test-e.ro

CAIET ELEV

Lecția 32

Aplicații integratoare. Proiect final: liste, fișiere, subprograme, interfață grafică

Lecția-proiect „Jurnalul de apă al gospodăriei” adună toate uneltele învățate — liste, subprograme, fișiere text, sortări și ferestre Tkinter — într-o singură aplicație.

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: Nu apare niciun construct nou: setul permis e cel de la L31 (liste și metode, def/return, open/with, tkinter, min/max/sum, slicing). Alegeri deliberate: sortările se scriu de mână (selecția maximului, metoda bulelor, lista de frecvențe), fără sort() și sorted(); fără clase proprii; fără atribuiri simultane — interschimbarea se face cu aux; fără global. Citirea listelor: n pe prima linie, apoi câte o valoare pe linie. Paritatea cu x % 2 == 0. Fișierele se deschid o singură dată, înaintea buclei de scriere. Codul Tkinter se rulează în IDLE/Thonny; în browser se testează doar logica (funcțiile fără ferestre). Fără diacritice în codul Python.

CUPRINS

1 Uneltele din trusă, înainte de construcție .....
2 Un proiect nu se scrie dintr-o bucată .....
3 Datele trăiesc mai mult decât programul .....
4 Din șiruri de litri, informații .....
5 Fereastra din fața jurnalului .....
6 Cine verifică modulul colegului .....
7 Cod ghidat · cinci probleme .....

SECȚIUNEA 1 · NE REAMINTIM UNELTELE PROIECTULUI

1 Uneltele din trusă, înainte de *construcție*

Astăzi nu înveți o unealtă nouă, ci le folosești pe toate odată. Înainte să ne apucăm de proiect, verificăm că fiecare unealtă — lista, subprogramul, fișierul, sortarea, fereastra — îți stă la îndemână.

În ultimele lecții ai strâns o trusă întreagă. O **listă** păstrează oricâte valori sub un singur nume: `v.append(x)` adaugă la final, `len(v)` spune câte sunt, `v[i]` dă elementul de pe poziția `i` (numerotarea începe de la 0). Un **subprogram** (`def`) primește date prin parametri și dă înapoi un rezultat cu `return`. Un **fișier text** se deschide cu `open(nume, mod)`: `'r'` pentru citire, `'w'` pentru scriere de la zero, `'a'` pentru adăugare la final; construcția `with` îl închide automat. Pentru **sortare** ai trei metode: selecția minimului (sau maximului), metoda bulelor și lista de frecvențe — ultima doar când valorile stau într-un interval mic. Iar în **Tkinter**, un `Button` cheamă funcția dată la `command=` de fiecare dată când e apăsat.

Citirea unei liste o facem mereu la fel: întâi `n`, apoi `n` valori, câte una pe linie.

```
n = int(input())
v = []
for i in range(n):
    v.append(int(input()))
```

DE REȚINUT

Metoda bulelor compară **vecini**: `v[i]` cu `v[i + 1]`. Dacă sunt în ordinea greșită, îi interschimbă printr-o variabilă auxiliară. După o singură trecere completă, cel mai mare element a ajuns sigur pe ultima poziție.

A EXERCIȚIUL A · GRILĂ

Ce știu deja

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Lista v = [3, 5]. Ce conține v după v.append(7)? `[3, 7]` `[7, 3, 5]` `[10]` `[3, 5, 7]`

2. Ce face modul 'a' în open('apa.txt', 'a')?

   `Creează o copie de rezervă a fișierului înainte de a-l modifica`

   `Adaugă ce scrii la sfârșitul fișierului, fără să șteargă ce era deja`

   `Deschide fișierul doar pentru citire, iar orice write produce eroare`

Șterge tot conținutul vechi al fișierului și scrie de la prima linie

3. Ai de sortat notele (numere întregi de la 1 la 10) a 1000 de elevi. Ce metodă e cea mai potrivită?

Selecția minimului, pentru că face mereu cele mai puține comparări

Lista de frecvențe, pentru că valorile stau într-un interval mic

Nicio metodă, pentru că notele nu se pot sorta fără sorted()

Metoda bulelor, pentru că se oprește devreme când lista e deja ordonată

4. În tk.Button(fereastra, text='Calculeaza', command=calculeaza), ce rol are command=calculeaza?

