

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

Carmen-Diana Cosman • Melinda-Emilia Coriteac
Amelia Stoian • Isabela-Patricia Coman
Miruna-Andreea Zăvelcă • Cristina Elena Anton
Violeta Marilena Grecea • Andreea Floriana Bontea

Informatică

curriculum de specialitate (CS), filiera teoretică,
profilul real, specializarea științe ale naturii

9

Manual pentru clasa a IX-a



Acest manual este proprietatea Ministerului Educației și Cercetării.

Acest proiect de manual școlar este realizat în conformitate cu Programa școlară aprobată prin Ordinul ministrului educației și cercetării nr. 6930/19.12.2025

Numărul de telefon **119** – număr unic de telefon la nivel național pentru cazurile de abuz împotriva copiilor.

Numărul de telefon **116.111** – număr de asistență pentru copii, în conformitate cu Legea nr. 272/2004 privind protecția și promovarea drepturilor copilului, republicată, cu modificările și completările ulterioare.

Ministerul Educației și Cercetării

Carmen-Diana Cosman • Melinda-Emilia Coriteac • Amelia Stoian
Isabela-Patricia Coman • Miruna-Andreea Zăvelcă • Cristina Elena Anton
Violeta Marilena Grecea • Andreea Floriana Bontea

Informatică

**curriculum de specialitate (CS), filiera teoretică,
profilul real, specializarea științe ale naturii**

Manual pentru clasa a **IX**-a



București, 2026

Manualul școlar a fost aprobat prin Ordinul de Ministru al Educației și Cercetării nr. 5779 din data de 09.10.2026. Manualul este distribuit elevilor în mod gratuit, atât în format tipărit, cât și în format digital, și este transmisibil timp de patru ani școlari, începând din anul școlar 2026-2027.

Inspectoratul Școlar
Școala / Colegiul / Liceul

ACEST MANUAL A FOST FOLOSIT DE:

Anul	Numele elevului	Clasa	Anul școlar	Aspectul manualului*			
				Format tipărit		Format digital	
				la primire	la predare	la primire	la predare
1							
2							
3							
4							

* Pentru precizarea aspectului manualului se va folosi unul dintre următorii termeni: **nou, bun, îngrijit, neîngrijit, deteriorat.**

Cadrele didactice vor verifica dacă informațiile înscrise în tabelul de mai sus sunt corecte.

Elevii nu vor face niciun fel de însemnări pe manual.

Referenți științifici:

- **Angela Blaga:** profesor, grad didactic I în Informatică, doctor în Inginerie și management, la Colegiul Național „Alexandru Papiu Ilarian” Târgu Mureș, județul Mureș.
- **Octavia-Maria Bolojan:** șef de lucrări, doctor în Informatică, la Universitatea din Oradea, Facultatea de Inginerie Electrică și Tehnologia Informației, Departamentul de Calculatoare și Tehnologia Informației.
- **Horea-Gavril Oros:** lector universitar doctor, titular la Universitatea din Oradea, Facultatea de Informatică și Științe, Departamentul de Matematică și Informatică.
- **Delia Tușe:** lector universitar, doctor în Informatică, la Universitatea din Oradea, Facultatea de Informatică și Științe, Departamentul de Matematică și Informatică.

Manager proiect: Radu Răcăreanu

Redactor: Dominic Burghilea

Grafică și tehnoredactare: Adrian-Viorel Rașca

Coperta: Adrian-Viorel Rașca

Credite foto și video: Canva, Magnific

Platforma e-learning, activități digitale interactive,

voce, procesare sunet și animații:

IMAGINE INFINITY SRL

© 2026 – EDITURA SIGMA

Toate drepturile asupra prezentei ediții aparțin Editurii SIGMA. Nicio parte a acestei lucrări nu poate fi reprodusă fără acordul scris al Editurii SIGMA.

Editura SIGMA

Sediul central:

Str. Nicolae Cartojan, nr. 11-13-15, sector 2, București,
cod poștal 023815

Tel.: 0758 044 829; 0723 541 020

E-mail: office@editurasigma.ro

Site: www.editurasigma.ro

Distribuție:

Str. Nicolae Cartojan, nr. 11-13-15, sector 2, București,
cod poștal 023815

Tel.: 0758 044 825; 0758 044 828

E-mail: comenzi@editurasigma.ro

Cuvânt-înainte

Manualul de față a fost conceput pentru a vă sprijini în înțelegerea și utilizarea responsabilă a instrumentelor informaticii, printr-o abordare accesibilă și aplicată. Acesta își propune să vă ajute să descoperiți cum gândirea algoritmică poate fi un instrument util în orice domeniu — inclusiv în științele naturii.

Lucrarea este structurată în patru capitole: Fundamentele programării în Python și utilizarea subprogramelor, Prelucrarea datelor numerice și algoritmi de divizibilitate, Organizarea datelor în liste cu aplicații practice, respectiv Metode de sortare și eficiența algoritmilor.

Veți învăța prin rezolvare, nu prin memorare: fiecare lecție vă propune situații în care să aplicați direct ceea ce ați descoperit. Fișele de lucru și activitățile de autoevaluare vă oferă un spațiu de reflecție personală, util atât pentru progres, cât și pentru încredere în propriile capacități.

Vă încurajăm să folosiți acest manual ca pe un ghid practic și ca pe un punct de plecare pentru curiozitate și explorare. Folosiți programarea cu responsabilitate și discernământ — fiecare problemă rezolvată poate deveni o ocazie de a înțelege mai bine lumea din jurul vostru.

Autorii

Manualul digital

Varianta digitală a manualului conține pe lângă varianta tipărită și activități multimedia interactive de învățare (AMII). Acestea vin în completarea noțiunilor și exemplelor prezentate în manualul tipărit și sunt de trei tipuri:  AMII statice (desene, fotografii sau planșe didactice);  AMII dinamice (filme sau animații) și  AMII interactive (diverse exerciții, jocuri educative).

Titlul capitoului — Fișă de lucru — Autoevaluare! Ce am învățat?

Titlul lecției — **Lecția 16 – Sortarea prin selecția minimului**

Definiție — Sortarea reprezintă procesul de aranjare a elementelor unei liste într-o ordine specifică. Este una dintre operațiile fundamentale în informatică; datele ordonate pot fi citate și prelucrate mult mai eficient. Câte mai frecvente variante sunt **ordine crescătoare** și **ordine descrescătoare**. În această lecție vom sorta crescător folosind selecția minimului.

Conținutul lecției — **16.1. Sortarea unei liste**

Exemplu — **16.2. Ideea metodei**

Observație — **16.3. Stabilirea algoritmului pas cu pas**

Alte elemente grafice utilizate în manual

 Știați că!

 Reține!

 Investighează!

 Portofoliu

 Explorează

 Proiect

 Atenție!

CUPRINS

Cuvânt-înainte. Prezentarea manualului	3
Competențe generale și competențe specifice	5
Instrucțiuni privind organizarea modului de lucru al fișelor/aplicațiilor din manual	6
Fișă de observare sistematică a activității și a comportamentului elevului	7
Evaluare inițială	8

Capitolul 1 – Fundamentele programării (1.1.; 1.3.; 1.5.; 2.1.; 2.3.; 2.5.; 3.1.; 3.3.; 3.5.; 4.1.; 4.3.; 4.5.; 5.1.; 5.3.; 5.5.; 6.1.; 6.5.)

Lecția 1 – Etapele elaborării unui program	9
Fișa de lucru 1	10
Lecția 2 – Pseudocod și limbaje de programare	12
Fișa de lucru 2	14
Lecția 3 – Variabile, tipuri de date și structuri de control	15
Fișa de lucru 3	17
Lecția 4 – Subprograme: definiție, parametri, variabile locale/globale	18
Fișa de lucru 4	20
Lecția 5 – Subprograme predefinite pentru calcule și colecții	21
Fișa de lucru 5	22
Evaluare sumativă – Capitolul 1	23
Investighează! • Explorează • Portofoliu	24

Capitolul 2 – Strategii de rezolvare a problemelor. Prelucrări ale datelor numerice (1.1.; 1.2.; 2.1.; 2.2.; 3.1.; 3.2.; 4.2.; 5.2.; 6.1.; 6.2.)

Lecția 6 – Operații cu cifrele unui număr	25
Fișa de lucru 6	27
Lecția 7 – Algoritmi de divizibilitate: Divizorii unui număr; Primalitate	29
Fișa de lucru 7	31
Lecția 8 – Descompunerea în factori primi; CMMDC, CMMM (Euclid)	33
Fișa de lucru 8	36
Evaluare sumativă – Capitolul 2	38
Investighează! • Explorează • Portofoliu	39

Capitolul 3 – Date organizate în liste (1.1.; 1.3.; 2.1.; 2.3.; 3.1.; 3.3.; 4.1.; 4.3.; 5.1.; 5.3.; 6.1.; 6.3.; 6.5.)

Lecția 9 – Modelul conceptual liniar: caracteristici, acces direct vs secvențial; liste de frecvență	40
Fișa de lucru 9	42
Lecția 10 – Clasa list din Python: operatori specifici și metode de bază	45
Fișa de lucru 10	47
Lecția 11 – Metode avansate ale clasei list și parcurgerea elementelor	50
Fișa de lucru 11	52
Lecția 12 – Introducere în POO: clase, obiecte și instanțiere	54
Fișa de lucru 12	57
Lecția 13 – Fișiere text: deschidere, citire, scriere, închidere; clasa TextIOWrapper	59
Fișa de lucru 13	61
Lecția 14 – Generare sistematică: șiruri recurente și secvențe cu proprietăți date	63
Fișa de lucru 14	64
Evaluare sumativă – Capitolul 3	66
Investighează! • Proiect • Portofoliu	67

Capitolul 4 – Metode de sortare și eficiența algoritmilor (1.4.; 2.4.; 3.4.; 4.2.; 4.3.; 4.4.; 5.3.; 5.4.; 5.5.; 6.1.; 6.4.; 6.5.)

Lecția 15 – Eficiența algoritmilor: timp de execuție, estimarea operațiilor	68
Fișa de lucru 15	70
Lecția 16 – Sortarea prin selecția minimului	71
Fișa de lucru 16	73
Lecția 17 – Sortarea cu listă de frecvențe	74
Fișa de lucru 17	76
Evaluare sumativă – Capitolul 4	77
Investighează! • Proiect • Portofoliu	78
Evaluare finală	79
Indicații și răspunsuri	80

Competențe generale și competențe specifice

CG1 – Identifică principalele caracteristici ale modelelor conceptuale și operaționale ale dezvoltării produselor software, pentru înțelegerea fundamentelor programării

- CS 1.1. Identifică principalele caracteristici ale organizării datelor în cadrul unor modele conceptuale fundamentale - date simple sau liste, pentru a structura și accesa date în vederea prelucrării acestora
- CS 1.2. Identifică specificul, caracteristicile și etapele unor algoritmi specializați pe clase de probleme, pentru a îi putea utiliza în prelucrarea algoritmică a numerelor, sortarea sau generarea sistematică a unor secvențe de valori
- CS 1.3. Identifică specificul, caracteristicile și etapele unor strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor
- CS 1.4. Identifică principalele elemente ale limbajului de programare utilizate pentru prelucrarea datelor organizate în modele fundamentale - date simple sau liste, respectiv pentru transmiterea datelor de la și către un program, în vederea rezolvării eficiente a problemelor informatice
- CS 1.5. Identifică elementele de sintaxă și componentele din definiția și apelul subprogramelor și subprogramelor predefinite pentru operații uzuale, pentru a le utiliza în implementarea algoritmilor în limbaj de programare

CG2 – Explică principii care stau la baza modelelor conceptuale și operaționale ale dezvoltării produselor software, pentru a fundamenta în mod logic proiectarea și implementarea soluțiilor informatice

- CS 2.1. Explică principiile de organizare a datelor în cadrul unor modele conceptuale fundamentale - date simple sau liste, pentru a le gestiona eficient
- CS 2.2. Explică etapele algoritmilor specializați pe clase de probleme, pentru a îi putea utiliza în prelucrarea algoritmică a numerelor, sortarea sau generarea sistematică a unor secvențe de valori
- CS 2.3. Explică etapele unor strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor
- CS 2.4. Explică rolul principalelor elemente ale limbajului de programare utilizate pentru prelucrarea datelor organizate în modele fundamentale - date simple sau liste, respectiv pentru transmiterea datelor de la și către un program în vederea rezolvării eficiente a problemelor informatice
- CS 2.5. Explică mecanismul de executare a subprogramelor și subprogramelor predefinite pentru operații uzuale, pentru a le utiliza în implementarea algoritmilor în limbaj de programare

CG3 – Utilizează modele conceptuale și operaționale ale dezvoltării produselor software, în scopul obținerii de soluții informatice funcționale și eficiente

- CS 3.1. Utilizează modul de organizare și prelucrare a datelor în cadrul unor modele conceptuale fundamentale - date simple sau liste, pentru a rezolva probleme de gestionare a datelor
- CS 3.2. Aplică algoritmi specializați pe clase de probleme, pentru a îi putea utiliza în prelucrarea algoritmică a numerelor, sortarea sau generarea sistematică a unor secvențe de valori
- CS 3.3. Aplică strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor
- CS 3.4. Utilizează principalele elemente ale limbajului de programare utilizate pentru prelucrarea datelor organizate în modele fundamentale - date simple sau liste, respectiv pentru transmiterea datelor de la și către un program, în vederea rezolvării eficiente a problemelor informatice
- CS 3.5. Utilizează elementele de sintaxă și componentele din definiția și apelul subprogramelor și subprogramelor predefinite pentru operații uzuale, în implementarea algoritmilor în limbaj de programare

CG4 – Analizează caracteristicile și aplicabilitatea modelelor conceptuale și operaționale ale dezvoltării produselor software, pentru a selecta soluțiile cele mai potrivite în funcție de contextul informatic dat

- CS 4.1. Analizează caracteristicile, modul de prelucrare, avantajele și dezavantajele utilizării unor modele conceptuale fundamentale - date simple sau liste, pentru a alege structura adecvată în rezolvarea unor probleme concrete
- CS 4.2. Analizează comportamentul, aplicabilitatea, avantajele și dezavantajele utilizării unor algoritmi specializați pe clase de probleme pentru prelucrarea numerelor, sortarea sau generarea sistematică a unor secvențe de valori
- CS 4.3. Analizează aplicabilitatea, avantajele și dezavantajele utilizării unor strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor
- CS 4.4. Analizează aplicabilitatea, avantajele și dezavantajele utilizării unor elemente ale limbajului de programare, pentru a identifica soluția adecvată prelucrării datelor organizate în modele fundamentale - date simple sau liste, respectiv pentru transmiterea datelor de la și către un program, în vederea rezolvării eficiente a problemelor informatice
- CS 4.5. Analizează aplicabilitatea, avantajele și dezavantajele utilizării subprogramelor, în implementarea algoritmilor în limbaj de programare

CG5 – Evaluează corectitudinea și eficiența soluțiilor informatice, în vederea optimizării și asigurării funcționalității în diverse scenarii de utilizare

- CS 5.1. Argumentează alegerea unor modele conceptuale fundamentale - date simple sau liste - și a modurilor de prelucrare a acestora pentru rezolvarea unor probleme informatice concrete
- CS 5.2. Evaluează corectitudinea și eficiența soluțiilor cu algoritmi specializați pe clase de probleme pentru prelucrarea numerelor, sortarea sau generarea sistematică a unor secvențe de valori
- CS 5.3. Evaluează corectitudinea și eficiența algoritmilor care utilizează strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor
- CS 5.4. Evaluează corectitudinea și eficiența aplicațiilor care utilizează elemente specifice de limbaj de programare pentru prelucrarea datelor organizate în modele fundamentale - date simple sau liste, respectiv pentru transmiterea datelor de la și către un program, în vederea rezolvării eficiente a problemelor informatice
- CS 5.5. Evaluează corectitudinea și eficiența programelor care utilizează subprograme, pentru a rezolva probleme informatice

CG6 – Elaborează algoritmi și programe personalizate, pentru a crea soluții informatice coerente și adaptate cerințelor

- CS 6.1. Proiectează algoritmi personalizați care utilizează modele conceptuale fundamentale - date simple sau liste pentru organizarea și prelucrarea eficientă a datelor, utilizând algoritmi de bază, în vederea rezolvării unor probleme
- CS 6.2. Proiectează soluții cu aplicarea personalizată a algoritmilor specializați pe clase de probleme pentru prelucrarea numerelor, sortarea sau generarea sistematică a unor secvențe de valori
- CS 6.3. Proiectează modular algoritmi personalizați pentru rezolvarea problemelor
- CS 6.4. Implementează aplicații care integrează în mod personalizat elemente specifice de limbaj de programare pentru prelucrarea datelor organizate în modele fundamentale - date simple sau liste, respectiv pentru transmiterea datelor de la și către un program, în vederea realizării unor soluții funcționale și performante
- CS 6.5. Implementează aplicații în limbaj de programare care integrează subprograme personalizate, pentru a modulariza și organiza eficient codul în cadrul soluțiilor informatice

Instrucțiuni privind organizarea modului de lucru al fișelor/aplicațiilor din manual

Prezentul manual este organizat în patru capitole și 17 lecții, fiecare având la final o fișă de lucru.

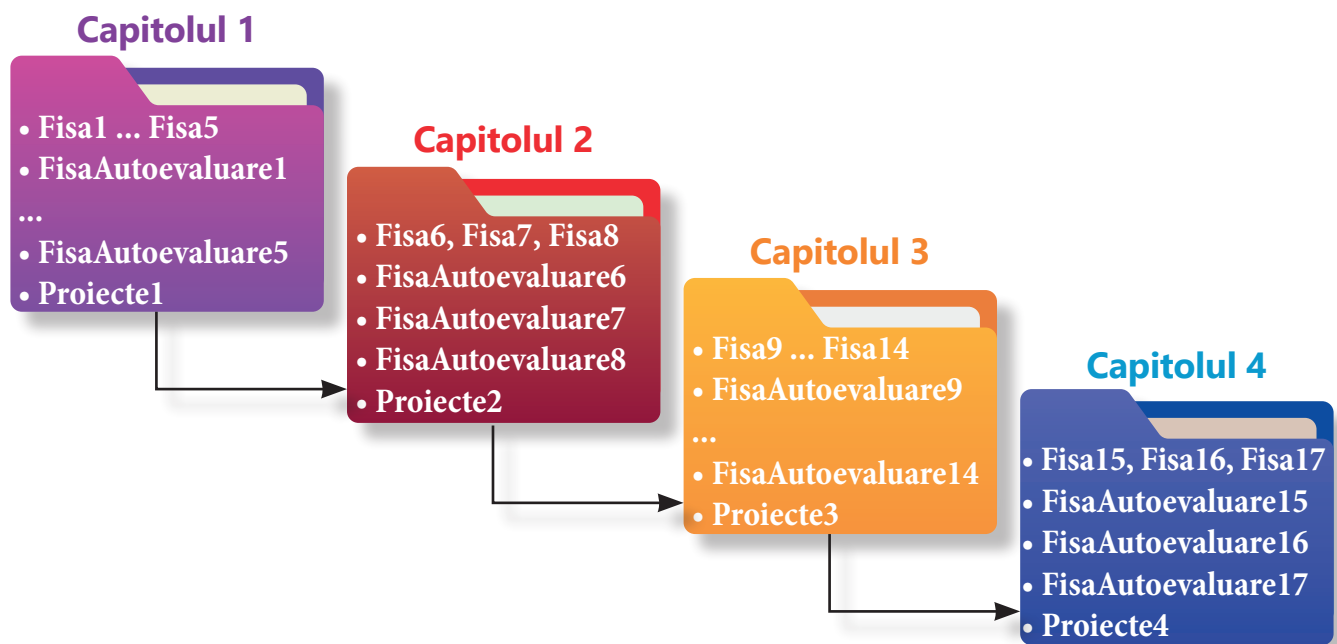
1. Creați pentru fiecare unitate un dosar numit **CapitolNr.**, unde nr. reprezintă numărul de ordine al capitolului (1, 2, 3 sau 4), într-un loc din memoria externă, unde aveți drept de scriere.

2. Creați pentru fiecare fișă de lucru un dosar numit **FisaNr.**, unde nr. reprezintă numărul de ordine al fișei de lucru. Fiecare dosar **FisaNr.** va fi salvat în dosarul **CapitolNr.** corespunzător. (Dosarele **Fisa1**, ..., **Fisa5** se vor salva în dosarul **Capitol1**, dosarele **Fisa6**, ..., **Fisa8** în dosarul **Capitol2**, dosarele **Fisa9**, ..., **Fisa14** în dosarul **Capitol3**, iar dosarele **Fisa15**, ..., **Fisa17** în dosarul **Capitol4**). În aceste dosare salvați fiecare fișier solicitat în fișele de lucru.

3. La finalul fiecărei lecții veți avea la dispoziție o fișă de autoevaluare. Fișele de autoevaluare 1, ..., 5 vor fi salvate în dosarul **FisaNr.** cu același număr aflat în dosarul **Capitol1**; Fișele de autoevaluare 6, ..., 8, în dosarul **Capitol2**; Fișele de autoevaluare 9, ..., 14, în dosarul **Capitol3**; Fișele de autoevaluare 15, ..., 17, în dosarul **Capitol4**. În aceste dosare salvați fiecare fișier solicitat în cerințe.

4. Pentru proiectele (Explorează/Investighează) pe care trebuie să le faceți pe parcurs, creați dosarele **Proiecte1**, **Proiecte2**, **Proiecte3**, respectiv **Proiecte4**, în care salvați fișierele solicitate.

La final, **portofoliul** vostru trebuie să aibă următoarea structură:



La finalul fiecărei lecții vă propunem să vă autoevaluați comportamentul, cu ajutorul fișei specifice, existentă la pagina următoare.

Fișă de observare sistematică a activității și a comportamentului elevului* *Informatică, specializarea științe ale naturii*



Nume și prenume: _____ Data: _____

Tema lecției: _____

Clasa: _____

Scală de evaluare:



1 = deloc



2 = puțin



3 = moderat



4 = bine



5 = foarte bine

Autoevaluare

Activitate / Criteriu	Elev	Profesor
Am fost atent la lecție	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am înțeles conceptele predate	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am răspuns la întrebări	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
M-am implicat activ	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am cerut ajutor când nu am înțeles	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am înțeles sarcinile de lucru	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am rezolvat exercițiile / problemele	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am lucrat eficient în echipă	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am ajutat colegii	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am colaborat constructiv	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am scris / testat codul corect	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Am depanat erorile independent	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Reflecție

 Ce am făcut bine: _____

 Ce pot îmbunătăți: _____

 O idee sau curiozitate din lecție: _____

* Această fișă te ajută să îți dai seama cât de bine ai înțeles lecția și cât de implicat ai fost. Nu este un test, ci o evaluare personală care te ajută să progresezi. Fișa se completează de elev și profesor, iar diferențele de opinie vor fi discutate pentru a identifica direcțiile necesare pentru progres.

Evaluare inițială

Creați dosarul numit *evaluare_initiala*. Rezolvați exercițiile într-un document: *raspunsuri.docx*.

1) (1p) Alegeți varianta corectă:

- a) Un algoritm poate fi infinit dacă problema este complexă;
- b) Un algoritm este o succesiune finită, clară și executabilă de pași care produce un rezultat;
- c) Pseudocodul este un limbaj de programare executabil;
- d) Variabilele de ieșire sunt datele citite de la tastatură.

2) (2p) Completați:

Cele 3 tipuri de erori într-un program sunt: erori de, erori de și erori de

Ce tip de eroare face ca programul să ruleze fără să se oprească, dar să producă rezultate greșite?

3) (3p) Scrieți valoarea și tipul fiecărei expresii C++:

Expresie	$7 / 2$	$7 \% 2$	$3 + 1.0$
Valoare			
Tip			

4) (1.5p) Scrieți ce afișează programul (fără a-l rula):

```
#include <iostream>
using namespace std;
int main()
{
    int n, s = 0;
    cin >> n; // se citește 473
    while (n > 0)
    {
        s += n % 10;
        n /= 10;
    }
    cout << s << endl;
    return 0;
}
```

Valoare afișată:

Ce calculează?

Ce se întâmplă pentru $n = 0$?

5) (1.5p) Scrieți un program C++ care citește n note (numere întregi) de la tastatură, și afișează media lor.

```
#include <iostream>
using namespace std;
int main()
{
    int n, note[100];
    cout << "Nr. note: ";
    cin >> n;
    for (int i = 0; i < n; i++) {
        // ... # citire
    }
    // ... # calcul medie
    // cout << ... << endl;
    return 0;
}
```

Barem de corectare: Se acordă **1 punct** din oficiu!

1) 1 punct pentru răspuns corect. **2) 2 puncte** - $0.5p \times 4$. **3) 3 puncte** - $0.5p/\text{celulă}$. **4) 1.5 puncte** (0.5 puncte valoare + 0.5 puncte ce calculează + 0.5 puncte $n=0$). **5) 1.5 puncte** ($1p$ citire + $0.5p$ medie).

Capitolul 1 – Fundamentele programării

Lecția 1 – Etapele elaborării unui program

„Nu mă tem de calculatoare. Mă tem de lipsa lor.” – **Isaac Asimov**

Informatica. Datele și algoritmul



Informatica este un complex de discipline care se ocupă cu prelucrarea informațiilor cu ajutorul calculatoarelor și este esențială în dezvoltarea *gândirii computaționale*.



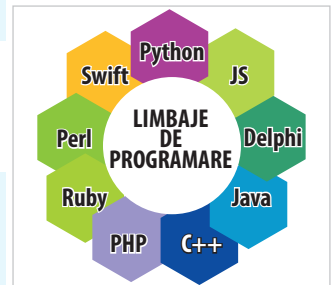
Gândirea computațională este abilitatea omului prin care acesta dezvoltă o metodă de rezolvare a problemelor prin secvențe de instrucțiuni și **algoritmi**. Principalele **caracteristici** sunt: *descompunerea, generalizarea, abstractizarea, algoritmizarea* și *evaluarea*.



Algoritmul este o succesiune finită de pași executați într-o ordine bine definită care prelucrează *date*, acestea identificându-se prin **nume**, **tip** și **valoare**.



Datele se clasifică în funcție de **tip** (numerice, logice, șir de caractere sau structurate), de **momentul utilizării** (intrare, intermediare/de manevră, ieșire) și de **valoare** (variabile și constante).



Criteriu	Categorii
după tip	numerice, logice, șiruri de caractere, structurate
după moment	de intrare, de manevră/intermediare, de ieșire
după valoare	variabile și constante
 Reține!	Un algoritm trebuie să fie finit, clar, executabil și să producă un rezultat pentru datele de intrare acceptate.

Etapele elaborării unui program

Pentru a realiza un program, care implementează un algoritm este necesară parcurgerea etapelor:

Etapă	Rol
Analiza problemei	identificarea datelor de intrare, de ieșire și a relațiilor dintre ele
Proiectarea modulară	descompunerea problemei în subprobleme, alegerea metodei de rezolvare și a modului de organizare a datelor
Implementarea	transpunerea algoritmului într-un limbaj de programare, respectând sintaxa acestuia
Testarea și depanarea	rularea programului pentru a identifica și corecta erorile
Analiza complexității/optimizarea	evaluarea eficienței din punctul de vedere al memoriei și timpului de execuție

Repere pentru aplicarea acestor etape în rezolvarea problemelor

Analiza: înțelegerea cerințelor problemei, stabilirea restricțiilor și a datelor de intrare/ieșire.

Proiectarea: schiță în pseudocod sau diagramă, alegerea structurilor de date potrivite și descompunerea problemei în subprobleme.

Implementarea: organizarea clară și lizibilă a codului.

Testarea: verificarea cu date normale, limită și eronate.

Criterii de elaborare a testelor

Pentru validarea datelor de intrare: verificăm dacă datele respectă tipul așteptat, dacă valorile se încadrează în intervalul permis și testăm valori limită sau invalide.

Pentru verificarea comportamentului algoritmului: testăm cazul general, cazurile speciale, comparăm rezultatele obținute cu cele așteptate și verificăm rulări repetate cu aceleași date.



Tipuri frecvente de erori

Tip eroare	Descriere
eroare de sintaxă	cod incorect din punct de vedere al regulilor limbajului
eroare de logică	programul rulează, dar produce rezultate greșite
eroare de execuție	programul se oprește în timpul rulării, de exemplu împărțire la zero

Exemplu de analiză



Mihai are un număr de n note la disciplina Informatică. Trebuie să determinați media notelor elevului și să verificați dacă a promovat sau nu la disciplina Informatică.



