trebui să numere valorile din [1, n], deci `5`)

```
n = int(input())
c = 0
i = 1
# numărăm câte valori sunt în [1, n]
while i < n:
    c = c + 1
    i = i + 1
print(c)
```

→ Afișează: `4` `5` `6`

De ce? _____

HINT *la MEDIE, gândește-te la* **ordinea operațiilor** *: ce se calculează primul, împărțirea sau adunarea? La NUMARARE, urmărește ultima valoare pe care o ia* `i` *înainte ca bucla să se oprească.*

SECȚIUNEA 3 · ANATOMIA UNUI TEST

3 Intrare → rezultat *așteptat*

Un **test** e o pereche simplă: **ce îi dai** programului (datele de intrare) și **ce ar trebui să-ți răspundă** (rezultatul așteptat).

> **REGULA DE AUR**
>
> Rezultatul așteptat îl calculezi **tu**, cu mintea ta, înainte de a rula programul. Dacă l-ai lua de la program, n-ai mai testa nimic — ai crede orice îți spune. Un set bun de teste acoperă **situații diferite**: la „semnul unui număr" vrei un pozitiv, un negativ și pe zero, fiindcă fiecare merge pe altă ramură.

A

EXERCIȚIUL A · COMPLETEAZĂ TESTELE

Programul „semnul unui număr"

COMPLETARE

Programul citește un întreg `n` și afișează `pozitiv` dacă `n > 0`, `negativ` dacă `n < 0`, `zero` dacă `n == 0`. Pentru fiecare intrare, scrie **rezultatul așteptat** (un singur cuvânt).

INTRARE · N	REZULTAT AȘTEPTAT (SCRII TU)
`7`	_____
`-3`	_____
`0`	_____
`-100`	_____
`1`	_____

B

EXERCIȚIUL B · TESTUL CARE DEMASCĂ

Un singur foc

GRILĂ

Cineva a scris „semnul unui număr", dar are un **bug ascuns**: pe intrarea `0` afișează `pozitiv` în loc de `zero`. Ai un singur test. Un test care „trece" nu prinde nimic — un test bun e cel care **are șansa să pice**. Bifează valoarea de test care demască bug-ul și explică de ce.

_____ *a)* `n = 7`

_____ *b)* `n = -3`

_____ *c)* `n = 0`

d) `n = 5`

De ce doar acest test prinde bug-ul? _____

SECȚIUNEA 4 · CAZURI LIMITĂ

4 Vânătoarea de *cazuri speciale*

Codul rezistă pe valorile obișnuite; cedează la **extreme**: pe zero, pe negative, exact la marginea intervalului, sau când primește ceva neașteptat. Aici stă aproape toată vânătoarea de bug-uri.

PATRU FELURI DE CAZURI

1. **Caz normal** — o valoare obișnuită, „din mijloc".
2. **Caz limită** — exact la graniță: prima sau ultima valoare acceptată, ori zero.
3. **Caz special** — situație rară, pe altă ramură (un singur element, valoarea care nu se împarte etc.).
4. **Date invalide** — ceva ce programul n-ar trebui să accepte deloc.

A

EXERCIȚIUL A · CLASIFICĂ

Clasificatorul de cazuri

ASOCIERE

Problemă: *„se citește un întreg `n` din intervalul [1, 100] și se verifică dacă e prim."* Scrie fiecare valoare de mai jos în coșul potrivit (o valoare poate încăpea în două coșuri).

`42` `100` `1` `0` `-5` `3.5` `200` `2`

CAZ NORMAL	CAZ LIMITĂ
valoare obișnuită, din mijloc	*la graniță · prima/ultima acceptată · zero*
CAZ SPECIAL	**DATE INVALIDE**
rar · pe altă ramură	*n-ar trebui acceptat*

HINT `200` și `-5` *sunt în [1, 100]?* `3.5` *e întreg?* `2` *e singurul prim par (special), iar* `1` *e la margine* **și** *caz special (nu e prim prin definiție).*

B

EXERCIȚIUL B · COMPLETEAZĂ

Ce ar trebui să răspundă?

