MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

EMANUELA CERCHEZ • MIHAI NAN • DANA LICA

IX

INFORMATICĂ

CURRICULUM DE SPECIALITATE (CS),
FILIERA TEORETICĂ,
PROFILUL REAL, SPECIALIZAREA ȘTIINȚE ALE NATURII



intuitext
grup SOFTWIN

Acest manual este proprietatea Ministerului Educației și Cercetării.

Manualul școlar a fost aprobat prin Ordinul ministrului educației și cercetării nr. 5779/09.10.2026.

Acest manual şcolar este realizat în conformitate cu Programa şcolară aprobată prin Ordinul ministrului educaţiei și cercetării nr. 6930/19.12.2025.



MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

EMANUELA CERCHEZ • MIHAI NAN • DANA LICA

IX

INFORMATICĂ

CURRICULUM DE SPECIALITATE (CS)
FILIERA TEORETICĂ,
PROFILUL REAL, SPECIALIZAREA ȘTIINȚE ALE NATURII

clasa a IX-a



intuitext
grup SOFTWIN

Disciplina: **Informatică**
Clasa: **a IX-a**
Număr de pagini: **84**

ACEST MANUAL A FOST FOLOSIT DE						
Anul	Numele elevului	Clasa	Școala	An școlar	Starea manualului*	
					la primire	la returnare
1						
2						
3						
4						

*Starea manualului se înscrie folosind termenii: *nou, bun, îngrijit, nesatisfăcător, deteriorat.*

Cadrele didactice vor controla dacă numele elevului este scris corect. Elevii nu trebuie să facă niciun fel de însemnări pe manual.

Copyright © 2026 – **Editura INTUITEXT**
Toate drepturile rezervate Editurii INTUITEXT.
Nicio parte din acest volum nu poate fi copiată fără permisiunea scrisă a Editurii INTUITEXT.

Editura INTUITEXT
București, b-dul Dimitrie Pompeiu 10A
Clădirea Conect 1, etaj 1, zona A, biroul nr. 2, sector 2

Departamentul vânzări:
Telefon: 0372.156.300
Fax: 021.233.07.63
vanzari@intuitext.ro
www.intuitext.ro

Referenți:
Prof. dr. ing. Florin POP, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie POLITEHNICA, București
Prof. gr. didactic I Mincă Carmen-Nicoleta, Colegiul Național de Informatică „Tudor Vianu”, București

Să exersăm!

Vei rezolva diferite sarcini de lucru pentru a folosi, în situații noi, ceea ce ai învățat.

Probleme rezolvate

Vei analiza și studia modele de rezolvare pentru sarcinile de lucru relevante.

AUTOEVALUARE

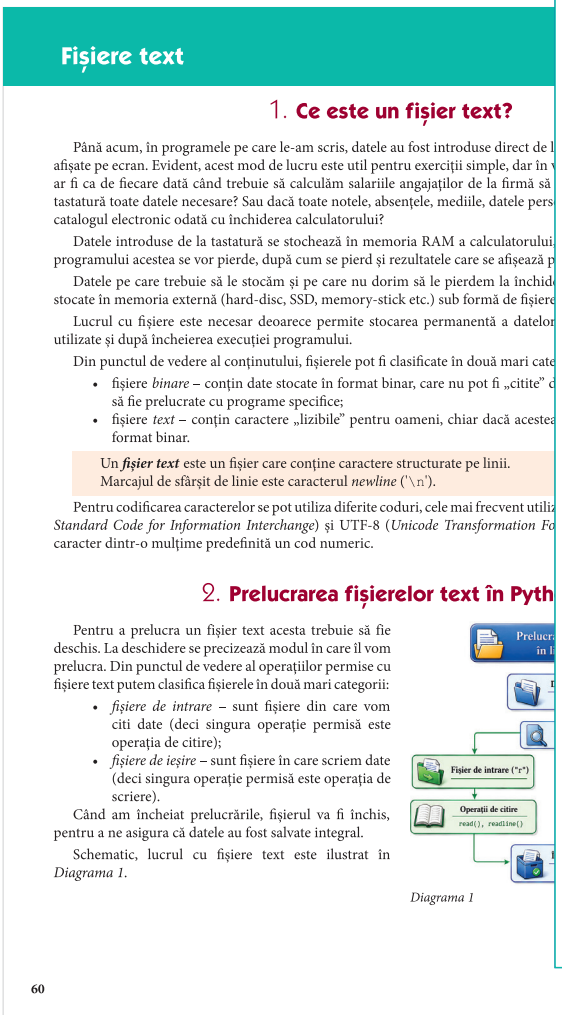
Vei reflecta asupra activității tale și a colaborării în echipă.

Portofoliu digital

Vei organiza programele și rezolvările realizate de tine și vei evalua propriul progres.

Evaluare

Vei aplica noțiunile învățate și îți vei verifica nivelul de înțelegere prin rezolvarea exercițiilor propuse.



Fișiere text

1. Ce este un fișier text?

Până acum, în programele pe care le-am scris, datele au fost introduse direct de l
afișate pe ecran. Evident, acest mod de lucru este util pentru exerciții simple, dar în v
ar fi ca de fiecare dată când trebuie să calculăm salariile angajaților de la firmă să
tastatură toate datele necesare? Sau dacă toate notele, absențele, mediile, datele pers
catalogul electronic odată cu închiderea calculatorului?

Datele introduse de la tastatură se stochează în memoria RAM a calculatorului
programului acestea se vor pierde, după cum se pierd și rezultatele care se afișează p

Datele pe care trebuie să le stocăm și pe care nu dorim să le pierdem la închid
stocate în memoria externă (hard-disc, SSD, memory-stick etc.) sub formă de fișiere

Lucrul cu fișiere este necesar deoarece permite stocarea permanentă a datelor
utilizate și după încheierea execuției programului.

Din punctul de vedere al conținutului, fișierele pot fi clasificate în două mari cate

- fișiere *binare* – conțin date stocate în format binar, care nu pot fi „citite" d
să fie prelucrate cu programe specifice;
- fișiere *text* – conțin caractere „lizibile" pentru oameni, chiar dacă acestea
format binar.

Un ***fișier text*** este un fișier care conține caractere structurate pe linii.
Marcajul de sfârșit de linie este caracterul *newline* ('\n').

Pentru codificarea caracterelor se pot utiliza diferite coduri, cele mai frecvent utiliz
Standard Code for Information Interchange) și UTF-8 (*Unicode Transformation Fo*
caracter dintr-o mulțime predefinită un cod numeric.

2. Prelucrarea fișierelor text în Pyth

Pentru a prelucra un fișier text acesta trebuie să fie deschis. La deschidere se precizează modul în care îl vom prelucra. Din punctul de vedere al operațiilor permise cu fișiere text putem clasifica fișierele în două mari categorii:

- *fișiere de intrare* – sunt fișiere din care vom citi date (deci singura operație permisă este operația de citire);
- *fișiere de ieșire* – sunt fișiere în care scriem date (deci singura operație permisă este operația de scriere).

Când am încheiat prelucrările, fișierul va fi închis, pentru a ne asigura că datele au fost salvate integral.

Schematic, lucrul cu fișiere text este ilustrat în *Diagrama 1*.

Diagrama 1

60

Fișa de autoevaluare

Vei măsura propriul tău progres în învățare. Vei identifica dificultățile întâmpinate.

Proiect

Vei lucra în echipă pentru a obține un produs informatic. Vei aprecia cum ai lucrat și cum ai colaborat.

Investigație

Vei descoperi răspunsuri pentru întrebările tale! Vei evalua soluția găsită și cooperarea cu colegii.

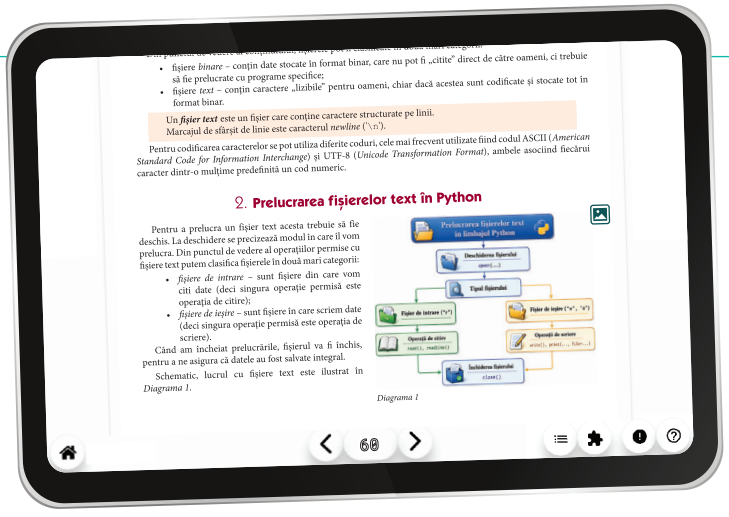
Observ!

Vei observa activ comportamentul tău din cadrul activităților, precum și respectarea sarcinilor primite.

Imagine în manualul digital

Film sau animație în manualul digital

Activitate interactivă în manualul digital



Acasă

Mergi la pagina

Cuprins interactiv

Activități de învățare

Răspunsuri

Ajutor

Cuprins

UNITATEA 1 Principii de elaborare a unui program. Fundamente ale elaborării algoritmilor 7

1. Ce este un algoritm? 7
2. Proprietăți caracteristice ale algoritmilor 7
3. Etapele rezolvării unei probleme 8
4. Date cu care lucrează algoritmii 9
5. Expresii 9
6. Fișă de autoevaluare 11
7. Reprezentarea algoritmilor 11
8. Structura secvențială 12
9. Probleme rezolvate 13
10. Fișă de autoevaluare 14
11. Structura alternativă (de decizie) 14
12. Probleme rezolvate 15
13. Fișă de autoevaluare 15
14. Structura repetitivă 16
15. Probleme rezolvate 18
16. Fișă de lucru aplicativă 20
17. Evaluare 21

UNITATEA 2 Principii de elaborare a unui program. Elemente de bază ale limbajului Python 22

1. Noțiuni introductive 22
2. Introducere în Programarea Orientată pe Obiecte (POO) 25
3. Variabile în limbajul Python 26
4. Expresii 29
5. Fișă de autoevaluare 31
6. Structura alternativă (de decizie) 32
7. Probleme rezolvate 33
8. Structura repetitivă 35
9. Probleme rezolvate folosind instrucțiunea `while` 35
10. Probleme rezolvate folosind instrucțiunea `for` 37
11. Instrucțiunile `break` și `continue` 38
12. Comentarii în limbajul Python 38
13. Evaluare 40

UNITATEA 3 Prelucrări ale datelor numerice 41

1. Prelucrarea cifrelor unui număr natural 41
2. Fișă de autoevaluare 44
3. Prelucrarea divizorilor numerelor naturale 45
4. Probleme rezolvate 48
5. Fișă de autoevaluare 49
6. Evaluare 51

UNITATEA 4 Subprograme 52

1. Utilitatea subprogramelor în rezolvarea problemelor complexe 52
2. Subprograme în limbajul Python 52
3. Biblioteci 55
4. Cum ne documentăm? 56
5. Arhitectura unei aplicații Python 56
6. Probleme rezolvate 56
7. Fișă de autoevaluare 57
8. Evaluare 59