Reține! Testarea și depanarea nu sunt etape opționale. Un program trebuie verificat cu date variate, nu doar cu un exemplu convenabil.

```
citeste n
suma <- 0
pentru fiecare nota executa
    citeste nota
    suma <- suma + nota
media <- suma / n
daca media >= 5 atunci
    afiseaza "promovat"
altfel
    afiseaza "nepromovat"
```



Fișa de lucru 1

Creați dosarul numit *Fisa1*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

1) Alegeți varianta corectă:

1a. Gândirea computațională include:

- a) Descompunere, generalizare, abstractizare, algoritmizare, evaluare;
- b) Programare, compilare, testare, livrare;
- c) Analiză, sinteză, memorare, reproducere;
- d) Pseudocod, schemă logică, Python, C++.

1b. Datele de manevră (intermediare) sunt:

- a) Datele citite de la tastatură;
- b) Datele afișate ca rezultat final;
- c) Datele folosite în calcule pe parcursul programului;
- d) Datele stocate în fișiere.

1c. Care tip de eroare face ca programul să se oprească brusc în timpul rulării?

- a) Eroare de sintaxă;
- b) Eroare de logică;
- c) Eroare de execuție;
- d) Eroare de compilare.

2) Completați:

- a) Cele 5 etape de elaborare a unui program, în ordine, sunt: → → → →
- b) Un algoritm trebuie să fie, ... și să producă un
- c) La testare verificăm programul cu trei categorii de date:, ... și

3) **Problema:** Se citește temperatura unui pacient și se afișează dacă are febră (temperatura > 37.0) sau nu.

Etapa	Date de intrare	Date de ieșire	Date de manevră	Condiția verificată	Un test cu date normale
Răspuns					



4) Identificați tipul fiecărei erori din fragmentele de mai jos și descrieți ce e greșit:

a)

```

algoritm medie
real a, b, medie
citește a, b
medie ← a + b / 2
scrie medie
sfârșit algoritm

```

Tip eroare:

Descriere:

b)

```

algoritm pozitiv
întreg n
citește n
dacă n > 0
    scrie "pozitiv"
sfârșit algoritm

```

Tip eroare:

Descriere:

5) **Evalueați afirmația:** Un elev susține că testarea este necesară doar când programul dă erori la rulare. În rest, dacă compilează, este corect. a) Sunteți de acord? Argumentați cu referire la tipurile de erori. b) Propuneți un contraexemplu concret: un program care compilează și rulează, dar produce un rezultat greșit.



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Enumerați cele 5 etape ale elaborării unui program și rolul fiecăreia într-un tabel.								
ÎNȚELEGERE	2) Explicați diferența dintre o eroare de logică și o eroare de execuție. Dați un exemplu pentru fiecare.								
APLICARE	3) Aplicați etapele de elaborare pentru problema: "Se citesc 3 note și se afișează media și dacă elevul este promovat." Completați un tabel cu date de intrare, ieșire, manevră și scrieți 3 teste.								
ANALIZĂ	4) Analizați: două echipe rezolvă aceeași problemă. Echipa A scrie cod direct, echipa B face analiza și pseudocod mai întâi. Comparați abordarea din perspectiva calității și timpului de depanare.								
EVALUARE	5) Evaluați următorul set de teste pentru programul "citește n , afișează dacă e par sau impar": test1: n=4 , test2: n=7 . Sunt suficiente? Ce cazuri lipsă puteți identifica?								
CREARE	6) Aplicați etapele de elaborare a unui program pentru problema: "Un elev citește n note și afișează media și dacă este premiant (medie ≥ 9.5).". Completați tabelul: <table border="1" data-bbox="862 1093 1413 1270"> <thead> <tr> <th>Etapă</th><th>Ce faceți la această etapă</th></tr> </thead> <tbody> <tr> <td>Analiza</td><td>date intrare, date ieșire, relații</td></tr> <tr> <td>Proiectare</td><td>descriere în cuvinte a pașilor</td></tr> <tr> <td>Testare</td><td>scrieți 3 seturi de date de test</td></tr> </tbody> </table>	Etapă	Ce faceți la această etapă	Analiza	date intrare, date ieșire, relații	Proiectare	descriere în cuvinte a pașilor	Testare	scrieți 3 seturi de date de test
Etapă	Ce faceți la această etapă								
Analiza	date intrare, date ieșire, relații								
Proiectare	descriere în cuvinte a pașilor								
Testare	scrieți 3 seturi de date de test								

Barem: 1p (din oficiu) + 1p (1) + 1p (2) + 1p (3) + 1p (4) + 2.5p (5) + 2.5p (6)

La finalul lecției, știu...		Da	Nu	Nu sunt sigur(ă)
1.	să enumăr și să explic rolul celor 5 etape de elaborare a unui program.			
2.	să identific datele de intrare, de manevră și de ieșire ale unei probleme.			
3.	să clasific erorile (sintaxă, logică, execuție) și dau un exemplu pentru fiecare.			
4.	să elaborez criterii de testare: date normale, limită și eronate.			
5.	să aplic etapele de elaborare pentru o problemă simplă.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 2 – Pseudocod și limbaje de programare

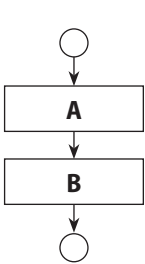
2.1. Reprezentarea algoritmilor

Un **algoritm** poate fi reprezentat prin schemă logică, pseudocod sau limbaj de programare. Fiecare mod de reprezentare are avantaje și limite.

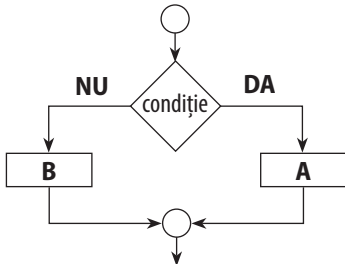
Mod de reprezentare	Caracteristici	Utilizare
schema logică	descriere grafică prin blocuri și săgeți	vizualizarea pașilor algoritmului
pseudocod	descriere textuală apropiată de limbajul natural	proiectarea algoritmului înainte de cod
limbaj de programare	instrucțiuni cu sintaxă strictă, execuție de calculator	implementarea efectivă a algoritmului

2.2. Scheme logice

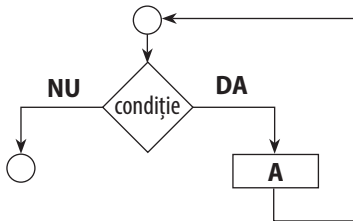
Schema logică este o metodă de descriere a algoritmilor prin reprezentare grafică. Structurile de control sunt descrise prin blocuri, iar ordinea lor de aplicare este marcată prin săgeți.



structura liniară



structura alternativă



structura repetitivă

2.3. Limbajul pseudocod

Pseudocodul este o modalitate intermediară de reprezentare a algoritmilor. El nu este executat direct de calculator, dar permite descrierea clară a pașilor înainte de implementarea într-un limbaj de programare.

Structuri de control în pseudocod

Structură	Formă în pseudocod
liniară	citire date instrucțiune 1 instrucțiune 2 afișare date
alternativă	dacă <condiție> atunci <instrucțiuni 1> altfel <instrucțiuni 2> sfârșit dacă
repetitivă	cât timp <condiție> execută <instrucțiuni> sfârșit cât timp pentru i <- val_inițială, val_finală, pas_execuție execută <instrucțiuni> sfârșit pentru

Structura repetitivă cu test final

repetă <instrucțiuni> până când <condiție>
execută <instrucțiuni> cât timp <condiție>

Exemplu – suma cifrelor unui cod

Miruna are o casetă care este închisă cu un lacăt ce are un cod format din 9 cifre. Pentru a deschide caseta, Miruna trebuie să determine suma cifrelor codului.

```
citeste cod
suma ← 0
cat timp cod > 0 executa
    cifra ← cod % 10
    suma ← suma + cifra
    cod ← cod // 10
sfarsit cat timp
scrie suma
```



Reține! Limbajul Pseudocod permite explicarea algoritmului fără a depinde de un anumit mediu de programare.

2.4. Limbaje de programare



Limbajul de programare reprezintă un mijloc de comunicare între programator și calculator.

Cu ajutorul limbajelor de programare pot fi implementați algoritmi în cadrul unor programe, prin care se rezolvă eficient și într-un mod clar o clasă de probleme.

Tip	Caracteristici
limbaj cu interpretor	instrucțiunile sunt executate pe rând; Python este un limbaj interpretat
limbaj cu compilator	programul este transformat înainte de execuție într-o formă executabilă

Transcriere din pseudocod în Python



Știați că?

Limbajele de nivel scăzut (limbaj de asamblare, cod mașină) sunt mai apropiate de procesor decât Python — instrucțiunile lor corespund direct operațiilor hardware, ceea ce le face rapide, dar greu de citit de oameni.

Interpretorul de comenzi (ex. Bash, Command Prompt) este diferit de un limbaj interpretat: prin el dai instrucțiuni punctuale sistemului de operare (copiezi fișiere, rulezi programe), fără să scrii un algoritm complet. Python, în schimb, este un limbaj interpretat — scrii un program cu logică complexă, pe care interpretorul îl execută linie cu linie.

Pseudocod	Python
citeste n suma ← 0 pentru i ← 1, n executa suma ← suma + i sfarsit pentru scrie suma	n = int(input("n=")) suma = 0 for i in range(1, n + 1): suma += i print(suma)

Acest exemplu are doar rol orientativ pentru a ilustra trecerea de la pseudocod la codul real. Conceptele specifice (cum ar fi *input*, *print*, structura *for* sau indentarea) vor fi studiate detaliat în lecțiile următoare.

Avantaje și limite

- Pseudocodul este intuitiv și ușor de urmărit, dar nu poate fi testat direct.
- Python permite rularea programului, dar cere respectarea strictă a sintaxei și a indentării.



Reține! Înainte de a scrie codul, este util să formulăm pașii algoritmului în pseudocod. Astfel, implementarea devine mai clară și mai ușor de testat.



Fișa de lucru 2

Creați dosarul numit *Fisa2*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

- 1) Scrieți în pseudocod un algoritm care citește două numere și afișează maximul lor.
- 2) Realizați schema logică pentru următorul algoritm: dacă x este par atunci afișează „par”, altfel afișează „impar”.
- 3) Transcrieți în Python structura de pseudocod de mai jos:

Pseudocod	Python
<pre> citește n suma ← 0 pentru i ← 1, n executa citește x suma ← suma + x sfarsit pentru scrie suma </pre>	<pre> n = int(input()) suma = 0 for i in range(____, ____): x = int(input()) ____ print(____) </pre>



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Enumerați cele 3 modalități de reprezentare a unui algoritm și pentru fiecare precizează un avantaj și o limită.
ÎNȚELEGERE	2) Explicați diferența dintre un limbaj interpretat și unul compilat. Încadrați limbajul Python în categoria corectă și justificați.
APLICARE	3) Scrieți în pseudocod și apoi în Python un algoritm care citește n numere și afișează câte sunt negative.
ANALIZĂ	4) Analizați: când este preferabil să folosești for în loc de while ? Dați un exemplu pentru fiecare caz.
EVALUARE	5) Un coleg susține că pseudocodul nu are nicio utilitate deoarece oricum trebuie să scrii codul în Python. Evaluați această afirmație și argumentați pro sau contra.
CREARE	6) Proiectați o soluție pentru "un calculator de medie școlară care citește n note și afișează media, nota maximă și dacă elevul este premiant (medie ≥ 9.5)". Parcurgeți toate etapele, inclusiv 3 teste.

Barem: $1p$ (din oficiu) + $1p$ (1) + $1p$ (2) + $2p$ (3) + $1p$ (4) + $2p$ (5) + $2p$ (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să enumăr modalitățile de reprezentare a unui algoritm cu avantaje și limite.			
2.	să transcriu un algoritm din pseudocod în Python.			
3.	să scriu și urmăresc pas cu pas o structură repetitivă.			
4.	să explic diferența dintre limbaj interpretat și compilat.			

5. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 3 – Variabile, tipuri de date și structuri de control

3.1. Arhitectura unui program Python



Limbajul **Python** este un limbaj interpretat, în care instrucțiunile sunt executate pe rând, de către interpretor. Cel mai simplu program afișează un șir de caractere cu ajutorul funcției `print()`.

```
print("Bună ziua!")
print("Învăț Python")
```

Vocabularul limbajului Python

- **setul de caractere** utilizat pentru scrierea programelor;
- **identificatori** folosiți pentru denumirea variabilelor, constantelor, funcțiilor sau modulelor;
- **cuvinte cheie** cu înțeles special: `if`, `else`, `elif`, `for`, `while`, `continue`, `break`, `return`;
- **separatori**: spațiul, indentarea, parantezele, `:`, `,` sau `;`
- **comentarii** introduse cu ajutorul caracterului `#`.

⚠ **Atenție!** În Python, indentarea este foarte importantă deoarece delimitează blocurile de instrucțiuni și face parte din sintaxă.

3.2. Variabile, citire și afișare

Pentru declararea unei **variabile** în Python, se alege un identificator, apoi i se atribuie o valoare. Tipul variabilei este stabilit automat în funcție de valoarea atribuită.

```
nume = input("Nume: ")
varsta = int(input("Vârsta: "))
print(f"Salut, {nume}! Ai {varsta} ani.")
```

Funcția `print()` este utilizată pentru a afișa mesaje sau valori pe ecran. În interiorul parantezelor putem include text, variabile sau șiruri formate (cu prefixul `f`), elementele fiind separate prin virgulă.

3.3. Tipuri de date



În limbajul Python există tipurile de date: *numerice*, *logice*, *șir de caractere*, *liste*, *dicționare* și *tuple*. Pentru această unitate ne interesează mai ales valorile numerice, valorile logice (`bool`), șirurile de caractere și listele.



Tip de date	Exemplu	Observații
<code>int</code>	<code>x = 7</code>	număr întreg
<code>float</code>	<code>media = 8.5</code>	număr real
<code>str</code>	<code>nume = "Ana"</code>	șir de caractere
<code>list</code>	<code>note = [10, 8, 9]</code>	colecție ordonată, indexată de la 0
<code>bool</code>	<code>promovat = True</code>	valoare logică: <code>True</code> sau <code>False</code>

Despre `input()`



Datele citite cu `input()` sunt salvate implicit sub forma unui șir de caractere. Pentru calcule, folosim conversii precum `int()` și `float()`.

Expresii și operatori



Expresia este o succesiune alcătuită din operanzi, legați prin operatori. *Exemplu:* $12 - 3 * 2$.
O expresie se evaluează ținând cont de precedența operatorilor.



Categorie	Operatori
aritmetici	+, -, *, **, /, //, %
relaționali	>, >=, <, <=, ==, !=

Categorie	Operatori
logici	not, and, or
atribuire	=, +=, -=, *=, /=, //=, %=



Reține! Aritmetici: ** (ridicare la putere), // (câtul împărțirii întregi), % (restul împărțirii / modulo). Operatorii % și // sunt foarte utilizați în prelucrarea cifrelor unui număr ($n \% 10$ extrage ultima cifră, iar $n // 10$ elimină ultima cifră). Atribuire compusă: +=, -=, *=, /=, //=, %= presupun actualizarea rapidă a unei variabile.

3.4. Structuri de control în Python

Structurile de control stabilesc ordinea în care sunt executate instrucțiunile unui program: liniar, alternativ sau repetitiv.



Structura alternativă

```
a = int(input("a = "))
b = int(input("b = "))
if a > b:
    print("primul copil este mai mare")
else:
    print("al doilea copil este mai mare")
```

```
if conditie1:
    # instrucțiuni dacă conditie1 este adevărată
elif conditie2:
    # instrucțiuni dacă conditie2 este adevărată
else:
    # instrucțiuni dacă nicio condiție anterioară
    # nu este adevărată
```



Structura repetitivă cu număr necunoscut de pași

```
n = int(input("n = "))
cnt = 0
while n > 0:
    cnt += 1
    n //= 10
print(cnt)
```

```
while conditie:
    # instrucțiuni executate repetat
    # atât timp cât conditia este True (adevărată)
```

Cazul cu test final (while și break): În situațiile în care condiția de oprire trebuie verificată la sfârșitul corpului repetitiv (echivalentul structurii repeat...until sau do...while din alte limbaje), în Python se folosește o buclă infinită controlată prin instrucțiunea break:

```
while True:
    # instrucțiuni executate cel puțin o dată
    if conditie_oprire:
        break # ieșire din buclă
```

Structura repetitivă cu număr cunoscut de pași

```
n = int(input("n = "))
produs = 1
for i in range(1, n + 1):
    produs *= i
print(produs)
```

```
for variabila in secventa:
    # instrucțiuni executate pentru fiecare element din secvență
```

Cu număr cunoscut de pași (folosind range): atunci când știm exact de câte ori se execută un bloc, generăm o secvență numerică:

```
for i in range(1, n + 1):
    # instrucțiuni repetate de n ori
```

⚠ Atenție! La fel ca în cazul structurii alternative, este foarte importantă indentarea instrucțiunilor din cadrul structurii repetitive.



Fișa de lucru 3

Creați dosarul numit *Fisa3*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.



- 1) Ce va afișa codul: `x = "10"; y = 5; print(int(x) + y)`?
- 2) Utilizând limbajul Python, citiți de la tastatură lungimea și lățimea unui dreptunghi și afișați aria folosind variabile potrivite.
- 3) Scrieți un program în Python care citește un număr și afișează dacă este pozitiv, negativ sau zero.
- 4) Scrieți un program în Python care citește *n* și afișează suma numerelor de la *1* la *n* folosind `for` și `range()`.



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Enumerați cele 5 tipuri de date de bază în Python cu câte un exemplu pentru fiecare.
ÎNȚELEGERE	2) Explicați diferența dintre operatorii <code>/</code> și <code>//</code> în Python. Calculați <code>17 / 3</code> și <code>17 // 3</code> și explicați rezultatele.
APLICARE	3) Scrieți un program Python care citește nota unui elev (real) și afișează calificativul: "Excelent" (≥ 9), "Bine" (≥ 7), "Suficient" (≥ 5) sau "Insuficient".
ANALIZĂ	4) Analizați: de ce <code>int('3.14')</code> generează eroare dar <code>int(3.14)</code> funcționează? Explicați diferența dintre conversia dintr-un șir și trunchierea unui float.
EVALUARE	5) Evaluați: <i>n</i> elev dorește să afle media aritmetică folosind împărțirea întreagă: <code>(a + b + c) // 3</code> . Este această variantă corectă pentru calculul mediei cu zecimale? Testați cu <code>a=7, b=8, c=9</code> (și apoi cu valori a căror sumă nu este divizibilă cu 3, de ex. 7, 8, 10).
CREARE	6) Proiectați un program "Calculator de factură" care citește prețul unitar, cantitatea și cota TVA (%), calculează și afișează subtotalul, valoarea TVA și totalul de plată.

Barem: $1p$ (din oficiu) + $1p$ (1) + $1p$ (2) + $2p$ (3) + $1p$ (4) + $2p$ (5) + $2p$ (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să identific tipul unei variabile Python.			
2.	să folosesc operatorii aritmetici inclusiv <code>//</code> , <code>%</code> și <code>**</code> .			
3.	să convertesc date între tipuri folosind <code>int()</code> , <code>float()</code> , <code>str()</code> .			
4.	să scriu structuri alternative (<code>if / elif / else</code>) corect indentate.			
5.	să scriu structuri repetitive pentru diferite cerințe.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 4 – Subprograme: definiție, parametri, variabile locale/globale

4.1. Ce este un subprogram?



Un **subprogram** (sau funcție în Python) este un bloc de instrucțiuni identificat printr-un nume, care rezolvă o sarcină specifică și poate fi reutilizat în mai multe locuri din program.

Rolul subprogramelor este de a permite organizarea modulară a codului: un program complex este descompus în subprograme mai mici, fiecare rezolvând o parte a problemei.

Avantaj	Explicație
Reutilizarea codului	un subprogram se scrie o singură dată și se apelează de oricâte ori
Lizibilitate	programul devine mai ușor de citit și înțeles
Testare și depanare	fiecare subprogram poate fi testat independent
Colaborare	membrii echipei pot lucra independent pe subprograme diferite

i 4.2. Structura unui subprogram în Python

Componentă	Descriere
Antetul (header)	prima linie, conține cuvântul cheie def , numele funcției și parametrii
Corpul (body)	liniile indentate sub antet; conține instrucțiunile care se execută la apel
return	instrucțiunea opțională care trimite înapoi rezultatul; fără return, funcția returnează None

```
def nume_functie(parametru1, parametru2):
    instructiune1
    instructiune2
    return rezultat # opțional
```

4.3. Parametri, argumente și returnarea rezultatelor



Parametrii sunt variabile prin care funcția primește date din exterior. La apel, valorile transmise se numesc **argumente**. Acestea permit utilizarea aceleiași funcții pentru valori diferite.



Funcția minimum

```
def minim(a, b):
    if a < b:
        return a
    else:
        return b

x = minim(3, 4)
y = minim(10, 7)
print(x, y)
```

1. Parametrii funcției (Datele de intrare)

În secvența **def minim(a, b):**, variabilele **a** și **b** reprezintă parametrii funcției. Ei se comportă ca niște variabile locale temporare care preiau valorile transmise în momentul apelului.

2. Corpul funcției și logica decizională

În funcție, se execută o structură alternativă pentru a compara valorile stocate în parametri.

3. Instrucțiunea **return**:

Cuvântul cheie **return** are un dublu rol:

- Întrerupe imediat execuția funcției.
- Trimite valoarea rezultată înapoi în locul din program unde a fost apelată funcția.

Valori implicite ale parametrilor

Există situații în care ne așteptăm ca un parametru să aibă o valoare în mod normal, dar vrem să îl putem modifica în situații speciale.

```
def calculeaza_nota(nota, oficiu = 1, maxim = 10):
    return min(nota + oficiu, maxim)

print(calculeaza_nota(3))           # 4
print(calculeaza_nota(10))          # 10
print(calculeaza_nota(3, 2, 5))     # 5
```



Reține! Instrucțiunea `return` transmite rezultatul calculat înapoi la locul din program de unde a fost apelat subprogramul. După executarea lui **`return`**, subprogramul se încheie imediat.

i 4.4. Mecanismul de execuție și variabilele locale/globale

Atunci când programul vede o funcție se oprește, transmite valorile parametrilor către antetul funcției, rulează corpul subprogramului cu valorile primite, returnează rezultatul în programul în care a fost apelat și continuă execuția de unde s-a întrerupt.

Pas	Ce se întâmplă
1	se oprește execuția din programul principal la întâlnirea apelului
2	se creează un spațiu de memorie nou pentru subprogram
3	valorile argumentelor se transmit parametrilor funcției
4	se execută corpul funcției și se returnează rezultatul
5	execuția programului principal continuă de unde s-a oprit

i 4.5. Variabile locale vs. variabile globale

Tip	Caracteristici
locale	sunt definite în interiorul unei funcții; există doar pe durata execuției funcției; parametrii sunt variabile locale
globale	sunt definite în afara oricărei funcții; pot fi citite din funcții, dar nu modificate fără cuvântul cheie global

```
tva = 0.19 # variabilă globală
def calculeaza_pret(pret):
    total = pret + pret * tva # total este variabilă locală
    return total

print(calculeaza_pret(100))
```

⚠ Atenție! Utilizarea excesivă a variabilelor globale poate îngreuna depanarea programului. De aceea, este preferabil ca funcțiile să primească date prin parametri și să returneze rezultate.



Fișa de lucru 4

Creați dosarul numit *Fisa4*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

- 1) Scrieți o funcție `cel_mai_mare(a, b, c)` care returnează maximul dintre trei numere.
- 2) La apelul `f(3, 5)`, valorile 3 și 5 sunt:
 - a) Parametri formali. b) Variabile globale.
 - c) Argumente. d) Constante.
- 3) Precizați ce variabile sunt locale și ce variabile sunt globale în secvența alăturată.

```
tva = 0.19
reducere = 0.10
def calculeaza_total(pret, cantitate):
    subtotal = pret * cantitate
    total = subtotal * (1 - reducere) * (1 + tva)
    return total
rezultat = calculeaza_total(100, 3)
print(rezultat)
```



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Enumerați componentele unui subprogram Python (antet, corp, return) și explicați rolul fiecăreia.
ÎNȚELEGERE	2) Explicați diferența dintre parametru și argument. De ce sunt utile valorile implicite ale parametrilor?
APLICARE	3) Scrieți o funcție <code>calculeaza_aria_triunghi(baza, inaltime)</code> și o funcție <code>calculeaza_perimetru_dreptunghi(a, b)</code> . Apelați-le și afișați rezultatele.
ANALIZĂ	4) Analizați avantajele și dezavantajele folosirii variabilelor globale față de transmiterea prin parametri. Când este acceptabilă folosirea unei variabile globale?
EVALUARE	5) O funcție <code>calculeaza_medie(lista)</code> este apelată cu o listă goală <code>[]</code> . Ce se întâmplă? Evaluati corectitudinea următorului cod și realizați îmbunătățiri dacă e necesar: <code>if len(lista) == 0: return 0.</code>
CREARE	6) Proiectați modular o aplicație "Calculator de factură cu discount". Definiți: <i>funcția</i> <code>calculeaza_subtotal(pret, cantitate)</code> , <i>funcția</i> <code>aplica_discount(subtotal, procent)</code> , <i>funcția</i> <code>calculeaza_tva(suma, cota=19)</code> , <i>funcția</i> <code>total_final(subtotal, discount, tva)</code> .

Barem: 1p (din oficiu) + 1p (1) + 1p (2) + 1p (3) + 1p (4) + 1p (5) + 4p (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să definesc și să apelez o funcție Python cu parametri și valoare returnată.			
2.	să identific variabilele locale și globale dintr-un program.			
3.	să explic mecanismul de execuție al unui apel de funcție.			
4.	să folosesc valori implicite ale parametrilor.			
5.	să identific diferența dintre return și print într-o funcție.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 5 – Subprograme predefinite pentru calcule și colecții

i 5.1. Subprograme predefinite pentru calcule matematice



Python include o serie de funcții încorporate care pot fi apelate direct pentru a efectua calcule matematice rapide sau pentru a converti tipurile de date între ele.

Funcție	Rol	Exemplu
<code>abs(x)</code>	returnează valoarea absolută	<code>abs(-25) → 25</code>
<code>math.sqrt(x)</code>	returnează radicalul unui număr, întotdeauna de tip float	<code>math.sqrt(16) → 4.0</code>
<code>round(x,n)</code>	rotunjește valoarea numerică la <i>n</i> zecimale, dacă <i>n</i> nu este specificat, rezultatul este un număr întreg	<code>round(15.7) → 16</code>
<code>int(x)</code>	convertește la număr întreg	<code>int("12") → 12</code>
<code>float(x)</code>	convertește la număr real	<code>float("3") → 3.0</code>
<code>str(x)</code>	convertește la șir de caractere	<code>str(25) → "25"</code>

Modulul math

Pentru operații matematice mai avansate, Python utilizează modulul *math*. Acesta trebuie importat la începutul programului.



math.sqrt(x) calculează **radicalul** și returnează întotdeauna un număr de tip float. Argumentul trebuie să fie pozitiv sau zero.

```
import math
x = -25
print(abs(x))
print(math.sqrt(abs(x)))
```

i 5.2. Subprograme predefinite pentru colecții de valori

Atunci când lucrăm cu seturi de date, cum ar fi listele, avem nevoie de mecanisme rapide pentru a extrage informații despre întregul grup de elemente. Python oferă subprograme predefinite care scu-

Funcție	Ce returnează	Exemplu
<code>len(colecție)</code>	numărul de elemente	<code>len([10, 8, 9]) → 3</code>
<code>min(colecție)</code>	cea mai mică valoare	<code>min([10, 8, 9]) → 8</code>
<code>max(colecție)</code>	cea mai mare valoare	<code>max([10, 8, 9]) → 10</code>
<code>sum(colecție)</code>	suma tuturor elementelor	<code>sum([10, 8, 9]) → 27</code>

tesc programatorul de scrierea unor bucle pentru calcule simple.



Analiza unei liste de note din catalog

```
note = [10, 8, 9, 7, 10]
print("Număr note:", len(note))
print("Nota minimă:", min(note))
print("Nota maximă:", max(note))
media = sum(note) / len(note)
print("Media:", round(media, 2))
```



Reține! Aceste funcții parcurg colecția și returnează rezultatul într-o formă clară, reducând codul scris manual pentru calcule simple.

Cazuri de atenție

- *min()* și *max()* nu pot fi aplicate pe o colecție vidă;
- *sum()* funcționează pe colecții de numere;
- *int("3.14")* generează eroare, deoarece șirul nu reprezintă un număr întreg valid.



Fișa de lucru 5

Creați dosarul numit *Fisa5*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.



1) Alegeți varianta corectă:

1a. Ce returnează `round(2.5)` în Python 3?

- a) 2 – rotunjire bancară (cel mai apropiat număr PAR); b) 3 – rotunjire clasică;
c) 2.5 – nu se rotunjește; d) eroare.

1b. Ce diferență există între `int(-3.9)` și `round(-3.9)`?

- a) Sunt identice: returnează -4; b) `int()` trunchiază spre zero → -3; `round()` rotunjește în jos → -4;
c) `int()` returnează -4; `round()` returnează -3; d) Nu există diferență pentru numere negative.