Creează o variabilă nouă numită calculeaza în interiorul butonului

Schimbă textul afișat pe buton în numele funcției calculeaza

Leagă funcția de buton: ea va fi apelată la fiecare apăsare

Execută funcția o singură dată, chiar în momentul creării butonului

B

EXERCIȚIUL B · DRY-RUN

O trecere a metodei bulelor

DRY-RUN

Urmărește o singură trecere a metodei bulelor. Valoarea lui i este dată; tu completezi ce valori compară programul (v[i] și v[i+1] înainte de eventuala interschimbare) și dacă face schimbul (DA/NU).

```
v = [7, 3, 9, 2]
i = 0
while i < len(v) - 1:
    if v[i] > v[i + 1]:
        aux = v[i]
        v[i] = v[i + 1]
        v[i + 1] = aux
    i = i + 1
```

i (dată)	v[i]	v[i+1]	schimb?	lista după pas (dată)
0	7	3	DA	[3, 7, 9, 2]
1				[3, 7, 9, 2]
2				[3, 7, 2, 9]
3	–	–	–	[3, 7, 2, 9]

HINT *La fiecare pas compari v[i] cu v[i+1] așa cum sunt înainte de schimb (ia-le din lista de pe rândul anterior) și scrii DA doar dacă v[i] > v[i+1].*

SPRIJIN · SCRIE CODUL

Suma valorilor dintr-o listă

SCRIERE COD

Se citește un număr natural n, apoi n numere naturale, câte unul pe linie. Memorează numerele într-o listă și afișează suma lor. Ex.: pentru n = 4 și valorile 10, 0, 25, 5 se afișează 40.

HINT *Dacă n = 0, lista rămâne goală și suma este 0.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

Câte valori pare?

SCRIERE COD

Scrie funcția numara_pare(v), care primește o listă de numere naturale și returnează câte dintre ele sunt pare. Programul citește n, apoi n numere naturale, câte unul pe linie, și afișează valoarea returnată de funcție. Ex.: pentru n = 5 și valorile 3, 8, 0, 7, 12 se afișează 3 (numerele pare sunt 8, 0 și 12).

HINT **Atenție.** *0 este număr par: 0 % 2 == 0. Funcția returnează numărul, iar programul principal îl afișează direct: print(numara_pare(v)).*

SOLUȚIA TA · PYTHON

2 SECȚIUNEA 2 · MODULUL DE PORNIRE: CUM ÎMPARȚI O APLICAȚIE

Un proiect nu se scrie dintr-o *bucată*

O aplicație de 200 de linii scrisă „de sus până jos” e greu de citit, greu de reparat și imposibil de împărțit pe echipă. Programatorii o taie în module — bucăți mici, fiecare cu treaba ei. Astăzi construim așa „Jurnalul de apă al gospodăriei”.

Proiectul nostru ține evidența litrilor de apă consumați zilnic de o gospodărie: utilizatorul introduce consumul zilei, aplicația îl salvează, calculează totalul și media, arată zilele de vârf și avertizează când consumul sare de o limită. Îl împărțim în patru module, fiecare cu **o singură responsabilitate**:

- **Date** — lista `consum`, câte o valoare (în litri) pentru fiecare zi;
- **Fișier** — `salveaza(v, nume)` și `incarca(nume)`: datele nu se pierd la închiderea programului;
- **Prelucrare** — `total(v)`, `medie_intreaga(v)`, `zi_maxima(v)`, `sorteaza_desc(v)`, `clasifica(x)`;
- **Interfață** — fereastra Tkinter: casetă de text, buton, etichetă cu totalul, listă cu istoricul.

Iată scheletul aplicației — antetele funcțiilor și ce face fiecare. Corpul lipsă e marcat cu `...`, adică „aici urmează codul”: Python acceptă scheletul, dar funcțiile nu fac încă nimic.

```python
# modulul Fisier
def salveaza(v, nume):      # scrie lista in fisier, cate o valoare pe linie
    ...
def incarca(nume):          # citeste fisierul si returneaza lista
    ...

# modulul Prelucrare
def total(v):               # returneaza suma valorilor
    ...
def medie_intreaga(v):      # returneaza total(v) // len(v)
    ...
def zi_maxima(v):           # returneaza numarul zilei cu consum maxim
    ...
def sorteaza_desc(v):       # ordoneaza lista descrescator
    ...

# modulul Interfata
def adauga():               # apelata de butonul Adauga
    ...
```

SEPARAREA LOGICII DE INTERFAȚĂ

Funcțiile de prelucrare **nu** conțin `input()` și nici `print()`: primesc datele prin parametri și dau rezultatul prin `return`. Doar modulul de interfață (sau programul principal) vorbește cu utilizatorul. Câștigi trei lucruri: poți testa fiecare funcție separat, poți schimba interfața (consolă sau fereastră) fără să atingi calculele, iar fiecare coleg din echipă poate lucra la modulul lui fără să-l strice pe al altuia.

GREȘEALA FRECVENTĂ

O funcție `total(v)` care face `print(s)` în loc de `return s` pare să meargă — vezi numărul pe ecran. Dar interfața nu-l poate prelua ca să-l pună în etichetă, iar altă funcție (de exemplu `medie_intreaga`) nu-l poate folosi. Funcția returnează; cine o apelează decide ce face cu rezultatul.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Aplicația de pe telefon a unei companii de apă citește datele contorului inteligent (modulul de date), le păstrează pe server (fișiere, baze de date), calculează consumul lunar (prelucrare) și îl afișează în grafice (interfață) — patru echipe diferite lucrează la cele patru părți. La fel, o brățară de fitness are un modul care citește senzorul, altul care numără pașii și altul care îi desenează pe ecran. Iar jocurile mari separă „motorul” de joc de meniurile pe care le vezi: pot schimba complet meniurile fără să rescrie fizica.