UNITATEA 5 Fișiere text 60

1. Ce este un fișier text? 60
2. Prelucrarea fișierelor text în Python 60
3. Probleme rezolvate 62
4. Fișă de autoevaluare 64
5. Evaluare 65

UNITATEA 6 Fundamentele structurilor de date liniare 66

1. Modelul conceptual liniar – listă 66
2. Fișă de autoevaluare 67
3. Clasa `list` din Python 67
4. Operatori specifici pentru liste 68
5. Metodele clasei `list` 70
6. Parcurgerea listelor 73
7. Fișă de autoevaluare 74
8. Metode de generare a elementelor unei liste 75
9. Fișă de autoevaluare 77
10. Metode de sortare a elementelor unei liste 78
11. Fișă de autoevaluare 80
12. Proiect: Decoderul Astro-Sort 81
13. Evaluare 83

COMPETENȚE GENERALE:

1. Identifică principalele caracteristici ale modelelor conceptuale și operaționale ale dezvoltării produselor software, pentru înțelegerea fundamentelor programării
2. 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
3. Utilizează modele conceptuale și operaționale ale dezvoltării produselor software, în scopul obținerii de soluții informatice funcționale și eficiente
4. 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
5. Evaluează corectitudinea și eficiența soluțiilor informatice, în vederea optimizării și asigurării funcționalității în diverse scenarii de utilizare
6. Elaborează algoritmi și programe personalizate, pentru a crea soluții informatice coerente și adaptate cerințelor

COMPETENȚE SPECIFICE:

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;

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;

1.3. Identifică specificul, caracteristicile și etapele unor strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor;

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;

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;

2.1. Explică principiile de organizare a datelor în cadrul unor modele conceptuale fundamentale – date simple sau liste, pentru a le gestiona eficient;

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;

2.3. Explică etapele unor strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor;

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;

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;

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;

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;

3.3. Aplică strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor;

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;

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;

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;

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;

4.3. Analizează aplicabilitatea, avantajele și dezavantajele utilizării unor strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor;

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;

4.5. Analizează aplicabilitatea, avantajele și dezavantajele utilizării subprogramelor, în implementarea algoritmilor în limbaj de programare;

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;

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;

5.3. Evaluează corectitudinea și eficiența algoritmilor care utilizează strategii de rezolvare a problemelor prin proiectarea modulară a algoritmilor;

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;

5.5. Evaluează corectitudinea și eficiența programelor care utilizează subprograme, pentru a rezolva probleme informatice;

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;

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;

6.3. Proiectează modular algoritmi personalizați pentru rezolvarea problemelor;

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;

6.5. Implementează aplicații în limbaj de programare care integrează subprograme personalizate, pentru a modulariza și organiza eficient codul în cadrul soluțiilor informatice.

Principii de elaborare a unui program. Fundamente ale elaborării algoritmilor

1

1. Ce este un algoritm?

Un ***algoritm*** reprezintă o metodă de rezolvare a problemelor de un anumit tip.

A rezolva o problemă înseamnă a obține, pentru anumite *date de intrare*, rezultatul problemei (*date de ieșire*):



Algoritmul este constituit dintr-o succesiune de operații care descriu, pas cu pas, modul de obținere a datelor de ieșire, plecând de la datele de intrare.

Se pot descrie algoritmi pentru rezolvarea problemelor din orice domeniu de activitate. De exemplu, orice rețetă de bucătărie poate fi considerată un algoritm prin care, plecând de la materiile prime, obținem printr-o succesiune finită de operații produsul finit.

Exemplu:

Presupunând că dispunem de un aragaz, o tigaie, 2 ouă, sare și ulei, să pregătim ochiuri.

„Date" de intrare: ouă, ulei, sare.
„Date" de ieșire: ochiuri.
Pas 1: Se pune tigaia pe foc.
Pas 2: Se toarnă uleiul în tigaie.
Pas 3: Așteptăm până când se încinge uleiul.
Pas 4: Spargem cu îndemânare ouăle în tigaie.
Pas 5: Așteptăm până când ouăle se rumenesc.
Pas 6: Dacă nu ținem regim, adăugăm sare.

Am descris o succesiune de pași, prin care, pentru orice „date" de intrare (ulei, sare, ouă), obținem rezultatul dorit (ochiuri). Fiecare pas constă din operații culinare specifice, care se execută în ordinea în care sunt specificate.

Să exersăm!

1 Descrie, punând în evidență succesiunea pașilor, un algoritm de copiere a unei zone de text dintr-un document Word într-un alt document Word.

2 Descrie, punând în evidență succesiunea pașilor, algoritmul pe care l-ai aplicat la ultima experiență efectuată în laboratorul de chimie.

2. Proprietăți caracteristice ale algoritmilor

Exemplele de la lecția precedentă generează în mod firesc două întrebări:

- *Pentru orice problemă există un algoritm de rezolvare?*

Răspunsul este NU! Există probleme pentru care se poate demonstra (lucru dificil!) că nu există algoritmi de rezolvare, dar și probleme pentru care nici nu s-a demonstrat că nu admit o metodă de rezolvare algoritmică, dar nici nu s-a descoperit (încă!) soluția algoritmică.

- *Orice succesiune de pași reprezintă un algoritm?*

Din nou, răspunsul este NU! Pentru a fi un algoritm, secvența trebuie să îndeplinească trei condiții:

1. Claritate – la fiecare moment, operația care urmează a fi executată este unic determinată, definită și realizabilă.
De exemplu, secvența „*Dacă plouă stau acasă sau merg la cinema*" nu este clară, deoarece, în cazul în care plouă, operația care se execută nu este unic determinată.

2. **Generalitate** (universalitate) – o secvență de pași reprezintă un algoritm de rezolvare a unei probleme dacă obține date de ieșire (rezultate) pentru orice date de intrare specifice problemei.

Secvența de pași pentru rezolvarea ecuației de gradul I este generală, deoarece conduce la rezolvarea ecuației `ax+b=0` pentru orice valori reale ale coeficienților `a` și `b`.

Dar, dacă am fi descris o secvență de pași care să rezolve numai ecuația `x+2=0`, aceasta nu ar fi fost un algoritm!

3. **Finitudine** – rezultatele problemei se obțin după un număr finit de pași.

De exemplu, problema „*Să se determine toate zecimalele numărului π*” nu are o soluție algoritmică, deoarece π este un număr irațional, care are o infinitate de zecimale. Dar dacă am enunța problema astfel: „*Fie n un număr natural dat. Să se determine primele n zecimale ale numărului π.*”, această problemă admite o soluție algoritmică, deoarece primele `n` zecimale se pot obține după un număr finit de pași.

În concluzie, deși nu putem defini cu rigurozitate noțiunea de algoritm, putem descrie această noțiune astfel:

Un ***algoritm*** este constituit dintr-o succesiune clară de operații realizabile, care au ca scop obținerea într-un timp finit a rezultatelor unei probleme, pentru orice set de date de intrare.

3. Etapele rezolvării unei probleme

Rezolvarea unei probleme constituie un proces complex, care comportă mai multe etape.

1. *Analiza problemei* în scopul stabilirii datelor de intrare, precum și a rezultatelor pe care trebuie să le obținem prin rezolvarea problemei.
2. *Elaborarea unui algoritm* de rezolvare a problemei.
3. *Implementarea* algoritmului într-un limbaj de programare.
4. *Verificarea corectitudinii* algoritmului propus.
   Un prim pas constă în testarea programului pe diverse seturi de date de test. Seturile de date de test trebuie elaborate cu atenție, astfel încât să acopere, pe cât posibil, toate variantele de execuție a algoritmului, inclusiv situații de excepție, și să verifice dacă fiecare subproblemă a problemei date este rezolvată corect.

   Testarea poate pune în evidență, eventual, omisiuni sau erori de concepție a algoritmilor, dar nu garantează corectitudinea algoritmului. Pentru aceasta ar trebui să testăm algoritmul pe toate seturile posibile de date de intrare, ceea ce este practic imposibil.
5. *Analiza complexității algoritmului*
   În general, există mai mulți algoritmi de rezolvare a unei probleme date. Pentru a alege cel mai bun algoritm, trebuie să analizăm acești algoritmi în scopul determinării eficienței lor și, pe cât posibil, a optimalității lor.

   Eficiența unui algoritm se evaluează din două puncte de vedere:

   1. din punctul de vedere al spațiului de memorie necesar pentru memorarea valorilor variabilelor care intervin în algoritm (complexitate spațiu).
   2. din punctul de vedere al timpului de execuție (complexitate timp).

Complexitatea spațiu o vom analiza cu precădere atunci când vom transpune algoritmul într-un limbaj de programare.

Pentru a estima *complexitatea timp* vom presupune că se lucrează pe un calculator „clasic”, în sensul că o singură instrucțiune este executată la un moment dat. Astfel, timpul necesar execuției programului depinde de numărul de operații elementare efectuate de algoritm.

Observație

Elaborarea algoritmilor nu este un proces liniar, adeseori este necesar să revenim la o anumită etapă și să o repetăm. De exemplu, după ce am demonstrat corectitudinea algoritmului și am analizat eficiența sa, ne putem pune problema de a optimiza algoritmul sau numai implementarea sa, caz în care trebuie să revenim la cea de a doua etapă, de proiectare a algoritmilor și de scriere a codului, etapă urmată în mod necesar de teste de corectitudine, eliminare a erorilor, demonstrații de corectitudine, teste de determinare a complexității, analiza teoretică a complexității etc.

4. Date cu care lucrează algoritmii

Orice algoritm lucrează cu date: date de intrare (datele pe care trebuie să le primească un algoritm din exterior), date de ieșire (datele pe care trebuie să le furnizeze algoritmul în exterior), precum și date de manevră (date temporare, necesare algoritmului).

Datele cu care lucrează algoritmii pot fi clasificate din mai multe puncte de vedere. O primă clasificare a datelor, în funcție de posibilitatea de a-și modifica valoarea, este:

1. **Constante** – date care nu își modifică valoarea

Exemple:

Constante întregi	`12 -13`
Constante reale	`2.5 -1.275`
Constante logice	`True` **(adevărat)** `False` **(fals)**
Constante șir de caractere	`"acesta este un sir de caractere"`

2. **Variabile** – date care își modifică valoarea.

Întrucât variabilele își modifică valoarea, pentru a ne referi la acestea trebuie să le asociem un nume.

Numele unei variabile este o succesiune formată din litere (mari sau mici), cifre sau caracterul `'_'` (*underscore*), primul caracter din succesiune fiind obligatoriu diferit de orice cifră.

Exemplu:

Pentru a rezolva o ecuație de forma `ax+b=0` vom utiliza două variabile `a` și `b`. Prin operația de citire, acestora li se asociază câte o valoare reală, introdusă de la tastatură. Utilizarea acestor două variabile este strict necesară, pentru a respecta generalitatea algoritmului. Dacă am utiliza două valori constante (de exemplu, `2` și `3.5`), secvența de operații ar fi rezolvat numai ecuația `2x+3.5=0`, deci nu ar fi avut utilitate.