1c. `sum([])` returnează: a) eroare; b) None; c) 0; d) [].

2) Scrieți în Python, folosind funcții predefinite, un program care citește o listă de n prețuri și calculează: prețul minim, maxim, suma totală și media.

3) Scrieți o funcție `distanța(x1, y1, x2, y2)` care calculează distanța dintre două puncte.



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Enumerați funcțiile predefinite pentru colecții (<code>len</code> , <code>min</code> , <code>max</code> , <code>sum</code>) cu rolul și un exemplu de apel pentru fiecare.
ÎNȚELEGERE	2) Explicați de ce <code>round(4.5)</code> returnează 4, nu 5 în Python 3. Numiți mecanismul și explicați utilitatea lui.
APLICARE	3) Scrieți o funcție <code>analizeaza_note(note)</code> care afișează media rotunjită la 2 zecimale, nota minimă, nota maximă și numărul de note. Apelați-o cu <code>note = [7, 8, 9, 10, 5]</code> .
ANALIZĂ	4) Analizați diferențele dintre <code>int()</code> și <code>round()</code> pentru numere reale. Calculați pentru <code>x = -3.7</code> și <code>x = 3.7</code> și explicați rezultatele.
EVALUARE	5) Evaluați: Un elev calculează media cu <code>sum(note) / len(note)</code> fără verificări. Identificați toate scenariile în care acest cod poate produce erori sau rezultate neașteptate.
CREARE	6) Proiectați o aplicație Python care citește temperaturile zilnice pentru o săptămână și afișează: temperatura medie, minimă, maximă, amplitudinea termică (max-min), și câte zile au depășit media.

Barem: 1p (din oficiu) + 1p (1) + 1p (2) + 1p (3) + 1p (4) + 2p (5) + 3p (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să folosesc <code>abs()</code> , <code>round()</code> , <code>int()</code> , <code>float()</code> , <code>str()</code> cu exemple.			
2.	să import modulul <code>math</code> și să-l folosesc.			
3.	să aplic <code>len()</code> , <code>min()</code> , <code>max()</code> , <code>sum()</code> pe liste de numere.			

4. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Evaluare sumativă - Capitolul 1

Rezolvați exercițiile într-un document: *evaluare_sumativa1.docx*.**1) (1.5p) Asociați fiecare etapă cu rolul său (scrieți litera în dreptul numărului):**

- | | |
|----------------------|-------------------------------------|
| 1. Analiză ____ | A. Scriere cod în Python |
| 2. Proiectare ____ | B. Identificare date intrare/ieșire |
| 3. Implementare ____ | C. Corectare erori |
| 4. Testare ____ | D. Verificare cu date variate |
| 5. Depanare ____ | E. Pseudocod și structuri de date |

2) (1p) Completați:

O funcție Python care nu are instrucțiunea return returnează

Variabilele definite în interiorul unei funcții se numesc variabile

Care este diferența între un parametru formal și un argument?

3) (1p) Calculați valorile (fără a rula codul):

abs(-15)	round(3.5)	round(4.5)	int(-3.9)

4) (1.5p) Completați subprogramul și apelați-l:

```
def calculeaza_medie(note):
    if ____ == 0:
        return None
    return ____ / ____
m = calculeaza_medie([7, 8, 9, 10])
print(m)
```

Ce afișează print(m):

Ce returnează calculeaza_medie([])?

De ce verificăm len(note) == 0?

5) (2p) Identificați și corectați cele 2 erori din codul următor:

```
def suma_patrate(a, b)
    rez = a**2 + b**2
return rez
print(suma_patrate(3, 4))
print(suma_patrate(5, 12))
```

Eroarea 1 (linia):

Corectare:

Eroarea 2 (linia):

Corectare:

6) (2p) Scrieți o funcție *converteste(suma, curs=5.0)* care primește o sumă în RON și cursul de schimb și returnează echivalentul în EUR rotunjit la 2 zecimale. Apelați-o pentru 250 RON și pentru 1000 RON cu curs 4.97.**Barem de corectare:** Se acordă **1 punct** din oficiu!

1) 1.5 puncte (0.3p/ asociere). **2) 1 punct**. **3) 1 punct** - 0.25p/celulă **4) 1.5 puncte** **5) 2 puncte** (1p/eroare)
6) 2 puncte (1p funcție + 1p apeluri)



Investighează!

(100 puncte)

Realizați, într-un fișier **investigheaza1.docx**, o investigație despre etapele elaborării unui program, parcurgând următorii pași:

- Alegeți o problemă simplă din viața reală (Exemplu: calculul mediei semestriale, conversia unei sume de bani) și parcurgeți toate cele 5 etape de elaborare a unui program: analiză, proiectare (pseudocod), implementare (Python), testare și depanare. **(20p)**
- Identificați și notați pentru fiecare etapă: ce ați făcut, ce dificultăți ați întâmpinat și cum le-ați rezolvat. **(20p)**
- Scrieți cel puțin 3 teste (date normale, date limită, date eronate) și notați rezultatele obținute. **(20p)**
- Descrieți un caz în care programul dvs. dă o eroare de execuție și explicați cum îl corectați. **(20p)**
- Reflecție finală: ce ați învățat despre procesul de elaborare a unui program? (5-8 rânduri) **(20p)**

Sugestie de structură: 1. Titlul investigației și ipoteza inițială. 2. Etapele parcurse (cu scurte capturi de ecran/ fragmente de cod). 3. Concluziile desprinse după testare.

20p	20p	20p	20p	20p
Analiză + proiectare	Implementare Python	Teste (3 cazuri)	Eroare identificată + corectată	Reflecție
Recomandare: Se lucrează individual. Fișierul se salvează în dosarul Capitol1.				



Explorează

(100 puncte)

Realizați, în echipă, o aplicație Python modulară numită **calculator_note.py**, care gestionează notele unui elev la toate disciplinele.

- Funcție **citire_note(disciplina)**: citește **n** note de la tastatură și le returnează într-o listă. **(20p)**
- Funcție **calculeaza_medie(note)**: returnează media rotunjită la 2 zecimale; tratează lista goală. **(20p)**
- Funcție **este_promovat(medie, prag=5.0)**: returnează **True/False** cu mesaj sugestiv. **(15p)**
- Funcție **afiseaza_raport(disciplina, note, medie)**: afișează un raport formatat cu toate informațiile. **(20p)**
- Program principal care apelează toate funcțiile pentru cel puțin 3 discipline. **(15p)**
- Aspectul codului: comentarii, nume sugestive de variabile, cod lizibil. **(10p)**

20p citire_note	20p calculeaza_medie	15p este_promovat	20p afiseaza_raport	25p Program+cod
Recomandare: Echipe de 3-4 elevi. Fiecare elev implementează cel puțin o funcție. Se prezintă în fața clasei.				



Portofoliu

(100 puncte)

Portofoliul digital reprezintă „cartea de vizită” a progresului vostru. Toate fișierele cerute vor fi salvate ordonat într-un dosar principal, pentru evaluarea ritmică și autoevaluarea competențelor. La finalul acestui capitol, portofoliul personal al elevului trebuie să cuprindă, în dosarul **Capitol1**, următoarele materiale:

- Dosarele **Fisa1**, **Fisa2**, **Fisa3**, **Fisa4**, **Fisa5**, unde fiecare dosar conține fișierele **raspunsuri.docx**, **FisaAutoevaluare1.docx**, **FisaAutoevaluare2.docx**, **FisaAutoevaluare3.docx**, **FisaAutoevaluare4.docx**, **FisaAutoevaluare5.docx**.
- Dosarul **Proiecte1** va conține: **investigheaza1.docx**, **calculator_note.py**.
- Fișierul **evaluare_sumativa1.docx**.

Modalitate de evaluare:

- Există dosarul **Capitol1** cu structura de directoare specificată – **20p**
- Există în dosare:
 - peste 95% din fișiere – **80p** • între 75% și 94% din fișiere – **60p**
 - între 50% și 74% din fișiere – **40p** • între 25% și 49% din fișiere – **20p** • sub 25% – **0p**

Capitolul 2 – Strategii de rezolvare a problemelor. Prelucrări ale datelor numerice

Lecția 6 – Operații cu cifrele unui număr

Prelucrarea cifrelor unui număr natural reprezintă o clasă importantă de probleme în algoritmică. Algoritmii din această categorie exploatează proprietățile sistemului pozițional de numerație: fiecare cifră poate fi extrasă prin operațiile de împărțire întreagă și modulo.

6.1. Scrierea pozițională a numerelor naturale și accesul la cifre

Orice număr natural poate fi privit ca o succesiune de cifre scrise în baza **10**. Exploatăm proprietățile sistemului pozițional prin două operații de bază:

Operație	Expresie Python	Exemplu
Extrage ultima cifră	<code>c = n % 10</code>	<code>473 % 10 = 3</code>
Șterge ultima cifră	<code>n = n // 10</code>	<code>473 // 10 = 47</code>
Adaugă cifra c la dreapta	<code>n = n * 10 + c</code>	<code>47 -> 479</code>
Adaugă cifra c la stânga	<code>n = c * pow(10, x) + n</code>	<code>7 + 456 -> 7456</code>

Repere pentru parcurgerea cifrelor unui număr

Pentru a prelucra cifrele unui număr natural, se aplică în mod repetat două operații aritmetice de bază:

- restul împărțirii la **10** (`n % 10`) → **extrage ultima cifră** a numărului
- împărțirea întreagă la **10** (`n // 10`) → **elimină ultima cifră**, scurtând numărul

 **Atenție!** Dacă `n = 0`, bucla `while n > 0` nu se execută deloc. Tratați întotdeauna acest caz separat, înainte de buclă, cu `if n == 0`.

6.2. Parcurgerea cifrelor de la dreapta la stânga

Aplicăm repetat extragerea și ștergerea ultimei cifre.

Bucla **`while n > 0`** se termină când nu mai există cifre.

! Cazul **`n = 0`**: Bucla nu se execută deloc. Tratați separat:

```
if n == 0: print(0)
```

```
n = int(input("n: ")) # citim n
if n == 0:
    print(0)
while n > 0:
    c = n % 10 # extrage ultima cifra
    print(c) # prelucreaza cifra
    n //= 10 # sterge ultima cifra
```

Iterație	n	<code>c = n % 10</code>	Cifra afișată
1	473	<code>473 % 10 = 3</code>	3
2	47	<code>47 % 10 = 7</code>	7
3	4	<code>4 % 10 = 4</code>	4
STOP	0	condiția <code>while n > 0</code> este falsă	Bucla se încheie

6.3. Construirea unui număr prin adăugarea de cifre

Pornind de la un număr existent, putem construi numere noi adăugând o cifră fie la sfârșitul său (la dreapta), fie la începutul său (la stânga).

Adăugarea unei cifre la dreapta

Adăugând cifra c la dreapta lui n , obținem un număr cu o cifră mai mult la final. Fiecare cifră veche urcă o poziție (valoarea sa pozițională se înmulțește cu 10), iar noua cifră ocupă poziția unităților.

Formula: $n_nou = n \times 10 + c$

Iterație	n	c	n_nou	Afișare
1	456	9	$456 \times 10 + 9 = 4569$	4569

```
n = 456
c = 9
n_nou = n * 10 + c
print(n_nou) # 4569

# Verificare pas cu pas:
# 456 * 10 = 4560 (toate cifrele
# vechi urca o pozitie)
# 4560 + 9 = 4569 (cifra noua ocupa
# pozitia unitatilor)
```

6.4. Oglinditul unui număr



Oglinditul unui număr natural este numărul obținut prin inversarea ordinii cifrelor sale.



De exemplu, oglinditul lui 1234 este 4321, iar al lui 1200 este 21 (zerourile de la început se pierd în reprezentarea numerică).

Formula: $ogl = ogl \times 10 + c$

La fiecare pas: deplasăm cifrele construite cu o poziție la stânga și adăugăm noua cifră la final.

! Atenție! n se MODIFICĂ în structura repetitivă. La final $n = 0$, nu valoarea inițială!

Dacă vrei să compari oglinditul cu n inițial, salvează n înainte de structura repetitivă.

Pașii algoritmului: Extrage cifra unităților din numărul curent cu $n \% 10$, adăugă cifra la sfârșitul oglinditului construit până în acel moment, apoi elimină cifra procesată prin $n // 10$. Procesul se repetă cât timp mai există cifre.

```
n = int(input("n = "))
inv = 0
while n > 0:
    inv = inv * 10 + n % 10
    n //= 10
print(inv)
```

6.5. Adăugarea unei cifre la stânga

Adăugând cifra c la stânga lui n , cifra nouă devine cea mai semnificativă. Valoarea sa este $c \times 10^x + n$, unde x este numărul de cifre ale lui n . Prin urmare, trebuie să determinăm câte cifre are n înainte de a aplica formula.

Formula: $n_nou = c \times 10^x + n$

n	c	nr_cifre	n_nou	Afișare
456	7	3	$7 \times 10^3 + 456$	7456

```
n = 456
c = 7
nr_cifre = len(str(n)) # nr_cifre = 3
n_nou = c * (10 ** nr_cifre) + n
print(n_nou) # 7456
```



Funcția `str()` convertește valoarea numerică într-un text, iar `len()` returnează lungimea acestuia. Aceasta este o modalitate rapidă în Python de a determina numărul de cifre al unui număr natural.

6.6. Parcurgerea cu contor de poziții

Când avem nevoie să știm poziția unei cifre (de la dreapta la stânga, de la 1), adăugăm un contor d separat.



$n = 12345 \rightarrow$ cifre pe poziții pare (2,4) = cifrele 4 și 2 $\rightarrow$ produsul = 8.

Iterație	n	p	d	c	poziție pară	Afișare
1	12345	1	1	5	nu	
2	1234	4	2	4	da	
3	123	4	3	3	nu	
4	12	8	4	2	da	
5	1	8	5	1	nu	
6	0					8

```
n = int(input())
p = 1
d = 1 # contor poziție
while n > 0:
    c = n % 10
    if d % 2 == 0: # poziție pară
        p = p * c
    d = d + 1
    n = n // 10
print(p)
```



Fișa de lucru 6

Creați dosarul numit *Fisa6*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

1) Citiți un număr natural n . Afișați suma cifrelor și numărul de cifre. *Exemplu:* $n = 12345$ afișează $s = 15$, $\text{cifre} = 5$; $n = 0$ afișează $s = 0$, $\text{cifre} = 1$ (caz particular!); $n = 9$ afișează $s = 9$, $\text{cifre} = 1$.

Parcurerea cifrelor și filtrarea acestora

Putem parcurge toate cifrele și putem prelucra doar o parte dintre acestea. Algoritmul alăturat determină produsul cifrelor nenule ale unui număr n , respectiv suma cifrelor ce au în față cifra 3.

2) Modificați „Numărul de cifre” astfel încât să afișezi numărul de cifre divizibile cu 3.

```
n = 456
nr_cifre = 1
while n > 9:
    c = n % 10 # extrage ultima cifră
    nr_cifre = nr_cifre + 1
    n /= 10 # șterge ultima cifră
print( nr_cifre )
```

```
n = 1035402
produs = 1
s = 0
while n > 0:
    c = n % 10
    if c != 0: # selectăm cifrele nenule
        produs = produs * c
    if n > 9 and n // 10 % 10 == 3:
        s += c
    n = n // 10
print( produs ) # 1*2*3*4*5 = 120
print ( s )
```

3) Modifică algoritmul pentru a număra câte cifre sunt urmate de o cifră nulă. *Exemplu:* pentru $n = 1035402$ cifra 1 și cifra 4 sunt urmate de o cifră 0, deci vom afișa 2.

4) Citiți un număr natural n și o cifră k . Determinați câte cifre sunt egale cu cifra k .

5) Construiți un număr format numai din cifrele pare ale unui număr citit, păstrând ordinea de apariție.



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE

1) Asociați fiecare expresie cu semnificația sa:

Expresie	Semnificație
$n // 10$	adaugă cifra 5 la sfârșitul numărului n
$n \% 10$	șterge ultima cifră din n
$n * 10$	adaugă o cifră 0 la finalul numărului n
$n = n * 10 + 5$	extrage ultima cifră din n

ÎNȚELEGERE

2) Ce valoare va afișa algoritmul pentru $n = 256$? Explicați ce calculează.

```
n = 256
s = 0
while n != 0:
    s = s + n % 10
    n = n // 10
print(s)
```

Variante: a) 256 b) 13 c) 62 d) 652

APLICARE

3) Scrieți un program Python care citește n și afișează:

- suma cifrelor lui n
- numărul de cifre pare ale lui n (*Exemplu:* $n = 12045 \rightarrow 2$ cifre pare: 2 și 4)

ANALIZĂ	4) Analizați algoritmul următor și identificați ce calculează, considerând pozițiile cifrelor numerotate de la 1 (dreapta -> stânga): <pre> p = 1 ; d = 1 while n > 0: c = n % 10 if d % 2 == 0: p = p * c d = d + 1 n = n // 10 print(p) </pre> a) Produsul cifrelor pare b) Produsul cifrelor de pe pozițiile pare (dreapta -> stânga) c) Produsul cifrelor de pe pozițiile pare (stânga -> dreapta) d) Produsul cifrelor de pe pozițiile impare
EVALUARE	5) Un elev susține că algoritmul de mai jos verifică corect dacă n este palindrom. Are sau nu dreptate? Argumentați și identificați cazul în care eșuează. <pre> n = int(input()) ogl = 0 while n > 0: c = n % 10 ogl = ogl * 10 + c n = n // 10 if ogl == n: print("DA") else: print("NU") </pre> Există un caz particular în care algoritmul funcționează totuși corect. Care este acela?
CREARE	6) CNP — extragerea lunii de naștere: Un CNP are forma: S AA LL ZZ JJ NNN C (13 cifre). Cifrele 3-4 reprezintă luna nașterii. Scrieți un program care citește un CNP ca număr întreg și afișează luna de naștere a persoanei. <i>Exemplu:</i> CNP = 1850315123456 -> luna = 03 (martie).

Barem: 2p (din oficiu) + 1p (1) + 1p (2) + 1p (3) + 0.5p (4) + 2p (5) + 2.5p (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să extrag ultima cifră a unui număr și să o șterg folosind operatorii % și //.			
2.	să parcurg toate cifrele unui număr și să calculez suma, produsul sau numărul lor.			
3.	să adăug o cifră la dreapta unui număr și la stânga.			
4.	să construiesc oglinditul unui număr.			
5.	să filtrez cifrele după o condiție (pare, impare, nenule) și să le prelucrez selectiv.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
 (cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 7 – Algoritmi de divizibilitate: Divizorii unui număr; Primalitate

7.1. Divizorii unui număr

Ce este un divizor?

Un număr natural d este **divizor** al lui n dacă $n \% d == 0$, adică d împarte exact pe n , fără rest.

 Mulțimea divizorilor lui n se notează $D(n)$.

EX.

$D(12) = \{1, 2, 3, 4, 6, 12\}$ | $D(7) = \{1, 7\}$ | $D(36) = \{1, 2, 3, 4, 6, 9, 12, 18, 36\}$



Algoritmul de bază — parcurgere $1 \dots n$

Fie n un număr natural. Dorim să afișăm în ordine toți divizorii acestuia.

Idee de rezolvare: testăm fiecare valoare d de la 1 la n și o afișăm dacă divide n .

```
# varianta lentă
n = int(input())
d = 1
while d <= n:
    if n % d == 0:
        print(d)
    d = d + 1
```



Algoritmul optimizat — parcurgere $1 \dots \sqrt{n}$

Fie n un număr natural. Să se determine suma divizorilor numărului n .



Dacă $d \mid n$, atunci și $(n // d) \mid n$. Divizorii vin în perechi $(d, n//d)$.

Ajunge să parcurgem d de la 1 la $\sqrt{n}$, reducând numărul de operații de la n la $\sqrt{n}$. Avem un caz particular însă: numerele pătrate perfecte pentru care radicalul lor îl adunăm o singură dată.

```
# varianta optimizată
n = int(input())
s = 0
d = 1
while d * d < n:
    if n % d == 0:
        s = s + d + n // d
    d = d + 1
if d * d == n:
    s = s + d
print(s)
```

Comparație algoritmică: Să analizăm cele două metode propuse pentru determinarea sumei divizorilor:

Criteriu	Varianta 1: $1 \dots n$	Varianta 2: $1 \dots \sqrt{n}$	Varianta preferabilă
Nr. operații	n operații	$\approx \sqrt{n}$ operații	V2
Număr pași pentru $n = 1\,000\,000$	$1\,000\,000$ pași	$1\,000$ pași	V2
Implementare	Simplă	Puțin mai complexă	V1
Cazul $d^2 = n$	Nu e problemă	Necesită verificare	V1

Aplicații practice – Contexte reale de utilizare a divizorilor:

- Tăierea unei benzi de L cm în bucăți întregi, fără pierderi: lungimile posibile ale bucăților = divizorii lui L ;
- Distribuirea a N caiete: numărul de elevi posibili = orice divizor d al lui N (fiecare primește N/d caiete).
- Ambalarea a N produse în număr minim de pachete, dar numărul căutat să fie minimum 2: căutăm divizorul $d > 1$ al lui N care maximizează N/d (nr. produse/pachet), deci cel mai mic $d > 1$.

7.2. Numărul perfect



Un **număr perfect** este acel număr care are suma divizorilor proprii (fără el însuși) egală cu numărul.

EX.

$6 = 1 + 2 + 3$

```

n = int(input())
s = 1 # 1 este întotdeauna divizor propriu (exclue n însuși)
d = 2
while d * d < n:
    if n % d == 0:
        s += d + n // d
    d += 1
if d * d == n:
    s += d
if n > 1 and n == s:
    print("Numarul este perfect")
else:
    print("Numarul nu este perfect")

```

7.3. Primalitate

7.3.1. Ce este un număr prim



Număr prim: $n > 1$ care nu are alți divizori decât 1 și n . Exemple: 2, 3, 5, 7, 11, 13, 97, 101.

Număr compus: $n > 1$ care are cel puțin un divizor d cu $1 < d < n$. Exemple: $4=2^2$, $6=2 \times 3$, $15=3 \times 5$.

Cazuri speciale: 0 și 1 nu sunt nici numere prime, nici numere compuse – acestea nu intră în clasificare.



7.3.2. Determinarea primalității

Un număr este prim dacă și numai dacă **nu are niciun divizor d cu $2 \leq d \leq \sqrt{n}$** . Nu are sens să căutăm divizori mai mari decât $\sqrt{n}$, dacă un astfel de d ar exista, atunci n/d ar fi un al doilea divizor $\leq \sqrt{n}$. Pentru analiză, prezentăm și varianta lentă. Ambele variante pornesc cu $d=2$ și avansează câte un pas. **Condiția de oprire** este cea care diferențiază eficiența:

 Python – V1 parcurgere $1 \dots n$ (lentă)	 Python – V2 parcurgere $1 \dots \sqrt{n}$ (optimizată)
<pre> n = int(input("n = ")) d = 2 # Opre: am găsit un divizor SAU d a # ajuns la n while d < n and n % d != 0: d += 1 if d == n and n > 1: # niciun divizor găsit print("PRIM") else: print("Nu este prim") # n prim => d merge până la n-1 </pre>	<pre> n = int(input("n = ")) d = 2 # Opre: d*d > n (am depășit sqrt) SAU am # găsit divizor while d * d <= n and n % d != 0: d += 1 if d * d > n and n > 1: # niciun divizor găsit print("PRIM") else: print("Nu este prim") # n prim => d merge până la sqrt(n) </pre>

7.3.3. Comparație algoritmică

n	Este prim?	Pași V1 ($1 \dots n$)	Pași V2 ($1 \dots \sqrt{n}$)	Raport V1/V2
7	DA	5	2	2.5×
25	NU	3	3	1× (egal)
97	DA	95	8	~12×
1 009	DA	1 007	31	~32×
1 000 003	DA	~ 10^6	~1000	~1000×

7.3.4. Aplicații practice – Contexte reale de utilizare a numerelor prime

- Securitatea pe Internet: Numerele prime sunt folosite în metode de protejare a datelor. Este ușor să înmulțești două numere prime, dar foarte greu e procesul invers.
- Hashing: se folosește ca să transformi rapid date (cuvinte, parole, fișiere) într-un „cod” scurt.

- Numerele prime sunt ca niște „cărămizi de bază” ale tuturor numerelor. Vom vedea în lecția care urmează descompunerea numerelor în produs de factori primi.



Reține!

- ✓ Divizorii lui n apar în perechi $(d, n//d)$, deci ajunge să căutăm până la $\sqrt{n}$.
- ✓ Un număr este prim dacă nu are niciun divizor d cu $2 \leq d \leq \sqrt{n}$.
- ✓ Cazurile speciale: 0 și 1 nu sunt nici prime, nici compuse.
- ✓ Condiția $d*d \leq n$ este preferată în cod față de $d \leq \text{sqrt}(n)$ pentru eficiență și acuratețe.



Fișa de lucru 7

Creați dosarul numit *Fisa7*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

- 1) Fie n un număr natural. Să se afișeze divizorii acestuia, în ordine crescătoare.
- 2) Scrieți un program care citește n ($n > 1$) și afișează: numărul de divizori ai lui n , cel mai mare divizor propriu al lui n și suma divizorilor impari proprii ai lui n .
- 3) O bandă de L cm trebuie tăiată în bucăți egale (lungimi naturale, fără pierderi). Afișați toate lungimile posibile, dacă banda se taie în cel puțin 3 bucăți.
- 4) Se citește un număr n . Determinați în câte moduri n caiete pot fi distribuite egal la un grup de cel puțin 2 și cel mult 10 elevi. Afișați fiecare variantă (nr. elevi, nr. caiete/elev).
- 5) O școală a primit o donație de k cărți. Acestea trebuie împărțite, în mod egal, mai multor clase. Știind că fiecare clasă trebuie să primească cel puțin 5 cărți și cel mult 20 de cărți, scrieți un algoritm care să afișeze numărul de clase și numărul de cărți primite de fiecare clasă în toate situațiile posibile.



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Care dintre variante reprezintă definiția corectă a unui divizor? a) d este divizor al lui n dacă $n \% d > 0$ b) d este divizor al lui n dacă $n \% d == 0$ c) d este divizor al lui n dacă $d \% n == 0$ d) d este divizor al lui n dacă $n // d == 0$
ÎNȚELEGERE	2) Ce afișează algoritmul următor pentru $n = 12$? <pre> s = 0 ; d = 1 while d * d < n: if n % d == 0: s = s + d + n // d d += 1 if d * d == n: s = s + d print(s) </pre> Variante: a) 26 b) 28 c) 16 d) 12

APLICARE	3) Scrieți un program care citește n și afișează suma divizorilor impari ai lui n . 4) Scrieți un program care verifică dacă n are divizori proprii, afișând un mesaj corespunzător. <i>Exemplu:</i> 6 divizorii proprii: 2, 3 – DA; 13 -> nu are divizori proprii - NU.
ANALIZĂ	5) Sunt dați doi algoritmi de verificare a primalității unui număr n . Analizați-i și identificați o diferență și o asemănare privind eficiența. <pre style="border: 1px solid #ccc; padding: 5px; width: 45%;"> # Algoritm A d = 2 e_prim = True while d <= n - 1: if n % d == 0: e_prim = False d += 1 </pre> <pre style="border: 1px solid #ccc; padding: 5px; width: 45%;"> # Algoritm B d = 2 e_prim = True while d <= n // 2: if n % d == 0: e_prim = False d += 1 </pre> Care este mai eficient? Estimați numărul de pași pentru $n = 97$ (prim) în fiecare variantă.
EVALUARE	6) Un elev verifică primalitatea cu while $d \leq n // 2$. Este mai bună sau mai slabă decât while $d * d \leq n$? Argumentați cu numărul de pași pentru $n = 97$.
CREARE	7) Se dă un număr natural n reprezentând un lot de pachete de biscuiți. Determinați toate modalitățile prin care pachetele pot fi împărțite în cutii egale, astfel încât în fiecare cutie să fie cel puțin 2 pachete și cel mult 10 pachete. Afișați numărul de cutii și numărul de pachete per cutie pentru fiecare variantă. 8) Un magazin are m fructe de același fel. Acestea se ambalează în mod egal, fără rest, în plase. Se cere să se afișeze toate variantele de ambalare în care: - Fiecare plasă conține același număr de fructe. - Numărul de plase trebuie să fie un număr par cuprins între 4 și 12.

Barem: 1p (din oficiu) + 0.5p (1) + 0.5p (2) + 1p (3) + 1p (4) + 1p (5) + 1p (6) + 2p (7) + 2p (8)

	<i>La finalul lecției, știu...</i>	<i>Da</i>	<i>Nu</i>	<i>Nu sunt sigur(ă)</i>
1.	să explic ce este un divizor și să dau exemple.			
2.	să implementez algoritmul de parcurgere $1 \dots n$ în Python.			
3.	să implementez algoritmul optimizat $1 \dots \sqrt{n}$ pentru suma/afișarea divizorilor.			
4.	de ce algoritmul optimizat este mai eficient și să calculez numărul de pași.			
5.	să implementez și să explic algoritmul numărului perfect.			
6.	să verific primalitatea unui număr.			

7. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
 (cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 8 – Descompunerea în factori primi; CMMDC, CMMMC (Euclid)

8.1. Descompunerea în factori primi

Descompunerea în factori primi

Teorema fundamentală a aritmeticii garantează că orice număr natural $n > 1$ se scrie **unic** ca produs de numere prime: $n = p_1^{a_1} \cdot p_2^{a_2} \cdot \dots \cdot p_n^{a_n}$. De exemplu: $360 = 2^3 \cdot 3^2 \cdot 5^1$.

Spre deosebire de algoritmul de determinare a divizorilor, unde n rămâne fix, aici n se transformă la fiecare pas: când găsim un factor prim, îl scoatem complet prin împărțiri repetate și continuăm căutarea în numărul rămas. Pentru **360**: extragem **2** de **trei** ori ($360 \rightarrow 180 \rightarrow 90 \rightarrow 45$), apoi căutăm factori în **45**, nu în **360**.

Varianta naivă – lentă: Pornim cu $d = 2$ și creștem d până când n ajunge la **1**. La fiecare d , dacă divide n , îl extragem complet și afișăm factorul cu puterea lui. Funcționează, dar pentru un număr prim foarte mare d va trebui să ajungă până la n însuși.

Varianta optimizată: După ce am scos toți factorii, poate rămâne în n un factor prim negăsit - dar **cel mult unul singur**. Iată de ce: dacă ar rămâne doi factori p și q , cel mai mic dintre ei, d , ar satisface $d \times d \leq p \times q \leq n$, deci $d \leq \sqrt{n}$ - și bucla l-ar fi găsit deja, **contradicție**. Prin urmare ne oprim când $d \times d > n$: toți factorii care apar de două ori sau mai mult au fost deja extrași, reducând verificările de la n la doar $\sim \sqrt{n}$.

i Python – V1 parcurgere 1...n (lentă)

```
n = int(input())
d = 2

while n > 1:
    if n % d == 0:
        p = 0
        while n % d == 0:
            n //= d
            p += 1
        print(f"{d}^{p}", end=" ")
    d += 1
# Cel mai rău caz: n prim => d merge până
# la n-1
```

i Python – V2 parcurgere 1...√n (optimizată)

```
n = int(input("n = "))
d = 2
while d * d <= n:      # parcurgem pana la √n
    p = 0              # puterea factorului d
    while n % d == 0:  # cat timp d divide n
        n //= d        # impartim n la d
        p += 1         # crestem puterea
    if p > 0:          # d a fost factor
        print(f"{d}^{p}", end="\n")
    d += 1
if n > 1:              # a ramas un factor prim
    print(f"{n}^{1}", end="")
# Cel mai rău caz: n prim => d merge până
# la √n
```

n	Factorizare	Pași naiv	Pași optim
84	$2^2 \times 3 \times 7$	82	8
360	$2^3 \times 3^2 \times 5$	358	18
97	97^1 (prim)	96	9
1000003	1000003^1 (prim)	1000002	~1000



Aplicații practice ale descompunerii

Descompunerea în factori primi stă la baza multor algoritmi fundamentali:

Numărul de divizori și suma lor

Pornind de la descompunerea în factori primi, putem calcula numărul de divizori, respectiv suma acestora.

Simplificarea fracțiilor

$\text{cmmdc}(a,b)$ prin factorizare $\rightarrow a/\text{cmmdc}$ și b/cmmdc sunt termenii reduși la forma cea mai simplă.

8.2. CMMDC – Algoritmul lui Euclid

Algoritmul lui Euclid este cea mai eficientă metodă clasică de calcul al celui mai mare divizor comun (CMMDC) al două numere nenule, fără a fi nevoie să le determinăm factorii primi. Ideea lui se bazează pe faptul că divizorii comuni se păstrează atunci când reducem problema la una mai simplă.



CMMDC: Pentru două numere întregi pozitive a și b , cel mai mare divizor comun al lor, notat cu $\text{cmmdc}(a,b)$, este cel mai mare $d > 0$ astfel încât d divide atât a cât și b .



Exemple:

$\text{cmmdc}(12,8)=4$ (divizorii comuni sunt 1, 2, 4)
 $\text{cmmdc}(17,5)=1$ (numere prime între ele)
 $\text{cmmdc}(48,18)=6$

Proprietăți utile:

$\text{cmmdc}(a,0) = a$ pentru $a \geq 0$.

$\text{cmmdc}(a,b) = \text{cmmdc}(b,a)$ (comutativitate).

Dacă $d = \text{cmmdc}(a,b)$ atunci există numere întregi x, y cu $ax+by=d$ (identitatea lui Bézout).

Rățiunea inițială – Algoritmul cu scăderi

Regula fundamentală: $\text{cmmdc}(a, b) = \text{cmmdc}(a - b, b)$. Procedura continuă prin scăderi repetate până când cele două numere devin egale. Atunci acel număr este **CMMDC**.



Exemplu: $\text{cmmdc}(48, 18)$:

$48 - 18 = 30 \rightarrow \text{cmmdc}(30, 18)$

$30 - 18 = 12 \rightarrow \text{cmmdc}(12, 18)$

$18 - 12 = 6 \rightarrow \text{cmmdc}(6, 12)$

$12 - 6 = 6 \rightarrow \text{cmmdc}(6, 6) = 6$

Optimizarea – Algoritmul cu împărțiri

Regula fundamentală: $\text{cmmdc}(a, b) = \text{cmmdc}(b, a \bmod b)$.



Procedura continuă prin împărțiri repetate până când restul devine 0, moment în care ultimul număr nenul este **CMMDC**.



Exemplu: $\text{cmmdc}(48, 18)$:

$48 \bmod 18 = 12 \rightarrow \text{cmmdc}(18, 12)$

$18 \bmod 12 = 6 \rightarrow \text{cmmdc}(12, 6)$

$12 \bmod 6 = 0 \rightarrow \text{cmmdc}(6, 0) = 6$

Comparând cele două metode:

- Cu scăderi: 4 pași
- Cu împărțiri: 3 pași

Diferența devine uriașă pentru numere mari.



Exemplu: $\text{cmmdc}(1.000.000, 1)$

- Cu scăderi: 999.999 operații
- Cu împărțiri: 1 operație ($1.000.000 \bmod 1 = 0$)

Explicația intuitivă

Imaginează două șnururi: unul lung de **48** cm și unul de **18** cm. Vrei să le tai în bucăți egale cât mai mari.

– Tai bucăți de **18** cm din cel de **48** $\rightarrow$ îți rămâne **12** cm. Acum ai șnururile de **18** cm și **12** cm.

– Tai bucăți de **12** cm din cel de **18** $\rightarrow$ îți rămâne **6** cm. Acum ai **12** cm și **6** cm.

– Tai bucăți de **6** cm $\rightarrow$ rest **0**.

Ultima bucată care „intră perfect” în ambele este **6** cm, adică **CMMDC**.

Aplicabilitate – alte probleme din viața reală:

– *Distribuirea materialelor*: Laboratorul are două role de sârmă, una de **72** cm și una de **120** cm. Se vor tăia bucăți identice, cât mai lungi, fără rest. Care poate fi lungimea maximă?

– *Împărțirea exercițiilor pe capitole*: Un manual are un capitol cu **42** de exerciții și altul cu **63**. Se vor crea seturi de fișe astfel încât fiecare fișă să conțină același număr de exerciții din fiecare capitol. Care este numărul maxim de exerciții din fișă?

– *Acoperirea unei suprafețe dreptunghiulare cu pătrate de latură maximă* (aplicabilă la pavarea suprafețelor).

CMMDC – Cu scăderi repetate	CMMDC – optim
<pre>a, b = map(int, input().split()) # a,b pozitive while a != b: if a > b: a = a - b else: b = b - a print(a)</pre>	<pre>a, b = map(int, input().split()) # a,b pozitive while b > 0: r = a % b a = b b = r print(a)</pre>



`a, b = map(int, input().split())` citește două numere de pe aceeași linie, separate prin spațiu: `input().split()` împarte textul după spații, iar `map(int, ...)` transformă fiecare parte în număr întreg.

CMMC – Cel mai mic multiplu comun



Cel mai mic multiplu comun al două numere este cel mai mic număr strict pozitiv care este multiplu al ambelor. Cu alte cuvinte, este primul număr la care „se întâlnesc” cele două serii de multipli, multiplii lui **a** și multiplii lui **b**. Se calculează eficient folosind relația dintre **CMMC** și **CMMDC**:

$$\text{CMMC}(a, b) = a * b / \text{CMMDC}(a, b)$$



Explicație intuitivă

Îți poți imagina două roți dințate: una face câte un „clic” la fiecare **4** secunde, cealaltă la fiecare **6** secunde. Momentul în care fac clic simultan pentru prima dată este **CMMC(4, 6) = 12 secunde**, deoarece **12** este primul multiplu comun al lui **4** și **6** (multiplii lui **4**: **4, 8, 12...**; multiplii lui **6**: **6, 12, 18...**). Astfel, CMMC ne spune la ce interval se aliniază două ritmuri, cicluri sau multiplicări.

Se poate observa că implementarea algoritmului lui Euclid modifică numerele introduse inițial, astfel că, pentru determinarea celui mai mic multiplu comun este necesară păstrarea unor copii ale valorilor inițiale, pentru a putea fi refolosite la final.	 CMMC <pre>a, b = map(int, input().split()) ca = a cb = b while b > 0: r = a % b a = b b = r print(ca * cb // a)</pre>
--	---

Aplicații practice

- Simplificarea fracțiilor: $a/b \rightarrow (a//\text{cmmdc}) / (b//\text{cmmdc}) \rightarrow 18/24 \rightarrow \text{cmmdc}=6 \rightarrow 3/4$
- Semafoare sincronizate: $\text{CMMC}(45, 60) = 180 \rightarrow$ ambele verzi după 180 secunde
- $\text{CMMC}(a,b,c) = \text{CMMC}(\text{CMMC}(a,b), c)$. Cu alte cuvinte, calculăm mai întâi CMMC pentru primele două numere, apoi aplicăm din nou formula pentru rezultatul obținut și al treilea număr. Procedura se poate extinde pentru oricâte numere, aplicând formula pas cu pas.



Exemplu: $\text{CMMC}(4, 6, 10)$

$$\text{CMMC}(4, 6) = 4 \times 6 / \text{CMMDC}(4, 6) = 24 / 2 = 12$$

$$\text{CMMC}(12, 10) = 12 \times 10 / \text{CMMDC}(12, 10) = 120 / 2 = 60$$

$$\text{Deci } \text{CMMC}(4, 6, 10) = 60$$



Fișa de lucru 8

Creați dosarul numit *Fisa8*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

- 1) Afișați descompunerea în factori primi a două numere **a** și **b** (de exemplu: $60 = 2^2 \cdot 3 \cdot 5$; $45 = 3^2 \cdot 5$).
- 2) Determinați dacă cele două numere **a** și **b** sunt prime între ele (de exemplu: 5, 14, $\text{cmmdc}(5,14)=1$).
- 3) Ana și Robert au **120** de caiete și **96** de pixuri. Ei vor să formeze pachete identice (care conțin același număr de caiete și același număr de pixuri), folosind **toate** obiectele. Care este cel mai mare număr de pachete care pot fi formate? Câte caiete și câte pixuri va conține fiecare pachet?
- 4) Vrem să construim un pătrat folosind dreptunghiuri egale, cu dimensiunile de **15** cm și **20** cm. Care este latura celui mai mic pătrat care poate fi format prin alăturarea acestor dreptunghiuri? Câte dreptunghiuri sunt necesare pentru a forma acest pătrat?
- 5) Doi atleți aleargă pe un stadion circular. Primul atlet termină o tură completă în **12** minute, iar al doilea atlet termină o tură în **15** minute. Dacă ambii atleți au plecat de la linia de start simultan la ora **09⁰⁰**, după câte minute se vor întâlni din nou la linia de start pentru prima dată și la ce oră se va întâmpla acest lucru?



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE

- 1) Care dintre următoarele reprezintă descompunerea corectă în factori primi a lui 36?
 - a) $36 = 2 \cdot 18$
 - b) $36 = 2^2 \cdot 3^2$
 - c) $36 = 4 \cdot 9$
 - d) $36 = 2^3 \cdot 3$
- 2) Care dintre următoarele afirmații despre CMMDC și CMMMC este corectă?
 - a) $\text{CMMDC}(a, b)$ este întotdeauna mai mare decât a și b
 - b) $\text{CMMDC}(a, b) \times \text{CMMMC}(a, b) = a \times b$
 - c) $\text{CMMMC}(a, b)$ este întotdeauna mai mic decât $a + b$
 - d) $\text{CMMDC}(a, b) = a$ dacă a divide pe b , altfel $\text{CMMDC}(a, b) = b$

ÎNȚELEGERE

- 3) Urmăriți execuția algoritmului de factorizare pentru **n = 84** și completați tabelul:

n	d	p	dmax	Algoritm
84				<pre> n = int(input()) d = 2; dmax = 0 while d * d <= n: p = 0 while n % d == 0: n = n // d; p += 1 if p > 0: dmax = d d += 1 if n > 1: dmax = n print(dmax) </pre>

De ce linia **if n > 1: dmax = n** se află în AFARA buclei while $d * d \leq n$?

	4) Completați tabelul pentru CMMDC cu scăderi ($a=48, b=36$): <table><tr><th>n</th><th>a</th><th>b</th><th>Algoritm</th></tr><tr><td>84</td><td></td><td></td><td>$a, b = \text{map}(\text{int}, \text{input}().\text{split}())$</td></tr><tr><td></td><td></td><td></td><td>while $a \neq b$:</td></tr><tr><td></td><td></td><td></td><td> if $a > b$: $a = a - b$</td></tr><tr><td></td><td></td><td></td><td> else:</td></tr><tr><td></td><td></td><td></td><td> $b = b - a$</td></tr><tr><td></td><td></td><td></td><td>print(a)</td></tr></table> De ce bucla se oprește când $a == b$ și nu când $a == 0$?	n	a	b	Algoritm	84			$a, b = \text{map}(\text{int}, \text{input}().\text{split}())$				while $a \neq b$:				if $a > b$: $a = a - b$				else :				$b = b - a$				print (a)
n	a	b	Algoritm																										
84			$a, b = \text{map}(\text{int}, \text{input}().\text{split}())$																										
			while $a \neq b$:																										
			if $a > b$: $a = a - b$																										
			else :																										
			$b = b - a$																										
			print (a)																										
APLICARE	5) Scrieți un program care afișează cel mai mic factor prim al lui n. <i>Exemplu:</i> $n=20 \rightarrow 2$; $n=17 \rightarrow 17$. <i>Hint:</i> opriți-vă imediat ce găsiți primul factor. 6) Citiți a și b și afișați fracția a/b adusă la forma ireductibilă, folosind CMMDC. <i>Exemplu:</i> $\frac{18}{24} \Rightarrow \frac{3}{4}$; $\frac{36}{48} \Rightarrow \frac{3}{4}$																												
ANALIZĂ	7) Factorizare – variante A (până la $\sqrt{n}$) vs. B (până la n): <pre># Varianta A d = 2 ; contor = 0 while d*d <= n: if n%d==0: contor += 1 while n%d==0: n//=d d += 1 if n>1: contor += 1</pre> <pre># Varianta B d = 2 ; contor = 0 while d <= n: if n%d==0: contor += 1 while n%d==0: n//=d d += 1</pre> Urmăriți manual ambele variante pentru $n = 12$ și $n = 13$. Produc același rezultat? Estimați numărul de pași pentru $n = 1.000.003$ (prim). Care e mai eficientă?																												
EVALUARE	8) Andrei merge la antrenament la fiecare 4 zile, Bogdan la fiecare 6 zile. Un coleg susține că se vor întâlni după CMMDC(4,6)=2 zile. Identificați ce calcul trebuie făcut și verificați manual rezultatul corect.																												
CREARE	9) Semafoare sincronizate: – Semaforul A se schimbă la fiecare 45 s, B la fiecare 60 s. Ambele sunt verzi la $t=0$. Scrieți un program care determină după câte secunde vor fi din nou verzi simultan. – Generalizați problema pentru n semafoare. <i>Hint:</i> $CMMMC(a,b,c) = CMMMC(CMMMC(a,b), c)$.																												

Barem: 1p (din oficiu) + 0.5p (1) + 0.5p (2) + 0.5p (3) + 0.5p (4) + 1p (5) + 2p (6) + 1p (7) + 1p (8) + 2p (9)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	aplica algoritmul de descompunere în factori primi.			
2.	să explic principiul algoritmului lui Euclid (scăderi $\rightarrow$ împărțiri).			
3.	să calculez CMMDC și CMMMC cu formula.			
4.	să identific erori logice și să le argumentez.			
5.	să proiectez soluții pentru probleme practice: fracții, semafoare.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
 (cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Evaluare sumativă - Capitolul 2

Rezolvați exercițiile într-un document: *evaluare_sumativa2.docx*.

1) (1p) **Alegeți varianta corectă:** Algoritmul lui Euclid cu împărțiri calculează:

a) CMMMC(a, b); b) $a * b$; c) CMMDC(a, b); d) $a + b$.

Care este valoarea lui CMMDC(48, 18)?

2) (1p) Urmăriți algoritmul pentru $n = 132$ și completați tabelul:

Iterația	n	c=n%10	suma	n după //10

```
n = int(input())
suma = 0
while n > 0:
    c = n % 10
    suma += c
    n = n // 10
print(suma)
```

Ce calculează algoritmul?

3) (1.5p) **Completați programul** care verifică dacă n este număr perfect:

```
n = int(input())
s = 1
d = 2
while d * d < n:
    if n % d == 0:
        s += ____
        d += 1
    if d * d == n:
        s += ____
print(n > 1 and n == s)
```

Completați spațiile:

Testați pentru $n = 6$:

De ce pornim $s = 1$?

4) (1.5p) **Completați programul CMMDC cu împărțiri și calculați:**

```
a, b = map(int, input().split())
while b != 0:
    a, b = ____, ____
print(a)
```

CMMDC(48, 18):

CMMDC(100, 75):

CMMMC(12, 18) = ...

5) (1.5p) Un elev susține că pentru a verifica dacă un număr n este prim, este suficient să testăm toți divizorii până la $n // 2$. Un alt elev propune să testăm doar până la $\sqrt{n}$.

a) Explicați de ce este suficient să verificăm divizorii doar până la $\sqrt{n}$ și nu până la $n // 2$

b) Estimați numărul de pași pentru $n = 97$ (număr prim) în fiecare variantă:

- până la $n // 2$:
- până la $\sqrt{n}$:

6) (2.5p) O rețetă folosește cantitățile a și b pentru două ingrediente. Scrieți un program Python care simplifică raportul $a:b$ la forma minimă, folosind CMMDC.

Barem de corectare: Se acordă **1 punct** din oficiu!

1) 1 punct. 2) 1 punct (0.25p/celulă + 0.25p/concluzie). 3) 1.5 puncte. 4) 1.5 puncte. 5) 1.5 puncte (1p/eroare).

6) 2.5 puncte (1.25p cmmdc + 1.25p simplificare).



Investighează!

(100 puncte)

Realizați, într-un fișier **investigheaza2.docx**, o investigație despre eficiența algoritmilor de calcul al CMMDC și al divizorilor unui număr, parcurgând următorii pași:

- Implementați în Python cei doi algoritmi pentru CMMDC (cu scăderi și cu împărțiri). Rulați algoritmi pentru 5 perechi de numere și notați rezultatele. **(20p)**
- Numărați manual pașii (iterațiile) pentru fiecare algoritm pe perechile (48,18) și (1000,1). Care este mai eficient și de ce? **(25p)**
- Implementați doi algoritmi pentru suma divizorilor lui n : varianta până la n și varianta până la rădăcina din n . Testați pentru $n=36$ și $n=100$. **(25p)**
- Construiți un tabel comparativ cu numărul de pași pentru ambele variante ($n=10, 100, 1000$). Concluzie. **(20p)**
- Notați o problemă din viața reală în care CMMDC sau CMMMC ar fi utile. (3-4 rânduri) **(10p)**

20p	20p	20p	20p	20p
CMMDC 2 variante	Numărare pași	Divizori 2 variante	Tabel comparativ	Aplicație reală
Recomandare: Se lucrează individual. Fișierul se salvează în dosarul <i>Capitol2</i> . Se pot include capturi de ecran.				



Explorează

(100 puncte)

Realizați, în echipă, o aplicație Python **calculator_fractii.py** care efectuează operații cu fracții. Salvați în dosarul **Capitol2**.

- Funcție *simplifica(numarator, numitor)*: reduce fracția la forma ireductibilă folosind CMMDC. **(20p)**
- Funcții *aduna(a1,b1,a2,b2)*, *scade(a1,b1,a2,b2)*: calculează suma/diferența și returnează fracția simplificată. **(25p)**
- Funcții *inmulteste(a1,b1,a2,b2)*, *imparte(a1,b1,a2,b2)*: calculează produsul/câtul și returnează fracția simplificată. **(25p)**
- *Meniu interactiv*: utilizatorul introduce două fracții și alege operația (1-adunare, 2-scădere, 3-înmulțire, 4-împărțire, 0-ieșire). **(20p)**
- *Tratarea erorilor*: împărțire la zero, numitor zero la intrare. **(10p)**

20p simplifica	25p aduna/scade	25p inmulteste/imparte	20p Meniu	10p Erori
Recomandare: Echipa de 3-4 elevi. Fiecare elev implementează cel puțin o funcție. Se prezintă în fața clasei.				



Portofoliu

(100 puncte)

La finalul acestui capitol, portofoliul personal al elevului trebuie să cuprindă, în dosarul **Capitol2**, următoarele materiale:

- Dosarele **Fisa6**, **Fisa7** și **Fisa8**, unde fiecare dosar conține fișierele: **raspunsuri.docx**, **FisaAutoevaluare6.docx**, **FisaAutoevaluare7.docx** și **FisaAutoevaluare8.docx**.
- Dosarul **Proiecte2** va conține: **investigheaza2.docx**, **calculator_fractii.py**.
- Fișierul **evaluare_sumativa2.docx**.

Modalitate de evaluare:

- Există dosarul **Capitol2** cu structura de directoare specificată – **20p**
- Există în dosare:
 - peste 95% din fișiere – **80p**
 - între 75% și 94% din fișiere – **60p**
 - între 50% și 74% din fișiere – **40p**
 - între 25% și 49% din fișiere – **20p**
 - sub 25% – **0p**

Capitolul 3 – Date organizate în liste

Lecția 9 – Modelul conceptual liniar: caracteristici, acces direct vs secvențial; liste de frecvență

1. Date simple vs. date compuse (liste)



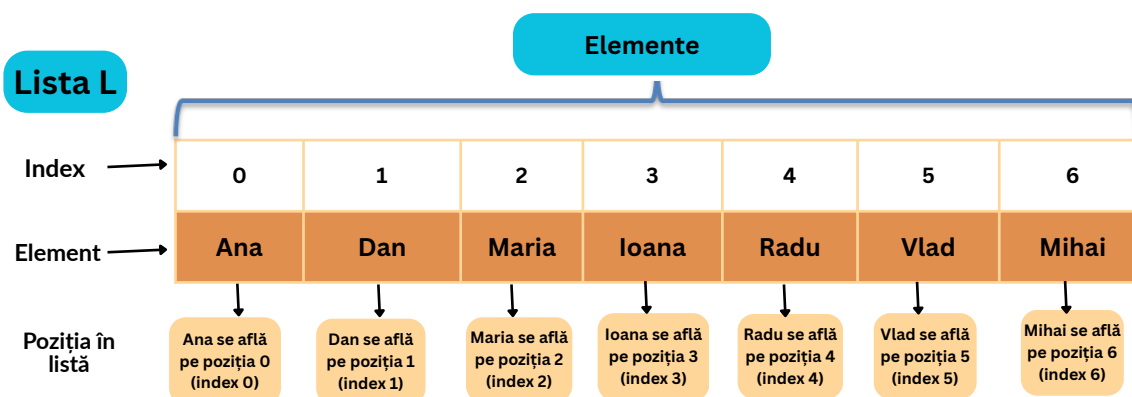
În informatică, datele pot fi organizate la nivel elementar sub forma unor *date simple* sau sub forma unor *date compuse*, care formează colecții liniare (liste/tablouri).

Date simple – exemple	Date compuse – exemple
un număr: 23	o listă de note: [8, 9, 10]
un caracter: 'A'	un nume complet, format din nume și prenume: "Popa Ana"
o valoare logică: True	o dată calendaristică sub forma zi, lună, an: 12 ianuarie 2026

Modelul conceptual liniar este adesea reprezentat prin structuri de date de tip *listă* sau *tablouri unidimensionale* (*șiruri de valori*). Este o formă de organizare secvențială a informațiilor, în care elementele sunt dispuse unul după celălalt.



Lista reprezintă o structură compusă de date care stochează, într-o anumită ordine, un număr finit de elemente. Ordinea elementelor este dată de poziția fiecărui element în listă, această poziție putând fi identificată printr-un *index*.



În limbajul de programare C++, listele/tablourile cuprind elemente de același tip. În limbajul de programare Python, listele pot cuprinde elemente de tipuri diferite, ca în exemplul de mai sus.

Limbajul Python				Limbajul C++					
0	1	2	3	0	1	2	3	4	5
lista=[1, 3.50, "a", "este luni"]				v=[12, 2, 3, 6, 9, 1]					

0 listă are următoarele caracteristici conceptuale:

- **structură de date compusă**: grupează mai multe valori;
- **organizare secvențială (liniaritate)**: fiecare element are un predecesor (cu excepția primului) și un succesor (cu excepția ultimului);
- **ordine bine definită**: indexată de la 0;
- **acces controlat**: lista permite atât accesul direct, cât și pe cel secvențial;
- **dimensiune variabilă**: într-o listă se pot adăuga elemente sau se pot șterge elemente.

Exemplu:

0 1 2 3 4 5

Fie lista $L=[4, 8, 15, 2, 32, 23]$. În această listă:

- 4 este primul element
- 23 este ultimul element
- 8 are predecesorul 4 și succesorul 15

Elementul 15 are indexul 2, deci $L[2]=15$.**Exemple de liste din viața reală:**

- ✓ lista cărților de la bibliotecă
- ✓ lista invitațiilor la un eveniment
- ✓ lista notelor unui elev la informatică
- ✓ lista comenzilor dintr-un magazin online
- ✓ lista elevilor dintr-o clasă
- ✓ lista melodiilor unui artist

Operațiile de bază care se pot realiza cu o listă sunt: crearea, parcurgerea, inserarea unui element, ștergerea unui element, căutarea unei valori în listă și accesul la elemente.

2. Acces direct vs. acces secvențial

Accesul la elementele unei liste reprezintă modul în care sunt localizate acestea. După tipul de acces, listele sunt: cu **acces direct** și cu **acces secvențial**.

Accesul direct oferă posibilitatea de a localiza imediat un element pe baza indexului său sau a unei adrese, fără a fi nevoie de parcurgerea elementelor precedente.

Accesul secvențial presupune parcurgerea elementelor în ordine, pornind de la începutul listei și avansând succesiv. În acest fel, elementele sunt accesate unul după celălalt.

Tip acces	Definiție	Exemplu	Avantaje	Dezavantaje
Acces direct	Localizare imediată prin index, fără parcurgere	$L[2] = 21$ din $L=[7,14,21,28]$	Acces rapid la un element	Inserări/ștergeri pot fi ineficiente
Acces secvențial	Parcurgere în ordine, element cu element	Parcurs lista de la $L[0]$ până la final	Inserări/ștergeri eficiente	Acces lent la elemente oarecare din listă

3. Repere de prelucrare a listelor

Fără memorare (parcurgere simplă):

Se prelucrează elementele pe rând, fără a le păstra. Util când ne interesează doar rezultatul final (suma, contor etc.)

```
n = int(input())
s = 0
for i in range(n):
    x = int(input())
    s += x
print("Suma:", s)
```

Cu memorare (se construiește lista):

Elementele se păstrează pentru prelucrări ulterioare (*Exemplu:* calculul mediei notelor la un test).

```
n = int(input())
note = []
for i in range(n):
    x = int(input())
    note.append(x)
media = sum(note) / len(note)
for nota in note:
    if nota > media:
        print(nota)
```

4. Liste de frecvență



O **listă de frecvență** este o structură de date care asociază fiecărei valori distincte dintr-un set de date numărul de apariții (frecvența) acesteia într-un șir. Ea transformă date brute, neorganizate, într-o reprezentare compactă și ușor de interpretat.