A

EXERCIȚIUL A · ASOCIERE

Fiecare bucată la modulul ei

ASOCIERE

Scrie în căsuța din fața fiecărui element din stânga litera variantei potrivite. Fiecare literă se folosește o singură dată.

- ☐ consum = []
- ☐ salveaza(v, nume)
- ☐ medie_intreaga(v)
- ☐ adauga() — apelată de buton

A Modulul Date | B Modulul Prelucrare | C Modulul Fișier | D Modulul Interfață

B

EXERCIȚIUL B · ORDONARE

Ce se întâmplă când apeși „Adaugă”

ORDONARE

Numerotează pașii de la 1 la 6, în ordinea corectă. Pașii îți apar amestecați.

- ☐ Utilizatorul scrie consumul zilei în caseta de text și apasă butonul „Adaugă”.
- ☐ Numărul este adăugat la finalul listei consum cu append().
- ☐ Textul este verificat și, dacă e valid, transformat în număr cu int().
- ☐ Eticheta afișează noul total, calculat de funcția total(consum).
- ☐ Butonul apelează funcția legată prin command, care preia textul cu get().
- ☐ Lista actualizată este salvată în fișierul apa.txt.

C

EXERCIȚIUL C · SCRIE CODUL

Primele funcții de prelucrare: total și medie

SCRIERE COD

Scrie funcțiile total(v), care returnează suma elementelor listei v, și medie_intreaga(v), care returnează partea întreagă a mediei aritmetice, folosind total(v). Programul citește n (n ≥ 1), apoi n consumuri zilnice în litri, câte unul pe linie, și afișează pe prima linie totalul, iar pe a doua linie media întreagă. Ex.: pentru n

= 3 și consumurile 300, 420, 360 se afișează 1080 pe prima linie și 360 pe a doua.

HINT **Atenție.** *Funcțiile nu conțin print și nici input — ele doar returnează. Afișarea o face programul principal, direct din ce returnează funcțiile: print(total(v)) și print(medie_intreaga(v)).*

SOLUȚIA TA · PYTHON

SECȚIUNEA 3 · MODULUL FIȘIER: JURNALUL NU SE PIERDE LA ÎNCHIDERE

3 Datele trăiesc mai mult decât *programul*

O listă dispare în clipa în care închizi programul. Ca jurnalul de apă să-și amintească și mâine ce ai scris azi, modulul Fișier îl salvează pe disc și îl reîncarcă la nevoie.

Stabilim întâi **formatul fișierului**: `apa.txt` conține câte o valoare pe linie, în ordinea zilelor. Un format simplu și fix e un contract între colegi: cine scrie modulul de salvare și cine scrie modulul de încărcare trebuie să respecte același format, altfel datele nu se mai potrivesc.

Salvarea parcurge lista și scrie fiecare valoare, transformată în text, urmată de caracterul de linie nouă:

```python
def salveaza(v, nume):
    with open(nume, 'w') as f:
        for x in v:
            f.write(str(x) + '\n')
```

Încărcarea face drumul invers: `readlines()` returnează o listă de **șiruri de caractere** (`['300\n', '0\n', '250\n']`), iar `int()` le transformă în numere — spațiile și caracterul `'\n'` de la capăt sunt ignorate de `int()`.

```python
def incarca(nume):
    v = []
    with open(nume, 'r') as f:
        for linie in f.readlines():
            v.append(int(linie))
    return v
```

DE REȚINUT

Modul `'w'` golește fișierul în momentul deschiderii și scrie de la zero; modul `'a'` păstrează ce era și scrie la sfârșit. Pentru jurnal ai două strategii corecte: fie rescrii toată lista cu `'w'` după fiecare modificare, fie adaugi doar valoarea zilei cu `'a'`. Atenție la prima rulare: dacă fișierul nu există încă, `open(nume, 'r')` dă eroare. În secțiunea despre interfață vei vedea cum îl creăm gol, cu modul `'a'`, înainte de citire.

CAPCANA DESCHIDERII ÎN BUCLĂ

Dacă pui `with open(nume, 'w')` **în interiorul** buclei `for`, fișierul se golește la fiecare pas și rămâne în el doar ultima valoare. Deschide fișierul **o singură dată**, înaintea buclei, și scrie toate valorile în interiorul aceluiași `with`.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Stațiile automate de monitorizare a aerului și contoarele inteligente scriu măsurătorile în fișiere-jurnal, linie cu linie, câte o linie la câteva minute. Când exporți din aplicația băncii extrasul de cont „în CSV”, primești tot un fișier text cu câte o tranzacție pe linie. Iar jocurile salvează progresul într-un fișier ca să-l poți continua a doua zi.

A

EXERCIȚIUL A · GRILĂ

Fișiere, pe scurt

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Fișierul b.txt conține trei linii: 300, 0, 250. Ce returnează f.readlines()?

