test-e.ro

CAIET ELEV

Lecția 28

Fișiere text. Clasa TextIOWrapper. Citire și scriere în fișiere

Fișierul text ca șir de caractere păstrat pe disc: cele trei etape ale lucrului cu un fișier (deschidere, transfer, închidere), funcția open() cu modurile "r", "w" și "a", clasa predefinită TextIOWrapper cu metodele read(), readline(), readlines(), write() și close(), noțiunea de sfârșit de fișier, construcția with open(...) as ...

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 și pe listă, break/continue, %, //, ord()/chr(), liste, len(), indexare, metodele listei, def/return, import math, min()/max()/sum(), slicing, atribuire simultană, plus open() și with — nou-permise chiar la lecția 28. Neutilizate deliberat: sorted() și .sort() (ordonarea se face tot prin metoda selecției, cu variabilă temporară la interschimbare) și .strip() (int() ignoră singur trecerea la linie nouă). Fără diacritice în codul Python și fără global.

CUPRINS

1 Ce știm deja .....
2 Deschidem primul fișier .....
3 Citim linie cu linie .....
4 Scriem în fișier .....
5 Lanțul complet, cu with .....
6 De ce merită un fișier .....
7 Cod ghidat · zece probleme .....

SECȚIUNEA 1 · CE ȘTIM DEJA: LISTE, SUBPROGRAME, CRONOMETRU

1 Ce știm *deja*

Înainte de a deschide primul fișier, ne asigurăm că uneltele sunt la locul lor: o listă în care adunăm valori, un subprogram care întoarce un rezultat și cronometrul cu care măsurăm cât durează o prelucrare. Toate trei revin în lecția de azi, în roluri noi.

O **listă** este colecția în care ținem, în ordine, mai multe valori de același fel. O construim pornind de la lista goală și adăugând element cu element cu `append()`, o parcurgem cu `for x in lista:` când ne interesează valorile și cu `for i in range(len(lista)):` când ne interesează pozițiile, iar `sum()`, `min()` și `max()` ne dau imediat suma, minimul și maximul.

Un **subprogram** definit cu `def` grupează o prelucrare sub un nume și, prin `return`, trimite rezultatul înapoi în locul din care a fost apelat. Nu afișează nimic de la sine: cine îl apelează decide ce face cu valoarea primită. Asta ne va folosi azi ca să separăm curat „citirea datelor” de „prelucrarea lor”.

Aceeași idee în Python (cu sintaxă reală):

```
def suma_pozitive(lista):
    s = 0
    for x in lista:
        if x > 0:
            s = s + x
    return s
valori = [4, -2, 7, 0]
print(suma_pozitive(valori))
```

Cronometrul îl avem din lecția despre eficiență: `from time import perf_counter` ne dă un moment de timp, iar diferența a două astfel de momente, luate înainte și după o prelucrare, este durata ei în secunde.

> **ATENȚIE**
>
> `sum()`, `min()` și `max()` cer o listă **nevidă**. Pe o listă goală, `max([])` oprește programul cu eroare. Când nu ești sigur că lista are elemente, verifică întâi `len(lista) > 0`. Capcana asta revine azi în forță, pentru că vom lucra des cu liste construite din date pe care programul nu le controlează.

A

EXERCIȚIUL A · GRILĂ

Liste, subprograme și cronometru

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce afișează print(len([3, 0, 7, 0]))?

   - 2 — pentru că elementele egale cu zero nu sunt numărate de len()
   - 3 — pentru că len() întoarce indicele ultimului element al listei
   - 10 — numărul obținut prin adunarea tuturor elementelor listei
   - 4

2. Ce returnează sum([5, -2, 0, 7])?

   - 4 — numărul de elemente al listei, nu suma valorilor lor
   - 7 — cel mai mare element din listă, nu suma elementelor ei
   - 10
   - 14 — suma valorilor absolute ale tuturor elementelor listei

3. Un subprogram definit cu def care se termină cu return c:

   - afișează pe ecran valoarea lui c, exact cum ar face instrucțiunea print
   - întoarce valoarea lui c în locul din care a fost apelat
   - creează o variabilă globală numită c, vizibilă în tot restul programului
   - oprește definitiv execuția întregului program, nu doar a subprogramului

4. perf_counter() din modulul time se folosește pentru:

   - a afișa data și ora curentă în formatul folosit de calculator
   - a opri programul pentru un număr dat de secunde, ca o pauză
   - a genera valori numerice aleatoare pentru testarea programelor
   - a măsura durata în secunde a unei porțiuni de program

B

EXERCIȚIUL B · DRY-RUN

Acumulare pe o listă

DRY-RUN

Urmărește bucla pas cu pas. Coloana x este dată (valoarea curentă din listă); tu completezi coloanele s și c, adică valorile celor două variabile după executarea corpului buclei. Ambele coloane sunt numerice. Ultimul rând arată starea în care bucla se oprește, pentru că lista s-a terminat.

```
1  valori = [4, 0, 7, 3]
2  s = 0
3  c = 0
```

```
for x in valori:
    s = s + x
    if x % 2 != 0:
        c = c + 1
print(s, c)
```

pas	x	s (completeaza)	c (completeaza)
înainte de buclă	—	0	0
1	4		
2	0		
3	7		
4	3		
bucla se oprește (lista s-a terminat)	—		

HINT *La fiecare pas s crește cu valoarea curentă x, iar c crește cu 1 doar atunci când x % 2 != 0.*

SPRIJIN · SCRIE CODUL

Suma și maximul unei liste

SCRIERE COD

Se citește un număr natural n (n >= 1), apoi n numere naturale, câte unul pe linie. Construiește lista celor n valori și afișează pe același rând suma lor și cea mai mare dintre ele, separate printr-un spațiu. Ex.: pentru n = 3 și valorile 5, 2, 9: suma este 5 + 2 + 9 = 16, iar maximul este 9, deci se afișează 16 9.