Observații

1. Se face diferență între literele mici și literele mari. De exemplu, `Produs` și `produs` sunt nume de variabile diferite.
2. Deși este simplu de utilizat variabile cu nume scurte, în practică este recomandat să alegem nume de variabile semnificative, care să descrie semnificația acestora. De exemplu, `temperatura`, `nume_elev`, `nr_zile`. În cazul în care numele variabilei este format din mai multe cuvinte, pentru lizibilitate mărită separăm cuvintele prin `'_'`.

Să exersăm!

1 Care dintre următoarele nume de variabile sunt greșite? Justifică răspunsul!

a. `Clasa` **b.** `a+b` **c.** `_5`

d. `12A` **e.** `nume_si_prenume_elev` **f.** `Varsta elev`

2 Într-un algoritm este necesară o variabilă în care să memorezi temperatura medie înregistrată în România în anul 2026. Propune o denumire semnificativă pentru această variabilă.

5. Expresii

O expresie este constituită dintr-o succesiune de operanzi, conectați prin operatori. Un operand poate fi o constantă, o variabilă, sau o expresie încadrată între paranteze rotunde. Operatorii desemnează operațiile care se execută asupra operanzilor.

Evaluarea unei expresii presupune calculul valorii acesteia, prin înlocuirea valorilor variabilelor care intervin ca operanzi în expresie și efectuarea operațiilor specificate de operatori.

În continuare, ne vom reaminti de 3 categorii de operatori pentru expresiile utilizate în reprezentarea algoritmilor.

Operatori aritmetici

Pentru exemplificare vom considera următoarele variabile din *Tabelul 1:* `a` (cu valoarea `12`), `b` (cu valoarea `5`), `c` (cu valoarea `1.5`), `d` (cu valoarea `-4`).

Observații

Pentru a calcula partea întreagă a unui număr real se utilizează notația matematică (încadrarea expresiei între paranteze drepte); de exemplu, partea întreagă a numărului real memorat în variabila `x` este `[x]`.

Pentru a calcula radicalul unui număr se utilizează, de asemenea, notația matematică.

Tabelul 1

Operator	Semnificație	Exemplu de expresie	Valoarea expresiei
+ (unar)	Același semn algebric	`+a`	`12`
– (unar)	Semn algebric schimbat	`-a` `-d`	`-12` `4`
+ (binar)	Adunare	`a+b`	`17`
– (binar)	Scădere	`a-b`	`7`
* (binar)	Înmulțire	`b*c` `b*d`	`7.5` `-20`
/ (binar)	Împărțire	`a/b` `a/d`	`2.4` `-3.0`
% (binar)	Restul împărțirii întregi	`a%b`	`2`

Operatori relaționali

Operatorii relaționali sunt operatori binari și descriu relația de ordine sau de egalitate dintre cei doi operanzi: < (mai mic), > (mai mare), ≤ (mai mic sau egal), ≥ (mai mare sau egal), = (egal), ≠ (diferit). Vezi *Tabelul 2*.

Valoarea unei expresii care conține un operator relațional este întotdeauna de tip logic (deci poate fi `True` sau `False` – adevărat sau fals).

Pentru exemplificare vom considera următoarele variabile: `a` (cu valoarea `12`), `b` (cu valoarea `5`), `c` (cu valoarea `1.5`), `d` (cu valoarea `5.0`).

Tabelul 2

Expresie	Valoarea expresiei
`a < b`	`False`
`b > c`	`True`
`a ≥ 0`	`True`
`b = d`	`True`
`False < True`	`True`
`"anatol" < "anapoda"`	`False`
`"Ana" < "ana"`	`True`
`'a' ≥ 'A'`	`True`

Observații

Pentru valorile logice `False<True`.

Pentru compararea caracterelor vom considera ordinea codurilor ASCII[1].

Operatori logici

Operatorii logici definesc operațiile logice studiate la matematică și se aplică operanzilor logici:

- *negație logică* – `not` (operator unar);
- *conjuncție logică* – `and` (operator binar);
- *disjuncție logică* – `or` (operator binar).

Valoarea unei expresii care utilizează operatori logici este logică și poate fi determinată ca în *Tabelul 3*.

Tabelul 3

x	y	not x	x or y	x and y
False	False	True	False	False
False	True	True	True	False
True	False	False	True	False
True	True	False	True	True

1. American Standard Code for Information Interchange (cod prin care se asociază fiecărui caracter dintr-o mulțime predefinită de `128` de caractere un număr între `0` și `127`). De exemplu, litera `'A'` are asociat codul `65`, litera `'B'` are codul `66`, litera `'a'` are codul `97`, caracterul `'0'` are codul `48`, caracterul spațiu are codul `32` etc.

Evaluarea unei expresii

În procesul de evaluare a unei expresii se respectă regulile de bază, învățate la matematică (în primul rând se evaluează expresiile dintre parantezele rotunde; apoi se execută operațiile în ordinea priorității lor; dacă există mai multe operații cu aceeași prioritate, se execută în ordine, de la stânga la dreapta). Operatorii studiați, în ordinea descrescătoare a priorității, sunt cei din *Tabelul 4*.

Tabelul 4

Operatori	Simbol
Operatorii unari semn algebric	`+ -`
Operatorii aritmetici multiplicativi	`* / %`
Operatorii aritmetici aditivi	`+ -`
Operatorii relaționali	`< > ≤ ≥ = ≠`
Negația logică	`not`
Conjuncție logică	`and`
Disjuncție logică	`or`

Exemple:

1. Fie `x` un număr natural. Expresia `x%10` are ca valoare restul împărțirii numărului `x` la `10` (care este de fapt ultima cifră a lui `x`).
2. Fie `n` un număr natural. Expresia `n*(n+1)/2` reprezintă suma numerelor naturale mai mici sau egale cu `n`.
3. Expresia `x%2=0` are valoarea `True` dacă și numai dacă numărul memorat în variabila `x` este par.
4. Expresia `x>0 and y>0` are valoarea `True` dacă și numai dacă valorile memorate în variabilele `x` și `y` sunt ambele strict pozitive.

6. Fișă de autoevaluare

1 Fie `n` o variabilă întreagă cu valoarea `32735`. Evaluează expresia `n%10+[n/10]%10+[n/100]%10`.

2 Să considerăm `a`, `b` și `c` trei variabile întregi, având valorile `5`, `7` și respectiv `8`.

Evaluează expresia `a≤b and b<c or c%2=1`.

3 Scrie o expresie care are valoarea `True` dacă și numai dacă valoarea memorată în variabila reală `x` este în intervalul `[a,b]` (`a` și `b` variabile reale).

4 Scrie o expresie care are valoarea `True` dacă și numai dacă numărul natural memorat în variabila `x` este par și are două cifre.

5 Scrie o expresie logică care să aibă valoarea `True` dacă și numai dacă valorile variabilelor `a`, `b` și `c` sunt în ordine strict crescătoare.

7. Reprezentarea algoritmilor

Principiile programării structurate

Creșterea complexității aplicațiilor a impus la începutul anilor '70 apariția unei noi paradigme în programare: *programarea structurată*. Scopul era de a dezvolta noi tehnici de programare, care să permită dezvoltarea unor programe fiabile, ușor de elaborat în echipă, ușor de depanat, de întreținut și de reutilizat.

Un prim principiu al programării structurate este *modularizarea*. Pentru proiectarea unor aplicații complexe, este necesară descompunerea problemei care trebuie rezolvată în subprobleme relativ independente, pentru fiecare dintre aceste subprobleme scriindu-se module de program mai simple. Fiecare modul efectuează un set de prelucrări specifice și este relativ independent de celelalte module, cu care comunică prin intermediul unui set de parametri, care constituie interfața. Avantajele sunt multiple. Cum la orice firmă se lucrează în echipă, modulele de program pot fi implementate de mai mulți programatori. Modificarea unui modul nu afectează celelalte module. Fiecare modul poate fi implementat, testat, depanat, modificat, independent de celelalte.

Un alt principiu fundamental este *structurarea datelor și a prelucrărilor*.

Programatorul are posibilitatea de a-și grupa datele în colecții, organizate după anumite reguli, denumite *structuri de date*. Prelucrările asupra datelor sunt structurate separat.

Teorema programării structurate (Böhm-Jacopini)

Orice algoritm poate fi descris prin compunerea a trei structuri fundamentale: *structura liniară* (secvențială), *structura alternativă* și *structura repetitivă*.

Reprezentarea algoritmilor în pseudocod

Pentru ca o secvență de operații să constituie un algoritm, aceasta trebuie să fie clară, adică la orice moment operația care urmează a fi executată trebuie să fie *unic determinată*, *definită* și *realizabilă* (să poată fi efectuată la momentul respectiv, cu mijloacele disponibile).

Apare întrebarea: *Care sunt operațiile definite, cu ajutorul cărora să putem descrie algoritmi?*

Un răspuns posibil ar fi: Operațiile definite sunt instrucțiunile limbajului de programare X. Este un răspuns acceptabil pentru toți cei care cunosc acest limbaj de programare. Dar pentru ceilalți?

Este nevoie de o metodă universală de reprezentare a algoritmilor, ulterior fiecare programator având posibilitatea de a implementa algoritmii în limbajul pe care îl preferă. De-a lungul timpului s-au impus două modalități de reprezentare a algoritmilor: *schemele logice* și *limbajele de tip pseudocod*.

Schemele logice constituie o metodă de reprezentare grafică, foarte sugestivă, dar cu o serie de dezavantaje: se dă o egală importanță componentelor principale ca și detaliului. Prin urmare, schemele logice devin deosebit de stufoase și greu de urmărit. Pentru aplicațiile mai complexe, când este necesară modularizarea, este practic imposibil de pus în evidență legăturile dintre module în cadrul schemei logice.

Din acest motiv, s-a impus o altă metodă de reprezentare a algoritmilor: pseudocodul.

Un limbaj de tip ***pseudocod*** este un ansamblu de convenții, respectate în mod sistematic, care definesc operațiile permise (denumite și instrucțiuni) pentru reprezentarea algoritmilor.

8. Structura secvențială

Operația de citire

citește variabila$_1$, variabila$_2$, ..., variabila$_n$

Efect: Prin operația de citire (denumită și operație de intrare) se preiau succesiv valori de la tastatură și se asociază, în ordine, variabilelor specificate.

Operația de scriere

scrie expresie$_1$, expresie$_2$, ..., expresie$_n$

Efect: Operația de scriere (denumită și operație de ieșire) presupune evaluarea în ordine a expresiilor specificate și afișarea pe ecran a valorilor lor pe aceeași linie.

Operația de atribuire

variabila ← expresie

Efect: se evaluează expresia, apoi se atribuie valoarea expresiei variabilei din membrul stâng.

Observație

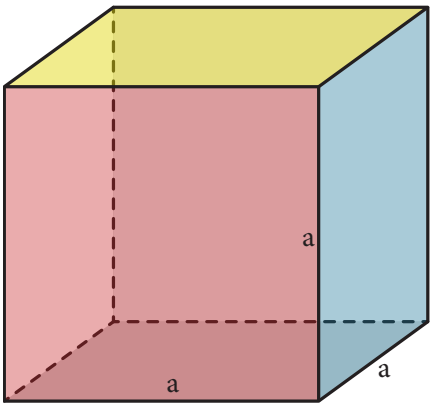
Pentru claritate, putem insera într-un algoritm comentarii, mici texte explicative. Începutul unui comentariu este marcat de succesiunea de caractere // și continuă până la sfârșitul liniei.