COMPLETARE

Pentru aceeași problemă („e `n` prim?", cu `n` din [1, 100]), scrie ce ar trebui să răspundă un program **corect** pe fiecare caz. Răspunde cu: `prim`, `nu e prim` sau `invalid`.

INTRARE · N	RĂSPUNS CORECT (SCRII TU)
`1`	_____
`2`	_____
`100`	_____
`0`	_____
`-5`	_____

HINT `1` *nu e prim prin definiție;* `0` *și* `-5` *sunt sub intervalul [1, 100], deci nici nu ar trebui acceptate.*

5 SECȚIUNEA 5 · URMĂRIREA VALORILOR

Depanare „de mână" *(dry-run)*

Ai prins că un program minte. Dar **unde** minte? Faci ce face orice programator: **urmărești valorile variabilelor pas cu pas**, exact cum le-ar calcula calculatorul, și te uiți unde apare prima abatere. Asta e un *dry-run* (rulare „pe hârtie").

A

EXERCIȚIUL A · TRASEAZĂ EXECUȚIA

Unde este greșeala?

DRY-RUN

Programul ar trebui să calculeze **n!** (factorialul) = 1·2·3·…·n. Pentru `n = 4` corect este `24`, dar programul afișează `6`. Trasează execuția pentru `n = 4`: după fiecare trecere prin buclă scrie valorile lui `i` și `p`. Primul rând e completat ca exemplu.

```
1  n = int(input())
2  p = 1
3  i = 1
4  while i < n:
5      p = p * i
6      i = i + 1
```

```
7  print(p)
```

Pas	i	p
înainte de buclă	1	1
după trecerea 1		
după trecerea 2		
după trecerea 3		

→ Programul afișează: _____

→ Bug-ul e pe linia: _____ corectată: _____

HINT *condiția `i < n` oprește bucla când `i` ajunge la `n` — deci ultima valoare cu care se înmulțește nu e niciodată `n` însuși. Ce condiție ar include și `n`?*

SECȚIUNEA 6 · CAZ INTEGRAT

6 Testează *complet* · anul bisect

Punem totul la lucru pe o problemă cu reputație: **anul bisect**. Regula pare simplă, dar are straturi — exact terenul unde testarea superficială se înșală.

REGULA ANULUI BISECT

Un an e bisect dacă e divizibil cu **4**, **dar nu** cu **100** — *cu excepția* anilor divizibili cu **400**, care sunt totuși bisecți. Deci `2016` e bisect, `2019` nu; `2100` **nu** e (÷100, dar nu ÷400), iar `2400` **este** (÷400).

A

EXERCIȚIUL A · ANCHETĂ

Detectivul de bug-uri

COMPLETARE

Un coleg a scris un program de bisect care verifică **doar** dacă anul e divizibil cu 4 (uită de regula lui 100 și 400). Găsește un an care **demască** bug-ul: completează, pentru fiecare an, ce răspunde programul buggy și ce e adevărat. Încercuiește anul la care cele două diferă.

AN	PROGRAM BUGGY (DOAR ÷4)	ADEVĂRUL (REGULA COMPLETĂ)
`2016`		
`1900`		
`2000`		
`2019`		

HINT *programul buggy spune „bisect" pentru orice an ÷4. Bug-ul iese la un an care e ÷4* **și ÷100 dar nu ÷400** *: acolo adevărul spune „nu e bisect", dar programul spune „bisect". Care dintre anii de sus e așa?*

B

EXERCIȚIUL B · ACOPERIRE

Construiește bateria de teste

ASOCIERE

Regula are **patru ramuri**. Un set bun le atinge pe toate, o dată fiecare, fără teste de prisos. Alege **un singur** an pentru fiecare ramură din lista de mai jos și scrie-l pe linie.