- `Doar prima linie, '300\n', fără celelalte două`
- `Un singur șir de caractere cu tot conținutul: '300\n0\n250\n'`
- `Lista de numere [300, 0, 250], gata de adunat`
- `Lista de șiruri ['300\n', '0\n', '250\n']`

2. Fișierul a.txt există și conține 10 linii. Ce se întâmplă cu ele după with open('a.txt', 'w') as f: f.write('5\n')?

- `Sunt șterse la deschidere; fișierul conține doar linia 5`
- `Programul se oprește cu eroare, pentru că fișierul nu era gol`
- `Rămân toate, iar linia 5 se adaugă la sfârșit, ca a unsprezecea`
- `Prima linie este înlocuită cu 5, iar celelalte nouă rămân neschimbate`

3. De ce preferăm with open(...) as f: în locul lui f = open(...)?

- `Pentru că with închide automat fișierul la ieșirea din bloc`
- `Pentru că with transformă automat fiecare linie citită în număr`
- `Pentru că fără with nu se pot folosi modurile 'w' și 'a'`
- `Pentru că with citește fișierul mai repede decât open simplu`

B

EXERCIȚIUL B · COMPLETARE

Cuvintele-cheie ale fișierelor

COMPLETARE

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

`read` `readlines` `'r'` `'w'` `'a'` `with`

Construcția _____ open(...) as f închide automat fișierul. Modul _____ golește fișierul și scrie de la zero, modul _____ adaugă la sfârșit, iar metoda _____ returnează toate liniile sub forma unei liste de șiruri.

C

EXERCIȚIUL C · SCRIE CODUL

Salvează și reîncarcă jurnalul

SCRIERE COD

Se citește n, apoi n consumuri zilnice în litri, câte unul pe linie. Scrie valorile în fișierul apa.txt, câte una pe linie. Apoi redeschide fișierul pentru citire, citește toate liniile și afișează pe prima linie câte linii ai citit, iar pe a doua linie suma valorilor citite din fișier. Ex.: pentru n = 3 și consumurile 300, 0, 250, fișierul va avea 3 linii, deci se afișează 3 și apoi 550.

HINT **Atenție.** *Deschide fișierul pentru scriere o singură dată, înaintea buclei — altfel rămâne în el doar ultima valoare.*

HINT *Pentru n = 0, fișierul rămâne gol: se afișează 0 și 0.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 4 · MODULUL PRELUCRARE: CE NE SPUN CIFRELE DIN JURNAL

4 Din șiruri de litri, *informații*

Un jurnal cu 30 de numere nu spune mare lucru dacă doar îl privești. Modulul Prelucrare îl transformă în răspunsuri: cât s-a consumat în total, în ce zi a fost vârful și care au fost zilele cele mai risipitoare.

Modulul Prelucrare adună algoritmii pe care îi știi deja și îi pune la lucru pe datele jurnalului. **Totalul** e o buclă-acumulator; **media întreagă** e `total(v) // len(v)`. **Ziua cu consum maxim** se află cu un algoritm de căutare a maximului care ține minte și poziția. Atenție la numerotare: pentru calculator prima zi e la poziția 0, pentru om e „ziua 1” — de aceea salvăm `i + 1`.

```python
def zi_maxima(v):
    maxim = v[0]
    zi = 1
    i = 1
    while i < len(v):
        if v[i] > maxim:
            maxim = v[i]
            zi = i + 1
        i = i + 1
    return zi
```

Pentru **clasamentul zilelor** sortăm lista descrescător. Alegem metoda după date: consumurile în litri pot lua sute de valori diferite, iar un jurnal are cel mult câteva zeci de zile, deci o metodă O(n²) — selecția maximului sau metoda bulelor — e perfect potrivită. Lista de frecvențe ar cere un vector de frecvențe cât cea mai mare valoare posibilă (mii de poziții) pentru doar câteva zeci de valori.

SELECȚIA MAXIMULUI (SORTARE DESCRESCĂTOARE)

La pasul `i` cauți poziția celui mai mare element din porțiunea `v[i], v[i+1], ..., v[n-1]` și îl interschimbi cu `v[i]`. După pasul `i`, primele `i + 1` poziții conțin deja cele mai mari valori, în ordine.

ALEGEREA METODEI DE SORTARE

Valori întregi pe un interval mic (note 1–10, niveluri de poluare 0–5) → lista de frecvențe, O(n + k). Valori oarecare, listă scurtă → selecția sau metoda bulelor, O(n²). Metoda bulelor păstrează ordinea inițială a valorilor egale — util când sortezi apartamente după consum și vrei ca la egalitate să rămână în ordinea numerelor lor. Dacă intervalul mic conține și valori negative (de exemplu temperaturi de la −20 la 40 de grade), lista de frecvențe se folosește în continuare: valoarea `t` se numără la poziția `t + 20`, deci ajung 61 de poziții.