HINT *Enunțul garantează n >= 1, deci lista nu va fi niciodată goală și max() se poate aplica fără verificare suplimentară.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

Un subprogram care numără

SCRIERE COD

Se citește un număr natural n (n >= 1), apoi n numere naturale, câte unul pe linie. Scrie un subprogram numara_pare(lista) care primește o listă de numere și returnează câte dintre ele sunt pare. Afișează valoarea returnată de subprogram pentru lista citită. Ex.: pentru n = 5 și valorile 1, 2, 3, 4, 5, numerele pare sunt 2 și 4, deci se afișează 2.

HINT **Atenție.** *Subprogramul returnează, nu afișează: print se scrie o singură dată, în programul principal, pe valoarea primită de la numara_pare.*

HINT *Zero este par (0 % 2 == 0), deci intră în numărătoare.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · CE ESTE UN FIȘIER TEXT. OPEN() ȘI CLASA TEXTIOWRAPPER

2 Deschidem primul *fișier*

Un fișier text este un șir de caractere păstrat pe disc, care rămâne acolo și după ce programul s-a terminat. Ca să ajungem la el avem nevoie de o singură funcție, `open()`, de un mod de lucru și de disciplina de a închide ce am deschis.

Un **fișier text** este o succesiune de caractere memorată pe disc, sub un nume. Caracterul special `\n` (trecerea la linie nouă) împarte această succesiune în **linii**. Spre deosebire de o listă, care trăiește doar cât timp rulează programul, fișierul supraviețuiește: îl scrii azi, îl citești mâine, cu alt program.

Lucrul cu un fișier are mereu **trei etape, în ordinea asta**: **deschiderea**, **transferul datelor** și **închiderea**. Nu se sare peste niciuna.

Deschiderea se face cu funcția `open(nume, mod)`. Atenție la ce întoarce: **nu conținutul fișierului**, ci un **obiect** prin care se face transferul. Obiectul acesta aparține clasei predefinite `TextIOWrapper` — e „capătul de furtun” montat pe fișier, prin care curg caracterele.

Aceeași idee în Python (cu sintaxă reală):

```
f = open("date.txt", "w")
f.write("42")
f.close()
```

> **CELE TREI MODURI**
>
> `"r"` deschide un fișier **existent** pentru **citire**; dacă fișierul nu există, programul se oprește cu `FileNotFoundError`. `"w"` deschide pentru **scriere** și **șterge complet** conținutul vechi; dacă fișierul nu există, îl creează gol. `"a"` deschide tot pentru scriere, dar **păstrează** conținutul și adaugă textul nou **la sfârșit**.

Verificat în Python: un fișier care conținea `abc12`, redeschis cu `"w"` și scris cu `write("7")`, ajunge să conțină exact `7`. Același fișier, deschis apoi cu `"a"` și scris cu `write("89")`, ajunge `789`.

Închiderea se face cu metoda `close()`. Nu e o formalitate: până la `close()`, ce ai scris poate fi încă într-un tampon din memorie, nu pe disc. După `close()`, obiectul rămâne, dar e închis — `f.closed` devine `True` (verificat) și orice încercare de citire sau scriere prin el dă eroare.

Cea mai simplă metodă de citire este `read()`: întoarce **tot** conținutul fișierului, ca un **singur șir de caractere**. Pentru un fișier care conține doar `42`, `read()` întoarce șirul `'42'` — text, nu număr. Ca să-l poți folosi în calcule, îl treci prin `int()`.

> **ATENȚIE**
>
> `open("date.txt", "w")` pe un fișier care conține deja ceva **șterge tot**, instantaneu și fără avertisment, chiar dacă nu apuci să scrii nimic după. Dacă vrei să păstrezi ce era, modul corect e `"a"`, nu `"w"`.

> **UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ**
>
> Fișierele text sunt peste tot, sub nume pe care le vezi zilnic: un fișier `.csv` exportat din catalogul electronic cu notele clasei este un fișier text cu o linie per elev; setările jocurilor pe care le instalezi stau în fișiere de configurare text pe care le poți deschide cu orice editor de text simplu; iar subtitrarea `.srt` a unui film este un fișier text în care fiecare replică ocupă câteva linii, cu momentul de început și de sfârșit.

A

EXERCIȚIUL A · GRILĂ

open(), modurile de lucru și TextIOWrapper

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce tip de obiect returnează open("date.txt", "w")?

   - un șir de caractere care conține tot conținutul fișierului deschis
   - un număr întreg care arată câte caractere are fișierul deschis
   - un obiect din clasa TextIOWrapper
   - o listă cu toate liniile fișierului, în ordinea în care apar în el

2. Un fișier care conține deja 12345 este deschis cu open("date.txt", "w"). Ce se întâmplă cu conținutul vechi?

   - programul se oprește cu eroare, pentru că fișierul nu este gol
   - este șters complet, iar fișierul rămâne gol
   - rămâne neatins, iar textul nou se adaugă la sfârșitul lui
   - rămâne neatins, iar textul nou se adaugă la începutul lui

3. De ce se apelează close() după ce s-a terminat scrierea?

   - pentru a șterge de pe disc fișierul care tocmai a fost folosit de program
   - pentru ca datele scrise să ajungă sigur pe disc și resursa să fie eliberată
   - pentru a transforma conținutul fișierului dintr-un text într-o listă de numere
   - pentru a muta cursorul de citire înapoi la prima linie a fișierului text

B

EXERCIȚIUL B · ASOCIERE

Apelul și efectul lui asupra fișierului

ASOCIERE

Leagă fiecare apel din coloana din stânga de efectul lui exact din coloana din dreapta. Opțiunile din dreapta sunt amestecate. Scrie în căsuța din fața fiecărui element din stânga litera variantei potrivite. Fiecare literă se folosește o singură dată.

☐	open("f.txt", "r")
☐	open("f.txt", "w")
☐	open("f.txt", "a")
☐	f.close()

A păstrează conținutul existent și adaugă textul nou la sfârșit

B încheie lucrul cu fișierul și trimite pe disc ce era în tampon

C deschide un fișier existent pentru citire; dacă lipsește, dă eroare

D creează fișierul sau îi șterge tot conținutul, apoi permite scrierea

C

EXERCIȚIUL C · SCRIE CODUL

Dus-întors printr-un fișier

SCRIERE COD

Se citește un număr natural n. Scrie-l în fișierul date.txt, închide fișierul, apoi redeschide-l în modul de citire și citește-i tot conținutul cu read(). Afișează pe primul rând valoarea citită din fișier, iar pe al doilea rând dublul ei. Ex.: pentru n = 7, fișierul date.txt ajunge să conțină textul 7; programul îl citește înapoi și afișează 7 pe primul rând și 14 pe al doilea.