Parcurgerea instrucțiunilor în secvență, în ordinea specificării lor, reprezintă o ***structură liniară*** (secvențială).

9. Probleme rezolvate

Cub

1 Fie `a` un număr real, citit de la tastatură, care reprezintă lungimea laturii unui cub (*Imaginea 1*). Să se scrie un algoritm care să calculeze și să afișeze volumul și suprafața totală a cubului.



Imaginea 1

Soluție

```
citește a
V ← a*a*a
scrie "Volumul cubului este ", V
S ← 6*a*a
scrie "Suprafața totală a cubului este ", S
```

Compus chimic

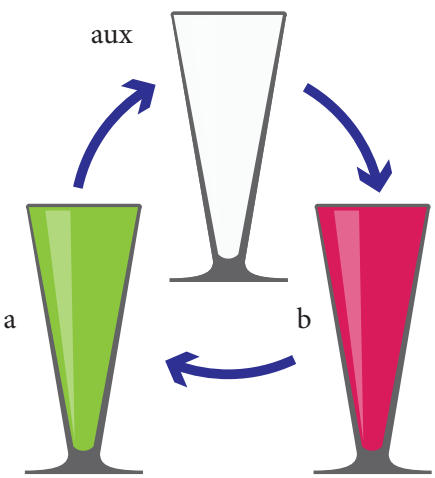
2 Un grup de cercetători studiază un compus chimic descoperit pe planeta Marte. În urma analizelor efectuate, au dedus că o moleculă din acest compus este formată din `nC` atomi de carbon, `nO` atomi de oxigen și `nH` atomi de hidrogen. Știind că masa atomului de carbon este `12`, masa atomului de oxigen este `16`, iar masa atomului de hidrogen este `1`, să se scrie un algoritm care să calculeze și să afișeze masa moleculară a acestui compus.

Soluție

```
citește nC, nO, nH
m ← nC*12+nO*16+nH
scrie "Masa moleculară a compusului este ", m
```

Schimb

3 Se citesc numerele reale `a` și `b`. Să se interschimbe valorile variabilelor `a` și `b`, apoi să se afișeze (*Imaginea 2*).



Imaginea 2

Soluție

Să ne imaginăm că variabila `a` este un pahar cu suc de mere, iar variabila `b` un pahar cu suc de zmeură. Trebuie să schimbăm conținuturile celor două pahare. Singura soluție fără „pierderi” este de a utiliza un pahar auxiliar (variabila `aux`). Vom turna sucul de mere în paharul auxiliar (atribuim variabilei `aux` valoarea variabilei `a`). Acum paharul cu suc de mere este gol, turnăm în el sucul de zmeură (atribuim variabilei `a` valoarea variabilei `b`). Paharul cu suc de zmeură (variabila `b`) a devenit disponibil, turnăm în el sucul de mere din paharul auxiliar (atribuim variabilei `b` valoarea variabilei `aux`).

Această metodă este denumită sugestiv „*regula celor trei pahare*”, deși analogia nu este absolut perfectă (când atribuim unei variabile valoarea altei variabile, această valoare se va găsi după atribuire în ambele, ceea ce nu se întâmplă atunci când turnăm conținutul unui pahar de suc în alt pahar de suc).

```
citește a, b                  // 1
aux ← a                       // 2
a ← b                         // 3
b ← aux                       // 4
scrie "a= ", a, "b= ", b      // 5
```

Pentru a înțelege mai bine acest algoritm să urmărim execuția lui pentru un set particular de date de intrare *(Tabelul 5)*. În acest scop, am numerotat instrucțiunile de la `1` la `5`. De exemplu, să presupunem că de la tastatură se introduc valorile `3` și `7`.

Tabelul 5

Linia	a	b	aux	Explicație
1	3	7	?	Se citesc de la tastatură valorile `3` și `7` și se atribuie în ordine variabilelor `a` și `b`.
2	3	7	3	Se copiază valoarea variabilei `a` în variabila `aux`.
3	7	7	3	Variabilei `a` i se atribuie valoarea variabilei `b`.
4	7	3	3	Variabilei `b` i se atribuie valoarea variabilei `aux` (în care am „salvat” valoarea inițială a variabilei `a`).
5	7	3	3	Pe ecran va apărea `a=7 b=3`

10. Fișă de autoevaluare

1 Ce va afișa următorul algoritm, dacă se citesc valorile `7` și `23`?

```
citește a, b
a ← a+b
b ← a-b
a ← a-b
scrie "a= ", a, " b= ", b
```

2 Fie `x1`, `x2`, `x3`, `x4`, `x5` cinci valori reale. Scrie un algoritm care să folosească o singură variabilă suplimentară pentru a permuta circular valorile celor cinci variabile (adică în final `x1` să aibă valoarea inițială a variabilei `x2`, `x2` valoarea inițială a variabilei `x3`, `x3` valoarea inițială a variabilei `x4`, `x4` valoarea inițială a variabilei `x5`, iar `x5` valoarea inițială a variabilei `x1`).

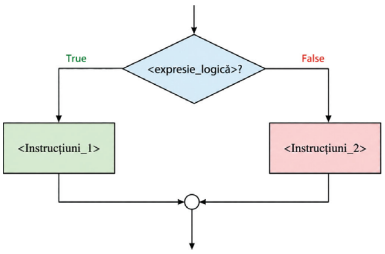
3 Fie `a`, `b` și `c` trei numere reale, care reprezintă lungimile laturilor unui triunghi. Scrie un algoritm care să calculeze și să afișeze perimetrul și aria triunghiului.

4 Doi colegi (Valeriu și Ioan) pleacă simultan din orașele în care locuiesc, unul către celălalt. Știind că distanța dintre cele două orașe este `D` km, că Valeriu merge cu viteza `v1` km/h, iar Ioan merge cu viteza `v2` km/h (`D`, `v1`, `v2` numere reale), scrie un algoritm care calculează după cât timp se întâlnesc cei doi colegi și la ce distanță de orașul în care locuiește Valeriu.

5 Fie A și B două puncte în plan specificate prin coordonatele lor carteziene. Scrie un algoritm care să calculeze și să afișeze lungimea segmentului AB.

11. Structura alternativă (de decizie)

```
dacă <expresie_logică> atunci
     <Instrucțiuni_1>
altfel
     <Instrucțiuni_2>
```



Imaginea 3

Efect:

Se evaluează expresia logică (*Imaginea 3*).

Dacă valoarea expresiei este `True`, atunci se execută `Instrucțiuni_1`.

Dacă valoarea expresiei este `False`, se execută `Instrucțiuni_2`.

Observații

1. Dacă pe ramura `altfel` nu este necesară efectuarea niciunei operații, această ramură poate lipsi.
2. `Instrucțiuni_1` și `Instrucțiuni_2` sunt subordonate instrucțiunii de decizie. În cazul în care o instrucțiune este subordonată unei alte instrucțiuni, aceasta se scrie indentat față de instrucțiunea care o subordonează.

Selectarea instrucțiunii ce urmează să fie executată în funcție de valoarea unei expresii reprezintă o ***structură alternativă***.

12. Probleme rezolvate

Maxim 2

1 Se citesc de la tastatură două numere întregi `x` și `y`. Scrie un algoritm care determină și afișează cea mai mare dintre valorile citite (valoarea maximă).

Soluție

Vom compara cele două valori și vom reține în variabila `maxim` cea mai mare dintre acestea.

Observație

Nu am testat decât condiția `x>y`. Evident, este suficient. Dacă am pus întrebarea „*Moneda a căzut cu stema în sus?*” și ni s-a răspuns „*Nu!*”, este inutil să mai întrebăm dacă a căzut cu cealaltă față în sus.

```
citește x, y
┌dacă x > y atunci
│      maxim ← x
│altfel
│      maxim ← y
└■
scrie maxim
```

Maxim 3

2 Se citesc de la tastatură trei numere întregi `x`, `y` și `z`. Scrie un algoritm care determină și afișează cea mai mare dintre valorile citite (valoarea maximă).

Soluție

Vom considera că prima valoare citită este maximul. Apoi vom compara succesiv fiecare valoare cu maximul și atunci când găsim o valoare mai mare decât maximul, actualizăm maximul.

```
citește x, y, z
maxim ← x
┌dacă y > maxim atunci
│      maxim ← y
└■
┌dacă z > maxim atunci
│      maxim ← z
└■
scrie maxim
```

Să exersăm!

3 Modifică algoritmul precedent pentru a calcula maximul a patru valori.

13. Fișă de autoevaluare

1 Rescrie corect următoarea secvență de instrucțiuni, respectând regulile de indentare și precizează ce valoare va avea variabila x, știind că variabilele `a`, `b` și `c` au valorile `3`, `5` și respectiv `7`.

```
dacă a-[b/2]<0 atunci x←1 altfel dacă a+b-[c/2]<b atunci x←2 altfel dacă a%b+c>b atunci x←3 altfel x←4
```

2 Rescrie corect următoarea secvență de instrucțiuni, respectând regulile de indentare și precizează ce valoare inițială ar putea avea variabila `x`, astfel încât la sfârșitul execuției secvenței de instrucțiuni variabila `y` să aibă valoarea `2`.

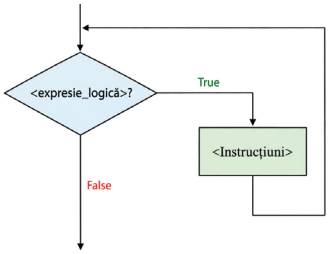
```
dacă x>3 atunci dacă x<7 atunci dacă x%2=0 atunci y←1 altfel y←2 altfel y←3 altfel y←4
```

3 Un elev este declarat promovat la bacalaureat dacă la fiecare dintre cele `3` probe de examen a luat cel puțin nota `5`, iar media sa generală este cel puțin `6`. Date fiind cele `3` note pe care elevul le-a obținut la bacalaureat, scrie un algoritm care să verifice dacă elevul a promovat sau nu examenul.

14. Structura repetitivă

Instrucțiunea repetitivă condiționată anterior

```
cât timp <expresie_logică> execută
         <Instrucțiuni>
```



Imaginea 4

Efect:

Pas 1: se evaluează expresia logică (*Imaginea 4*);

Pas 2: dacă valoarea expresiei este `False`, se iese din instrucțiunea `cât timp`;

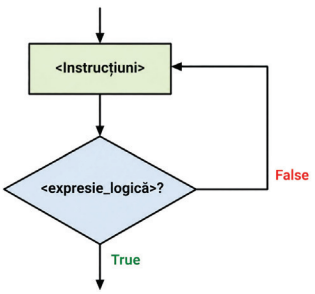
Pas 3: dacă valoarea expresiei este `True`, se execută `Instrucțiuni`, apoi se revine la Pas 1.