STRICT MAI MARE, NU MAI MARE SAU EGAL

Cu `v[i] > maxim`, la egalitate rămâne **prima** zi cu consum maxim. Cu `>=`, ar rămâne ultima. Enunțul spune care se cere — citește-l.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Factura de utilități îți arată consumul mediu zilnic și luna cu consum maxim — exact calculele din acest modul. Aplicațiile de monitorizare a calității aerului îți arată „ora cu cea mai mare poluare” din ziua respectivă. Iar clasamentele de tip „top 5 zile cele mai călduroase ale verii” sunt o sortare descrescătoare urmată de afișarea primelor valori.

A

EXERCIȚIUL A · DRY-RUN

În ce zi a fost vârful?

DRY-RUN

Urmărește căutarea zilei cu consum maxim. Coloanele i și v[i] sunt date; tu completezi valorile lui maxim și zi după ce s-a executat corpul buclei pentru acel i.

```
v = [320, 410, 290, 410, 450]
maxim = v[0]
zi = 1
i = 1
while i < len(v):
    if v[i] > maxim:
        maxim = v[i]
        zi = i + 1
    i = i + 1
```

i (dată)	v[i] (dată)	maxim	zi
–	–	320	1
1	410	410	2
2	290		
3	410		
4	450		
5	–	450	5

HINT *maxim și zi se schimbă doar când v[i] este strict mai mare decât maxim — atunci zi devine i + 1. Altfel rămân la fel ca pe rândul de deasupra.*

B

EXERCIȚIUL B · SCRIE CODUL

Zilele de vârf, în ordine

SCRIERE COD

Scrie funcția sorteaza_desc(v), care ordonează descrescător lista v prin metoda selecției maximului. Programul citește n (n ≥ 1), apoi n consumuri zilnice, câte unul pe linie, apelează funcția și afișează valorile ordonate pe un singur rând, separate prin spațiu. Ex.: pentru n = 5 și consumurile 320, 410, 290, 0, 380 se afișează 410 380 320 290 0.

HINT *Scrie sortarea de mână, fără sort() sau sorted() — scopul e să folosești metoda învățată.*

HINT **Atenție.** *Interschimbarea se face cu o variabilă auxiliară, în trei instrucțiuni separate:*

SOLUȚIA TA · PYTHON

PERFORMANȚĂ · SCRIE CODUL

Cât mai am voie să consum mâine?

SCRIERE COD

Familia și-a propus ca media consumului zilnic să nu depășească m litri. Se citește n (n ≥ 0), numărul de zile trecute, apoi n consumuri, câte unul pe linie, apoi m. Afișează câți litri poate consuma familia, cel mult, în ziua următoare, astfel încât media celor n + 1 zile să nu depășească m. Dacă obiectivul este deja ratat, afișează 0. Ex.: pentru consumurile 300, 420, 360 (total 1080) și m = 350: în 4 zile sunt permiși 4 · 350 = 1400 de litri, deci mâine se pot consuma cel mult 1400 − 1080 = 320 de litri.

HINT *De data asta nu calculezi media dintr-un total, ci faci drumul invers: pornești de la media dorită și afli valoarea care lipsește.*

HINT **Curiozitate.** *Același raționament îl folosești când afli ce notă îți trebuie la ultima lucrare ca să ai media dorită.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · MODULUL INTERFAȚĂ: BUTOANE CARE CHEAMĂ FUNCȚII

5 Fereastra din fața *jurnalului*

Utilizatorul nu vrea să scrie în consolă. Vrea o casetă, un buton „Adaugă” și un total care se actualizează singur. Modulul Interfață face exact asta — și, dacă e bine construit, doar asta.

Fereastra jurnalului are cinci piese: o etichetă cu instrucțiunea, o casetă `Entry` în care se tastează consumul, butonul „Adaugă”, o etichetă cu totalul și un `Listbox` cu istoricul zilelor. Toată „inteligența” stă în funcția `adauga()`, legată de buton — iar ea e scurtă, pentru că **cheamă** funcțiile din celelalte module în loc să calculeze singură.

Aplicația completă (rulează în IDLE sau Thonny pe calculatorul tău — în browser, Tkinter nu poate deschide ferestre):

```python
import tkinter as tk
from tkinter import messagebox

def incarca(nume):
    with open(nume, 'a') as f:    # creeaza fisierul gol daca nu exista inca
        pass
    v = []
    with open(nume, 'r') as f:
        for linie in f.readlines():
            v.append(int(linie))
    return v

def total(v):
    s = 0
    for x in v:
        s = s + x
    return s

def valoare_valida(text):
    if len(text) == 0:
        return False
    for c in text:
        if ord(c) < 48 or ord(c) > 57:
            return False
    return int(text) <= 2000

def salveaza(v, nume):
    with open(nume, 'w') as f:
        for x in v:
            f.write(str(x) + '\n')

def adauga():
    text = caseta.get()
    if valoare_valida(text):
        x = int(text)
        consum.append(x)
        salveaza(consum, 'apa.txt')
        istoric.insert(tk.END, 'Ziua ' + str(len(consum)) + ': ' + text + ' litri')
        eticheta.config(text='Total: ' + str(total(consum)) + ' litri')
        if x > 500:
            messagebox.showwarning('Atentie', 'Consum ridicat azi!')
```

```
    else:
        messagebox.showerror('Eroare', 'Scrie un numar intre 0 si 2000')
    caseta.delete(0, tk.END)

fereastra = tk.Tk()
fereastra.title('Jurnalul de apa')
tk.Label(fereastra, text='Litri consumati azi:').pack()
caseta = tk.Entry(fereastra)
caseta.pack()
tk.Button(fereastra, text='Adauga', command=adauga).pack()
eticheta = tk.Label(fereastra, text='Total: 0 litri')
eticheta.pack()
istoric = tk.Listbox(fereastra, width=30)
istoric.pack()

consum = incarca('apa.txt')       # jurnalul salvat la rularile anterioare
for i in range(len(consum)):
    istoric.insert(tk.END, 'Ziua ' + str(i + 1) + ': ' + str(consum[i]) + ' litri')
eticheta.config(text='Total: ' + str(total(consum)) + ' litri')

fereastra.mainloop()
```