HINT **Atenție.** *write() primește obligatoriu un șir de caractere, deci numărul se trece prin str(n). Invers, read() întoarce tot un șir, deci înainte de înmulțire îl treci prin int().*

HINT *Fișierul se închide după scriere și se deschide din nou pentru citire: sunt două deschideri distincte, cu două moduri diferite.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 3 · CITIREA PE LINII: READLINE(), READLINES() ȘI SFÂRȘITUL DE FIȘIER

3 Citim linie cu *linie*

`read()` ne-a dat tot conținutul deodată, ca un bloc de text. De cele mai multe ori însă datele vin organizate pe linii — o temperatură pe zi, un elev pe rând — și avem nevoie să le luăm una câte una. Aici intră `readline()` și `readlines()`, plus o întrebare nouă: de unde știe programul că fișierul s-a terminat?

Când citim un fișier, în spate există un **cursor invizibil**: o poziție care arată de unde continuă următoarea citire. La deschidere, cursorul e la începutul fișierului; fiecare citire îl mută mai departe.

Metoda `readline()` întoarce **exact o linie**, de la cursor până la prima trecere la linie nouă **inclusiv**, și mută cursorul după ea. Pentru un fișier cu liniile 5, 0 și 8, apelurile succesive întorc `'5\n'`, apoi `'0\n'`, apoi `'8\n'` (verificat în Python). Când cursorul a trecut de ultima linie, `readline()` nu dă eroare: întoarce **șirul vid** `""`. Asta este, concret, **sfârșitul de fișier** — nu un semn special în fișier, ci pur și simplu faptul că nu mai e nimic de întors.

De aici tiparul standard de parcurgere, care se scrie mereu la fel:

```
g = open("date.txt", "r")
s = 0
linie = g.readline()
while linie != "":
    s = s + int(linie)
    linie = g.readline()
g.close()
```

```
8  print(s)
```

Metoda `readlines()` face totul dintr-un foc: întoarce o **listă** cu toate liniile fișierului, fiecare cu `\n`-ul ei — `['5\n', '0\n', '8\n']` — pe care apoi o parcurgi cu `for`, ca pe orice listă. Pentru un fișier gol întoarce lista goală `[]` (verificat), iar `readline()` întoarce direct `''`.

TREI GRANULAȚII DE CITIRE

`read()` — tot fișierul, într-un singur șir. `readline()` — o linie pe apel, cursorul avansează, `""` la sfârșit. `readlines()` — toate liniile deodată, într-o listă. Alegi după cum vrei să prelucrezi: `readlines()` când ai nevoie de toate valorile în memorie, `readline()` când vrei să treci o dată prin fișier fără să-l ții tot.

ATENȚIE

Linia citită **conține și `\n`-ul**, deci `'5\n'` nu este egal cu `'5'`. Pentru numere nu e o problemă: `int("5\n")` întoarce `5`, pentru că `int()` ignoră spațiile albe de la capete (verificat în Python). Dar dacă ai compara direct șiruri, comparația ar eșua fără motiv aparent.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Un fișier de jurnal (*log*) al unui server crește la milioane de linii: se citește cu `readline()`, linie cu linie, tocmai ca să nu fie nevoie să încapă tot în memorie. Aplicația de contacte a telefonului importă un fișier `.vcf` în care fiecare contact ocupă câteva linii citite pe rând. Iar un fișier `.csv` cu notele clasei se încarcă de obicei cu `readlines()`, pentru că toate liniile trebuie oricum ținute deodată ca să calculezi media.

A

EXERCIȚIUL A · DRY-RUN

Bucla care se oprește la șirul vid

DRY-RUN

Urmărește parcurgerea fișierului. Coloana „linie citită” este dată (ce întoarce readline() la fiecare pas); tu completezi coloanele nr și s, adică valorile celor două variabile după executarea corpului buclei. Ambele sunt numerice. Ultimul rând arată starea în care condiția linie != "" devine falsă.

```
1  g = open("date.txt", "r")
2  nr = 0
3  s = 0
4  linie = g.readline()
5  while linie != "":
6      nr = nr + 1
7      s = s + int(linie)
8      linie = g.readline()
```

```
 9  g.close()
10  print(nr, s)
```

pas	linie citita	nr (completeaza)	s (completeaza)
înainte de buclă	'5\n'	0	0
1	'5\n'		
2	'0\n'		
3	'8\n'		
bucla se oprește (readline() a întors '')	''		

HINT *La fiecare trecere nr crește cu 1, iar s crește cu valoarea numerică a liniei citite; int('0\n') este 0, deci s nu se schimbă la pasul al doilea.*

B

EXERCIȚIUL B · COMPLETARE

Cele trei metode de citire

COMPLETARE

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

`plin` `readlines` `vid` `liste` `readline` `read`

Metoda _____ întoarce o singură linie din fișier, iar când s-a ajuns la sfârșit întoarce șirul _____ ; metoda _____ întoarce toate liniile deodată, sub forma unei _____ .

C

EXERCIȚIUL C · SCRIE CODUL

Câte linii și ce sumă?

SCRIERE COD

Se citește un număr natural n (n >= 0), apoi n numere naturale, câte unul pe linie. Scrie cele n valori în fișierul date.txt, câte una pe linie, apoi redeschide fișierul și parcurge-l cu readline() până la sfârșitul de fișier. Afișează pe același rând numărul de linii găsite și suma valorilor de pe ele, separate printr-un spațiu. Ex.: pentru n = 3 și valorile 4, 5, 6, fișierul are 3 linii, iar suma lor este 4 + 5 + 6 = 15, deci se afișează 3 15.