Observații

1. `Instrucțiuni` se execută repetat, cât timp valoarea expresiei este `True`. Pentru ca ciclul să nu fie infinit, este obligatoriu ca instrucțiunile care se execută să modifice cel puțin una dintre variabilele care intervin în expresie, astfel încât aceasta să poată lua valoarea `False`.
2. Dacă expresia are de la început valoarea `False`, `Instrucțiuni` nu se execută nici măcar o dată.
3. `Instrucțiuni` sunt subordonate instrucțiunii `cât timp`, deci sunt scrise indentat față de aceasta (*Imaginea 4*).

Instrucțiunea repetitivă condiționată posterior

```
repetă
      <Instrucțiuni>
până când <expresie_logică>
```



Imaginea 5

Efect:

Pas 1: se execută `Instrucțiuni`;

Pas 2: se evaluează expresia logică;

Pas 3: dacă valoarea expresiei este `True` se iese din instrucțiunea repetitivă; dacă valoarea expresiei este `False`, se revine la Pas 1 (*Imaginea 5*).

Observații

1. Instrucțiunea se execută repetat, cât timp valoarea expresiei este `False`. Pentru ca ciclul să nu fie infinit, este obligatoriu ca instrucțiunile care se execută să modifice cel puțin una dintre variabilele care intervin în expresie, astfel încât aceasta să poată lua valoarea `True`. Deoarece evaluarea expresiei se face după execuția instrucțiunii, instrucțiunea se execută cel puțin o dată.
2. Instrucțiunea repetitivă condiționată posterior poate fi transformată într-o secvență de cod echivalentă care utilizează instrucțiunea repetitivă condiționată anterior și reciproc (*Tabelul 6*).

Tabelul 6

Structura repetitivă	Transformare

`cât timp <expresie_logică> execută`
`  <Instrucțiuni>`
	`dacă <expresie_logică> atunci`
	`  repetă`
	`  <Instrucțiuni>`
	`  până când <not expresie_logică>`

`repetă`
`  <Instrucțiuni>`
`până când <expresie_logică>`
	`<Instrucțiuni>`
	`cât timp <not expresie_logică> execută`
	`  <Instrucțiuni>`

Observații

1. Pentru prima transformare a fost necesară testarea inițială a expresiei deoarece, spre deosebire de instrucțiunea `cât timp`, instrucțiunea `repetă ... până când` efectuează o dată `Instrucțiuni` înainte de a testa expresia. Din același motiv, la a doua transformare am executat mai întâi o dată `Instrucțiuni` înainte de a intra în structura repetitivă.
2. Ieșirea din structura repetitivă `repetă ... până când` se face atunci când expresia logică are valoarea `True`, în timp ce ieșirea din structura repetitivă `cât timp` se face atunci când expresia logică are valoarea `False`. Din acest motiv, la transformare a fost necesar să negăm condiția.

Instrucțiunea repetitivă cu număr cunoscut de pași

```
pentru <contor> ← <expInit>, <expFin>, <pas> execută
       <Instrucțiuni>
```

Efectul în cazul în care `pas>0`:

Pas 1: se inițializează variabila `contor` cu valoarea expresiei `expInit`

Pas 2: dacă valoarea variabilei `contor` este mai mică sau egală cu valoarea expresiei `expFin` atunci:
- se execută `Instrucțiuni`;
- se adună `pas` la valoarea variabilei `contor`;
- se revine la Pas 2;

Pas 3: dacă valoarea variabilei `contor` este mai mare decât valoarea expresiei `expFin` atunci se iese din instrucțiunea repetitivă.

Observații

1. Dacă `pas>0`, numărul de iterații (numărul de executări pentru `Instrucțiuni`) este egal cu `[(expFin - expInit)/pas]+1`.
2. Valoarea implicită pentru `pas` este `1`. Prin urmare, pentru această valoare nu este obligatoriu să specificăm `pas`.

Efectul în cazul în care `pas<0`:

Pas 1: se inițializează variabila `contor` cu valoarea expresiei `expInit`

Pas 2: dacă valoarea variabilei `contor` este mai mare sau egală cu valoarea expresiei `expFin` atunci:
- se execută `Instrucțiuni`;
- se adună `pas` la valoarea variabilei `contor`;
- se revine la Pas 2;

Pas 3: dacă valoarea variabilei `contor` este mai mică decât valoarea expresiei `expFin` atunci se iese din instrucțiunea repetitivă.

Instrucțiunea repetitivă cu număr cunoscut de pași poate fi transformată într-o secvență de cod echivalentă care utilizează instrucțiunea repetitivă condiționată anterior sau instrucțiunea repetitivă condiționată posterior. Să exemplificăm transformarea care utilizează instrucțiunea `cât timp` (*Tabelul 7*):

Tabelul 7

pas>0	pas<0

`contor ← expInit`
`cât timp contor ≤ ExpFin execută`
`  <Instrucțiuni>`
`  contor ← contor+pas`
	`contor ← expInit`
	`cât timp contor ≥ ExpFin execută`
	`  <Instrucțiuni>`
	`  contor ← contor+pas`

Executarea repetată a unei instrucțiuni, controlată de valoarea unei expresii, reprezintă o ***structură repetitivă***.

15. Probleme rezolvate

Secvență

1 Se consideră următorul algoritm descris în pseudocod:

```
citește n, a                          // 1
i ← 1; nr ← 0                         // 2
┌cât timp i < n execută                // 3
│          citește b                   // 4
│          i ← i+1                     // 5
│          ┌dacă b%a = 0 atunci        // 6
│          │     nr ← nr+1
│          └■
│           a ← b                      // 7
└■
scrie nr                              // 8
```

a. Ce se va afișa dacă se citesc valorile 6, 4, 8, 7, 21, 3, 6?

b. Introdu o succesiune de cel puțin trei valori astfel încât algoritmul să afișeze valoarea 0.

c. Descrie în limbaj natural care este efectul acestui algoritm.

Soluție

a. Se cere să verificăm efectul algoritmului pentru un set particular de date de intrare. În acest caz vom executa instrucțiunile algoritmului pas cu pas, urmărind valorile variabilelor care intervin în algoritm. Rezultatul obținut va fi 3.

b. Vom alege o secvență de exact 3 valori. Prima valoare din secvență este egală cu numărul de valori care urmează. Deci, va trebui să fie 2. Pentru ca valoarea variabilei nr să rămână 0 (așa cum este inițial) va trebui să alegem două valori, astfel încât condiția din instrucțiunea 6 să nu fie îndeplinită (adică prima valoare să nu dividă pe cea de a doua valoare). O soluție posibilă este 2 5 7.

c. Urmărind algoritmul pas cu pas, deducem că se citește mai întâi o valoare n, apoi se citesc succesiv n numere naturale. Permanent avem memorate în variabilele a și b două numere consecutiv citite dintre cele n. La fiecare pas se testează dacă primul număr (cel memorat în variabila a) divide pe cel de-al doilea (cel memorat în variabila b) și dacă da, valoarea variabilei nr este incrementată (mărită cu 1). Prin urmare, algoritmul numără în variabila nr câte perechi de elemente consecutive din secvența citită au proprietatea că primul element divide pe cel de-al doilea element din pereche.

Factorial

2 Fie n un număr natural. Scrie un algoritm care să calculeze n! (se citește „n factorial”).

```
citește n
f ← 1
┌pentru i ← 1, n execută
│        f ← f * i
└■
scrie f
```

Soluție

Prin definiție, n!=1*2*...*n, pentru orice n>0, iar 0!=1. Observăm că n! se obține înmulțind succesiv toate numerele naturale mai mici sau egale cu n. Pentru a calcula acest produs, vom utiliza o variabilă f. Inițial variabila f va fi 1 (pentru că 1 este elementul neutru la înmulțire), apoi vom parcurge numerele naturale de la 2 la n, înmulțind succesiv variabila f cu fiecare dintre aceste numere.

Să exersăm!

3 Urmărește pas cu pas execuția algoritmului anterior pentru n=6.

4 Scrie un algoritm echivalent care să utilizeze instrucțiunea repetitivă cât timp.

Putere

5 Fie n un număr natural și x un număr real. Scrie un algoritm care calculează pe x^n.

Soluție

$$x^n = \underbrace{x*x*...*x}_{\text{de } n \text{ ori}} = \underbrace{(x*x*...*x)}_{\text{de } n-1 \text{ ori}} *x = x^{n-1}*x$$

Evident, $x^0=1$.

Vom utiliza o variabilă p în care vom calcula puterea. La fiecare pas i (i=1, ..., n) vom înmulți valoarea variabilei p cu x.

```
citește n, x
p ← 1
┌pentru i ← 1, n execută
│        p ← p * x
└■
scrie p
```

Maxim

6 Se citește de la tastatură un număr natural nenul `n`, apoi se citesc succesiv `n` valori reale. Scrie un algoritm care determină și afișează cea mai mare valoare reală citită.

Soluție

O primă problemă este declararea datelor de intrare: ni se dau `n` valori reale, numai că de data aceasta `n` nu este constant, ci variabil (citit de la tastatură). Cum să declarăm un număr variabil de variabile? Nu putem și nici nu este necesar, pentru că nu este nevoie să citim fiecare număr într-o variabilă separată. Vom citi numerele succesiv în aceeași variabilă (să o numim `a`) și imediat ce le-am citit le vom compara cu elementul maxim.

Pentru a intui mai bine algoritmul de calcul al maximului, să ne imaginăm un șir de elevi, care intră în clasă unul câte unul. Am dori ca pe măsură ce intră să reținem care este cel mai înalt. Evident, primul care intră poate fi considerat cel mai înalt (de altfel, este singurul pe care l-am văzut până acum). Când intră pe ușă un nou elev, compar înălțimea lui cu înălțimea celui pe care eu îl consider cel mai înalt de până acum. Dacă noul elev este mai înalt, reține înălțimea lui ca etalon pentru viitoarele comparații (el este cel mai înalt de până acum). Prin urmare, noi reținem permanent maximul dintre cei intrați deja.

Această idee o vom utiliza în continuare: în variabila `maxim` vom reține la fiecare moment cea mai mare valoare citită până la momentul respectiv.

```
citește n, maxim // inițializăm maxim cu primul element citit
i ← 1            // i indică numărul de valori citite
┌cât timp i < n execută
│          // cât timp nu am citit toate cele n valori
│          citește a                // citesc o nouă valoare
│          i ← i+1                  // număr valoarea citită
│         ┌dacă maxim < a atunci // compar cu maxim
│         │     maxim ← a
│         └■
└■
scrie "maxim= ", maxim
```

7 Rescrie algoritmul precedent utilizând o instrucțiune repetitivă cu număr cunoscut de pași.

8 Modifică algoritmul precedent astfel încât să calculeze cel mai mic dintre elementele citite.

9 Modifică algoritmul precedent astfel încât să calculeze cele mai mari două elemente din cele citite.

10 Scrieți un algoritm echivalent, înlocuind structura repetitivă `cât timp` cu o structură repetitivă cu test final.

Stație meteorologică

11 Se citește un număr natural `n`, apoi o succesiune de `n` valori întregi reprezentând temperaturile înregistrate la o stație meteorologică pe parcursul a `n` zile consecutive. Scrie un algoritm care să determine numărul maxim de zile consecutive pentru care temperaturile au fost negative.