`2024` `2012` `1900` `2000` `2023` `2021`

R1 **÷4, dar nu ÷100** → bisect · an ales: _____

R2 **÷100, dar nu ÷400** → nu e bisect · an ales: _____

R3 **÷400** → bisect · an ales: _____

R4 **nu ÷4** → nu e bisect · an ales: _____

HINT *testul minim complet are exact* **4** *ani — câte unul pe ramură. Anii care cad pe aceeași ramură (ex.* `2024` *și* `2012`*) sunt redundanți: unul singur ajunge.*

7

SECȚIUNEA 7 · COD GHIDAT

Scrie cod care *trece testele*

Acum ești de cealaltă parte: tu scrii programul, iar testele îl verifică. Fiecare problemă are 5 cazuri de test alese special — printre ele se ascund cazurile-limită. Înainte să scrii, gândește-te: ce se întâmplă pe valoarea cea mai mică? Pe zero? La marginea intervalului? Afișează **doar** ce se cere, pe o

singură linie, fără text în plus.

1

PROBLEMA 7.1 · VALIDARE

Validează intrarea

SCRIERE COD

Citește un întreg `n` și verifică dacă se află **în intervalul [10, 1000]**. Afișează `valid` dacă da, altfel `invalid`.

HINT *margini: `10` și `1000` sunt valide; `9` și `1001` nu. Dar numerele negative? Testele le vor verifica.*

SOLUȚIA TA · PYTHON

2

PROBLEMA 7.2 · PROVOCARE

Cel mai mare divizor comun

SCRIERE COD

Se citesc două numere naturale `a` și `b`, pe câte o linie. Afișează **cel mai mare divizor comun**. **Atenție:** pot fi și `0`! Dacă unul singur e `0`, rezultatul e celălalt (`cmmdc(12, 0) = 12`); dacă **ambele** sunt `0`, afișează exact `nu exista`.

HINT *testează dinadins cazul în care unul e `0` și cazul în care amândouă sunt `0`. Aici te bazezi pe ce știi din gimnaziu (căutarea divizorilor) — accentul e pe* **cum testezi** *.*

SOLUȚIA TA · PYTHON

3

PROBLEMA 7.3

Suma primelor cifre

SCRIERE COD

Se citesc două numere naturale nenule, `x` și `y`, pe câte o linie. Afișează suma **primei cifre** (cea mai din stânga) a fiecăruia. Exemplu: `x = 347`, `y = 82` → `11` (3 + 8).

HINT *ca să ajungi la prima cifră, împarte repetat la 10 (`// 10`) până rămâne un singur rând de cifre. Afișează* **doar** *numărul cerut, nu `3 + 8 = 11`.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 8 · EVALUARE

8 Evaluare *formativă*

Un recapitulativ scurt, pentru tine: verifică ce ai reținut despre testare, validare și depanare. E formativă — te ajută să vezi unde mai ai de lucru.

A

EXERCIȚIUL A · RECAPITULARE

Adevărat sau fals?

GRILĂ

Încercuiește A (adevărat) sau F (fals).

1. O eroare logică oprește programul cu mesaj roșu. A F
2. Rezultatul așteptat al unui test îl calculezi tu, înainte de a rula programul. A F
3. Cinci teste cu valori „din mijloc" acoperă și cazurile-limită. A F
4. Un dry-run înseamnă să notezi valorile variabilelor pas cu pas, pe hârtie. A F
5. Un test bun e cel care are șansa să **pice**, nu unul care trece sigur. A F

B

EXERCIȚIUL B · REFLECȚIE

Ce ai vânat azi?

NOTIȚE

Alege un program pe care l-ai scris recent și descrie **trei teste** cu care l-ai verifica: un caz normal, un caz-limită și un caz care ar putea să-l pice. Pentru fiecare scrie intrarea și rezultatul așteptat.

CELE TREI TESTE ALE MELE