HINT **Atenție.** *write() nu adaugă singură trecerea la linie nouă. Dacă scrii doar str(x), toate numerele se lipesc într-o singură linie. Scrie str(x) + "\n".*

HINT *Pentru n = 0 fișierul rămâne gol; readline() întoarce direct șirul vid, bucla nu se execută niciodată și se afișează 0 0.*

SOLUȚIA TA · PYTHON

4 SECȚIUNEA 4 · SCRIEREA ÎN FIȘIER: WRITE(), MODUL „W” FAȚĂ DE „A”

Scriem în *fișier*

Am citit; acum producem. Scrierea are o singură metodă, `write()`, și două capcane pe care le vom lovi azi din plin: scrie doar șiruri de caractere și nu trece singură la linie nouă.

Metoda `write(text)` scrie în fișier, de la poziția curentă, șirul primit ca argument. Două reguli, ambele verificate în Python:

Prima: primește obligatoriu un șir de caractere. Un număr trebuie trecut prin `str()`. `f.write(42)` nu merge; `f.write(str(42))` merge. În schimb, `write()` întoarce un număr — câte caractere a scris efectiv: `write("abc")` întoarce `3`, `write("12")` întoarce `2`.

A doua, mai perfidă: `write()` nu adaugă trecerea la linie nouă. Dacă scrii pe rând `"12"`, `"5"` și `"7"`, fișierul conține `1257` — o singură linie. Ca să obții trei linii, trebuie să adaugi tu `\n`:

```
1  f = open("date.txt", "w")
```

```
f.write(str(12) + "\n")
f.write(str(5) + "\n")
f.write(str(7) + "\n")
f.close()
```

ATENȚIE

Uitarea lui `"\n"` nu produce nicio eroare — produce un fișier greșit, care abia mai târziu dă rezultate absurde: `readlines()` găsește **o** linie în loc de trei, iar `int()` pe ea întoarce numărul monstruos `1257`. Exact asta vei repara la exercițiul de remediere din secțiunea 7.

„W" DISTRUGE, „A" CONTINUĂ

Modul `"w"` golește fișierul în momentul deschiderii, deci se folosește când produci un rezultat de la zero. Modul `"a"` scrie mereu la sfârșit, deci se folosește când adaugi la un istoric care trebuie păstrat. Verificat: un fișier cu `7`, deschis cu `"a"` și scris cu `write("89")`, ajunge `789`.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Un joc care salvează progresul rescrie fișierul de salvare cu `"w"` — vrei ultima stare, nu istoricul tuturor. Aplicația de fitness care înregistrează fiecare alergare adaugă cu `"a"` o linie nouă la sfârșitul jurnalului, fără să atingă ce era. Iar un program care exportă catalogul clasei într-un fișier `.csv` scrie linie cu linie, cu `"\n"` la fiecare elev — dacă îl uită, tot catalogul ajunge pe un singur rând, imposibil de deschis corect într-un program de calcul tabelar.

A

EXERCIȚIUL A · GRILĂ

Ce face și ce nu face write()

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce returnează apelul f.write("abc")?

   `True, ca semn că scrierea s-a încheiat fără nicio problemă`

   `șirul "abc", exact așa cum a fost el scris în fișierul text`

   `None, pentru că write() doar scrie și nu întoarce nimic util` `3`

2. Un fișier este deschis cu "w", apoi se execută f.write("12") și f.write("7"). Ce conține fișierul?

   `12 și 7, scrise una sub alta, pe două linii separate` `127, totul pe o singură linie`

   `12 7, cu un spațiu pus automat între cele două scrieri`

   `doar 7, pentru că a doua scriere o înlocuiește pe prima`

3.

Un program scrie 3 numere cu f.write(str(x)), fără să adauge altceva. Câte elemente va avea lista întoarsă apoi de readlines()?

- `0, pentru că fișierul nu conține niciun caracter de linie nouă`
- `1`
- `4, cele 3 numere plus o linie goală la sfârșitul fișierului`
- `3, câte o linie pentru fiecare număr scris în fișier`

B

EXERCIȚIUL B · SCRIE CODUL

Raportul divizorilor

SCRIERE COD

Se citește un număr natural n (n >= 1). Scrie în fișierul divizori.txt toți divizorii lui n, în ordine crescătoare, câte unul pe linie. Redeschide apoi fișierul, citește-l cu readlines() și afișează pe același rând câți divizori are n și suma lor, separate printr-un spațiu. Ex.: pentru n = 6, fișierul divizori.txt conține liniile 1, 2, 3 și 6; sunt 4 divizori, iar suma lor este 1 + 2 + 3 + 6 = 12, deci se afișează 4 12.

HINT **Atenție.** *Fiecare divizor se scrie cu str(d) + "\n". Fără "\n", pentru n = 6 fișierul ar conține 1236 și readlines() ar găsi o singură linie.*

HINT *Numărul de divizori se obține direct cu len() pe lista întoarsă de readlines() — nu e nevoie de un contor separat.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · WITH OPEN(...) AS ...: ÎNCHIDEREA AUTOMATĂ ȘI LANȚUL COMPLET

5 Lanțul complet, cu *with*

Un `close()` uitat înseamnă date rămase în tampon și un fișier blocat. Python are o construcție care rezolvă problema definitiv: `with`. Cu ea scriem azi lanțul întreg — citim dintr-un fișier, prelucrăm în memorie, scriem rezultatul în alt fișier.

Construcția `with open(nume, mod) as f:` deschide fișierul, îl leagă de variabila `f` și îl **închide automat** la ieșirea din bloc. „Automat” înseamnă: și când blocul se termină normal, **și când în bloc apare o eroare**. Verificat în Python: imediat după bloc, `f.closed` este `True`, fără să fi scris vreun `close()`.

Aceeași idee în Python (cu sintaxă reală):

```
with open("date.txt", "r") as f:
    linii = f.readlines()
for linie in linii:
    print(int(linie))
```

REGULA LUI WITH

Tot ce ai nevoie **din** fișier se ia **în interiorul** blocului; prelucrarea se face **după**, pe datele deja aduse în memorie. În afara blocului, `f` există, dar e închis: orice citire prin el dă eroare.