Soluție

Vom citi succesiv cele `n` valori de temperatură și vom determina în variabila `lg` lungimea secvenței curente formate din temperaturi negative. Când citim un număr negativ, lungimea secvenței curente `lg` va crește cu `1`. Când citim un număr pozitiv, deducem că secvența curentă de temperaturi negative s-a terminat și comparăm lungimea ei cu lungimea maximă `lgmax`, actualizând `lgmax` dacă este cazul. Deoarece ultima secvență nu se termină cu întâlnirea unui element pozitiv, va trebui, la final, să comparăm și lungimea acesteia cu `lgmax`.

```
citește n
lg ← 0; lgmax ← 0
┌pentru i ← 1, n execută
│        citește t
│        ┌dacă t < 0 atunci
│        │     lg ← lg+1
│        │altfel
│        │      ┌dacă lg > lgmax atunci
│        │      │     lgmax ← lg
│        │      └■
│        │      lg ← 0
│        └■
└■
┌dacă lg > lgmax atunci
│     lgmax ← lg
└■
scrie lgmax
```

Să exersăm!

12 Modifică algoritmul astfel încât să determine și numărul de ordine al zilei în care începe cea mai lungă secvență de zile cu temperaturi negative. Dacă există mai multe astfel de secvențe, se va afișa ziua de început a primei secvențe în ordine cronologică.

16. Fișă de lucru aplicativă

1 Se consideră algoritmul alăturat descris în pseudocod:

```
citește a, b
i ← a; x ← 0
┌cât timp i ≤ b execută
│          x ← x+i
│          i ← i+1
└■
scrie x
```

a. Ce valoare va fi afișată pe ecran dacă se introduc valorile `1` și `10`?

b. Ce valori ar putea fi introduse pentru `a` și `b` astfel încât algoritmul să afișeze valoarea `22`?

c. Dă exemplu de valori distincte care ar putea fi introduse astfel încât algoritmul să afișeze `0`.

d. Scrie un algoritm echivalent mai eficient. Doi algoritmi sunt considerați echivalenți dacă pentru orice date de intrare ei produc aceleași rezultate.

2 Ce valoare inițială ar trebui să aibă variabila `x`, astfel încât după execuția următoarei secvențe de instrucțiuni să se afișeze valoarea `640`?

```
d ← 2
┌cât timp d < 50 execută
│          x ← x*d
│          d ← d*d
└■
scrie x
```

Medie

3 Se citește de la tastatură un număr natural nenul `n`, apoi se citesc succesiv cele `n` note obținute de un elev la disciplina informatică. Scrie un algoritm care determină și afișează media la disciplina informatică a elevului, precum și mesajul `PROMOVAT` (dacă media, după rotunjire, este mai mare sau egală cu `5`), respectiv mesajul `CORIJENT`, în caz contrar.

Pătrate perfecte

4 Se citește de la tastatură un număr natural `n`. Scrie un algoritm eficient care să afișeze toate pătratele perfecte mai mici decât `n`. Analizează complexitatea algoritmului.

Poluare

5 Pentru a monitoriza poluarea aerului cu particule fine în zona în care locuiesc am instalat pe balcon un senzor PM `2.5` (pentru particule cu diametrul ≤ `2.5` μm) care înregistrează la fiecare oră nivelul de poluare. Dacă nivelul de poluare este >`50` μg/m³, se consideră că există un risc serios pentru sănătate. Completează zonele lacunare din algoritmul următor, astfel încât să citească cele `n` valori ale nivelului de poluare înregistrate pe parcursul a `n` ore și să determine în variabila `lgmax` lungimea celei mai lungi perioade de timp în care poluarea a fost un risc serios pentru sănătate.

```
citește n
lg ← 0; lgmax ← 0
┌pentru i ← 1, n execută
│       citește p
│       ┌dacă p > 50 atunci
│       │     ...
│       │altfel
│       │     ...
│       └■
└■
┌dacă lg > lgmax atunci
│     lgmax ← lg
└■
```

Temperaturi

6 La o stație meteorologică s-au înregistrat `n` valori de temperatură pe parcursul unui an calendaristic. Scrie un algoritm care să citească valoarea `n`, apoi cele `n` valori de temperatură înregistrate și care să rezolve cerințele:

a. să determine temperatura minimă, temperatura maximă și temperatura medie;

b. să determine cea mai lungă secvență de zile consecutive în care temperaturile au alternat ca semn.

17. Evaluare

Timp de lucru: 50 de minute
Se acordă 10 puncte din oficiu

1 Selectați numele de variabile corecte din lista următoare:

a. `10x`
b. `2B||!2B`
c. `Nr elevi`
d. `Șoim`
e. `'x'`
f. `Clasa_9A` *10 p*

2 Care dintre următoarele expresii au valoarea `True` dacă și numai dacă numărul întreg memorat în variabila `x` aparține mulțimii {`0`, `1`}?

a. `x = 0 and x = 1`
b. `x = 0 or x = 1`
c. `not (x ≠ 0 and x ≠ 1)`
d. `not (x ≠ 0 or x ≠ 1)` *10 p*

3 Scrie o expresie care să aibă valoarea `True` dacă și numai dacă numerele întregi nenule memorate în variabilele `x` și `y` au semne contrare. *10 p*

4 Ioan are un teren dreptunghiular cu lățimea de `x` metri și lungimea de `y` metri (`x`, `y` numere naturale). Un metru de gard costă `P` lei (`P` număr natural). Scrie un algoritm care să determine și să afișeze suma de bani pe care Ioan trebuie să o aibă pentru a-și împrejmui întreg terenul. *10 p*

5 Rescrie corect următoarea secvență de instrucțiuni, respectând regulile de indentare și precizează ce se va afișa pe ecran, știind că variabila `a` are valoarea `7`, iar variabila `b` are valoarea `11`.

```
dacă a%2=1 atunci dacă b>a atunci dacă b+a%2=1 atunci scrie 1 altfel scrie 2
altfel scrie 3 altfel scrie 4
```

10 p

6 Se consideră algoritmul de mai jos, reprezentat în pseudocod, unde `n` este un număr natural nenul.

```
citește n
p ← 0; i ← 1
cât timp i ≤ n execută
    m ← i
    cât timp m%2 = 0 execută
        m ← [m/2]
    dacă m = 1 atunci
        p ← i
    i ← i+1
scrie p
```

A. Selectează variantele corecte:
- **a.** algoritmul conține o structură alternativă inclusă într-o structură repetitivă;
- **b.** algoritmul conține o structură repetitivă inclusă într-o structură repetitivă;
- **c.** algoritmul conține o structură repetitivă inclusă într-o structură alternativă;
- **d.** algoritmul conține o structură repetitivă urmată de o structură alternativă. *5 p*

B. Scrie valoarea afișată în urma executării algoritmului, dacă se citește numărul `12`. *10 p*

C. Scrie cel mai mic și cel mai mare număr care pot fi citite, astfel încât, pentru fiecare dintre acestea, valoarea afișată să fie `16`. *10 p*

7 Numerele piramidale se definesc ca fiind sumele parțiale ale șirului pătratelor perfecte `1`, `4`, `9`, `16`, `25`, ...
De exemplu, primele `5` numere piramidale sunt `1`, `5`, `14`, `30`, `55`. Scrie un algoritm care să afișeze primele `n` numere piramidale. *15 p*

AUTOEVALUARE

Reflectează asupra activității tale și completează o fișă asemenea celei de mai jos:

a. Prin rezolvarea exercițiilor am învățat .... .

b. Mi-a fost ușor să ...., dar am avut dificultăți la .... . Pentru a depăși aceste probleme, este nevoie să .... .

Principii de elaborare a unui program. Elemente de bază ale limbajului Python

2

Acest capitol are rolul de a asigura armonizarea experiențelor anterioare ale elevilor în domeniul programării și o tranziție unitară către limbajul de programare prevăzut de programa clasei a IX-a.

1. Noțiuni introductive

Ce este un limbaj de programare?

În capitolul precedent, pentru a descrie algoritmii am utilizat pseudocodul, un ansamblu de convenții ușor de înțeles și de utilizat de către oameni. Din păcate, calculatoarele nu „înțeleg" (deocamdată) un astfel de limbaj. Pentru a rezolva o problemă cu ajutorul calculatorului, trebuie să transpunem (să implementăm) algoritmul de rezolvare a problemei într-un limbaj de programare.

Descrierea unui algoritm într-un limbaj de programare se numește ***program***.

Un limbaj de programare reprezintă un mijloc de comunicare între programator și calculator. Pentru a defini cu rigurozitate un limbaj de programare, trebuie să descriem două aspecte ale limbajului:

- **sintaxa** – reprezintă totalitatea regulilor pe baza cărora se obțin elementele limbajului;
- **semantica** – definește semnificația construcțiilor sintactic corecte (a elementelor limbajului).

Observați că aceste aspecte se studiază și în cazul unui limbaj natural (de exemplu, limba română). Dar spre deosebire de limba română, unde dacă facem o greșeală gramaticală sau utilizăm greșit o expresie într-un context, sunt șanse destul de mari ca interlocutorul să ne înțeleagă, atunci când comunicăm cu un calculator, trebuie să respectăm cu exactitate regulile de comunicare. Calculatorul nu face presupuneri, nu ghicește ceea ce ați fi dorit să-i spuneți. Dacă nu vă „exprimați" corect, nu veți obține de la calculator decât mesaje de eroare.

Limbajele de programare pot fi clasificate după mai multe criterii.

După nivelul de abstractizare:

- Limbaje de nivel scăzut (*low-level*) – sunt limbaje dependente de arhitectura *hardware* a calculatorului (specifice unui tip de procesor), oferind control direct asupra memoriei și resurselor. Utilizarea acestor limbaje necesită cunoștințe detaliate despre funcționarea internă a calculatorului; aceste limbaje sunt dificil de citit și de scris pentru oameni, dar au performanță ridicată. De exemplu, limbajul mașină (singurul limbaj pe care un procesor îl „înțelege") utilizează instrucțiuni scrise în binar, specifice procesorului sau limbajul de asamblare (utilizează mnemonice[1] pentru a reprezenta instrucțiunile din limbajul mașină, păstrând însă o corespondență directă cu *hardware*-ul).
- Limbaje de nivel înalt (*high-level*) – sunt limbaje mai apropiate de limbajul natural, independente de hardware. Sunt mai ușor de învățat și de utilizat, având o sintaxă lizibilă și sunt portabile (codul poate rula pe platforme diferite). De exemplu, C++, Python, Java, JavaScript etc.

După modul de execuție a codului:

- Limbaje compilate – codul sursă este tradus integral în limbaj mașină de un *compilator* obținându-se un fișier executabil. De exemplu, C, C++.
- Limbaje interpretate – un program numit *interpretor* citește și execută codul linie cu linie în timp real (din acest motiv, codul executabil este mai lent decât în cazul limbajelor compilate). De exemplu, Python, JavaScript.

1. În limbajul de asamblare, mnemonicele sunt „nume" lizibile pentru instrucțiunile scrise în binar din limbajul mașină (de exemplu, `ADD`, `MOV`, `JMP`).

Ce este Python?

Python este un limbaj de programare de nivel înalt creat de programatorul olandez Guido van Rossum. Versiunea 1.0 a fost lansată oficial în 1994, la ora actuală fiind unul dintre cele mai utilizate limbaje de programare.