Profesorul de informatică dorește să știe cum s-au descurcat cei 30 de elevi dintr-o clasă la primul test. Notele primite de elevi sunt: 7, 5, 4, 8, 9, 9, 10, 7, 8, 7, 4, 5, 7, 9, 10, 4, 6, 9, 8, 9, 3, 6, 7, 8, 9, 9, 7, 8, 10, 9. Pentru a afla anumite valori, va contoriza numărul de apariții ale fiecărei posibile note în lista de frecvență, astfel:

Notă posibilă	1	2	3	4	5	6	7	8	9	10
Frecvența	0	0	1	3	2	2	6	5	8	3

Pe baza datelor de mai sus, profesorul poate trage diverse concluzii, cum ar fi: nota care a fost obținută de cele mai multe ori, câți elevi din clasă necesită măsuri remediale urgente, rata de promovare, numărul elevilor cu notă maximă.

Algoritmul de construire a unei liste de frecvență:

Pași	Pasul 1: Se creează o listă (vector) de dimensiune fixă (de exemplu, de la 0 la 10 pentru note). Pasul 2: Se inițializează lista cu 0 pentru fiecare element. Pasul 3: Se parcurge setul de date de intrare, element cu element. Pasul 4: Pentru fiecare element, se incrementează cu 1 frecvența în listă, la indexul egal cu valoarea sa.
-------------	--

```
# Exemplu: frecvența notelor (1-10) a 30 elevi
note = [7, 5, 4, 8, 9, 9, 10, 7, 8, 7, 4, 5, 7, 9, 10, 4, 6, 9, 8, 9, 3, 6, 7, 8, 9, 9, 7, 8, 10, 9]
frecv = [0] * 11 # indexuri 0..10
for nota in note:
    frecv[nota] += 1
print("Nota 9 apare de", frecv[9], "ori") # 8 ori
print("Nota maxima:", max(note)) # 10
```

Avantaje

- ✓ Organizare clară a datelor de același fel
- ✓ Eficiența din punct de vedere al memoriei
- ✓ Calcul rapid al frecvenței datelor (%)
- ✓ Identificarea ușoară a unor valori (valoarea cu frecvența maximă, cu frecvența minimă, media ponderată)
- ✓ Bază pentru reprezentări grafice (histogramă, diagramă circulară)
- ✓ Viteză de procesare a unui volum mare de date
- ✓ Util în analize statistice și rapoarte.

Dezavantaje

- ✗ Nu păstrează ordinea originală a datelor
- ✗ Datele individuale dispar, deci se poate pierde contextul (nu mai știm ce notă a luat un anumit elev)
- ✗ Ineficiență pentru valori unice



Fișa de lucru 9

Creați dosarul numit *Fisa9*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

1) **Alegeți varianta corectă:** O listă este o structură de date care conține:

- a) Un singur element de tip numeric;
- b) Un număr finit de elemente ordonate;
- c) Doar texte, fără numere;
- d) Elemente fără o ordine definită.

2) **Completați:**

- a) Primul element al unei liste Python are indexul, iar ultimul element poate fi accesat cu indexul
- b) Accesul la un element prin specificarea poziției sale se numește acces, iar parcurgerea elementelor unul câte unul se numește acces
- c) O listă este o listă care nu conține niciun element, creată cu [] sau list().

3) Pentru $L = [4, 8, 15, 16, 23]$, **scrie valorile:** $L[0]$: $L[2]$: $L[-1]$: $L[-2]$: $\text{len}(L)$:

4) **Explicați diferența dintre acces direct și acces secvențial, folosind un exemplu din viața reală (nu din programare).**

Acces direct:

Acces secvențial:

Exemplu din viața reală:

5) **Construiți lista de frecvență** pentru notele $[7, 8, 7, 10, 6, 8, 7]$, știind că notele posibile sunt între 1 și 10.

1	2	3	4	5	6	7	8	9	10

Pe baza listei de frecvență, determinați: Nota cu cea mai mare frecvență: Numărul de elevi cu nota ≥ 5 : Numărul de elevi cu nota maximă (10):

6) **Scrieți o secvență Python care:**

- creează o listă cu 5 valori, numere reale;
- afișează lista;
- afișează suma elementelor ei;
- afișează numărul de elemente.

```
L = [____, ____, ____, ____, ____]
print(____)
print("Suma:", ____ (L) )
print("Numar elemente:", ____ (L) )
```



7) **Care dintre următoarele scenarii necesită memorarea datelor (construirea listei) și care nu?**

Scrieți DA sau NU și justificați în 1-2 propoziții.

Scenariu	DA/NU	Justificare
Verificare pentru a stabili dacă numărul citit este par.		
Afișarea celui mai mare număr dintr-un șir de n numere		
Afișarea notelor în ordine inversă față de cea în care au fost introduse		
Afișarea sumei unor numere citite pe rând		



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE

- Modelul conceptual liniar se caracterizează prin:
 - Relații complexe între elemente;
 - Organizare ierarhică;
 - Organizare secvențială;
 - Număr infinit de elemente.
- Accesul direct la un element se referă la:
 - Acces aleator lent;
 - Utilizarea unui index;
 - Parcurgerea tuturor elementelor;
 - Existența unui succesor.

ÎNȚELEGERE

- Completați: Accesul la elementele unei liste poate fi: și
Dimensiunea variabilă a unei liste se referă la:
- Scrieți 3 argumente prin care justificați că lista este o structură compusă de date care stochează, într-o anumită ordine, un număr finit de elemente, plecând de la caracteristicile conceptuale ale listei.

APLICARE	5) Un profesor dorește să analizeze notele a 30 de elevi: 7, 5, 4, 8, 9, 9, 10, 7, 8, 7, 4, 5, 7, 9, 10, 4, 6, 9, 8, 9, 3, 6, 7, 8, 9, 9, 7, 8, 10, 9 - Construiți lista de frecvență (nota 1-10 și nr. apariții); - Determinați nota obținută de cele mai multe ori.
ANALIZĂ	6) Care dintre scenariile următoare necesită memorarea datelor într-o listă și care nu? Argumentați. - Înregistrarea temperaturilor zilnice maxime, dintr-o lună, pentru a afla ziua cea mai caldă; - Verificarea dacă un număr este impar; - Contorizarea numerelor impare dintr-un șir de n numere; - Găsirea perechilor de numere impare consecutive dintr-un șir.
EVALUARE	7) Un elev susține că următorul program construiește corect lista de frecvență pentru notele clasei și determină nota cea mai frecventă: <pre>note = [7, 8, 9, 7, 10, 7, 9, 8] frecv = [0] * 10 # indexuri 0..9 for nota in note: frecv[nota] += 1 print("Nota maxima ca frecventa:", frecv.index(max(frecv)))</pre> a) Identificați eroarea din cod și explicați de ce produce rezultate greșite sau o eroare, la rulare. b) Testați cu notele de mai sus. Ce se întâmplă când nota = 10? Argumentați. c) Scrieți varianta corectată a programului. <i>Hint: gândește-te la dimensiunea listei frecv și la indexul maxim posibil.</i>
CREARE	8) Un magazin online a primit 10 recenzii pentru un produs, cu valori între 1 și 5 stele: [5, 4, 5, 3, 5, 4, 2, 5, 4, 3]. Construiți lista de frecvență și răspundeți: a) Câte recenzii de 5 stele a primit produsul? b) Care este numărul de stele dat cel mai des? c) Câte recenzii sunt pozitive (valoare ≥ 4)?

Barem: 1p (din oficiu) + 0.5p (1) + 0.5p (2) + 0.5p (3) + 0.5p (4) + 2p (5) + 2p (6) + 1p (7) + 2p (8)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să definesc o listă și să enumăr caracteristicile ei conceptuale.			
2.	diferența dintre accesul direct și cel secvențial.			
3.	să recunosc situațiile în care prelucrarea fără memorare este suficientă și când este necesară memorarea datelor într-o listă.			
4.	să folosesc liste de frecvență.			
5.	să identific o eroare într-un algoritm cu listă de frecvență și să o corectez.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
 (cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 10 – Clasa list din Python: operatori specifici și metode de bază



În Python, **list** este o clasă predefinită folosită pentru a stoca o colecție ordonată și modificabilă de elemente. O **listă ordonată** înseamnă că păstrează ordinea introducerii, iar **modificabilă** înseamnă că i se pot adăuga, schimba sau șterge elemente.



10.1. Crearea și accesarea listelor

Tipuri de liste:	Indexare pozitivă și negativă:
<pre>lista_vida = [] lista_homogena = ["a", "b", "c"] lista_neomog = [1, 3.5, "text", True] lista_din_constructor = list(range(5))</pre>	<pre>L=["Ana", "Maria", "Ion", "Ema", "Victor"] # 0 1 2 3 4 # -5 -4 -3 -2 -1 print(L[2]) # Ion print(L[-1]) # Victor print(L[-3]) # Ion</pre>

Pentru a accesa elemente, se folosește **operatorul de indexare** sau **operatorul de feliere (slicing)**.

Operatorul de indexare este reprezentat prin paranteze pătrate, sub forma `ID_lista[index]`.

Operatorul de feliere (slicing) este reprezentat prin semnul `:`, fiind folosit pentru a accesa un grup de elemente consecutive din listă sub forma `[start:stop:pas]`.

Sintaxa	Semnificație
<code>L[0]</code>	primul element
<code>L[-1]</code>	ultimul element
<code>L[i:j]</code>	elementele cu index în intervalul [i,j)
<code>L[i:]</code>	de la index i până la final
<code>L[:i]</code>	primele i elemente
<code>L[:]</code>	toată lista
<code>L[::2]</code>	elementele de la indici pari
<code>L[::-1]</code>	lista inversată
<code>L[-x:]</code>	ultimele x elemente



10.2. Operatori: in, not in, +, *, comparatori

Apartenență:	Comparatori (lexicografic):
<pre>L = ["Ana", "Maria", "Ion"] print("Ana" in L) # True print("Ema" not in L) # True</pre>	<pre>print([3,2,1] == [3,2,1]) # True print([1,2,3] > [3,2,1]) # False (1<3) print([1,2] < [1,2,3]) # True (mai scurta)</pre>
Concatenare și multiplicare:	Modificare prin index și slicing:
<pre>L1 = [1,2,3] ; L2 = [4,5,6] print(L1 + L2) # [1,2,3,4,5,6] print([0]*3) # [0,0,0] print(L1*2) # [1,2,3,1,2,3]</pre>	<pre>L = [1,2,3,4,5] L[2] = 99 # L=[1,2,99,4,5] L[1:3] = [10,20] # L=[1,10,20,4,5]</pre>



10.3. Metode esențiale ale clasei list

Obiectele de tip listă au **metode de bază, predefinite pentru analiză și manipulare**: adăugare/inserare, ștergere/eliminare elemente din listă, extinderea listei (are ca efect modificarea lungimii listei), numărare.

Metoda	Efect	Exemplu
<code>append(x)</code>	Adaugă x la sfârșit	<code>L.append(4) -> [1, 2, 3, 4]</code>
<code>insert(i, x)</code>	Inserează x în poziția i , deplasând elementele existente spre dreapta	<code>L.insert(1, 9) -> [1, 9, 2, 3, 4]</code>
<code>pop(i)</code>	Șterge și returnează elementul de pe poziția i	<code>L.pop(0) -> șterge primul</code>
<code>remove(x)</code>	Șterge prima apariție a lui x	<code>L.remove(2) -> șterge primul 2</code>
<code>clear()</code>	Șterge toate elementele	<code>L.clear() -> []</code>
<code>index(x)</code>	Prima poziție a lui x	<code>L.index(3) -> 2</code> (daca <code>L=[1, 2, 3]</code>)
<code>count(x)</code>	Nr. de apariții ale lui x	<code>L.count(2) -> 1</code>
<code>sort()</code>	Sortează lista crescător/descrescător	<code>L.sort()</code> sau <code>L.sort(reverse=True)</code>
<code>copy()</code>	Creează o copie a listei	<code>L2 = L.copy()</code>
<code>extend(L)</code>	Adaugă elementele din lista L	<code>L.extend([4, 5])</code>
<code>reverse()</code>	Inversează elementele listei	<code>L.reverse()</code>

Exemple detaliate ale metodelor

Metoda	Cod	Cum funcționează	Lista
<code>append(x)</code> – adaugă un element la sfârșitul listei	<pre>L = [1, 2, 3] L.append(4) print(L) # [1, 2, 3, 4]</pre>	Adaugă elementul x după ultimul element existent, modificându-se astfel lista.	Înainte: <code>L = [1, 2, 3]</code> După: <code>L.append(4) -> L = [1, 2, 3, 4]</code>
<code>insert(i, x)</code> – inserează un element la o poziție specificată	<pre>L = ["Ana", "Ion", "Ema"] L.insert(1, "Maria") print(L) # ["Ana", "Maria", "Ion", "Ema"]</pre>	Inserează elementul x la indexul i . Toate elementele de la indexul i încolo se deplasează cu o poziție spre dreapta.	Înainte: <code>L = ["Ana", "Ion", "Ema"]</code> După: <code>L.insert(1, "Maria") -> L = ["Ana", "Maria", "Ion", "Ema"]</code>
<code>pop(i)</code> – șterge și returnează elementul de la poziția i	<pre>L = [10, 20, 30, 40] val = L.pop(1) print(val) # 20 print(L) # [10, 30, 40]</pre>	Șterge elementul de la indexul i din listă și îl returnează. Dacă i lipsește, șterge și returnează ultimul element (implicit $i=-1$).	Înainte: <code>L = [10, 20, 30, 40]</code> După: <code>val = L.pop(1) -> val = 20, L = [10, 30, 40]</code>
<code>remove(x)</code> – șterge prima apariție a valorii x	<pre>L = [3, 1, 4, 1, 5] L.remove(1) # șterge primul 1 print(L) # [3, 4, 1, 5]</pre>	Caută prima apariție a valorii x în listă și o șterge. Nu returnează nimic. Dacă valoarea nu există, ridică <code>ValueError</code> .	Înainte: <code>L = [3, 1, 4, 1, 5]</code> După: <code>L.remove(1) -> L = [3, 4, 1, 5]</code> (primul 1 șters)
<code>clear()</code> – șterge toate elementele listei	<pre>L = [1, 2, 3, 4, 5] print(len(L)) # 5 L.clear() print(L) # [] print(len(L)) # 0</pre>	Șterge toate elementele din listă, lăsând-o vidă. Obiectul listă rămâne în memorie, doar conținutul devine <code>[]</code> .	Înainte: <code>L = [1, 2, 3, 4, 5]</code> După: <code>L.clear() -> L = []</code>

<code>index(x)</code> – returnează poziția primei aparitii a valorii x	<pre>L = ["mere", "pere", "prune", "pere"] print(L.index("pere")) # 1</pre>	Returnează indexul primei apariții a valorii x . Dacă x nu se află în listă, ridică ValueError . Acceptă parametri opționali de start și stop pentru a limita căutarea.	Înainte: <code>L = ["mere", "pere", "prune", "pere"]</code> După: <code>L.index("pere")</code> -> 1 (prima apariție)
<code>count(x)</code> – numără aparițiile valorii x în listă	<pre>note = [7, 8, 9, 7, 10, 7, 9] print(note.count(7)) # 3 print(note.count(10)) # 1 print(note.count(5)) # 0 (fără eroare)</pre>	Numără de câte ori apare valoarea x în listă. Returnează 0 dacă x nu se află în listă (fără eroare). Căutarea este case-sensitive pentru string-uri.	Înainte: <code>note = [7, 8, 9, 7, 10, 7, 9]</code> După: <code>note.count(7)</code> -> 3
<code>sort()</code> – sortează lista	<pre>L = [3, 1, 4, 1, 5, 9, 2] L.sort() print(L) # [1, 1, 2, 3, 4, 5, 9] L.sort(reverse=True) print(L) # [9, 5, 4, 3, 2, 1, 1]</pre>	Sortează elementele listei (modifică lista originală). Implicit sortează crescător. Acceptă parametri: reverse = True pentru descrescător, key = funcție pentru criteriu personalizat.	Înainte: <code>L = [3, 1, 4, 1, 5, 9, 2]</code> După: <code>L.sort()</code> -> <code>L = [1, 1, 2, 3, 4, 5, 9]</code>
<code>copy()</code> – returnează o copie independentă a listei	<pre>L1 = [1, 2, 3] L2 = L1.copy() L2.append(4) print(L1) # [1, 2, 3] (nemodificată) print(L2) # [1, 2, 3, 4]</pre>	Creează și returnează o copie superficială (shallow copy) a listei.	Înainte: <code>L1 = [1, 2, 3]</code> După: <code>L2 = L1.copy()</code> -> <code>L2 = [1, 2, 3]</code> , obiecte separate
<code>extend(L)</code> – adaugă toate elementele unei liste L la sfârșitul listei	<pre>L1 = [1, 2, 3] L2 = [4, 5, 6] L1.extend(L2) print(L1) # [1, 2, 3, 4, 5, 6] print(L2) # [4, 5, 6] (nemodificată)</pre>	Adaugă fiecare element din iterabil (listă, tuple, string etc.) la sfârșitul listei curente, individual.	Înainte: <code>L1 = [1, 2, 3]</code> ; <code>L2 = [4, 5, 6]</code> După: <code>L1.extend(L2)</code> -> <code>L1 = [1, 2, 3, 4, 5, 6]</code>



Fișa de lucru 10

Creați dosarul numit *Fisa10*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

1) Completează cu metoda sau operatorul potrivit:

`L. ____ (x)` – adaugă x la sfârșitul listei

`L. ____ (i, x)` – inserează x la poziția i

`L. ____ (i)` – șterge și returnează elementul de la poziția i

`L. ____ (x)` – șterge prima apariție a valorii x

`L. ____ (x)` – returnează poziția primei apariții a lui x

`L. ____ ()` – șterge toate elementele listei

2) Fără a rula codul, scrieți ce se afișează:

<code>L = [3, 1, 4, 1, 5, 9]</code>	<code>L[2]:</code> _____
<code>print(L[2])</code>	<code>L[-1]:</code> _____
<code>print(L[-1])</code>	<code>L[1:4]:</code> _____
<code>print(L[1:4])</code>	<code>L[::-1]:</code> _____
<code>print(L[::-1])</code>	<code>print(L):</code> _____
	<code>len(L):</code> _____

3) Fără a rula codul Python de mai jos, scrieți ce se va afișa în fiecare dintre cele 2 situații:

<code>culori=["alb", "verde"]</code>	<code>numere=[10, 20, 30, 40]</code>
<code>print(culori*2)</code>	<code>numere.pop(1)</code>
	<code>print(numere)</code>

4) Fără a rula codul, scrieți ce se afișează:

<code>L = ["ana", "ion", "ema"]</code>	<code>L[2]:</code> _____
<code>L.append("mihai")</code>	<code>L[-1]:</code> _____
<code>L.insert(1, "dan")</code>	<code>L[1:4]:</code> _____
<code>L.pop(2)</code>	<code>L[::-1]:</code> _____
<code>print(L)</code>	<code>print(L):</code> _____
<code>print(len(L))</code>	<code>len(L):</code> _____

5) Completați programul care gestionează o listă de produse dintr-un depozit:

```
depozit1 = ["mere", "pere", "prune"]
depozit2 = ["capsuni", "zmeura"]
# 1. Combina cele doua liste într-o listă nouă
total = _____
# 2. Adaugă "struguri" la sfârșitul listei total
_____
# 3. Verifică dacă "banane" se află în listă
if _____ :
    print("Avem banane")
else:
    print("Nu avem banane")
# 4. Afișează poziția produsului "pere" în lista total
poz = _____
print(f"Pere se afla la pozitia {poz}")
# 5. Numără de câte ori apare "mere" în lista total
nr = _____
print(f"Mere apare de {nr} ori")
```

Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Se dă lista L=[10,20,30,40,50]. Ce va afișa print(L[1:4])? a) [10,20,30,40] b) [20,30,40,50] c) [10,20,30] d) [20,30,40] 2) Dacă avem L=["ananas", "mere", "pere", "struguri", "nuci"], care este rezultatul L[:2]? a) ["ananas", "pere", "nuci"] b) ["ananas", "mere"] c) ["ananas"] d) invers
ÎNȚELEGERE	3) Fără a rula codul, scrieți ce se va afișa: <pre>culori = ["alb", "verde"] print(culori * 2)</pre> <pre>numere = [10, 20, 30, 40] numere.pop(1) print(numere)</pre>



APLICARE	4) Fie lista $L=["\text{Teo}","Ema","Ana","Alin","Mara"]$. Scrieți instrucțiunile necesare pentru: - adaugă "Sara" la sfârșitul listei; - inserează "Victor" la începutul listei; - șterge "Ema" din listă; - extinde lista cu primele 3 persoane din lista inițială; - afișează lista finală.
ANALIZĂ	5) Un elev scrie următorul program pentru a șterge primul element cu valoarea 2 și ultimul element din listă: <pre>L = [1, 2, 3, 2, 4] L.remove(2) L.pop() print(L)</pre> a) Ce afișează programul? Urmăriți pas cu pas. b) Ce se întâmplă dacă valoarea 2 nu există în listă la apelul <code>remove()</code> ? c) Modificați programul astfel încât să șteargă TOATE aparițiile valorii 2, folosind un <code>while</code> .
EVALUARE	6) Un elev scrie următorul program pentru a șterge toate aparițiile valorii 2 dintr-o listă: <pre>L = [1, 2, 3, 2, 4, 2] for x in L: if x == 2: L.remove(x) print(L)</pre> Rezultat așteptat: $[1, 3, 4]$ Rezultat real: $[1, 3, 4, 2]$ (<i>surprinzător!</i>) a) Explicați de ce programul nu șterge toate aparițiile lui 2. b) Propuneți două variante corecte de rezolvare (una cu <code>while</code> , una cu list comprehension). <pre># Varianta 1 - while: L = [1, 2, 3, 2, 4, 2] ... # Varianta 2 - list comprehension: L = [1, 2, 3, 2, 4, 2] L = ...</pre>
CREARE	7) Miniproiect - lista de cumpărături: Creați un program Python care gestionează o listă de cumpărături cu un meniu: a) afișare listă; b) caută și șterge un produs; c) adaugă produs; d) exit. Meniul rulează până la apăsarea tastei 0.

Barem: 1p (din oficiu) + 0.5p (1) + 0.5p (2) + 0.5p (3) + 2p (4) + 1p (5) + 2p (6) + 2.5p (7)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să accesez elemente dintr-o listă prin indexare pozitivă și negativă.			
2.	să extrag subliste cu operatorul de slicing.			
3.	să verific apartenența unui element la o listă cu <code>in</code> și <code>not in</code> .			
4.	să folosesc operatorii <code>+</code> , <code>*</code> și operatorii relaționali pe liste.			
5.	cum să aplic metode de bază asupra listelor.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 11 – Metode avansate ale clasei list și parcurgerea elementelor

11.1. Sortarea avansată cu sort(key=...)

În Python, există și metode avansate ale clasei **list**. *Metoda sort()* devine o metodă avansată dacă se utilizează parametrul **key**: **ID_lista.sort(key=funcție)**. Acesta permite sortări personalizate, prin care se specifică după ce criteriu vor fi sortate elementele.

Instrucțiune	Efect
L.sort(key=len)	ordonează textele după lungime
L.sort(key=abs)	ordonează numerele după valoarea absolută

Parametrul **key**: sortează după un criteriu personalizat:

```
L = ["ananas", "nuci", "pepene", "struguri"]
L.sort(key=len)      # L=["nuci", "ananas", "pepene", "struguri"]

L2=[3, 9, -6, 7, -8, 5]
L2.sort(key=abs)     # L2=[3, 5, -6, 7, -8, 9]
```



11.2. Inversarea ordinii

Metoda reverse() inversează ordinea elementelor existente, fără să le sorteze.

```
L = [3, 1, 4, 2] # sort descrescator:
L.sort(reverse=True)
print(L)        # [4, 3, 2, 1]
# reverse: INVERSEAZA ordinea, nu sorteaza!
L2 = [3, 1, 4, 2]
L2.reverse()
print(L2)       # [2, 4, 1, 3]
```

11.3. Copierea listelor: atribuire vs. copy() vs. slicing

Listele sunt obiecte modificabile/mutabile. Dacă atribuim o listă unei alte variabile, nu se creează automat o copie, ci două nume pentru aceeași listă.

Metoda	Ce face	Efect
Atribuire (L2=L1)	Nu creează copie – doar alias (este referită aceeași zonă de memorie)	Orice modificare în L2 afectează și L1
copy()	Creează copie superficială, independentă	L1 rămâne nemodificată la schimbările din L2
Slicing [:]	Creează copie (similar cu copy())	L2 = L1[:] - echivalent cu L1.copy()

Copiere prin atribuire	Copiere prin metoda copy()
<pre>L1 = [1, 2, 3] L2 = L1 # creez un alias al listei L1 L2.append(4) # adaug 4 la sfârșitul # obiectului unic print(L1) # L1=[1, 2, 3, 4] print(L2) # L2=[1, 2, 3, 4]</pre>	<pre>L1 = [1, 2, 3] L2 = L1.copy() L2.append(4) # adaug 4 la sfârșitul # listei L2 print(L1) # L1=[1, 2, 3] print(L2) # L2=[1, 2, 3, 4]</pre>

11.4. Operatorii == și is

Operatorul de **egalitate** `==` verifică dacă două liste au același conținut. Operatorul de **identitate** `is` verifică dacă două variabile indică exact același obiect în memorie.

Operatorul `is` vs. `==`

`==` verifică CONȚINUTUL (element cu element), `is` verifică dacă e indicat ACELAȘI OBIECT în memorie.

```
L1=[1,2,3] ; L2=L1 ; L3=[1,2,3]
print(L1==L2) # True (același conținut)
print(L1 is L2) # True (același obiect)
print(L1==L3) # True (același conținut)
print(L1 is L3) # False (obiecte diferite)
```

11.5. Moduri de parcurgere a listelor

Tip	Sintaxa	Când se folosește	Exemplu
Parcurgere element cu element	<code>for x in lista:</code>	Util când ne interesează valoarea, nu poziția	<code>for n in [5,2,3]: s+=n</code>
Parcurgere prin index	<code>for i in range(len(lista)):</code>	Util când avem nevoie și de poziție	<code>L=[1,2,3]</code> <code>for i in range(len(L)):</code> <code>print(i, L[i])</code>
Parcurgere cu <code>enumerate()</code>	<code>for i,x in enumerate(lista):</code>	Acces simultan la index și valoare	<code>L=[1,2,3]</code> <code>for i,n in enumerate(L):</code> <code>print("pozitia", i,</code> <code>"valoarea", n)</code>

11.6. Crearea listelor cu list comprehensions

Sintaxa: [expresie `for` element `in` iterabil `if` condiție]

```
# Clasic:
patrate = []
for x in range(5):
    patrate.append(x**2)
# [0,1,4,9,16]

# Cu list comprehension:
patrate = [x**2 for x in range(5)]

# Cu filtru:
pare = [x for x in range(10) if x%2==0]
# [0,2,4,6,8]
```

Exemple practice:

```
# Notele peste medie:
note = [7,8,5,9,6,10]
media = sum(note)/len(note)
peste_medie = [n for n in note if n > media]

# Cuvinte lungi:
cuvinte = ["mere", "pepene", "par"]
lungi = [c for c in cuvinte if len(c) > 4]
# ["pepene"]

# Primele n patrate perfecte nenule:
n = 5
L = [i**2 for i in range(1, n+1)]
# [1,4,9,16,25]
```



Fișa de lucru 11

Creați dosarul numit *Fisa11*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier raspunsuri.docx.

1) Ce afișează următorul cod?

```
L = ["ananas", "mere", "pere", "struguri", "nuci"]
copie = L[1:4]
L[2] = "banane"
print(copie)
```

- a) ["mere", "banane", "struguri"] b) ["mere", "pere", "struguri"]
c) ["ananas", "mere", "pere"] d) ["mere", "pere", "nuci"]

2) Completați cu răspunsul corect:

`L2 = L1` — L2 este un al lui L1 (același obiect în memorie)

`L2 = L1.copy()` — L2 este o independentă.

`L1 == L2` — verifică dacă listele au aceleași

`L1 is L2` — verifică dacă listele sunt în memorie.

3) Fără a rula codul, scrieți ce se afișează:

```
A = [1, 2, 3]
B = A[:]
C = A
A.append(4)
print(B)
print(C)
print(A is B)
print(A is C)
```

B:

C:

A is B:

A is C:

4) Se dă lista cu invitați la o petrecere: `L = ["Teo", "Ema", "Ana", "Alin", "Mara"]`. Scrieți ce afișează fiecare instrucțiune (fără a rula):

Instrucțiune Python	Rezultat afișat
<code>print(L[::-1])</code>	
<code>print(sorted(L))</code>	
<code>print(L) # dupa sorted</code>	
<code>L2 = L.copy() ; L2.sort() ; print(L)</code>	
<code>print(L[1::2])</code>	
<code>print([x for x in L if len(x) == 3])</code>	

5) Analizați algoritmul și răspundeți la întrebări:

```
n = int(input())
L = [i**2 for i in range(1, n+1)]
print(L)
```

a) Ce conține lista L pentru $n = 5$?

b) Rescrieți același algoritm FĂRĂ list comprehension (varianta clasică cu for și append).

```
n = int(input())
L = []
for i in range(1, n+1):
    ... # completează
print(L)
```

c) Modificați list comprehension-ul pentru a păstra DOAR pătratele numerelor impare.