Cu `with` scriem acum **lanțul complet de prelucrare**, tiparul care apare în aproape orice program serios de azi înainte:

1. **citirea** — deschizi sursa în `"r"` și aduci liniile în memorie;
2. **transformarea** — treci fiecare linie prin `int()` și construiești lista de valori;
3. **prelucrarea** — calculezi ce ai nevoie (sumă, maxim) și, dacă e cazul, ordonezi lista prin metoda selecției, învățată deja;
4. **scrierea** — deschizi destinația în `"w"` și scrii rezultatul, câte o valoare pe linie.

Sursa și destinația sunt **două fișiere diferite**, deschise separat, pentru că un fișier deschis în `"r"` nu poate fi scris, iar unul deschis în `"w"` și-a pierdut deja conținutul.

ATENȚIE

Nu deschide același fișier simultan în `"r"` și în `"w"`: modul `"w"` îi golește conținutul chiar în momentul deschiderii, deci ai citi dintr-un fișier care tocmai a fost șters. Scrie rezultatul în alt fișier.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

> Aplicația care îți exportă extrasul de cont citește fișierul brut de la bancă și scrie un al doilea fișier, curat, doar cu tranzacțiile din luna cerută. Un editor de subtitrări citește `.srt`-ul original și produce alt fișier cu timpii decalați cu două secunde. Iar programul de la secretariatul școlii citește lista neordonată a elevilor și scrie într-un fișier nou aceeași listă, ordonată alfabetic — sursa rămâne intactă, ca dovadă.

A

EXERCIȚIUL A · GRILĂ

Blocul with și lanțul de prelucrare

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce avantaj are with open(...) as f: față de f = open(...) urmat de f.close()?

   - `fișierul poate fi deschis în același timp și pentru citire, și pentru scriere`
   - `fișierul rămâne deschis până la sfârșitul programului, ca să nu se piardă date`
   - `fișierul se citește mult mai repede, pentru că datele intră direct în listă`
   - `fișierul se închide automat la ieșirea din bloc, chiar dacă apare o eroare`

2. După ce blocul with open("date.txt", "r") as f: s-a încheiat, cât valorează f.closed?

   - `False, pentru că închiderea se face doar prin apelul explicit close()`
   - `True`
   - `0, pentru că fișierul are zero caractere rămase necitite în memorie`
   - `None, pentru că variabila f nu mai există după terminarea blocului`

3. Într-un program care citește dintr-un fișier și scrie rezultatul în altul, de ce se deschid două fișiere diferite?

   - `pentru că într-un fișier nu se poate scrie mai mult de o linie odată`
   - `pentru că un fișier deschis în modul "r" nu poate fi și scris`
   - `pentru că modul "w" citește datele, iar modul "r" le scrie pe disc`
   - `pentru că o listă din Python nu poate păstra mai mult de 100 de valori`

B

EXERCIȚIUL B · SCRIE CODUL

Lanțul complet: citește, prelucrează, scrie

SCRIERE COD

Se citește un număr natural n (n >= 1), apoi n numere naturale, câte unul pe linie. Scrie valorile în fișierul date.txt, câte una pe linie. Folosind construcția with, citește-le înapoi într-o listă, determină suma și maximul, ordonează lista crescător prin metoda selecției și scrie valorile ordonate în fișierul rezultat.txt, câte una pe linie. Redeschide rezultat.txt și afișează pe primul rând suma și maximul

separate printr-un spațiu, iar pe al doilea rând valorile ordonate, separate prin câte un spațiu. Ex.: pentru n = 4 și valorile 5, 2, 9, 1: suma este 17, maximul este 9, iar lista ordonată este 1, 2, 5, 9. Se afișează 17 9 pe primul rând și 1 2 5 9 pe al doilea.

HINT **Atenție.** *rezultat.txt trebuie să fie un fișier diferit de date.txt. Dacă ai deschide date.txt în modul "w" ca să scrii rezultatul, i-ai șterge conținutul înainte de a apuca să-l citești.*

HINT *Ordonarea se face prin metoda selecției, ca la lecțiile anterioare, cu variabilă temporară la interschimbare — nu cu funcții gata făcute.*

SOLUȚIA TA · PYTHON

PERFORMANȚĂ · SCRIE CODUL

Două fișiere deodată

SCRIERE COD

Se citește un număr natural n (n >= 0), apoi n numere naturale, câte unul pe linie. Distribuie valorile în două fișiere: cele pare în pare.txt, cele impare în impare.txt, câte una pe linie, ambele fișiere fiind deschise în același timp. Redeschide apoi cele două fișiere și afișează pe primul rând câte valori are pare.txt și suma lor, iar pe al doilea rând aceleași două informații pentru impare.txt. Ex.: pentru n = 5 și valorile 1, 2, 3, 4, 5: în pare.txt ajung 2 și 4 (2 valori, suma 6), iar în impare.txt ajung 1, 3 și 5 (3 valori, suma 9). Se afișează 2 6 pe primul rând și 3 9 pe al doilea.

HINT **Atenție.** *Ambele fișiere se deschid în modul "w" înainte de bucla de distribuire, nu în interiorul ei. Dacă le-ai deschide la fiecare valoare, fiecare deschidere ar șterge ce scrisese cea dinainte și ar rămâne o singură valoare în fiecare fișier.*

HINT *Pentru n = 0 ambele fișiere se creează goale, iar programul afișează 0 0 pe amândouă rândurile. Zero este par, deci intră în pare.txt.*

HINT **Curiozitate.** *Blocurile with se pot imbrica: unul în interiorul altuia ține ambele fișiere deschise simultan și le închide, la ieșire, în ordine inversă deschiderii.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 6 · FIȘIER FAȚĂ DE TASTATURĂ: TIMPUL, MODELUL ȘI CAZURILE-LIMITĂ

6 De ce merită un *fișier*

Am învățat cum se lucrează cu fișiere. Rămâne întrebarea de ce: măsurăm diferența de timp față de citirea de la tastatură, învățăm să proiectăm modelul datelor înainte de a scrie cod și pregătim programul pentru cele trei situații care îl pot doborî — fișier inexistent, fișier gol, fișier cu o singură linie.