Caracteristicile limbajului Python

- portabilitate (este un limbaj multiplatformă – rulează atât pe Windows, Linux, cât și pe macOS);
- este un limbaj interpretat;
- suportă mai multe paradigme de programare (programare procedurală, programare orientată pe obiecte și elemente de programare funcțională), ceea ce îi conferă versatilitate, având numeroase domenii de aplicabilitate, cum ar fi inteligența artificială și învățarea automată, analiza și prelucrarea datelor, dezvoltarea aplicațiilor web, automatizarea proceselor;
- dispune de un ecosistem bogat: are o vastă bibliotecă standard, precum și module externe (de exemplu, *NumPy*, *Pandas*, *Scikit-learn* etc.) care extind funcționalitatea limbajului dincolo de operațiile de bază și măresc productivitatea programatorilor;
- există o comunitate largă, care asigură suport și resurse educaționale.

Atenție!

Limbajul Python este *case-sensitive* (adică în scrierea codului se face distincție între literele mari și mici).

Ce este un mediu de programare?

Mediul de programare este ansamblul de programe și instrumente care îi permit unui programator să scrie, să testeze, să ruleze și să corecteze programe.

Cu ajutorul unui mediu de programare, programatorul poate **să scrie codul sursă al programului** într-un mod organizat și clar. Editorul de cod evidențiază sintaxa limbajului, oferă sugestii și ajută la evitarea greșelilor de tastare, ceea ce face procesul de scriere mult mai eficient.

De asemenea, mediul permite **organizarea fișierelor proiectului**. Fiecare proiect poate conține mai multe fișiere sursă, resurse și biblioteci, iar mediul de programare oferă un mod structurat de a le gestiona, astfel încât programul să fie ușor de întreținut și extins.

Un alt rol esențial este acela de **rulare a programului și verificare a rezultatelor**. Mediul de programare oferă un interpretor sau un compilator care transformă codul scris într-o aplicație funcțională și afișează rezultatele într-o fereastră specială, permițând programatorului să vadă imediat dacă programul funcționează corect.

În final, mediul de programare ajută la **identificarea și corectarea erorilor**. Mesajele de eroare și instrumentele de depanare permit programatorului să descopere rapid problemele din cod și să le remedieze, astfel încât programul să devină funcțional.

Mediul de programare PyCharm

PyCharm este un mediu de programare (IDE – *Integrated Development Environment*) destinat limbajului de programare Python. PyCharm oferă un editor de cod prietenos care evidențiază sintaxa limbajului Python, oferă sugestii pentru completarea instrucțiunilor și semnalează erorile de scriere încă din momentul tastării. De asemenea, permite organizarea proiectelor și fișierelor, astfel încât programele să fie ușor de gestionat chiar și atunci când devin complexe.

Mediul de programare PyCharm poate fi utilizat gratuit. Există și alte medii de programare pentru Python (de exemplu, IDLE Python, Spyder etc.).

Instalarea mediului de programare PyCharm

Pentru a folosi PyCharm, mai întâi trebuie să îl descărcăm de pe site-ul oficial. Pentru acest lucru, putem căuta pe Internet „*PyCharm Download*” și descărca versiunea potrivită sistemului de operare pe care îl utilizăm. După ce descărcarea s-a finalizat, lansăm în execuție programul de instalare. Instalarea este asistată de un *wizard* (o succesiune de ferestre de dialog care ne ghidează în procesul de instalare).

Componentele interfeței PyCharm

După ce am instalat PyCharm, îl putem deschide și putem crea un proiect nou. În cadrul unui proiect, vom organiza toate fișierele care conțin codul Python. Fiecare fișier (modul) reprezintă un program sau o parte a programului.

Gestionarea proiectelor în PyCharm

1 Observă în *Imaginea 1* cum poți gestiona un proiect în PyCharm.

Crearea unui proiect

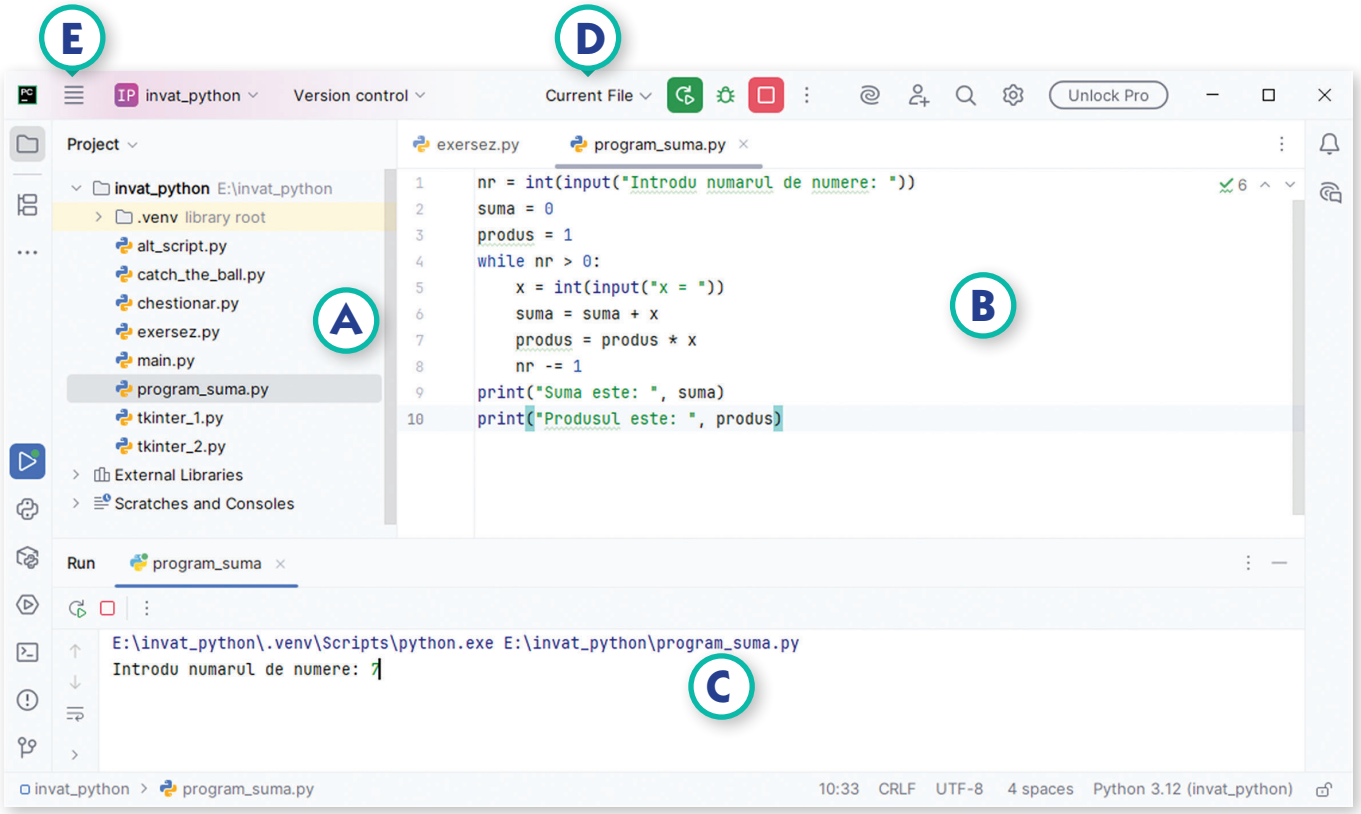
Din meniul principal (*Main menu* – zona E) selectăm opțiunea *New Project*. Va apărea o fereastră de dialog în care specificăm în zona *Location* dosarul în care dorim să fie salvate fișierele proiectului (PyCharm va crea automat dosarul).

Deschiderea unui proiect existent

Din meniul principal (*Main menu*) selectăm opțiunea *Open*, apoi selectăm dosarul proiectului.

Închiderea unui proiect

Din meniul principal (*Main menu*) selectăm opțiunea *Close Project*.



Imaginea 1

Gestionarea fișierelor Python în PyCharm

Crearea unui fișier

După ce am deschis un proiect, pentru a adăuga un fișier Python nou trebuie să facem clic dreapta pe folderul proiectului în *Project Explorer* sau să acționăm butonul + și să alegem *New* → *Python File*, apoi să introducem numele fișierului. Numele fișierului trebuie să fie descriptiv și să nu conțină spații. Toate fișierele Python au extensia `.py`.

> Un fișier cu extensia `.py` se numește ***modul***.

După ce fișierul a fost creat, acesta va apărea în zona *Project Explorer* (A), iar zona de editor de cod (B) va fi activă, gata să scriem instrucțiunile Python.

Salvarea fișierelor

Salvarea fișierelor în PyCharm se face automat, dar dacă dorim să forțăm salvarea putem utiliza combinația de taste `CTRL+S`.

Editorul de cod

Editorul de cod (B) este zona principală în care vom scrie, edita și salva programele Python.

Executarea programului

După ce am creat fișierul Python și am scris codul, trebuie să rulăm programul pentru a vedea rezultatele.

Pentru a executa programul, se acționează butonul ▷ *Run* sau combinația `Shift + F10`.

Programul se va executa, iar rezultatul va apărea în fereastra *Run Console* (C) din partea de jos a PyCharm. Dacă există erori, PyCharm le subliniază în cod și oferă detalii în consola de rulare, astfel încât să putem corecta problemele.

Atenție la configurația de rulare (Run Configuration)

Lângă butonul *Run* se află zona *Run Configuration* (D), care stabilește ce fișier Python este rulat și cum este rulat programul. În PyCharm, fiecare fișier Python poate avea propria configurație de rulare. Dacă este selectată o altă configurație, PyCharm va rula un alt fișier, chiar dacă în editor este deschis un program diferit. De aceea, înainte de a apăsa *Run*, este important să verificăm numele fișierului afișat în *Run Configuration* și dacă acesta corespunde programului pe care dorim să îl rulăm sau să îl depanăm.

În *Imaginea 1*, configurația selectată este *Current File*, ceea ce înseamnă că va fi rulat fișierul deschis în editor.

Depanarea programelor în PyCharm

Depanarea este procesul prin care identificăm și corectăm erorile dintr-un program. În PyCharm, depanarea (*debugging*-ul) ne permite să executăm programul pas cu pas, să vedem valorile variabilelor în timpul rulării și să înțelegem mai bine cum funcționează algoritmul.

Pornirea procesului de depanare se poate realiza în următoarele moduri:

1. folosind butonul special *Debug*;
2. prin *clic* dreapta în editor sau pe numele fișierului și selectarea opțiunii *Debug*;
3. din meniul *Run* → *Debug*.

2. Introducere în Programarea Orientată pe Obiecte (POO)

Această paradigmă de programare a apărut în contextul în care era necesară creșterea productivității în domeniul *software*, principalul avantaj al POO fiind modularizarea (modulele pot fi dezvoltate separat, pot fi modificate separat fără a afecta funcționalitatea întregii aplicații, pot fi reutilizate în alte aplicații, pot fi extinse ulterior). Ideea principală care stă la baza acestui stil de programare este de a organiza codul utilizând clase și obiecte.

În dicționar, termenul *clasă* este definit ca o mulțime de elemente cu însușiri comune. Prin însușiri comune putem înțelege atribute comune sau comportamente comune.