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Care este variabila care ia valoarea fiecărui element din listă pe rând? <pre>for x in numere: print(x)</pre> a) print; b) in; c) x; d) numere.
ÎNȚELEGERE	2) Ce se afișează în urma executării programului de mai jos? <pre>L = ["ananas", "mere", "pere", "struguri", "nuci"] copie_L = L[1:4] L[2] = "banane" print(copie_L) print(L[2] == copie_L[1])</pre> a) ["mere", "pere", "struguri"] și True; b) ["ananas", "mere", "pere"] și False; c) ["mere", "pere", "struguri"] și False; d) ["mere", "banane", "struguri"] și True.
APLICARE	3) Fie lista persoanelor la o petrecere: L=["Teo", "Ema", "Ana", "Alin", "Mara"]. Scrieți programul Python care afișează: invitații în ordinea sosirii; invitații în ordinea inversă sosirii; invitații în ordine alfabetică, FĂRĂ a modifica lista inițială; primii doi și ultimii doi invitați sosiți.
ANALIZĂ	4) Se citește de la tastatură o listă cu n numere naturale. Scrie câte un program Python pentru fiecare cerință: a) Afișează câte elemente din listă sunt mai mari decât ultimul element. <i>Exemplu:</i> [3, 7, 2, 9, 5] → 2 (elementele 7 și 9); b) Afișează poziția primului element par din listă. Dacă nu există, afișează -1. <i>Exemplu:</i> [3, 7, 2, 9, 4] → 2; c) Afișează poziția ultimului element impar din listă. Dacă nu există, afișează -1. <i>Exemplu:</i> [3, 7, 2, 9, 4] → 3.
EVALUARE	5) Analizați codul de mai jos și identificați diferența dintre aliasuri și copii: <pre>A = [1, 2, 3] B = A[:] print(A is B) print(A == B)</pre> <pre>fructe = ["mar", "para"] lista_noua = fructe lista_noua[0] = "banana" print(fructe[0])</pre> Explicați de ce, în cele două exemple, comportamentele sunt diferite.
CREARE	6) Se citește o listă cu n numere reprezentând notele elevilor. Care sunt numerele de identificare ale elevilor cu nota peste media clasei? <i>Hint: notează elevii cu numerele 1, 2, ..., n. Determină media, apoi afișează indicii celor cu nota > media.</i>

Barem: 1p (din oficiu) + 0.5p (1) + 0.5p (2) + 2.5p (3) + 2.5p (4) + 1p (5) + 2p (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să sortez o listă cu criterii personalizate (key=, reverse=).			
2.	diferența dintre copierea prin atribuire (alias) și copierea independentă cu copy() sau slicing.			
3.	diferența dintre operatorul is și ==.			
4.	să parcurg o listă prin cele 3 metode (element, index, enumerate).			
5.	să creez liste cu list comprehensions.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 12 – Introducere în POO: clase, obiecte și instanțiere

12.1. De ce POO? De la procedural la orientat pe obiecte

Până acum am scris cod procedural: secvențe de instrucțiuni executate una după alta. Pe măsură ce programele devin mai complexe, **POO** (Programarea Orientată pe Obiecte) ne permite să modelăm realitatea prin entități numite **obiecte**, care combină **date** (atribute) și **subprograme** (metode).

12.2. Clasa

Analogie: Clasa = tiparul, Obiectul = produsul

Clasa: șablonul sau planul de construcție. Definește structura și comportamentul unui tip de obiect. (Exemplu: proiectul (șablonul) unei case)

Obiectul: instanță concretă a clasei, creată din șablon, cu propriile date specifice. (Exemplu: casa propriu-zisă)

Membrii clasei: Date (atribute) + Metode (subprograme asociate obiectului)

Definirea unei clase

 **Clasa** este un tipar care definește structura și comportamentul unui tip de obiecte. Ea specifică ce date poate stoca un obiect și ce operații poate efectua, fără a crea obiecte concrete. Clasele se definesc cu ajutorul cuvântului cheie **class**.

Caracteristici ale unei clase:

- Se definește o singură dată, dar poate fi folosită pentru a crea oricâte obiecte este nevoie;
- Conține două categorii de membri: **date** (atribute) și **subprograme** (metode);
- În Python se definește cu cuvântul cheie **class**, urmat de numele clasei (cu majusculă);
- Analogie: clasa este ca un tipar/proiect al unei case – tiparul nu este casă, dar folosind același tipar/proiect se creează case identice ca formă.

```
class Elev:           # definirea clasei - ȘABLONUL
    pass              # (corp minim, fără membri deocamdată)
```

12.3. Membrii clasei – date și metode

Membrii clasei sunt componentele definite în interiorul unei clase. Există două categorii:

Date (atribute):

 Datele sunt variabilele care aparțin unui obiect și stochează informații despre acesta. Se definesc în constructorul clasei folosind **self.nume_atribut = valoare**.

- Fiecare obiect are propriile valori ale atributelor, independente de alte obiecte;
- Sunt accesate din exterior prin sintaxa **obiect.atribut**.

Metode:

Metodele sunt subprogramele (funcțiile) definite în interiorul clasei, care descriu comportamentul obiectelor. Primul lor parametru este întotdeauna **self** – referința la obiectul curent.

- Metodele pot accesa și modifica atributele obiectului prin **self**;
- Sunt apelate din exterior prin sintaxa **obiect.metoda(argumente)**.

12.4. Constructorul clasei (`__init__`) și operatorul **self**

Constructorul¹ este o metodă specială a clasei care se execută **automat** în momentul instanțierii unui obiect – adică imediat ce scriem **obiect = NumeClasa(argumente)**. Rolul său este de a inițializa starea internă a obiectului prin atribuirea unor valori specifice atributelor sale încă din momentul creării.

**Operatorul **self**:**

self este modul prin care un obiect se referă la el însuși în interiorul metodelor. Când apelăm o metodă pe un obiect, Python transmite automat acel obiect ca prim argument (**self**), astfel încât metoda știe despre care obiect concret este vorba.

```
class Animal:
    def __init__(self, nume, varsta):
        #inițializarea atributelor specifice fiecărei instanțe
    self.nume = nume # atribut - persistă în memorie
    self.varsta = varsta # atribut - persistă în memorie
    def descriere(self, x):
        # self.varsta = atribut al obiectului
        # x = parametru extern, există doar pe durata acestei metode
        return self.varsta, x

# Crearea obiectelor (instanțiere) prin apelarea constructorului
caine = Animal("Bobby", 2)
pisica = Animal("Katy", 5)

print(caine.nume) # Accesăm atributul nume al primului obiect - Bobby
print(pisica.nume) # Accesăm atributul nume al celui de-al doilea obiect - Katy
print(caine.descriere(7)) # (2, 7)

class Elev:
    def __init__(self, nume, nota_medie): # constructor
        self.nume = nume # dată (atribut)
        self.nota_medie = nota_medie # dată (atribut)
    def saluta(self): # metodă
        print(f"Bună! Sunt {self.nume}.")
    def este_promovat(self): # metodă
        return self.nota_medie >= 5
```

Metoda **`__init__`** este un constructor special care se apelează automat în momentul creării unui obiect și este folosită pentru a inițializa atributele acestuia.

¹ În POO, **constructorul** (sau metoda de inițializare) este mecanismul prin care un obiect nou creat își stabilește starea inițială și primește valorile pentru atributele sale. Fără această etapă de inițializare, atributele obiectului nu ar fi definite sau nu ar conține date valide pentru a putea fi utilizate în program.

12.5. Metode speciale (Dunder Methods)

Pe lângă metodele obișnuite, Python oferă **metode speciale (dunder methods)** (dunder este abrevierea de la double underscore). Acestea încep și se termină cu `__` și au roluri predefinite în limbaj — pot fi apelate direct prin operatorul sau funcția corespunzătoare, fără a le numi explicit:

Metodă specială	Rol și declanșator
<code>__init__(self, ...)</code>	Constructorul – se apelează automat la instanțiere: <code>Animal("Bobby", 2)</code>
<code>__str__(self)</code>	Specifică ce text să fie afișat când obiectul este convertit la șir de caractere – se apelează automat de <code>print()</code> : <code>print(obiect)</code>

```
class Animal:
    def __init__(self, nume, varsta):
        self.nume = nume
        self.varsta = varsta
    def __str__(self):          # apelat de str()
        return f"{self.nume} are {self.varsta} ani."
# Creăm trei obiecte (instanțe ale clasei)
pufu = Animal("Pufu", 3)
pufi = Animal("Pufu", 5)
bobi = Animal("Bobi", 2)
# Afișăm obiectele (print apelează automat funcția str())
print(pufu)                  # Pufu are 3 ani.      (apelează __str__)
print(pufi)                  # Pufu are 5 ani.      (apelează __str__)
print(bobi)                  # Bobi are 2 ani.      (apelează __str__)
```

Funcțiile pot fi apelate cu denumirea lor, dar și direct, folosind simbolul respectiv (de exemplu, `print(obiect)` apelează automat **metoda** `__str__`).

12.6. Instanțierea unei clase predefinite



Instanțierea este procesul prin care se creează un obiect concret pe baza unei clase.

```
# Instanțierea clasei list (predefinită):
note = list()
note.append(10)
print(note)

# Instanțierea clasei date din biblioteca datetime:
from datetime import date
azi = date.today()          # metodă de clasă care creează un obiect
nastere = date(2008, 3, 15) # instanțiere cu argumente: an, lună, zi
```

Element	Rol
<code>date</code>	clasa predefinită importată din biblioteca <code>datetime</code>
<code>azi</code>	obiectul creat prin metoda <code>today()</code>
<code>year, month</code>	atribute ale obiectului
<code>strftime()</code>	metodă pentru formatarea datei

Scrierea **`obiect.metoda()`** indică faptul că metoda aparține obiectului. De exemplu, `note.append(10)` adaugă o valoare în obiectul listă `note`.

12.7. Accesul la membrii unui obiect

Accesul la membrii unui obiect se realizează prin operatorul punct (`.`), care permite atât citirea/modificarea atributelor, cât și apelul metodelor:

```
nume = "informatică"
print(nume.upper())      # metodă a obiectului de tip str
print(nume.capitalize())
```

```
note = [8, 10, 7]
note.append(9)
note.sort()
print(note)
```

Modelare simplă cu obiecte predefinite

Putem organiza informații despre elevi folosind clase predefinite: lista pentru notele elevului, șiruri pentru nume, numere pentru valori.

```
nume = "Ana"
note = [9, 10, 8]
media = sum(note) / len(note)
print(nume, round(media, 2))
```

Sintaxă	Descriere
<code>obiect.atribut</code>	Citirea valorii unui atribut al obiectului
<code>obiect.atribut = val</code>	Modificarea valorii unui atribut al obiectului
<code>obiect.metoda()</code>	Apelul unei metode fără argumente suplimentare
<code>obiect.metoda(arg1, arg2)</code>	Apelul unei metode cu argumente



Reține! În această etapă este important să recunoaștem obiectele și să folosim metodele lor. Definirea unor clase proprii se poate realiza ulterior, după ce stăpânim lucrul cu clase predefinite.



Fișa de lucru 12

Creați dosarul numit *Fisa12*. Rezolvați cerințele următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

- 1) Care este diferența principală dintre o clasă și un obiect? Folosiți o analogie din viața reală.
- 2) Dacă definim o clasă **Caine** cu un atribut **nume**, scrieți codul necesar pentru a crea două obiecte diferite (doi câini cu nume diferite).
- 3) Scrieți o clasă numită **Elev** care să aibă atributele **nume** și **medie**. Adăugați o metodă numită **este_premiant()** care să returneze **True** dacă media este mai mare sau egală cu **9**, altfel **False**.
- 4) Definiți o clasă **Masina** cu datele: **marca**, **model**, **an_fabricatie**. Adăugați o metodă **varsta()** și o metodă **descriere()** care afișează toate datele.
- 5) Creați 3 obiecte de tip **Masina**. Accesați atributele și metodele fiecăruia. Modificați **an_fabricatie** al unui obiect și verificați dacă celelalte nu s-au schimbat.



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Ce rol are parametrul <code>self</code> în interiorul unei metode? a) Reprezintă o variabilă globală accesibilă de oriunde. b) Este o referință către instanța curentă a clasei. c) Este un cuvânt cheie care oprește execuția codului.
ÎNȚELEGERE	2) Ce se va afișa și de ce? <pre>class Carte: def __init__(self, titlu): self.titlu = titlu c1 = Carte("Ion") c2 = c1 c2.titlu = "Baltagul" print(c1.titlu)</pre> Explicați de ce c1.titlu s-a schimbat, deși am modificat c2.
APLICARE	3) Identifică eroarea din codul următor și explică de ce nu va rula: <pre>class Masina: def __init__(nume): self.nume = nume auto = Masina("Dacia")</pre> Scrieți o clasă Elev cu attributele nume și medie. Adăugați o metodă este_premiant() care returnează True dacă media ≥ 9.
ANALIZĂ	4) Scrieți o clasă Piesa care modelează o melodie dintr-un playlist cu: - Constructor cu: titlu, artist, durata (în secunde) - Metoda durata_minute() care returnează durata în minute (cu rotunjire în sus dacă trebuie) - Metoda descriere() care afișează: "Titlu - Artist (X min)" Creați două obiecte și afișați descrierea pentru ambele.
EVALUARE	5) Ce se întâmplă dacă încercăm să accesăm un atribut al unui obiect care nu a fost definit în __init__ ? Demonstrați cu un exemplu de cod și explicați eroarea obținută.
CREARE	6) Proiectați o clasă Temperatura cu: - Atribut valoare (în grade Celsius) - Metoda in_fahrenheit() care returnează valoarea convertită: $F = C * 9/5 + 32$ - Metoda descriere() care afișează: "20°C = 68.0°F" Creați două obiecte cu temperaturi diferite și afișați descrierea pentru fiecare.

Barem: 1p (din oficiu) + 1p (1) + 1p (2) + 1p (3) + 2.5p (4) + 1p (5) + 2.5p (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	diferența dintre o clasă și un obiect.			
2.	ce este constructorul (<code>__init__</code>).			
3.	să definesc o clasă cu attribute și metode în Python.			
4.	ce este instanțierea și cum se realizează.			
5.	cum se accesează membrii unui obiect.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
 (cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 13 – Fișiere text: deschidere, citire, scriere, închidere; clasa TextIOWrapper

13.1. De ce folosim fișiere?



Variabilele păstrează datele doar cât timp programul rulează. **Fișierele** permit stocarea permanentă a informațiilor, astfel încât datele să poată fi citite din nou la o rulare ulterioară.

Un fișier text conține caractere: litere, cifre, spații și linii. În programele simple, îl folosim pentru a citi date de intrare sau pentru a salva rezultate.

Fișiere text:

- Stochează informații ca caractere (.txt, .csv, .py, .html);
- Pot fi deschise cu editoare de text;
- Folosite la schimbul de date în Python.

Fișiere binare:

- Stochează informații în format codificat;
- Imagini, audio, video, executabile;
- Nu se pot citi direct ca text.



13.2. Etapele lucrului cu un fișier

Etapă	Descriere
deschidere	programul stabilește legătura cu fișierul
citire/scriere	datele sunt transferate din fișier în program sau invers
închidere	legătura se eliberează; modificările sunt salvate

13.3. Deschiderea și închiderea unui fișier

Sintaxe generale:

```
nume_variabila_pt_fisier=open(cale\nume_fisier, mod, codificare)
nume_variabila_pt_fisier.close()
```

```
fisier = open("date.txt", "r", encoding="utf-8")
continut = fisier.read()
fisier.close()
```

Blocul with (recomandat)

Blocul with închide automat fișierul la sfârșitul blocului, chiar și în caz de eroare:

Sintaxa: `with open(cale/nume_fisier, mod, encoding) as nume_variabila_pentru_fisier:`

```
with open("date.txt", "r", encoding="utf-8") as fisier:
    continut = fisier.read()
    print(continut)
# fisierul este inchis automat dupa with
```

Mod	Semnificație
"r"	citire - fișierul trebuie să existe
"w"	scriere - creează fișierul sau șterge conținutul vechi
"a"	adăugare - scrie la finalul fișierului

⚠ Atenție! Modul "w" rescrie fișierul. Dacă fișierul avea conținut, acesta se pierde.



13.4. Citirea datelor din fișier

Pentru citire folosim metode ale obiectului fișier. Alegem metoda în funcție de cum sunt organizate datele.

Metodă	Rol
read()	citește tot conținutul fișierului ca un singur șir de caractere
readline()	citește o singură linie
readlines()	citește toate liniile într-o listă

```
with open("mesaj.txt", "r", encoding="utf-8") as f:
    continut = f.read()
print(continut)
```

Citirea unei liste de valori

Dacă într-un fișier avem numere separate prin spații, putem citi tot conținutul și apoi transforma valorile în numere întregi.

Forma recomandată: Instrucțiunea **with** închide automat fișierul, chiar dacă apar erori. De aceea este preferată în programele moderne.

```
with open("numere.txt", "r") as f:
    text = f.read()
    valori = text.split()
    numere = []
    for x in valori:
        numere.append(int(x))
    print(numere)
```

13.5. Scrierea datelor în fișier

```
with open("rezultat.txt", "w", encoding="utf-8") as f:
    f.write("Salut din Python!\n")
    f.write("Date importante")
```

13.6. Clasa TextIOWrapper



Atunci când deschidem un fișier text cu **open()**, Python creează un obiect de tip **TextIOWrapper**. Acest obiect oferă metode de citire, scriere și închidere.

```
f = open("date.txt", "r", encoding="utf-8")
print(type(f))
f.close()
```

Metodă	Rol
f.read()	citește conținutul întregului fișier
f.readline()	citește o singură linie
f.readlines()	citește toate liniile într-o listă
f.write(text)	scrie text în fișier
f.writelines(listă)	scrie o listă de string-uri
f.close()	închide fișierul

Erori frecvente

- fișierul nu există atunci când este deschis în modul **"r"**;
- uităm să transformăm textul citit în număr cu **int()** sau **float()**;
- folosim modul **"w"** și ștergem din greșeală conținutul vechi;
- uităm să închidem fișierul, dacă nu folosim **with**.

Aplicație: calculul mediei notelor dintr-un fișier

```
with open("note.txt", "r") as f:
    valori = f.read().split()
    note = []
    for x in valori:
        note.append(int(x))
    media = sum(note) / len(note)
    with open("media.txt", "w") as f:
        f.write(str(round(media, 2)))
```



Reține! Fișierele text sunt utile când programul trebuie să păstreze date după terminarea execuției sau să prelucreze un volum mai mare de date.



Fișa de lucru 13

Creați dosarul numit *Fisa13*. Rezolvați cerințele următoare, în limbajul Python notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

1) Jurnalul zilnic. Completați programul Python de mai jos, care:

- Citește de la tastatură 3 activități din ziua curentă
- Le scrie într-un fișier "**jurnal.txt**", fiecare pe o linie nouă
- Redeschideți fișierul și afișați conținutul cu numerele rândurilor (1. activitate, 2. activitate, ...)

```
# Pasul 1 - scriere
with open("jurnal.txt", "w") as f:
    for i in range(3):
        act = input(f"Activitate {i+1}: ")
        ... # scrie in fisier

# Pasul 2 - citire si afisare numerotata
with open("jurnal.txt", "r") as f:
    for nr, linie in enumerate(f, 1):
        ... # afiseaza cu numar de rand
```

2) Urmărește execuția și completează ce se va afișa dacă fișierul "**note.txt**" conține:

```
8
10
7
9
6
```

```
with open("note.txt", "r") as f:
    note = []
    for linie in f:
        n = int(linie.strip())
        note.append(n)

print(len(note))
print(max(note))
print(sum(note) / len(note))
```

Instrucțiune	Valoare afișată
<code>print(len(note))</code>	
<code>print(max(note))</code>	
<code>print(sum(note)/len(note))</code>	

Ce se întâmplă dacă fișierul conține și o linie goală? Cum se protejează codul?



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Enumerați etapele în lucrul cu un fișier text și metodele corespunzătoare în Python. 2) Care mod de deschidere trebuie folosit pentru a adăuga text la sfârșitul unui fișier existent, fără a șterge conținutul vechi? a) "r"; b) "w"; c) "a"; d) "x".
-------------------	---

ÎNȚELEGERE	3) Explicați mecanismele de citire și scriere într-un fișier. Ce tip returnează expresia <code>open("date.txt", "r")</code> ? De ce este utilă cunoașterea tipului? <pre>fisier = open("date.txt", "r") print(type(fisier))</pre>
APLICARE	4) Creați un fișier text și scrieți în el 3 nume de orașe. Citiți un fișier linie cu linie și afișați conținutul. Calculați suma numerelor memorate într-un fișier text (un număr pe fiecare linie).
ANALIZĂ	5) Comparați cele două variante de lucru cu fișiere: <pre># Varianta 1: f = open("date.txt", "r") continut = f.read() f.close()</pre> <pre># Varianta 2: with open("date.txt", "r") as f: continut = f.read()</pre> Ce avantaje oferă Varianta 2? Ce se întâmplă dacă apare o eroare în Varianta 1?
EVALUARE	6) Un elev scrie următorul program care ar trebui să citească numere dintr-un fișier și să calculeze suma: a) Ce eroare apare la rulare și de ce? Identificați linia cu problema. b) Corectați codul. Ce conversie este necesară și de ce? c) Ce se întâmplă dacă fișierul conține o linie goală la sfârșit? Cum tratați acest caz? <i>Hint: datele citite cu <code>readline()</code> sau <code>for linie in f</code> sunt întotdeauna de tip <code>str</code>.</i> <pre>with open("numere.txt", "r") as f: suma = 0 for linie in f: suma += linie print("Suma:", suma)</pre>
CREARE	7) Scrieți un program Python care citește <i>n</i> numere de la tastatură, le salvează în fișierul "numere.txt" câte unul pe linie, apoi recitește fișierul și afișează minimul, maximum și media numerelor citite.

Barem: 1p (din oficiu) + 1p (1) + 1p (2) + 1p (3) + 2p (4) + 1p (5) + 1.5p (6) + 1.5p (7)

	<i>La finalul lecției, știu...</i>	<i>Da</i>	<i>Nu</i>	<i>Nu sunt sigur(ă)</i>
1.	să explic ce este clasa <code>TextIOWrapper</code> .			
2.	să deschid un fișier cu codificare specificată folosind parametrul <code>encoding="utf-8"</code> .			
3.	să citesc eficient un fișier mare linie cu linie, fără a încărca tot conținutul în memorie.			
4.	să accesez proprietățile unui obiect <code>TextIOWrapper</code> .			
5.	să identific și să tratez cazurile limită: fișier gol, fișier inexistent, date invalide.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
 (cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 14 – Generare sistematică: șiruri recurente și secvențe cu proprietăți date

14.1. Ce este generarea sistematică?

În informatică, apar frecvent situații în care trebuie să construim pas cu pas liste de valori respectând o anumită regulă. Acest proces poartă numele de **generare sistematică**.



Metode de generare sistematică. Există trei metode principale de generare a elementelor unei liste, fiecare adaptată unui tip specific de problemă:

Metoda	Descriere	Exemplu
1. Prin formulă	Fiecare termen se calculează direct printr-o expresie matematică ce depinde de indice.	$a_n = 2 \cdot n \rightarrow 0, 2, 4, 6 \dots$
2. Cu proprietăți	Se testează fiecare valoare; se include în secvență doar dacă satisface o condiție dată.	numere prime, pătrate perfecte
3. Recurentă	Fiecare termen depinde de termenii anteriori ai secvenței. Nu se poate calcula direct.	șirul lui Fibonacci, dobânda compusă

14.2. Generare prin formulă directă

Când între indicele n și valoarea termenului există o relație matematică directă, fiecare element poate fi calculat **independent** de ceilalți termeni, fără a cunoaște valorile anterioare. Programul parcurge valorile lui n și calculează direct fiecare termen folosind formula dată.

Multiplii lui 2 $a_n = 2 \cdot n$	Pătrate perfecte $a_n = n^2$
<pre>n = int(input()) for i in range(n): print(2 * i)</pre>	<pre>n = int(input()) for i in range(1, n+1): print(i * i)</pre>

14.3. Generare cu proprietăți date

Uneori dorim să selectăm din mulțimea numerelor naturale doar pe cele care satisfac o anumită condiție (proprietate). Algoritmul parcurge valorile și include în secvență doar pe cele care trec testul condiției.

Numere pare Condiția: $i \% 2 == 0$	Numere prime Condiția: dacă nu are divizori	Multiplii de 3 sau 5 Două condiții legate cu or
<pre>n = int(input()) for i in range(n): if i % 2 == 0: print(i)</pre>	<pre>n = int(input()) for i in range(n): if i >= 2: este_prim = True for d in range(2, i): if i % d == 0: este_prim = False break if este_prim: print(i)</pre>	<pre>n = int(input()) for i in range(n): if i % 3 == 0 or i % 5 == 0: print(i)</pre>

14.4. Șiruri recurente

Un **șir recurent** este un șir în care fiecare termen depinde de unul sau mai mulți termeni anteriori. Spre deosebire de formula directă, termenul **nu poate fi calculat** fără a cunoaște valorile precedente.

Recurența de ordinul 1. Fiecare termen depinde de imediat precedentul. Este necesară o singură variabilă pentru starea curentă.

```
n = int(input("n = "))
a = 3
for i in range(n):
    print(a)
    a = a * 2
```

⚠ Atenție! În șirurile recurente termenii inițiali se afișează înaintea termenilor calculați.

Recurența de ordinul 2 – Șirul Fibonacci. Fiecare termen depinde de cei doi termeni anteriori. Sunt necesare **două variabile de stare**.

În șirul Fibonacci, fiecare termen se obține ca sumă a celor doi termeni anteriori. Termenii inițiali sunt 0 și 1.

```
n = int(input("n = "))
a, b = 0, 1
for i in range(n):
    print(a)
    a, b = b, a + b
```

i	0	1	2	3	4	5	6
termen	0	1	1	2	3	5	8



Reține! Generarea sistematică este utilă atunci când avem nevoie de întreaga secvență, nu doar de un singur termen. Alegem metoda în funcție de regula de formare: formulă, proprietate sau recurență.



Fișa de lucru 14

Creați dosarul numit *Fisa14*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.



I. Alegeți varianta corectă:

- 1) Un șir recurent:
 - a) depinde de termenii anteriori;
 - b) este calculat direct;
 - c) nu are regulă.
- 2) Șirul Fibonacci este:
 - a) de ordin 1;
 - b) de ordin 2;
 - c) fără recurență.

II. Observați și formulați regula:

- 3) Aplicați principiul de generare recurentă și construiți următoarele șiruri cu n termeni:
 - a) Primul termen este 2, iar fiecare termen se obține adunând 3 la termenul anterior.
 - b) Primul termen este 5, iar fiecare termen se obține înmulțind termenul anterior cu 2.
 - c) Primii doi termeni sunt 1 și 4, iar fiecare termen este suma celor doi termeni anteriori.
- 4) Se consideră șirul: 1, 3, 6, 10, 15, ...
 - a) Calculați următorii 3 termeni;
 - b) Precizați regula de formare;
 - c) Indicați tipul (recurent / formulă).
- 5) Se consideră șirul: $a_0 = 10$, $a_n = a_{n-1} - 2$
 - a) Scrieți primii 6 termeni;
 - b) Precizați tipul recurenței.

III. Realizați un program în Python pentru fiecare enunț în parte:

- 6) Numere impare consecutive. Se dă un număr natural n . Să se afișeze primele n numere impare pozitive.
- 7) Puteri ale lui 3. Se dă un număr natural n . Să se afișeze primele n puteri ale lui 3, începând cu 3^0 .