Timpul. Imaginează-ți registrul unei stații meteo: 365 de temperaturi, una pe zi. Citite dintr-un fișier și transformate în numere, cele 365 de valori ajung în memorie în **0,00009 secunde** (măsurat cu `perf_counter()`). Aceleași 365 de valori introduse de la tastatură cer un om care să apese taste: la o

valoare pe secundă, peste **6 minute** — de aproape patru milioane de ori mai mult. Diferența nu vine din viteza procesorului, ci din faptul că fișierul **nu are nevoie de om**. Bonus: fișierul poate fi rulat de o mie de ori, identic; tastarea, nu.

Măsurarea se scrie exact ca la lecția despre eficiență:

```
from time import perf_counter
t0 = perf_counter()
with open("meteo.txt", "r") as f:
    linii = f.readlines()
valori = []
for linie in linii:
    valori.append(int(linie))
t1 = perf_counter()
print(t1 - t0)
```

Modelul conceptual. Înainte de prima linie de cod, răspunzi la trei întrebări despre fișier: *ce reprezintă o linie?* (o temperatură, în grade Celsius, număr întreg), *în ce ordine vin liniile?* (cronologic, de la 1 ianuarie), *ce valori sunt valide?* (între −40 și 50; orice altceva e o eroare de măsurare). Abia apoi scrii codul. Un fișier prost înțeles produce un program corect sintactic și greșit în fond — de exemplu unul care calculează media unor date pe care le-a citit în altă ordine decât cea reală.

> **CELE TREI CAZURI-LIMITĂ**
>
> **Fișier inexistent:** `open(nume, "r")` oprește programul cu `FileNotFoundError` (verificat). **Fișier gol:** `readlines()` întoarce lista goală `[]`, iar `readline()` întoarce direct `""` — bucla nu se execută niciodată și orice `max()` pe lista rezultată dă eroare. **Fișier cu o singură linie:** lista are exact un element, deci un program care presupune „cel puțin două valori” (ca o comparație între primul și al doilea) se rupe.

> **ATENȚIE**
>
> Un program testat doar pe fișiere „normale” pare corect. Cele trei cazuri de mai sus nu sunt curiozități: fișierul gol apare la prima rulare a oricărei aplicații, iar fișierul lipsă apare de fiecare dată când cineva mută un folder. Testează-le explicit.

> **UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ**
>
> Stațiile meteo automate scriu zilnic o linie într-un fișier și trimit odată pe an fișierul întreg spre prelucrare — nimeni nu tastează 365 de valori. Aplicația de pe telefon care numără pașii salvează un fișier pe lună și îl citește instantaneu când deschizi graficul. Iar un magazin online

își importă lista de prețuri dintr-un fișier de la furnizor: dacă fișierul lipsește sau vine gol, programul trebuie să spună asta clar, nu să afișeze zero produse ca și cum ar fi normal.

A

EXERCIȚIUL A · GRILĂ

Timp, fișier gol, fișier inexistent

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1.

De ce este mult mai rapid un program care citește 365 de valori dintr-un fișier decât unul care le cere de la tastatură?

- `pentru că fișierele text sunt păstrate întotdeauna comprimate pe disc`
- `pentru că valorile dintr-un fișier sunt deja numere, nu șiruri de caractere`
- `pentru că funcția open() folosește mai multe procesoare în același timp`
- `pentru că nu așteaptă un om care tastează, ci ia datele direct de pe disc`

2. Ce returnează readlines() pentru un fișier gol?

- `None, pentru că nu există nicio linie care să fie întoarsă`
- `[""], adică o listă cu un singur element, șirul vid`
- `[]`
- `0, adică numărul de linii găsite în fișierul gol deschis`

3. Ce se întâmplă dacă deschizi în modul "r" un fișier care nu există?

- `open() întoarce None, iar programul continuă cu fișierul închis`
- `programul se oprește cu eroarea FileNotFoundError`
- `programul așteaptă ca utilizatorul să introducă datele de la tastatură`
- `fișierul este creat automat, gol, iar programul continuă normal`

B

EXERCIȚIUL B · ORDONARE

Etapele încărcării datelor dintr-un fișier

ORDONARE

Numerotează pașii de la 1 la 5, în ordinea corectă. Pașii îți apar amestecați.

- [ ] Închizi fișierul cu close() sau lași blocul with să-l închidă automat
- [ ] Transferi conținutul din fișier în memorie, cu readline() sau readlines()
- [ ] Proiectezi modelul conceptual: ce reprezintă o linie, în ce ordine vin liniile, ce valori sunt valide

- [ ] Transformi liniile de text în valorile de care ai nevoie, de exemplu cu int()
- [ ] Deschizi fișierul cu open(), precizând numele și modul de lucru

C

EXERCIȚIUL C · SCRIE CODUL

Fișier gol, o singură linie sau mai multe

SCRIERE COD

Se citește un număr natural n (n >= 0), apoi n numere naturale, câte unul pe linie. Scrie valorile în fișierul date.txt, câte una pe linie, apoi citește-l cu readlines(). Dacă fișierul este gol, afișează FISIER GOL. Dacă are exact o linie, afișează O SINGURA LINIE urmat de un spațiu și de valoarea de pe acea linie. Altfel, afișează pe același rând numărul de linii și suma valorilor, separate printr-un spațiu. Ex.: pentru n = 0 nu se scrie nimic, fișierul rămâne gol și se afișează FISIER GOL. Pentru n = 1 și valoarea 42 se afișează O SINGURA LINIE 42. Pentru n = 3 și valorile 1, 2, 3 se afișează 3 6.

HINT **Atenție.** *Mesajele se scriu fără diacritice, exact ca în enunț: FISIER GOL și O SINGURA LINIE.*

HINT *Numărul de linii se citește direct cu len() pe lista întoarsă de readlines(); testul len(linii) == 0 este chiar verificarea de fișier gol.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 7 · COD GHIDAT

7 Scrie cod care *trece testele*

Zece 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.