DE CE PORNIM CU INCARCA?

Dacă programul ar începe cu `consum = []`, la prima apăsare pe „Adaugă” funcția `salveaza` ar deschide `apa.txt` în modul `'w'` și ar scrie doar valoarea nouă: toate zilele salvate ieri s-ar pierde. De aceea, înainte de `mainloop()`, aplicația reîncarcă jurnalul din fișier, completează istoricul și afișează totalul. Prima linie din `incarca` deschide fișierul în modul `'a'` doar ca să-l creeze, gol, la prima rulare — altfel `open(nume, 'r')` ar da eroare pentru un fișier inexistent.

Observă că `valoare_valida`, `total`, `incarca` și `salveaza` nu știu nimic despre fereastră: le poți copia într-un program de consolă și merg la fel. Tocmai de aceea pe `valoare_valida` o poți scrie și testa chiar aici, în browser.

DE REȚINUT

`caseta.get()` returnează mereu **text** (tip `str`), chiar dacă utilizatorul a tastat cifre. Înainte de `int(text)` verifici că textul chiar e un număr — altfel, la un „3a5” tastat din greșeală, programul se oprește cu eroare.

FĂRĂ PARANTEZE LA COMMAND

Scrii `command=adauga`, nu `command=adauga()`. Cu paranteze, funcția s-ar executa o singură dată, la crearea butonului, iar la clic nu s-ar mai întâmpla nimic.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Orice formular online face ce face `valoare_valida`: câmpul „cod poștal” îți refuză literele, câmpul „vârstă” refuză valorile imposibile — înainte ca datele să ajungă la server. Termostatul inteligent din casă are aceeași structură: ecranul cu butoane doar trimite comenzi, iar un modul separat decide când pornește centrala. Iar aplicația de la ATM te avertizează („sumă peste limita zilnică”) exact ca `messagebox.showwarning` din jurnalul nostru.

A

EXERCIȚIUL A · ASOCIERE

Piesele ferestrei

ASOCIERE

Leagă fiecare componentă Tkinter de rolul ei în fereastra jurnalului, apoi apasă „Verifică”. Scrie în căsuța din fața fiecărui element din stânga litera variantei potrivite. Fiecare literă se folosește o singură dată.

☐	Entry
☐	Button
☐	Listbox
☐	messagebox.showwarning

A `Arată istoricul, câte o zi pe fiecare linie`

B `Caseta în care utilizatorul tastează consumul zilei`

C `Declanșează funcția legată prin command la fiecare clic`

D `Deschide o fereastră de avertizare pentru consum ridicat`

B

EXERCIȚIUL B · GRILĂ

Cum comunică fereastra cu codul

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Utilizatorul tastează 350 în caseta caseta. Ce tip are valoarea returnată de caseta.get()?

`Șir de caractere (str), care trebuie convertit cu int()`

`Valoarea logică True, pentru că, în casetă, există un text`

- Număr întreg (int), pentru că utilizatorul a tastat doar cifre
- Listă cu trei elemente, câte unul pentru fiecare cifră tastată

2. Ce face istoric.insert(tk.END, 'Ziua 3: 420 litri')?

- Șterge ultimul rând din listă și îl mută la începutul ei
- Înlocuiește tot conținutul listei cu un singur rând nou
- Scrie rândul în fișierul apa.txt, fără să-l afișeze în fereastră
- Adaugă un rând nou la sfârșitul listei afișate în fereastră

3. De ce funcția valoare_valida(text) poate fi testată în browser, deși aplicația folosește Tkinter?

- Pentru că nu folosește nicio componentă Tkinter: primește text și returnează True sau False
- Pentru că are return, iar funcțiile cu return rulează oriunde, chiar și cu ferestre
- Pentru că e scrisă înainte de mainloop(), iar acolo Tkinter e încă inactiv
- Pentru că browserul poate rula orice fereastră Tkinter dacă e mică

C

EXERCIȚIUL C · SCRIE CODUL

Logica din spatele butonului: validarea

SCRIERE COD

Scrie funcția valoare_valida(text), care returnează True dacă textul primit reprezintă un consum acceptat și False în caz contrar. Un consum e acceptat dacă textul nu e gol, conține numai cifre și numărul format nu depășește 2000. Programul citește o linie de text și afișează pe același rând textul citit, un spațiu, apoi DA dacă e acceptat sau NU în caz contrar. Ex.: pentru textul 350 se afișează 350 DA; pentru textul 3a5 se afișează 3a5 NU.