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Completați cu următorii 3 termeni și șirurile: a) 2, 5, 8, ..., ..., ...; b) 3, 6, 12, ..., ..., ...; c) 1, 1, 2, 3, 5, ..., ..., ...; d) 4, 9, 16, 25, ..., ..., ... 2) Un șir recurent: a) Depinde de termenii anteriori; b) Este calculat direct; c) Nu are regulă.
ÎNȚELEGERE	3) Completați următorii 3 termeni: a) $a_0 = 2, a_n = a_{n-1} + 5 \rightarrow \dots$ b) $a_0 = 1, a_n = 2 \cdot a_{n-1} \rightarrow \dots$ c) $a_0 = 0, a_1 = 1, a_n = a_{n-1} + a_{n-2} \rightarrow \dots$ Explicați de ce pentru șirul Fibonacci sunt necesare două variabile de stare și nu una singură.
APLICARE	4) Scrieți un program Python care generează primii n termeni ai șirului: $a_1 = 3, a_n = 2 \cdot a_{n-1}$. 5) Un copil poate urca o scară fie cu o treaptă, fie cu două trepte odată. Numărul de moduri în care poate ajunge la treapta n respectă o regulă asemănătoare șirului Fibonacci. Să se determine numărul de moduri. <i>Exemplu:</i> $n = 5 \rightarrow 8$
ANALIZĂ	6) Se consideră șirul: 1, 3, 6, 10, 15, ... • Identificați regula de generare • Scrieți relația de recurență (a_n în funcție de a_{n-1}) • Există și o formulă directă? Scrieți-o. • Care variantă este mai eficientă? Argumentați.
EVALUARE	7) Evaluați corectitudinea algoritmului următor care generează primii n termeni Fibonacci: <pre>def fibonacci(n): lista = [] a, b = 0, 1 for i in range(n): lista.append(a) a, b = b, a + b return lista</pre> Testați cu $n = 0, n = 1, n = 2$ și $n = 7$. Produce rezultatele corecte? Argumentați.
CREARE	8) Se citesc: suma inițială (capital), rata dobânzii anuale (%) și numărul de ani. Generați o listă cu soldul la sfârșitul fiecărui an și afișați după câți ani soldul depășește dublul capitalului inițial. <i>Formula:</i> $\text{sold}(n) = \text{sold}(n-1) \cdot (1 + \text{rata}/100)$ <i>Justificați:</i> De ce este mai eficientă generarea recurentă (păstrăm doar termenul anterior) față de recalcularea de la zero pentru fiecare termen?

Barem: 1p (din oficiu) + 0.5p (1) + 0.5p (2) + 0.5p (3) + 1p (4) + 1.5p (5) + 2p (6) + 1p (7) + 2p (8)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să generez o secvență de valori prin formula directă.			
2.	să calculez termeni ai unui șir.			
3.	să recunosc un șir recurent.			
4.	să generez șirul Fibonacci (recurența de ordinul 2).			

5. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Evaluare sumativă - Capitolul 3

Rezolvați exercițiile într-un document: *evaluare_sumativa3.docx*.

1) (1p) Alegeți varianta corectă pentru fiecare:

L.sort() returnează:

a) lista sortată; b) None; c) True; d) eroare.

L2 = L1 creează:

a) copie independentă; b) alias (același obiect); c) copie profundă; d) lista nouă.

2) (1p) Calculează valorile (fără a rula):

```
note = [7, 8, 9, 7, 10, 7, 9]
frecv = [0] * 11
for nota in note:
    frecv[nota] += 1
```

frecv[7]	frecv[10]	note.count(9)	max(note)

3) (1.5p) Analizați codul și răspundeți:

```
L1 = [1, 2, 3]
L2 = L1
L3 = L1.copy()
L1.append(4)
print(L2)
print(L3)
print(L1 is L2)
print(L3 == L1)
```

print(L2):

print(L3):

L1 is L2:

L3 == L1:

De ce L2 și L1 diferă de L3?

4) (1p) Completați programul care citește note dintr-un fișier și afișează media și nota maximă:

```
try:
    with open("note.txt", __) as f:
        note = [int(linie.strip()) for linie in f if linie.strip()]
        print("Media:", __ / __)
        print("Maxima:", __ (note))
except FileNotFoundError:
    print("Fișierul nu exista!")
```

5) (2.5p) Generați primii n termeni ai șirului Fibonacci. Tratați cazurile $n=0$, $n=1$ și $n \geq 2$. Testați pentru $n=7$ (rezultat: 0 1 1 2 3 5 8).

6) (2p) Scrieți o clasă **Produs** cu atributele nume și pret, metoda *aplica_discount(procent)* care scade prețul și metoda *descriere()* care afișează "Nume: X RON". Creați două obiecte, aplicați un discount de 10% la primul și afișați ambele.

Barem de corectare: Se acordă **1 punct** din oficiu!

1) 1 punct. 2) 1 punct – 0.25p/celulă. 3) 1.5 puncte – 0.3p/rând. 4) 1 punct. 5) 2.5 puncte (0.5p/cazuri + 1p cod).

6) 2 puncte (1p clasă + 0.5p metode + 0.5p apeluri).



Investighează!

(100 puncte)

Realizați, într-un fișier **investigheaza3.docx**, o investigație despre comportamentul listelor Python și al fișierelor text în situații reale.

- Investigați diferența dintre atribuire ($L2=L1$) și copiere ($L2=L1.copy()$): scrieți 3 exemple de cod, ru-lați-le și explicați rezultatele cu referire la conceptele de alias și copie independentă. **(25p)**
- Construiți o listă de frecvență pentru răspunsurile la un sondaj de satisfacție (1-5 stele) cu cel puțin 20 de valori generate aleator. Afișați histograma text (Exemplu: nota 3: ***). **(25p)**
- Scrieți un program care generează **15** numere aleatoare, le salvează în "date.txt", le recitește și calculează media, min, max. Tratați cazul când fișierul e inexistent. **(25p)**
- Creați o clasă **Elev** cu atributele nume și lista de note. Adăugați metode: *adauga_nota(n)*, *media()*, *este_promovat()*. Testați metodele folosind **3** obiecte de tip elev. **(15p)**
- Reflecție: În ce situații ați folosi o listă de frecvență în loc să sortați lista? (4-5 rânduri) **(10p)**

25p	25p	25p	15p	10p
Alias vs. copie	Listă frecvență + histogramă	Fișiere + statistici	Clasa Elev	Reflecție
Recomandare: Se lucrează individual. Fișierul se salvează în dosarul Capitol3. Se includ capturi de ecran.				



Proiect

(100 puncte)

Realizați, în echipă, o aplicație Python **catalog_digital.py** care gestionează catalogul unei clase. Salvați în dosarul **Capitol3**.

- Clasa **Elev** cu: nume, listă de note, metode *adauga_nota(n)*, *media()*, *este_promovat(prag=5)*. **(20p)**
- Funcții pentru catalog: *adauga_elev(lista_elevi)*, *cauta_elev(nume)*, *sterge_elev(nume)*. **(20p)**
- Salvare și încărcare catalog dintr-un fișier "catalog.txt" (un elev pe linie: "Nume nota1 nota2 ..."). **(25p)**
- Raport statistic: media clasei, cel mai bun elev, numărul de corigenți, lista de frecvență a notelor. **(20p)**
- Meniu interactiv (adaugă, caută, șterge, salvează, raport, ieșire) care rulează până la opțiunea 0. **(15p)**

20p Clasa Elev	20p Funcții catalog	25p Fișier save/load	20p Raport statistic	15p Meniu
Recomandare: Echipe de 4-5 elevi. Fiecare elev are o responsabilitate clară (clasă, fișiere, meniu, raport). Se prezintă și se evaluează reciproc.				



Portofoliu

(100 puncte)

La finalul acestui capitol, portofoliul personal al elevului trebuie să cuprindă, în dosarul **Capitol3**, următoarele materiale:

- Dosarele **Fisa9**, **Fisa10**, **Fisa11**, **Fisa12**, **Fisa13** și **Fisa14**, unde fiecare dosar conține fișierele: *raspunsuri.docx*, *FisaAutoevaluare9.docx*, *FisaAutoevaluare10.docx*, *FisaAutoevaluare11.docx*, *FisaAutoevaluare12.docx*, *FisaAutoevaluare13.docx* și *FisaAutoevaluare14.docx*.
- Dosarul **Proiecte3** va conține: *investigheaza3.docx*, *catalog_digital.py*.
- Fișierul *evaluare_sumativa3.docx*.

Modalitate de evaluare:

- Există dosarul **Capitol3** cu structura de directoare specificată – **20p**
- Există în dosare:
 - peste 95% din fișiere – **80p**
 - între 75% și 94% din fișiere – **60p**
 - între 50% și 74% din fișiere – **40p**
 - între 25% și 49% din fișiere – **20p**
 - sub 25% – **0p**

Capitolul 4 – Metode de sortare și eficiența algoritmilor

Lecția 15 – Eficiența algoritmilor: timp de execuție, estimarea operațiilor

„Un program bun nu este doar corect, ci și eficient.”

15.1. Eficiența algoritmilor



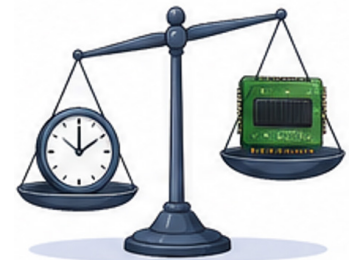
În informatică, rezolvarea corectă a unei probleme reprezintă doar primul pas. La fel de important este modul în care este obținut rezultatul, din punctul de vedere al **timpului** și al **resurselor utilizate**.

Un algoritm **eficient** permite prelucrarea rapidă a unui volum mare de date. În practică, diferențele dintre doi algoritmi pot deveni semnificative atunci când dimensiunea datelor crește. De exemplu, un program care sortează 10 valori poate părea rapid, dar pentru 10 000 de valori diferențele dintre metode devin vizibile.



Timp de execuție și memorie

- **timpul de execuție** – reprezintă durata necesară pentru rularea completă a algoritmului și este influențat de numărul de operații efectuate;
- **memoria utilizată** – reprezintă spațiul de memorie ocupat în timpul execuției algoritmului: variabile, liste și eventuale spații auxiliare.



În acest capitol nu vom folosi notație formală de complexitate. Vom estima numărul de operații prin numărare și prin compararea unor cazuri concrete.



15.2. Estimarea numărului de operații

Evaluarea eficienței nu se realizează doar în funcție de timpul măsurat în secunde, deoarece acesta depinde de calculatorul pe care rulează programul. În schimb, analizăm **numărul de operații** efectuate raportat la **dimensiunea datelor de intrare**, notată de obicei cu ***n***. Nu este nevoie să numărăm fiecare instrucțiune, ci operațiile dominante: comparații, parcurgeri ale listei sau actualizări repetate.

Eficiența unui algoritm este exprimată de obicei prin notația Big O, care arată modul în care crește numărul de operații în funcție de dimensiunea datelor de intrare ***n***. Exemple frecvente sunt: $O(1)$ – timp constant, $O(n)$ – timp liniar, $O(n^2)$ – timp pătratic.



Exemplu – parcurgerea liniară a unei liste

Pentru o listă cu ***n*** elemente, instrucțiunea din interiorul buclei se execută de ***n*** ori. Dacă $n = 8$, avem 8 verificări; dacă $n = 200$, avem 200 de verificări.

```
a = [7, 4, 9, 2, 6]
cnt = 0
for x in a:
    if x % 2 == 0:
        cnt += 1
print(cnt)
```

În exemplul anterior, lista este parcursă o singură dată. Numărul de verificări crește odată cu lungimea listei.

15.3. Caz favorabil, mediu și nefavorabil

În unele probleme, algoritmul poate lucra diferit în funcție de ordinea datelor. De aceea, evaluăm comportamentul pentru mai multe situații:

Caz	Semnificație	Exemplu
<i>favorabil</i>	situația în care algoritmul termină cel mai repede	elementul căutat este primul
<i>mediu</i>	situație obișnuită, fără ordine specială	elementul căutat apare la mijloc
<i>nefavorabil</i>	situația în care algoritmul face cele mai multe operații	elementul nu există în listă

EX.

Exemplu – căutarea unei valori într-o listă

```
a = [4, 7, 2, 9, 5]
x = int(input("x = "))
gasit = False
for v in a:
    if v == x:
        gasit = True
        break
print(gasit)
```

Dacă x este primul element, se face o singură comparație importantă. Dacă x nu se află în listă, se parcurg toate elementele. Instrucțiunea **break** oprește bucla atunci când rezultatul este deja cunoscut.

15.4. Estimarea operațiilor pentru bucle imbricate

 Când o buclă se află în interiorul altei bucle, instrucțiunile din interior se repetă de mai multe ori. Acesta este un reper important pentru înțelegerea metodelor de sortare.

n	for i in range(n)	două bucle imbricate
10	aprox. 10 pași	aprox. 100 pași
100	aprox. 100 pași	aprox. 10 000 pași
200	aprox. 200 pași	aprox. 40 000 pași



Știați că?

Numărul de operații crește diferit în funcție de algoritm:

- O parcurgere simplă a unei liste cu n elemente → aproximativ n operații
- Două bucle imbricate → aproximativ $n \times n$ operații
- Sortarea cu listă de frecvențe → aproximativ $n + k$ operații, unde k este dimensiunea intervalului de valori
- O formulă directă (exemplu: $n \times (n+1)/2$) → același număr de operații indiferent de n



Reține! Un algoritm corect nu este întotdeauna și potrivit pentru un volum mare de date. Înainte de a alege o metodă, estimăm numărul de operații și memoria folosită. În lecțiile următoare vom compara două metode de sortare: **selecția minimului** și **sortarea cu listă de frecvențe**.



Fișa de lucru 15

Creați dosarul numit *Fisa15*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

1) Completați spațiile libere: a) Eficiența unui algoritm se măsoară din punctul de vedere al și al utilizate. b) Completați tabelul cu numărul estimat de operații: c) Cele 3 cazuri de analiză a unui algoritm sunt:, ... și

n	10	100	200	500	1000
0 buclă (n op.)					
2 bucle imbricate (~n² op.)					



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Definiți eficiența unui algoritm din cele două puncte de vedere: timp de execuție și memorie.
ÎNȚELEGERE	2) Explicați de ce pentru $n=200$ două bucle imbricate fac aproximativ 40.000 de operații și nu 400 .
APLICARE	3) Calculați numărul de operații pentru cele două variante de calcul al sumei $1+2+...+n$: varianta cu buclă (n adunări) și varianta cu formula $n*(n+1)/2$ (2 operații). Completați un tabel pentru $n = 10, 100, 1000$.
ANALIZĂ	4) Analizați: pentru o listă cu $n=10$ elemente, algoritmul A face n operații și algoritmul B face $n*(n-1)/2$ operații. Calculați valorile și comentați diferența. Ce se întâmplă pentru $n=1000$? <pre># Algoritmul A - o parcurgere for x in a: print(x)</pre> <pre># Algoritmul B - două bucle imbricate for i in range(n): for j in range(i+1, n): print(a[i], a[j])</pre>
EVALUARE	5) Un elev susține că algoritmul mai scurt (mai puține linii de cod) este întotdeauna mai eficient. Evaluați această afirmație și construiți un contraexemplu.
CREARE	6) Proiectați un experiment: cum ați măsura și compara în Python timpul de execuție al celor doi algoritmi pentru suma $1..n$? Descrieți pașii și scrieți pseudocodul experimentului.

Barem: 1p (din oficiu) + 1p (1) + 1p (2) + 1p (3) + 1p (4) + 2p (5) + 3p (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să definesc eficiența unui algoritm din punct de vedere al timpului și al memoriei.			
2.	să estimez numărul de operații pentru o buclă și pentru două bucle imbricate.			
3.	să compar două variante ale aceluiași algoritm după numărul de operații.			

4. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 16 – Sortarea prin selecția minimului

16.1. Sortarea unei liste



Sortarea reprezintă procesul de aranjare a elementelor unei liste într-o ordine specificată. Este una dintre operațiile fundamentale în informatică: datele ordonate pot fi căutate și prelucrate mult mai eficient. Cele mai frecvente variante sunt **ordine crescătoare** și **ordine descrescătoare**. În această lecție vom sorta crescător folosind selecția minimului.

5 2 8 1

Listă nesortată: [5, 2, 8, 1]

1 2 5 8

Listă sortată crescător: [1, 2, 5, 8]



16.2. Ideea metodei

Scenariul lecției – Concursul de talente. Organizatorii trebuie să aranjeze finaliștii în ordinea punctajelor, de la cel mai mic la cel mai mare. Căutăm scorul cel mai mic din toată lista, îl plasăm pe primul loc, apoi repetăm procedeul pentru restul listei. Aceasta este exact ideea din spatele metodei **selecției minimului**.



Repere pentru aplicarea metodei



Algoritmul efectuează $n - 1$ iterații externe. La fiecare iterație i , de la 0 la $n - 2$, caută elementul minim din subșirul nesortat, de la poziția i la $n - 1$, și îl aduce pe poziția i printr-o singură interschimbare.

Zona sortată	Zona nesortată	Acțiune
vidă	[5, 2, 8, 1]	căutăm minimul
[1]	[2, 8, 5]	căutăm minimul rămas
[1, 2]	[8, 5]	căutăm minimul rămas
[1, 2, 5]	[8]	lista este sortată



La fiecare pas, elementele din stânga poziției curente sunt deja așezate în pozițiile finale.



16.3. Simularea algoritmului pas cu pas

Urmăriți lista $a = [5, 2, 8, 1]$. Căutăm minimul în zona nesortată și îl mutăm pe poziția curentă.

Pas	Lista înainte	Minimul găsit	Operație	Lista după
1	[5, 2, 8, 1]	1	schimb 5 cu 1	[1, 2, 8, 5]
2	[1, 2, 8, 5]	2	rămâne pe loc	[1, 2, 8, 5]
3	[1, 2, 8, 5]	5	schimb 8 cu 5	[1, 2, 5, 8]

Pseudocod

```

pentru i <- 0, n - 2 executa
    poz_min <- i
    pentru j <- i + 1, n - 1 executa
        daca a[j] < a[poz_min] atunci
            poz_min <- j
        sfarsit_daca
    sfarsit_pentru
    interschimba a[i] cu a[poz_min]
sfarsit_pentru

```



De ce se numește selecția minimului?

La fiecare parcurgere se selectează cel mai mic element din partea nesortată. Elementul selectat este așezat pe poziția corectă, iar zona sortată se mărește cu o poziție.

⚠ Atenție! Dacă lista are n elemente, sunt necesare $n - 1$ etape principale. La ultima poziție nu mai avem ce compara.

Activitate ghidată – Concursul de talente

Pentru punctajele [42, 31, 55, 28, 47], aplicați aceeași metodă: alegeți minimul, îl aduceți pe prima poziție liberă, apoi continuați doar în zona rămasă nesortată.

Etapă	Zonă sortată	Zonă nesortată	Minim ales
0	vidă	[42, 31, 55, 28, 47]	-
1	[28]	[31, 55, 42, 47]	28
2	[28, 31]	[55, 42, 47]	31
3	[28, 31, 42]	[55, 47]	42
4	[28, 31, 42, 47]	[55]	47

16.4. Implementare în Python

```
a = [5, 2, 8, 1]
n = len(a)
for i in range(n - 1):
    poz_min = i
    for j in range(i + 1, n):
        if a[j] < a[poz_min]:
            poz_min = j
    a[i], a[poz_min] = a[poz_min], a[i]
print(a)
```

Explicații

- **poz_min** memorează poziția celui mai mic element găsit până în acel moment; reținem poziția, nu valoarea, deoarece avem nevoie să interschimbăm elementele;
- în bucla interioară se caută minimul în zona nesortată;
- la finalul fiecărei etape, elementul minim este mutat pe poziția i .



16.5. Estimarea numărului de comparații

Pentru o listă cu n elemente, la prima etapă comparăm $n - 1$ elemente, apoi $n - 2$, apoi $n - 3$, până la 1 . Numărul total de comparații este:

Etapă	Număr de comparații
1	$n - 1$
2	$n - 2$
3	$n - 3$
...	...
ultima	1

Formula intuitivă: $(n - 1) + (n - 2) + \dots + 1 = n \cdot (n - 1) / 2$.

Pentru $n = 4$, avem $3 + 2 + 1 = 6$ comparații. Pentru $n = 200$, numărul este aproape **20 000 de comparații**.

Teste recomandate

- listă deja sortată: [1, 2, 5, 8];
- listă invers sortată: [8, 5, 2, 1];
- listă cu valori egale: [4, 4, 4];
- listă cu un singur element: [7].



Fișa de lucru 16

Creați dosarul numit *Fisa16*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

1) Pentru lista [6, 1, 4, 2], aplicați sortarea prin selecția minimului și scrieți lista după fiecare etapă.



2) Câte comparații face sortarea prin selecția minimului pentru $n = 5$? Scrieți suma etapă cu etapă (câte comparații se fac la fiecare pas) și calculați totalul.

3) Analizați comportamentul algoritmului pentru cazurile speciale:

Lista de intrare	Comparații	Schimburi	Observație
[1, 2, 3, 4, 5] (sortata)			
[5, 4, 3, 2, 1] (inversata)			
[3, 3, 3, 3] (valori egale)			
[7] (un singur element)			



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Descrieți în 3-4 propoziții ideea metodei de sortare prin selecția minimului. Ce se întâmplă la fiecare pas?
ÎNȚELEGERE	2) De ce algoritmul face exact $n-1$ pași externi și nu n ? Explicați, făcând referire la zona sortată și zona nesortată.
APLICARE	3) Aplicați sortarea prin selecția minimului pe lista [37, 25, 83, 18, 26] și completați un tabel cu zona sortată, zona nesortată și minimul ales la fiecare pas.
ANALIZĂ	4) Analizați: pentru o listă cu $n=8$ elemente, numărul de comparații este întotdeauna același indiferent de ordinea inițială? Justificați și calculați.
EVALUARE	5) Evaluați următoarea implementare a selecției minimului: în loc de a memora poziția minimului (poz_min), un elev memorează valoarea (val_min). Ce problemă apare la interschimbare? Cum ar corecta?
CREARE	6) Scrieți un program complet care citește n punctaje ale participanților la un concurs și le afișează sortate descrescător folosind selecția maximului. Afișați și clasamentul (locul 1, 2, ...).

```

a = [5, 2, 8, 1]
n = len(a)
for i in range(n - 1):
    val_min = a[i]
    for j in range(i + 1, n):
        if a[j] < val_min:
            val_min = a[j]
    a[i] = val_min
print(a)

```

Barem: 1p (din oficiu) + 1p (1) + 1p (2) + 1p (3) + 1p (4) + 2p (5) + 3p (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să descriu metoda de sortare prin selecția minimului.			
2.	să aplic algoritmul pas cu pas pe o listă dată (tabel cu zone sortată/nesortată).			
3.	să implementez sortarea cu selecția minimului în Python.			
4.	să calculez numărul de comparații folosind formula $n*(n-1)/2$.			
5.	Am întâmpinat dificultăți în: ...			

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Lecția 17 – Sortarea cu listă de frecvențe

17.1. Când putem folosi o listă de frecvențe?



O listă de frecvență asociază fiecărei valori distincte numărul de apariții ale acesteia într-un șir. Dacă valorile sunt numere întregi dintr-un interval cunoscut și mic, putem sorta fără să comparăm elementele între ele. Această tehnică stă la baza sortării cu *listă de frecvențe*.



Exemple de date potrivite: note între 1 și 10, niveluri de urgență între 1 și 5, coduri de defecțiune între 1 și 8, răspunsuri la un chestionar între 1 și 5. Datele brute sunt transformate într-o reprezentare compactă și ușor de interpretat.

Scenariul lecției – Catalogul de note.

7	10	8	7	5	10	8	7
---	----	---	---	---	----	---	---

Profesorul vrea să aranjeze notele elevilor în ordine crescătoare. În loc să compare notele între ele, trece prin catalog o singură dată și bifează căsuța corespunzătoare fiecărei note. Apoi rescrie catalogul, menționând fiecare notă de atâtea ori cât indică frecvența.



17.2. Pașii algoritmului

Pas	Descriere
1	creăm o listă de frecvențe de dimensiune fixă, potrivită intervalului valorilor posibile
2	parcurgem setul de date, element cu element, și incrementăm frecvența la indexul egal cu valoarea citită
3	reconstruim lista sortată parcurgând frecvențele în ordine și adăugând fiecare valoare de câte ori apare



Reține! Poziția din lista de frecvențe reprezintă valoarea numărată, iar conținutul acelei poziții reprezintă numărul de apariții. Pentru note 1..10 folosim `frecv[nota]`.



17.3. Construirea listei de frecvențe

```
note = [7, 10, 8, 7, 5, 10, 8, 7]
frecv = [0] * 11
for nota in note:
    frecv[nota] += 1
print(frecv)
```



Pentru datele de mai sus, valorile nenule din lista de frecvențe sunt:

Nota	1	2	3	4	5	6	7	8	9	10
Frecv.	0	0	0	0	1	0	3	2	0	2

17.4. Reconstruirea listei sortate

```
sortate = []
for nota in range(1, 11):
    for k in range(frecv[nota]):
        sortate.append(nota)
print(sortate)
```

5	7	7	7	8	8	10	10
---	---	---	---	---	---	----	----

Explicații

- prima buclă parcurge valorile posibile, de la 1 la 10;
- a doua buclă adaugă valoarea în lista sortată de câte ori a apărut;
- nu comparăm notele între ele, ci folosim direct numărul aparițiilor; ordinea inițială a datelor nu se mai păstrează, dar obținem rapid lista sortată.



Dacă notele sunt introduse de la tastatură una câte una, nu este obligatoriu să salvăm toate notele într-o listă. Putem actualiza direct lista de frecvență pe măsură ce citim datele.

```
frecv = [0] * 11
n = int(input("Număr de note: "))
for i in range(n):
    nota = int(input("nota = "))
    frecv[nota] += 1
```



17.5. Sortare descrescătoare și aplicații

Pentru ordine descrescătoare, parcurgem valorile posibile de la cea mai mare la cea mai mică.

```
sortate = []
for nota in range(10, 0, -1):
    for k in range(frecv[nota]):
        sortate.append(nota)
print(sortate)
```



Exemplu practic – triaj medical

Într-un sistem de triaj, gravitatea poate fi codificată folosind valori de la 1 la 5. Pentru un număr mare de pacienți, sortarea cu frecvențe este potrivită deoarece intervalul este limitat, iar frecvențele se calculează rapid.



17.6. Avantajele și dezavantajele sortării cu listă de frecvențe

Avantaje	Dezavantaje
viteză bună de procesare pentru volum mare de date, dacă intervalul este mic	nu se potrivește pentru valori reale, cum ar fi 7.5 sau 9.25
ușor de aplicat pentru note, scoruri, coduri	nu este eficientă dacă intervalul este foarte mare
nu face comparații între elementele listei și permite identificarea ușoară a frecvențelor	folosește memorie suplimentară pentru frecvențe

Comparație între cele două metode

Metodă	Când este potrivită?	Observație
selecția minimului	liste generale, valori oarecare	nu cere interval limitat
listă de frecvențe	valori întregi din interval mic	foarte bună pentru volum mare de date

Convenție de numărare: numărăm toate operațiile – incrementări, iterații și adăugări. Lista de frecvențe face aproximativ $2n+k$ operații, unde n este numărul de elemente și k este dimensiunea intervalului de valori. Selecția minimului face $n \times (n-1)/2$ comparații – de exemplu, pentru $n=1000$: lista de frecvențe $\rightarrow 2 \times 1000 + 10 = 2.010$ operații, selecția minimului $\rightarrow 1000 \times 999/2 = 499.500$ comparații.



Reține! Alegerea metodei depinde de date. Dacă avem valori generale, folosim o metodă de sortare prin comparații. Dacă valorile sunt întregi și intervalul este limitat, lista de frecvențe poate fi mult mai potrivită.



Fișa de lucru 17

Creați dosarul numit *Fisa17*. Rezolvați exercițiile următoare, notând răspunsurile corecte într-un fișier *raspunsuri.docx*.

- 1) Construiți lista de frecvențe pentru notele: [8, 7, 10, 8, 6, 7, 8]. Valorile posibile sunt 1..10.
- 2) Scrieți secvența Python care reconstruiește lista sortată crescător folosind lista de frecvențe.
- 3) Alegeți metoda potrivită și justificați: sortarea a 2 000 de răspunsuri la un chestionar, fiecare răspuns fiind o valoare între 1 și 5.
- 4) Precizați un caz în care sortarea cu listă de frecvențe nu este potrivită.



Autoevaluare! Ce am învățat?

[CUNOAȘTERE] [ÎNȚELEGERE] [APLICARE] [ANALIZĂ] [EVALUARE] [CREARE]

CUNOAȘTERE	1) Descrieți cei 3 pași ai algoritmului de sortare cu listă de frecvențe. Când este potrivit să îl folosim?
ÎNȚELEGERE	2) Explicați de ce poziția dintr-o listă de frecvențe reprezintă valoarea, iar conținutul reprezintă numărul de apariții. Folosiți ca exemplu notele [7, 8, 7, 10].
APLICARE	3) Scrieți codul Python complet pentru sortarea crescătoare a vârstelor participanților la un concurs. Sunt n participanți, iar vârstele sunt numere întregi cuprinse între 10 și 18 ani.
ANALIZĂ	4) Analizați: de ce sortarea cu lista de frecvențe nu face comparații între elementele listei? Ce face în schimb? Este aceasta o limitare sau un avantaj?
EVALUARE	5) Argumentați decizia: un sistem de gestionare a solicitărilor de mentenanță pentru o companie de lifturi trebuie să sorteze cererile după tipul defecțiunii (coduri 1-8). Justificați alegerea metodei cu lista de frecvențe comparativ cu selecția minimului.
CREARE	6) Proiectați un program complet pentru gestionarea solicitărilor de mentenanță ale companiei de lifturi: citește n cereri (fiecare cu un cod 1-8), sortează după urgență (8=urgent), afișează lista ordonată și un raport cu numărul de cereri din fiecare tip.