PROBLEMA 7.1 · COD GHIDAT

1 *Registrul de temperaturi*

SCRIERE COD

Se citește un număr natural n (n >= 1), apoi n numere întregi reprezentând temperaturile înregistrate de o stație meteo, câte una pe linie. Scrie-le în fișierul temperaturi.txt, câte una pe linie, apoi citește fișierul înapoi. Afișează pe primul rând cea mai mică și cea mai mare temperatură, separate printr-un spațiu, iar pe al doilea rând câte temperaturi au fost negative. Ex.: pentru n = 5 și temperaturile −3, 7, 0, 12, −8: minimul este −8, maximul este 12, iar temperaturile negative sunt −3 și −8, adică 2. Se afișează -8 12 pe primul rând și 2 pe al doilea.

HINT *Temperaturile pot fi negative, dar int() citește fără probleme și valorile cu semnul minus, atât de la tastatură, cât și dintr-o linie de fișier.*

HINT **Atenție.** *Zero nu este negativ, deci nu intră în numărătoarea de pe al doilea rând.*

SOLUȚIA TA · PYTHON

2

PROBLEMA 7.2 · COD GHIDAT

Copia filtrată

SCRIERE COD

Se citește un număr natural n (n >= 0), apoi n numere întregi, câte unul pe linie. Scrie-le în fișierul date.txt, câte unul pe linie. Citește apoi date.txt și copiază în fișierul copie.txt doar valorile mai mari decât 0, câte una pe linie. Redeschide copie.txt și afișează pe același rând câte valori au fost copiate și suma lor, separate printr-un spațiu. Ex.: pentru n = 5 și valorile 1, −2, 3, 0, 4: în copie.txt ajung 1, 3 și 4, adică 3 valori cu suma 8, deci se afișează 3 8.

HINT **Atenție.** *Zero nu este mai mare decât 0, deci nu se copiază.*

HINT *Dacă nicio valoare nu este pozitivă, copie.txt rămâne gol, readlines() întoarce lista goală și se afișează 0 0.*

SOLUȚIA TA · PYTHON

3

PROBLEMA 7.3 · COD GHIDAT

Fișierul întors pe dos

SCRIERE COD

Se citește un număr natural n (n >= 1), apoi n numere naturale, câte unul pe linie. Scrie-le în fișierul date.txt, câte unul pe linie. Citește apoi fișierul și scrie în invers.txt aceleași valori, dar în ordine inversă, câte una pe linie. Redeschide invers.txt și afișează valorile din el pe un singur rând, separate

prin câte un spațiu. Ex.: pentru n = 4 și valorile 1, 2, 3, 4, fișierul invers.txt conține liniile 4, 3, 2, 1, iar programul afișează 4 3 2 1.

HINT **Atenție.** *Ordinea inversă se obține parcurgând lista de linii de la ultimul indice către primul, cu range(len(linii) - 1, -1, -1) — nu prin oglindirea vreunui număr.*

HINT *Între valorile afișate se pune exact un spațiu, iar rândul nu se termină cu spațiu.*

SOLUȚIA TA · PYTHON

4

PROBLEMA 7.4 · REPARĂ CODUL

Programul care pierde liniile

SCRIERE COD

Programul de mai jos ar trebui să scrie în fișier cele n valori citite, câte una pe linie, apoi să le citească înapoi și să afișeze pe același rând numărul de linii și suma valorilor. Rulat pentru n = 3 și valorile 12, 5, 7, ar trebui să afișeze 3 24, dar afișează 1 1257. Găsește greșeala, repar-o și rescrie programul corect.

Cod stricat:

```
1  n = int(input())
2  f = open("date.txt", "w")
3  for i in range(n):
4      x = int(input())
```

```
    f.write(str(x))
f.close()
f = open("date.txt", "r")
linii = f.readlines()
f.close()
s = 0
for linie in linii:
    s = s + int(linie)
print(len(linii), s)
```

Ex.: pentru n = 3 și valorile 12, 5, 7, programul corect afișează 3 24, pentru că fișierul are 3 linii, iar 12 + 5 + 7 = 24.

HINT *Greșeala nu produce nicio eroare: fișierul se scrie, se citește și se afișează un rezultat — doar că un rezultat greșit. Verifică afirmația din teoria secțiunii 4 despre ce nu face write() de la sine.*

HINT **Curiozitate.** *Valoarea 1257 nu este întâmplătoare: e chiar textul 12, 5 și 7 lipit, citit apoi ca un singur număr.*

PROGRAMUL CORECTAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Două fișiere, o singură listă ordonată

SCRIERE COD

Se citesc două numere naturale n și m (n + m >= 1), apoi n numere naturale care se scriu în fișierul a.txt și m numere naturale care se scriu în fișierul b.txt, câte unul pe linie în fiecare fișier. Citește ambele fișiere, adună toate valorile într-o singură listă, ordonează-o crescător prin metoda selecției și scrie rezultatul în fișierul unit.txt, câte o valoare pe linie. Redeschide unit.txt și afișează pe primul rând câte valori conține, iar pe al doilea rând valorile lui, separate prin câte un spațiu. Ex.: pentru n = 2, m = 3, valorile 5 și 1 în a.txt și valorile 4, 2, 9 în b.txt: lista unită și ordonată este 1, 2, 4, 5, 9. Se afișează 5 pe primul rând și 1 2 4 5 9 pe al doilea.

HINT **Atenție.** *n sau m poate fi 0: atunci fișierul respectiv se creează gol, iar readlines() întoarce lista goală. Programul trebuie să funcționeze și așa.*

HINT *Ordinea de citire de la tastatură este: întâi n, apoi m, apoi cele n valori pentru a.txt, apoi cele m valori pentru b.txt.*

SOLUȚIA TA · PYTHON

Cele cinci probleme de mai jos nu sunt obligatorii toate. Dacă simți că ți-au scăpat bazele, începe cu **7.6** și **7.7**: sunt scurte și izolează câte o singură dificultate. Dacă ai rezolvat fără efort problemele 7.1–7.3, treci direct la **7.8** (singura care intră în progresul lecției), apoi la **7.9** și **7.10**.