HINT *O cifră are codul ASCII între 48 ('0') și 57 ('9'). Verifică fiecare caracter cu ord() înainte să folosești int(text).*

HINT **Atenție.** *Nu transforma textul în număr la citire (int(input())): la 3a5 programul s-ar opri cu eroare. Citește-l ca text, cu input(). Validarea e chiar despre cazurile de la margine, așa că aici trebuie să treacă toate cele 5 teste.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 6 · TESTARE, DOCUMENTARE ȘI EVALUARE ÎN ECHIPĂ

6 Cine verifică modulul *colegului*

Modulul tău merge pe exemplul din enunț. Dar merge și când familia a fost plecată în vacanță (consum 0)? Și exact la limita de 300 de litri? Înainte să predați proiectul, fiecare modul trece prin teste — ale voastre și ale colegilor.

Un modul se consideră terminat abia după ce a fost **testat separat**. Nu e de ajuns să încerci o singură valoare: alegi teste din fiecare categorie, pentru că erorile se ascund mai ales la margini.

> **CELE PATRU TIPURI DE TESTE**
>
> (1) **Caz obișnuit** — o valoare tipică (`clasifica(150)`). (2) **Caz limită** — exact pe prag și imediat după el (`clasifica(300)` și `clasifica(301)`); aici se văd confuziile între `<` și `<=`. (3) **Caz zero / listă goală** — `total([])` , un consum de 0 litri, un fișier gol. (4) **Date invalide** — ce tastează utilizatorul din greșeală (`"3a5"` , `"-40"` , textul gol).

Documentarea are două niveluri. În cod, fiecare funcție primește deasupra un comentariu scurt (fără diacritice) care spune ce primește și ce returnează. În afara codului, echipa scrie un document de prezentare: scopul aplicației, tema transversală aleasă și de ce contează, formatul fișierului de date, cum se pornește aplicația și cine a scris fiecare modul.

```
# primeste: x = consumul unei zile, in litri
# returneaza: 'ECONOMIC', 'NORMAL' sau 'RIDICAT'
def clasifica(x):
    ...
```

Lucrul în echipă urmează împărțirea pe module: un coleg scrie modulul Fișier, altul Prelucrarea, altul Interfața. Funcționează doar dacă antetele funcțiilor (numele și parametrii) sunt stabilite **împreună, la început** — ele sunt contractul dintre voi. La **coevaluare**, fiecare echipă verifică proiectul alteia după trei

criterii: *corectitudine* (trece testele din toate cele patru categorii?), *independența modulelor* (poți înlocui interfața fără să modifici prelucrarea?) și *claritate* (se înțelege codul fără explicații verbale?). Feedbackul bun e concret: „`medie_intreaga` dă eroare pe lista goală” ajută; „e ok” nu ajută.

TEMA TRANSVERSALĂ

Conform Legii 198/2023 (art. 88 alin. 10), proiectul se leagă de o temă cu relevanță socială. Fiecare echipă alege una: educație pentru mediu (consumul de apă sau nivelul de poluare — ca exemplul nostru), digitalizare (catalog școlar electronic), sănătate (jurnal de activitate fizică sau de alimentație) sau patrimoniu (catalog de obiective turistice din județul Prahova). La prezentarea din lecția următoare veți argumenta de ce aplicația voastră contează.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

În firmele de software, fiecare modificare a codului pornește automat sute de teste înainte să ajungă la utilizatori. Pe platformele de găzduire a codului, codul unui programator este citit și comentat de un coleg înainte de a fi acceptat („code review”) — exact coevaluarea voastră. Iar aplicațiile mari au documentație separată, pentru ca un programator nou venit în echipă să poată lucra din prima zi.

A

EXERCIȚIUL A · ASOCIERE

Ce fel de test e acesta?

ASOCIERE

Leagă fiecare apel de test de categoria din care face parte, apoi apasă „Verifică”. Pragurile de clasificare sunt 300 și 500 de litri. Scrie în căsuța din fața fiecărui element din stânga litera variantei potrivite. Fiecare literă se folosește o singură dată.

- ☐ clasifica(150)
- ☐ clasifica(300)
- ☐ total([])
- ☐ valoare_valida("3a5")

A Caz obișnuit · B Date invalide · C Caz limită · D Caz zero / listă goală

B

EXERCIȚIUL B · SCRIE CODUL

Clasificarea unei zile, testată la limite

SCRIERE COD

Scrie funcția clasifica(x), care primește consumul unei zile, în litri, și returnează ECONOMIC dacă x ≤ 300, NORMAL dacă 300 < x ≤ 500 și RIDICAT dacă x > 500. Programul citește x și afișează pe același rând valoarea citită, un spațiu și categoria. Ex.: pentru x = 300 se afișează 300 ECONOMIC, iar pentru x = 301 se afișează 301 NORMAL.