Barem: $1p$ (din oficiu) + $1p$ (1) + $1p$ (2) + $1p$ (3) + $1p$ (4) + $2p$ (5) + $3p$ (6)

	La finalul lecției, știu...	Da	Nu	Nu sunt sigur(ă)
1.	să descriu cei 3 pași ai algoritmului cu lista de frecvențe.			
2.	să construiesc lista de frecvențe și să reconstruiesc lista sortată.			
3.	să implementez în Python sortarea cu lista de frecvențe.			
4.	să identific situațiile potrivite pentru fiecare metodă (selecția minimului vs. frecvențe).			
5.	să argumentez alegerea metodei de sortare în funcție de caracteristicile datelor.			

6. Am întâmpinat dificultăți în: ...

Nu uitați să vă evaluați comportamentul și activitatea din timpul lecției!
(cu ajutorul fișei de observare a comportamentului și a activității, de la pagina 7)

La finalul lecției, m-am simțit:



Evaluare sumativă - Capitolul 4

Rezolvați exercițiile într-un document: *evaluare_sumativa4.docx*.

- 1) (0.5p) Alegeți varianta corectă:
 Selecția minimului pe $n=10$ face:
 a) 10 pași; b) 45 pași; c) 100 pași; d) 5 pași.
 Lista de frecvențe este potrivită când:
 a) Valorile sunt reale; b) Valorile sunt în interval limitat; c) Lista e deja sortată; d) $n < 10$.

- 2) (2p) Simulați sortarea prin selecția minimului pe $a = [5, 2, 8, 1, 4]$:

Pas	Lista înainte	Min (poz.)	Schimb	Lista după
0	[5, 2, 8, 1, 4]			
1				
2				
3				

Nr. total comparații (formula):

- 3) (1.5p) Completați algoritmul de sortare cu lista de frecvențe pentru note (1-10) și răspundeți:

```

note = [8, 7, 10, 8, 6, 7, 8]
frecv = [0] * ____
for nota in note:
    frecv[____] += 1
sortat = []
for v in range(1, ____):
    for k in range(frecv[v]):
        sortat.append(v)
print(sortat)

```

Afișează:

Câte operații face? (nu comparații între elemente!)

De ce nu face comparații între note?

- 4) (1.5p) Comparați eficiența celor două metode pentru $n=1000$ note cu valori 1-10:

Criteriu	Selecția minimului	Lista de frecvențe
Nr. operații (estimare)	~500.000	~2020
Memorie suplimentară		
Funcționează pt. valori reale?		

- 5) (1.5p) **Evaluati:** Un sistem de triaj medical primește **2000** de pacienți cu urgență codificată **1 – 5**. Un elev alege selecția minimului. Argumentați dacă alegerea este corectă sau nu și propuneți o alternativă.
Alegerea e corectă?
Argumentare (operații comparate):
Alternativa recomandată și motivul

- 6) (2p) **Scrieți programul Python** pentru un sistem de gestionare a solicitărilor de mentenanță: citește n cereri cu cod **1 – 6**, sortează descrescător (**6**=urgent), afișează lista și numărul de cereri de fiecare fel.

Barem de corectare: Se acordă **1 punct** din oficiu!**1) 0.5 puncte. 2) 2 puncte** (0.3p/rând tabel + 0.2p/formulă). **3) 1.5 puncte. 4) 1.5 puncte** (0.5p/rând).**5) 1.5 puncte. 6) 2 puncte** (1p sortare + 1p raport).



Investighează!

(100 puncte)

Realizați, într-un fișier **investigheaza4.docx**, o investigație comparativă între sortarea prin selecția minimului și sortarea cu lista de frecvențe.

- Implementați ambii algoritmi de sortare în Python. Testați-i pe aceeași listă cu **10** elemente și verificați dacă produc același rezultat. **(20p)**
- Numărați operațiile (comparații/incrementări) pentru fiecare algoritm pe liste cu $n=5$, $n=10$, $n=20$ elemente. Completați un tabel comparativ și formulați o concluzie. **(30p)**
- Identificați pentru fiecare algoritm: un scenariu în care este recomandat și unul în care NU este recomandat. Justificați, făcând referire la tipul și intervalul datelor. **(25p)**
- Investigați: Ce se întâmplă dacă aplicați sortarea cu lista de frecvențe pe note reale (Exemplu: 7.5, 8.25)? Explicați limitarea și propuneți o soluție. **(15p)**
- Concluzie finală: Pe baza investigației, când alegeți fiecare metodă? (5-6 rânduri) **(10p)**

20p	30p	25p	15p	10p
Implementare + test	Tabel operații	Scenarii recomandate/nerecomandate	Limitare frecvențe	Concluzie
Recomandare: Se lucrează individual. Fișierul se salvează în dosarul Capitol4. Se includ tabele cu numărul de operații.				



Proiect

(100 puncte)

Realizați, în echipă, o aplicație Python **festival.py** (o salvați în dosarul **Capitol4**) pentru gestionarea și sortarea solicitărilor primite de organizatorii unui festival școlar, care:

- Citește n solicitări, fiecare cu: ID (număr), tip solicitare (cod **1-8**), descriere (text). **(15p)**
- Determină și afișează numărul de solicitări pentru fiecare cod (**1-8**) folosind lista de frecvențe. Afișează codurile în ordine descrescătoare împreună cu frecvența lor. **(25p)**
- Raportează numărul de solicitări pentru fiecare tip (**1-8**) și tipul cu cele mai multe solicitări. **(20p)**
- Salvează solicitările sortate într-un fișier **solicitari.txt** și le încarcă la următoarea rulare. **(20p)**
- Afișează numărul de operații folosite de lista de frecvențe vs. selecția minimului pentru același n . **(20p)**

15p Citire cereri	25p Sortare frecvențe	20p Raport	20p Fișier save/load	20p Comparatie eficiență
Recomandare: Echipa de 3-4 elevi. Se recomandă testarea cu $n=50$ cereri generate aleator. Prezentare în fața clasei cu demonstrație live.				



Portofoliu

(100 puncte)

La finalul acestui capitol, portofoliul personal al elevului trebuie să cuprindă, în dosarul **Capitol4**, următoarele materiale:

- Dosarele **Fisa15**, **Fisa16** și **Fisa17**, unde fiecare dosar conține fișierele: **raspunsuri.docx**, **FisaAutoevaluare15.docx**, **FisaAutoevaluare16.docx** și **FisaAutoevaluare17.docx**.
- Dosarul **Proiecte4** va conține: **investigheaza4.docx**, **festival.py**.
- Fișierul **evaluare_sumativa4.docx**.

Modalitate de evaluare:

- Există dosarul **Capitol4** cu structura de directoare specificată – **20p**
- Există în dosare:
 - peste 95% din fișiere – **80p**
 - între 75% și 94% din fișiere – **60p**
 - între 50% și 74% din fișiere – **40p**
 - între 25% și 49% din fișiere – **20p**
 - sub 25% – **0p**

Evaluare finală

Creați dosarul numit *evaluare_finala*. Rezolvați exercițiile într-un document: *raspunsuri.docx*.

1) (2p) Asociați elementele de pe cele două coloane:

- | | |
|---------------------|--|
| 1. Eroare de logică | A. 4 |
| 2. Variabila locală | B. 2 |
| 3. CMMDC(12,8) | C. Program rulează, dar rezultatul este greșit |
| 4. round(2.5) | D. Există doar în funcție |

2) (1p) Calculați valorile:

sum([3,7,2,8])	max([3,7,2,8])	int(-3.9)	[1,2]+[3,4]

Ce afișează: `L=[1,2,3]; L2=L.copy(); L2.append(4); print(L) → ....`

3) (1.5p) Urmăriți execuția pentru $n = 84$ (factorizare):

```
d = 2
while d * d <= n:
    p = 0
    while n % d == 0:
        n //= d
        p += 1
    if p > 0:
        print(d, p)
    d += 1
if n > 1:
    print(n, 1)
```

Completați ce se afișează:
 Prima linie afișată:
 A doua linie:
 A treia linie:
 Ce factori primi are 84?.....

4) (1p) Scrieți o funcție *calculeaza_medie_sigura(note)* care verifică dacă lista nu este goală (returnează None dacă este goală) și altfel returnează media rotunjită la 2 zecimale. Apelați-o cu [7,8,9,10] și cu [] și salvați capturi de ecran din care să reiasă apelurile.

5) (1.5p) Scrieți un program Python care citește n note (numere întregi) de la tastatură și afișează media lor.

6) (2p) *Mini-proiect*: Scrieți un program Python complet care citește numărul de pași realizați zilnic dintr-un fișier numit *pasi.txt* (o valoare pe linie), calculează frecvența fiecărei valori, afișează valoarea cu frecvența maximă și scrie un raport în fișierul *raport.txt*. Raportul va conține: valoarea cu numărul maxim de pași înregistrat cel mai des, frecvența acesteia (de câte ori a apărut în fișier), precum și numărul total de zile analizate. Tratați situația în care fișierul este inexistent.

Barem de corectare: Se acordă **1 punct** din oficiu!

1) 2 puncte (0.5p/asociere). **2) 1 punct** – $0.25p \times 4$. **3) 1.5 puncte**. **4) 1 punct**. **5) 1.5 puncte**. **6) 2 puncte** (0.5p citire + 0.5p frecvențe + 0.5p raport + 0.5p erori).

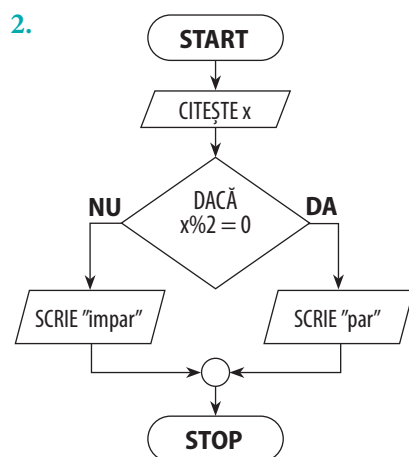
INDICAȚII ȘI RĂSPUNSURI

Capitolul 1

Fișa de lucru 1: 1. 1a-a); 1b-c); 1c-c), 2. a – Analiza problemei → Proiectarea algoritmului → Implementarea (Codificarea / Scrierea programului) → Testarea și depanarea → Întreținerea (Documentarea și livrarea); b – clar, finit, general, rezultat; c – normale, la limită, eronate. 3. temperatura, mesaj, nu sunt, temperatura >37, t=38 => "Pacientul are febră". 4. a – eroare logică – rezultatul afișat va fi greșit; b – eroare de sintaxă – lipsă atunci, sf.dacă din structura decizională. 5. a – Nu, nu sunt de acord. Faptul că un program compilează și rulează demonstrează doar că acesta nu are erori de **sintaxă**. Totuși, testarea este absolut necesară deoarece programul poate conține erori de **logică**; b – exemplu programul de la 4a, calculează media aritmetică greșit.

Fișa de lucru 2:

1. algoritm maxim_doua_numere
întreg a, b, max
citește a, b
dacă a > b atunci
 max ← a
altfel
 max ← b
sfârșit_dacă
scrie max
sfârșit_algoritm



3. 1, n+1, suma+=x, suma

Fișa de lucru 3: 1. 15

2. lungime = float(input("Introdu lungimea: "))
latime = float(input("Introdu lățimea: "))
aria = lungime * latime
print("Aria este:", aria)

3.

```
n = int(input("Introdu n: "))
suma = 0
for i in range(1, n + 1):
    suma = suma + i
print("Suma de la 1 la", n, "este:", suma)
```

Fișa de lucru 4:

1.

```
def cel_mai_mare(a, b, c):
    if a >= b and a >= c:
        return a
    elif b >= a and b >= c:
        return b
    else:
        return c
```

2. c). 3. globale: tva, reducere, rezultat, locale: pret, cantitate, subtotal, total.

Fișa de lucru 5:

1. 1a-a); 1b-c); 1c-c).

2.

```
n = int(input("Introduceți numărul de prețuri (n): "))
preturi = []
for i in range(n):
    pret = float(input(f"Introduceți prețul {i + 1}: "))
    preturi.append(pret)
if n > 0:
    p_min = min(preturi)
    p_max = max(preturi)
    s_total = sum(preturi)
    media = s_total / len(preturi)
    print("\n--- Rezultate ---")
    print(f"Prețul minim este: {p_min}")
    print(f"Prețul maxim este: {p_max}")
    print(f"Suma totală este: {s_total}")
    print(f"Media prețurilor este: {round(media, 2)}")
else:
    print("Nu au fost introduse prețuri.")
```

3.

```
import math
def distanta(x1, y1, x2, y2):
    dist_p = (x2 - x1)**2 + (y2 - y1)**2
    rezultat = math.sqrt(dist_p)
    return rezultat

# --- Exemplu de utilizare în Programul Principal ---
# Punctul A(1, 2) și Punctul B(4, 6)
d = distanta(1, 2, 4, 6)
print(f"Distanța dintre puncte este: {d}")
# Distanța va fi 5.0
```

Capitolul 2

Fișa de lucru 6:

- ```
n = int(input("n = "))
s = 0
nr = 0
if n == 0:
 s = 0
 nr = 1
else:
 temp = n
 while temp > 0:
 c = temp % 10
 s += c
 nr += 1
 temp //= 10
print("suma =", s)
print("cifre =", nr)
```
- ```
n=456
nr_cifre = 0
while n>0:
    c = n % 10
    if c % 3 == 0:
        nr_cifre = nr_cifre + 1
    n //= 10
print(nr_cifre)
```
- ```
n = int(input("n = "))
contor = 0
temp = n
while temp > 9:
 c_urmatoare = (temp // 10) % 10
 if c_urmatoare == 0:
 contor += 1
 temp //= 10
print(contor)
```
- ```
n = int(input("n = "))
k = int(input("k = "))
contor = 0
if n == 0:
    if k == 0:
        contor = 1
else:
    temp = n
    while temp > 0:
        c = temp % 10
        if c == k:
            contor += 1
        temp //= 10
print(contor)
```
- ```
n = int(input("n = "))
rezultat = 0
pozitie = 1
temp = n
while temp > 0:
 c = temp % 10
 if c % 2 == 0:
 rezultat = c * pozitie + rezultat
 pozitie *= 10
 temp //= 10
print(rezultat)
```

### Fișa de lucru 7:

- ```
n = int(input("n = "))
d = 1
while d <= n:
    if n % d == 0:
        print(d)
    d += 1
```
- ```
n = int(input("n = "))
nr_div = 0
cmmp = 1
suma_impairi = 0
d = 1
while d * d <= n:
 if n % d == 0:
 nr_div += 1
 if d != n:
 if d % 2 != 0:
 suma_impairi += d
 if d > cmmp and d != n:
 cmmp = d
 if d != n // d:
 mare = n // d
 nr_div += 1
 if mare != n:
 if mare % 2 != 0:
 suma_impairi += mare
 if mare > cmmp:
 cmmp = mare
 d += 1
print("Numar divizori:", nr_div)
print("Cel mai mare divizor propriu:", cmmp)
print("Suma divizorilor impari:", suma_impairi)
```
- ```
L = int(input("L = "))
d = 1
while d * d <= L:
    if L % d == 0:
        if L // d >= 3:
            print("lungime bucata:", d, "|",
                  "nr bucati:", L // d)
            mare = L // d
            if d != mare and L // mare >= 3:
                print("lungime bucata:", mare,
                      "| nr bucati:", L // mare)
    d += 1
```
- ```
n = int(input("n = "))
gasit = False
elevi = 2
while elevi <= 10:
 if n % elevi == 0:
 print("elevi:", elevi, "| caiete/ elev:", n // elevi)
 gasit = True
 elevi += 1
if not gasit:
 print("Nu exista nicio varianta de distributie.")
```

```

5. k = int(input("k = "))
gasit = False
carti_per_clasa = 5
while carti_per_clasa <= 20:
 if k % carti_per_clasa == 0:
 nr_clase = k // carti_per_clasa
 print("clase:", nr_clase, "| carti/
clasa:", carti_per_clasa)
 gasit = True
 carti_per_clasa += 1
if not gasit:
 print("Nu exista nicio distributie
posibila.")

```

### Fișa de lucru 8:

```

1. a = int(input("a = "))
b = int(input("b = "))
descompunerea lui a
print(a, "=", end=" ")
temp = a
d = 2
primul = True
while d * d <= temp:
 if temp % d == 0:
 putere = 0
 while temp % d == 0:
 putere += 1
 temp //= d
 if not primul:
 print("*", end=" ")
 if putere == 1:
 print(d, end=" ")
 else:
 print(str(d) + "^" + str(putere), end=" ")
 primul = False
 d += 1
 if temp > 1:
 if not primul:
 print("*", end=" ")
 print(temp, end=" ")
print()
descompunerea lui b
print(b, "=", end=" ")
temp = b
d = 2
primul = True
while d * d <= temp:
 if temp % d == 0:
 putere = 0
 while temp % d == 0:
 putere += 1
 temp //= d
 if not primul:
 print("*", end=" ")
 if putere == 1:
 print(d, end=" ")
 else:
 print(str(d) + "^" + str(putere), end=" ")
 primul = False
 d += 1

```

```

if temp > 1:
 if not primul:
 print("*", end=" ")
 print(temp, end=" ")
print()

```

```

2. a = int(input("a = "))
b = int(input("b = "))
x = a
y = b
while y > 0:
 r = x % y
 x = y
 y = r
cmmdc = x
if cmmdc == 1:
 print(a, "si", b, "sunt prime intre ele")
else:
 print(a, "si", b, "nu sunt prime intre
ele")
print("CMMDC =", cmmdc)

```

```

3. caiete = 120
pixuri = 96
a = caiete
b = pixuri
while b > 0:
 r = a % b
 a = b
 b = r
nr_pachete = a
print("Numarul maxim de pachete:", nr_pachete)
print("Caiete per pachet:", caiete // nr_pachete)
print("Pixuri per pachet:", pixuri // nr_pachete)

```

```

4. lat1 = 15
lat2 = 20
a = lat1
b = lat2
ca = a
cb = b
while b > 0:
 r = a % b
 a = b
 b = r
cmmdc = a
cmmmc = ca * cb // cmmdc
print("Latura patraturului:", cmmmc, "cm")
aria_patrat = cmmmc * cmmmc
aria_dr = lat1 * lat2
nr_dr = aria_patrat // aria_dr
print("Numar de dreptunghiuri:", nr_dr)

```

```

5. tural1 = 12
 tura2 = 15
 a = tural1
 b = tura2
 ca = a
 cb = b
 while b > 0:
 r = a % b
 a = b
 b = r
 crmdc = a
 minute = ca * cb // crmdc
 print("Se intalnesc dupa", minute, "minute")
 ora_start = 9
 minute_start = 0
 minute_total = minute_start + minute
 ora_final = ora_start + minute_total // 60
 minute_final = minute_total % 60
 if minute_final < 10:
 print("Ora intalnirii: "+str(ora_final)
 +":0" + str(minute_final))
 else:
 print("Ora intalnirii: " + str(ora_final)
 +":" + str(minute_final))

```

## Capitolul 3

### Fișa de lucru 9:

1. b). 2. a) 0, -1; b) direct, secvențial; c) vidă. 3. L[0] → 4, L[2] → 15, L[-1] → 23, L[-2] → 16, len(L) → 5; 4. localizezi imediat un element cunoscând poziția sa, fără a parcurge celelalte; parcurgi elementele unul după altul, de la început, până găsești ce cauți; o bibliotecă cu rafturi numerotate – dacă știi că o carte e la raftul 5 (acces direct) vs. a căuta cartea raft cu raft de la primul (acces secvențial);

5.

|   |   |   |   |   |   |   |   |   |    |
|---|---|---|---|---|---|---|---|---|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 0 | 0 | 0 | 0 | 1 | 3 | 2 | 0 | 1  |

Nota cu cea mai mare frecvență: 7 (apare de 3 ori)

Numărul de elevi cu nota >= 5: 7 (toți)

Numărul de elevi cu nota maximă (10): 1

```

6. L = [3, 7, 2, 9, 5]
 print(L)
 print("Suma:", sum(L))
 print("Numar elemente:", len(L))

```

7. NU – Nu trebuie să păstrăm numărul – verificăm și afișăm imediat; NU – Păstrăm doar un max\_curent actualizat la fiecare pas – nu avem nevoie de toată lista; DA – trebuie să păstrăm toate notele pentru a le parcurge de la sfârșit la început; NU – Adunăm pe parcurs într-o variabilă sumă – nu e nevoie să memorăm fiecare număr

### Fișa de lucru 10:

1. L.append(x), L.insert(i, x), L.pop(i), L.remove(x), L.index(x), L.clear().

2. L[2]: 4; L[-1]: 9; L[1:4]: [1, 4, 1]; L[::-1]: [9, 5, 1, 4, 1, 3]; print(L): [3, 1, 4, 1, 5, 9]; len(L): 6.

3. # situatia 1 # → ["alb", "verde", "alb", "verde"]

# situatia 2 # sterge elementul de la indexul 1 → 20 # → [10, 30, 40]

4. L[2]: "ema"; L[-1]: "mihai"; L[1:4]: ["dan", "ema", "mihai"]; L[: -1]: ["mihai", "ema", "dan", "ana"]; print(L): # ["ana", "dan", "ema", "mihai"]; print(len(L)): # 4

```

5. total = depozit1 + depozit2;
 total.append("struguri");
 if "banane" in total;
 poz = total.index("pere");
 nr = total.count("mere").

```

### Fișa de lucru 11:

1. b). 2. alias; copie; conținut; același obiect. 3. # [1, 2, 3]; # [1, 2, 3, 4]; # False; # True; [3, 5, -6, 7, -8, 9]. 4. ["Mara", "Alin", "Ana", "Ema", "Teo"]; ["Alin", "Ana", "Ema", "Mara", "Teo"]; ["Teo", "Ema", "Ana", "Alin", "Mara"]; ["Teo", "Ema", "Ana", "Alin", "Mara"]; ["Ema", "Alin"]; ["Teo", "Ema", "Ana"].

5. a) # L = [1, 4, 9, 16, 25];

```

b) n = int(input())
 L = []
 for i in range(1, n + 1):
 L.append(i ** 2)

 print(L)

```

c) L = [i\*\*2 for i in range(1, n + 1)]  
if i % 2 != 0 # pentru n=5 → [1, 9, 25]

### Fișa de lucru 12:

1. Clasa = șablonul, tiparul. Definește structura și comportamentul. Obiectul = instanța concretă creată din șablon. Analogie: clasa este rețeta unui tort – descrie ingredientele și pașii. Obiectul este tortul propriu-zis – creat după rețetă, cu valori concrete (ciocolată, 3 ouă etc.). Din aceeași rețetă poți face oricâte torturi diferite.

```

2. class Caine:
 def __init__(self, nume):
 self.nume = nume
 caine1 = Caine("Rex")
 caine2 = Caine("Azor")
 print(caine1.nume) # Rex
 print(caine2.nume) # Azor

```

```

3. class Elev:
 def __init__(self, nume, medie):
 self.nume = nume
 self.medie = medie
 def este_premiant(self):
 return self.medie >= 9
 e1 = Elev("Ana", 9.5)
 e2 = Elev("Ion", 7.2)
 print(e1.este_premiant()) # True
 print(e2.este_premiant()) # False

```

```

4. class Masina:
 def __init__(self, marca, model, an_fabricatie):
 self.marca = marca
 self.model = model
 self.an_fabricatie = an_fabricatie

```

```

def varsta(self):
 return 2026 - self.an_fabricatie
def descriere(self):
 print("Marca:", self.marca)
 print("Model:", self.model)
 print("An fabricatie:", self.an_fabricatie)
 print("Varsta:", self.varsta(),
 "ani")
m = Masina("Dacia", "Logan", 2018)
m.descriere()
Marca: Dacia
Model: Logan
An fabricatie: 2018
Varsta: 8 ani

```

```

5. m1 = Masina("Dacia", "Logan", 2018)
m2 = Masina("Volkswagen", "Golf", 2020)
m3 = Masina("Toyota", "Corolla", 2015)
print(m1.marca) # Dacia
print(m2.model) # Golf
print(m3.an_fabricatie) # 2015
print(m1.varsta()) # 8
m2.descriere()
m1.an_fabricatie = 2022
print(m1.an_fabricatie) # 2022 - modificat
print(m2.an_fabricatie) # 2020 - nemodificat
print(m3.an_fabricatie) # 2015 - nemodificat

```

#### Fișa de lucru 13:

```

1. # Pasul 1 - scriere
with open("jurnal.txt", "w") as f:
 for i in range(3):
 act = input(f"Activitate {i+1}: ")
 f.write(act + "\n")
Pasul 2 - citire si afisare numerotata
with open("jurnal.txt", "r") as f:
 for nr, linie in enumerate(f, 1):
 print(str(nr) + ". " + linie.strip())

```

Activitate 1: matematica

Activitate 2: informatica

Activitate 3: sport

1. matematica

2. informatica

3. sport

2. # 5; # 10; # 8.0; int(linie.strip()) generează ValueError;

Pentru o linie goală protejăm codul cu:

```

with open("note.txt", "r") as f:
 note = []
 for linie in f:
 if linie.strip():
 note.append(int(linie.strip()))

```

#### Fișa de lucru 14:

1. depinde de termenii anteriori. 2. de ordin 2.  
3. a) 2, 5, 8, 11, 14, 17, ...; b) 5, 10, 20, 40, 80, 160, ...; c) 1, 4, 5, 9, 14, 23, ... . 4. a) 21, 28, 36; b) 1+2=3, 3+3=6, 6+4=10, 10+5=15, 15+6=21 ...; c) recurent. 5. a) 10, 8, 6, 4, 2, 0; b) recurență de ordinul 1.

```

6. n = int(input("n = "))
termen = 1
for i in range(n):
 print(termen)
 termen += 2
7. n = int(input("n = "))
putere = 1
for i in range(n):
 print(putere)
 putere *= 3

```

## Capitolul 4

Fișa de lucru 15: 1. a) timpului de execuție, al memoriei;

| n                                       | 10  | 100    | 200    | 500     | 1000      |
|-----------------------------------------|-----|--------|--------|---------|-----------|
| O buclă (n op.)                         | 10  | 100    | 200    | 500     | 1000      |
| 2 bucle imbricate (~n <sup>2</sup> op.) | 100 | 10 000 | 40 000 | 250 000 | 1 000 000 |

c) favorabil, mediu și nefavorabil.

#### Fișa de lucru 16:

| 1. Pas | Lista înainte | Minimul găsit | Operație      | Lista după   |
|--------|---------------|---------------|---------------|--------------|
| 1      | [6, 1, 4, 2]  | 1             | schimb 6 cu 1 | [1, 6, 4, 2] |
| 2      | [1, 6, 4, 2]  | 2             | schimb 6 cu 2 | [1, 2, 4, 6] |
| 3      | [1, 2, 4, 6]  | 4             | rămâne pe loc | [1, 2, 4, 6] |

2. Numărul de comparații pentru n = 5

(n-1) + (n-2) + (n-3) + (n-4) = 4 + 3 + 2 + 1 = 10 comparații

Formula:  $n \times (n-1) / 2 = 5 \times 4 / 2 = 10$  comparații

#### Fișa de lucru 17:

| 1. Nota   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-----------|---|---|---|---|---|---|---|---|---|----|
| Frecvența | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 0 | 1  |

```

2. sortate = []
for nota in range(1, 11):
 for k in range(frecv[nota]):
 sortate.append(nota)
print(sortate)
→ [6, 7, 7, 8, 8, 8, 10]

```

3. Metoda potrivită: sortarea cu listă de frecvențe.

Justificare:

- valorile sunt **întregi** între 1 și 5 – interval mic și cunoscut;
- numărul de date este **mare** (2000) – lista de frecvențe este mult mai rapidă decât selecția minimului;
- selecția minimului ar face  $2000 \times 1999 / 2 \approx 2\,000\,000$  **comparații**, față de doar **2005 operații** cu lista de frecvențe.

4. Sortarea cu listă de frecvențe **nu este potrivită** când:

- valorile sunt **numere reale** – exemplu: note de tipul 7.5, 8.25 – nu pot fi folosite ca index;
- intervalul valorilor este **foarte mare** – exemplu: sortarea salariilor între 1 și 100 000 ar necesita o listă de frecvență cu 100 000 de poziții, risipind multă memorie;
- datele nu sunt **numere întregi** – exemplu: șiruri de caractere, nume.