PROBLEMA 7.6 · REMEDIERE

Un singur număr, dus și întors

SCRIERE COD

Se citește un număr natural n. Scrie-l în fișierul unu.txt, închide fișierul, apoi redeschide-l pentru citire și citește-i tot conținutul cu read(). Afișează valoarea citită din fișier. Ex.: pentru n = 5, fișierul unu.txt ajunge să conțină textul 5, iar programul afișează 5.

HINT **Atenție.** *write() primește obligatoriu un șir de caractere, deci numărul se trece prin str(n). Aici nu ai nevoie de niciun calcul după citire: afișezi direct șirul întors de read().*

HINT *Fără buclă, fără listă, fără trecere la linie nouă — o singură valoare, un singur drum dus-întors. Dacă problema 7.1 ți s-a părut prea mare dintr-odată, aceasta este pasul dinaintea ei.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.7 · REMEDIERE

Trei linii, nu una

SCRIERE COD

Se citesc trei numere naturale, câte unul pe linie. Programul de mai jos ar trebui să le scrie în fișierul trei.txt, câte unul pe linie, apoi să afișeze pe același rând câte linii găsește și suma valorilor de pe ele. Rulat cu 10, 20 și 30, ar trebui să afișeze 3 60, dar afișează 1 102030. Repară-l.

Cod stricat:

```python
a = int(input())
b = int(input())
c = int(input())
f = open("trei.txt", "w")
f.write(str(a))
f.write(str(b))
f.write(str(c))
f.close()
f = open("trei.txt", "r")
linii = f.readlines()
f.close()
s = 0
for linie in linii:
    s = s + int(linie)
print(len(linii), s)
```

Ex.: pentru 10, 20 și 30, programul corect afișează 3 60, pentru că fișierul are 3 linii, iar 10 + 20 + 30 = 60.

HINT **Atenție.** *Este aceeași greșeală ca la problema 7.4, dar aici cele trei scrieri sunt una sub alta, fără buclă: vezi cu ochiul liber ce lipsește la fiecare și repari în trei locuri identice.*

HINT *Rezultatul greșit 1 102030 nu este întâmplător: e chiar textul 10, 20 și 30 lipit, citit apoi ca un singur număr.*

SOLUȚIA TA · PYTHON

8

PROBLEMA 7.8 · COD GHIDAT

Două unelte pentru fișiere

SCRIERE COD

Se citește un număr natural n (n >= 1), apoi n numere naturale, câte unul pe linie. Scrie două subprograme: scrie_fisier(nume, lista), care scrie în fișierul cu numele dat valorile din listă, câte una pe linie, și citeste_fisier(nume), care deschide fișierul cu numele dat și returnează lista valorilor citite din el. Folosind cele două subprograme, scrie valorile în date.txt, citește-le înapoi și afișează pe același rând suma și maximul lor, separate printr-un spațiu. Ex.: pentru n = 3 și valorile 4, 9, 2: suma este 15, maximul este 9, deci se afișează 15 9.

HINT **Atenție.** *citeste_fisier trebuie să returneze lista, nu să o afișeze: programul principal decide ce face cu ea. Deschiderea și închiderea fișierului stau în interiorul fiecărui subprogram — asta este toată ideea: cine le folosește nu trebuie să știe cum lucrează ele.*

HINT *Odată scrise, cele două subprograme se pot refolosi la orice problemă cu fișiere din lecție: exact avantajul modularizării despre care s-a discutat la lecția 19.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.9 · PROVOCARE

Jurnalul care se reia

SCRIERE COD

Se citește un număr natural p (p >= 0), apoi p numere naturale: acestea formează jurnalul de ieri și se scriu în fișierul jurnal.txt, câte unul pe linie. Se citește apoi un cuvânt, RESET sau ADAUGA, și după el un număr natural k (k >= 0) urmat de k numere naturale. Dacă s-a citit RESET, jurnalul trebuie golit înainte de a scrie noile valori; dacă s-a citit ADAUGA, noile valori se pun la sfârșit, păstrând ce era deja. La final, redeschide jurnal.txt și afișează pe același rând câte linii are și suma valorilor de pe ele, separate printr-un spațiu. Cerință: Alegerea dintre cele două comportamente se face prin *modul de deschidere* a fișierului, nu prin recitirea și rescrierea conținutului vechi. Un singur mod golește, un singur mod adaugă la sfârșit — sunt exact cele două din secțiunea 4. Ex.: pentru p = 2 cu valorile 5 și 3, apoi ADAUGA, apoi k = 3 cu valorile 1, 1, 1: jurnalul are 5 linii cu suma 11, deci se afișează 5 11. Cu aceleași date, dar RESET în loc de ADAUGA, jurnalul are 3 linii cu suma 3, deci se afișează 3 3.

HINT *Pentru p = 0 jurnalul de ieri este gol, iar pentru k = 0 nu se adaugă nimic; combinația RESET cu k = 0 lasă un jurnal gol și afișează 0 0.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.10 · INVESTIGAȚIE

Cine e mai rapid, cu adevărat

SCRIERE COD

În secțiunea 6 am comparat citirea dintr-un fișier cu tastarea de la tastatură și am spus că diferența nu vine din viteza procesorului, ci din faptul că fișierul nu are nevoie de un om. Verifică singur afirmația, eliminând omul din ecuație.

Ce ai de făcut. Scrie, în mediul tău de lucru obișnuit (IDLE, VS Code, PyCharm), un program care generează valorile 1, 2, …, n pentru n = 20000, le scrie în fișierul mare.txt, apoi măsoară cu perf_counter() două durate: (1) timpul de citire a fișierului cu readlines() și de transformare a liniilor în numere; (2) timpul de calcul al aceleiași sume direct, fără fișier, dintr-o listă deja existentă în memorie. Afișează cele două durate și raportul lor.

De răspuns în scris. Care dintre cele două este mai lentă și de câte ori? Rămâne diferența la fel de mare ca cea dintre fișier și tastatură din secțiunea 6? Ce parte din diferența de acolo era, de fapt, timpul de reacție al omului și ce parte era munca efectivă a calculatorului?

REZULTATELE ȘI CONCLUZIA TA