HINT **Atenție.** *Testele verifică exact valorile de pe prag (300, 500) și imediat de după ele (301, 501). O confuzie între < și <= pică tocmai aici. De aceea, la acest exercițiu trebuie să treacă toate cele 5 teste.*

SOLUȚIA TA · PYTHON

7 SECȚIUNEA 7 · COD GHIDAT

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

Zile peste medie

SCRIERE COD

Se citește n (n ≥ 1), apoi n consumuri zilnice în litri, câte unul pe linie. Folosind o funcție total(v), afișează câte zile au avut un consum strict mai mare decât media aritmetică a celor n zile. Ex.: pentru 300, 420, 360, 0, 500 totalul e 1580, iar media e 316; peste ea sunt 420, 360 și 500, deci se afișează 3.

HINT *Ca să nu lucrezi cu numere zecimale, compară x * n > total în loc de x > total / n — e aceeași condiție.*

SOLUȚIA TA · PYTHON

2

PROBLEMA 7.2 · COD GHIDAT

Nivelurile de poluare, în ordine

SCRIERE COD

O stație de monitorizare a aerului a înregistrat n (n ≥ 1) niveluri de poluare, fiecare un număr întreg de la 0 (aer curat) la 5 (foarte poluat). Se citesc n, apoi cele n niveluri, câte unul pe linie. Afișează nivelurile în ordine crescătoare, pe un singur rând, separate prin spațiu, folosind sortarea cu listă de frecvențe. Ex.: pentru n = 7 și nivelurile 3, 0, 5, 3, 1, 0, 2 se afișează 0 0 1 2 3 3 5.

HINT *Scala 0–5 e stabilită pentru acest exercițiu. Valorile stau într-un interval mic, deci lista de frecvențe are doar 6 poziții: fr[k] numără de câte ori apare nivelul k.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.3 · COD GHIDAT

3 *Ziua cea mai economică din fișier*

SCRIERE COD

Se citește n (n ≥ 1), apoi n consumuri zilnice, câte unul pe linie. Scrie valorile în fișierul saptamana.txt, câte una pe linie, apoi redeschide fișierul, reconstruiește lista din liniile citite și afișează numărul zilei (numerotarea începe de la 1) cu cel mai mic consum. Dacă mai multe zile au același consum minim, afișează prima dintre ele. Ex.: pentru consumurile 300, 250, 400, 250, 310, 500, 280, minimul 250 apare în zilele 2 și 4, deci se afișează 2.

HINT **Atenție.** *Căutarea se face pe lista citită din fișier, nu pe cea citită de la tastatură — ca să verifici că salvarea și încărcarea funcționează.*

SOLUȚIA TA · PYTHON

4

PROBLEMA 7.4 · REPARĂ CODUL

Jurnalul care uită

SCRIERE COD

Programul de mai jos ar trebui să salveze cele n consumuri citite în fișierul jurnal.txt, apoi să afișeze câte linii are fișierul și suma valorilor din el. Pentru intrarea 3 ↲ 300 ↲ 420 ↲ 360 ar trebui să afișeze 3 și 1080, dar afișează 1 și 360. Găsește greșeala și corecteaz-o.

Cod stricat:

```
n = int(input())
v = []
for i in range(n):
    v.append(int(input()))
for x in v:
    with open('jurnal.txt', 'w') as f:
        f.write(str(x) + '\n')
with open('jurnal.txt', 'r') as f:
    linii = f.readlines()
s = 0
for linie in linii:
    s = s + int(linie)
print(len(linii))
print(s)
```

Ex.: pentru n = 4 și valorile 100, 0, 200, 50, programul corect afișează 4 și 350 (programul stricat afișează 1 și 50).

HINT *Recitește cardul „Capcana deschiderii în buclă” din secțiunea despre fișiere: programul acesta e chiar exemplul lui.*

PROGRAMUL CORECTAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Cea mai lungă serie de zile risipitoare

SCRIERE COD

Se citește n (n ≥ 1), apoi n consumuri zilnice, câte unul pe linie, apoi un prag p. Afișează lungimea celei mai lungi serii de zile consecutive în care consumul a fost strict mai mare decât p. Dacă nicio zi nu depășește pragul, afișează 0. Ex.: pentru consumurile 300, 520, 610, 280, 550, 600, 700 și p = 500, seriile peste prag sunt 520, 610 (lungime 2) și 550, 600, 700 (lungime 3), deci se afișează 3.

HINT *Ține două contoare: lungimea seriei curente (care revine la 0 când o zi nu depășește pragul) și cea mai lungă serie găsită până acum.*

SOLUȚIA TA · PYTHON