De exemplu, să considerăm clasa câinilor. Toți câinii au atribute și comportamente comune. Toți câinii au blană, deci un atribut interesant ar putea fi lungimea părului. Toți câinii latră – iată un comportament comun.

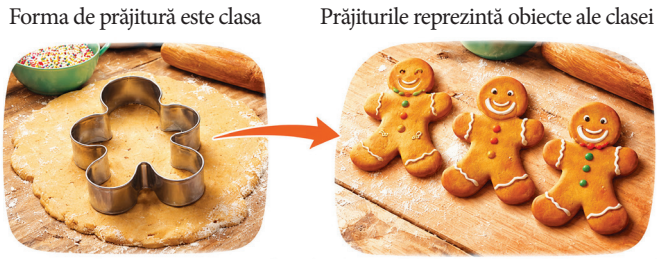
În esență, o clasă reprezintă exact același lucru și în cazul *POO*. Clasa definește caracteristicile comune (atribute, comportamente) ale obiectelor clasei.

O clasă conține:

- date membre (acestea corespund atributelor);
- funcții membre, denumite metode (acestea corespund comportamentelor).

Un ***obiect*** este o instanță a unei clase.

O analogie relevantă pentru a înțelege mai bine conceptele de clasă și obiect este analogia cu forma de prăjituri. Forma de prăjituri este clasa; ea definește aspectul tuturor prăjiturilor care vor fi tăiate cu forma respectivă. Prăjiturile pe care le tăiem cu forma respectivă reprezintă obiectele clasei (*Imaginea 2*).



Imaginea 2

Accesul la membrii unui obiect (date, metode) se realizează cu operatorul . astfel: **`obiect.membru`**.

Observație

În limbajul Python există numeroase clase predefinite (de exemplu, clasa `int`, clasa `str`, clasa `list` etc.), dar programatorul are posibilitatea de a-și defini propriile clase.

3. Variabile în limbajul Python

Conceptul de variabilă a fost introdus în contextul elaborării algoritmilor. În continuare vom introduce aspecte specifice variabilelor în limbajul Python.

Crearea variabilelor

În limbajul Python variabilele nu se declară. Variabila este creată atunci când primește o valoare (*Tabelul 1*).

Tabelul 1

Exemplu	Explicație
`varsta = 16`	Am creat o variabilă denumită `varsta` căreia i-am asociat valoarea `16`.
`raza = 2.5`	Am creat o variabilă denumită `raza` căreia i-am asociat valoarea `2.5`.
`ok = True`	Am creat o variabilă denumită `ok` căreia i-am asociat valoarea `True`.
`nume_elev1 = "Pop"`	Am creat o variabilă denumită `nume_elev1` căreia i-am asociat valoarea `"Pop"`.
`nume_elev2 = 'Ion'`	Am creat o variabilă denumită `nume_elev2` căreia i-am asociat valoarea `'Ion'`.

Un ***nume*** de variabilă în Python este o succesiune de litere, cifre și *underscore* (_), care nu poate începe cu o cifră și nu poate coincide cu un cuvânt rezervat al limbajului.

Recomandarea PEP 8[2] pentru numele de variabile este să fie scrise cu litere mici și, în cazul în care sunt formate din mai multe cuvinte, acestea să fie separate prin *underscore* (*snake case*). De exemplu, `medie_generala`.

Tipul unei variabile

Tipul variabilei definite este determinat dinamic, în timpul execuției programului, în funcție de valoarea asociată.

Un ***tip de date*** definește valorile pe care le pot lua variabilele de tipul respectiv, precum și operațiile care pot fi efectuate cu aceste valori.

Limbajul Python este un limbaj care folosește paradigma programării orientate pe obiecte, toate tipurile de date predefinite fiind clase.

2. PEP 8 (*Python Enhancement Proposal*) este ghidul oficial de stil pentru codul Python (reguli pentru a scrie cod Python clar, lizibil și profesionist).

Exemple de tipuri predefinite în limbajul Python (Tabelul 2):

Tabelul 2

Clasa	Explicație	Exemplu
`int`	Numere întregi (pozitive și negative), numărul maxim de cifre fiind limitat doar de memoria disponibilă.	Variabila `varsta`
`float`	Numere reale (raționale) scrise în forma `±parte_întreagă.parte_zecimală`	Variabila `raza`
`bool`	Valorile logice (`True`, `False`)	Variabila `ok`
`str`	Șiruri de caractere, valorile fiind succesiuni de caractere Unicode delimitate de ghilimele sau de apostrofuri.	Variabilele `nume_elev1` și `nume_elev2`

Observație

Pentru a determina tipul unei variabile/expresii putem apela funcția predefinită `type()`, specificând ca parametru numele variabilei, respectiv expresia.

Citirea variabilelor de la tastatură

Pentru a citi valori de la tastatură se utilizează funcția predefinită `input()`. Aceasta poate avea ca parametru un șir de caractere care se va afișa pe ecran înainte ca utilizatorul să introducă valoarea (*Tabelul 3*).

Tabelul 3

Exemplu	Explicație
`a=input("Variabila a= ")`	Pe ecran va apărea mesajul `Variabila a=` Apoi utilizatorul va introduce o valoare urmată de acționarea tastei *Enter*, valoare care va fi atribuită variabilei `a`.
`b = input()`	Pe ecran nu va apărea niciun mesaj; utilizatorul va introduce o valoare urmată de acționarea tastei *Enter*, valoare care va fi atribuită variabilei `b`.

Observație

Valorile citite de la tastatură sunt șiruri de caractere. Cu alte cuvinte, apelurile `type(a)`, respectiv `type(b)` vor returna ca rezultat `<class 'str'>`. Dacă dorim ca tipul variabilei să nu fie `str`, există posibilitatea ca la citire să convertim explicit tipul variabilei folosind o funcție predefinită de conversie de tip, cum ar fi `int()`, `float()`, `bool()`. Evident, pentru ca șirul de caractere să poată fi convertit la tipul dorit, conținutul său trebuie să corespundă cu tipul (de exemplu, pentru tipul `int` trebuie să conțină doar cifre, eventual precedate de semnul algebric).

Exemplu:

```
a = int(input("Variabila a= "))
b = float(input())
```

În acest caz tipul variabilei `a` va fi clasa `int`, iar tipul variabilei `b` va fi clasa `float`.

Afișarea pe ecran a unei valori

Valoarea unei expresii (și implicit valoarea unei variabile) poate fi afișată pe ecran apelând funcția predefinită `print()`.

`print(expresie`$_1$`, expresie`$_2$`, …, expresie`$_k$`)`

Efect

Se afișează în ordine, pe o singură linie, valorile expresiilor separate prin câte un singur spațiu.

Observație

Implicit valorile expresiilor sunt separate prin spațiu, iar la final se adaugă marcajul de sfârșit de linie (caracterul `'\n'`). Se poate schimba atât separatorul dorit (atribuind parametrului denumit `sep` șirul de caractere care va fi utilizat ca separator), cât și marcajul de final (atribuind parametrului denumit `end` șirul de caractere care va fi afișat la final). În *Tabelul 4* este prezentat un exemplu în care sunt utilizați parametrii `sep` și `end`.

Tabelul 4

Exemplu	Explicație
`a = 25`	Se creează variabila `a` cu valoarea `25`.
`b = input("b= ")`	Pe ecran va apărea mesajul `b=`, apoi utilizatorul va introduce o valoare, urmată de apăsarea tastei *Enter*, valoare care va fi atribuită variabilei `b` (de exemplu, `13`).
`print(a,type(a))`	Se afișează o linie pe care va fi scris: `25 <class 'int'>`
`print(b,type(b))`	Se afișează o linie pe care va fi scris: `13 <class 'str'>`
`print(a,b,sep=",")`	Se afișează o linie pe care valorile vor fi separate prin caracterul virgulă: `25,13`
`print("Am",b,"mere", end="!!!")` `print(a,"perfect",sep="...",end="?")`	Se afișează: `Am 13 mere!!!25...perfect?`

Afișarea cu formatare

Afișarea cu formatare permite includerea valorilor variabilelor/expresiilor direct în textul care urmează să fie afișat, într-un mod clar și ușor de citit.

Sintaxa

```
print(f"text{variabila/expresie}text{variabila/expresie}...")
```

Semantica

Caracterul `f` indică faptul că urmează scriere cu formatare. După litera `f` urmează un șir de caractere care conține textul ce urmează a fi afișat. În acest text pot fi incluse între paranteze acolade variabile sau expresii (pentru acestea, pe ecran va fi afișată valoarea variabilei sau expresiei respective), așa cum vei observa în *Tabelul 5*.

Tabelul 5

Exemplu	Ce se afișează pe ecran
`varsta = 16` `print(f"Elevul are {varsta} ani")`	`Elevul are 16 ani`

Observație

Se pot specifica și alte elemente de formatare în ceea ce privește valorile variabilelor sau expresiilor. De exemplu, dacă dorim să specificăm numărul de zecimale afișate pentru o valoare reală, după numele variabilei sau după expresie specificăm: `.<nr>f`, unde `nr` reprezintă numărul de zecimale dorite (*Tabelul 6*).

Tabelul 6

Exemplu	Ce se afișează pe ecran
`varsta = 17` `print(f"Elevul are {varsta/3:.3f} ani")`	`Elevul are 5.667 ani`

Concluzie

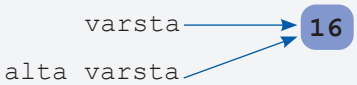
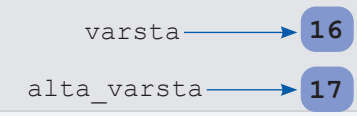
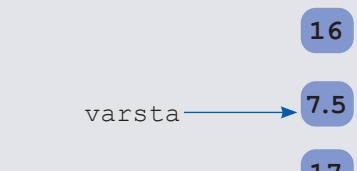
Să explicăm mai detaliat ce se întâmplă atunci când se execută o instrucțiune de forma

```
variabilă = expresie
```

1. Se creează un obiect al clasei corespunzătoare tipului expresiei; pentru a identifica în mod unic acest obiect, i se asociază un număr – identificator unic (este returnat de funcția predefinită `id()`).
2. Obiectul creat primește ca valoare valoarea expresiei (această valoare este memorată în zona de memorie alocată obiectului respectiv).
3. Variabila va indica obiectul creat (deci, variabila nu este obiectul, ci o referință la obiectul respectiv).

Urmărește exemplul din *Tabelul 7*:

Tabelul 7

Instrucțiuni	Explicație	Vizualizare
`varsta = 16`	Se creează un obiect al clasei `int`, i se atribuie valoarea `16`, iar variabila `varsta` va indica acest obiect.	
`alta_varsta = varsta`	Variabila `alta_varsta` va indica același obiect (cel cu valoarea `16`).	
`print(varsta, id(varsta))`	Pe ecran se afișează: `16 140736920963992` (a doua valoare afișată este identificatorul unic al obiectului, care se schimbă la fiecare execuție a codului)	
`print(alta_varsta, id(alta_varsta))`	Pe ecran se afișează: `16 140736920963992` (a doua valoare afișată este identificatorul unic al obiectului, deci cele două variabile indică același obiect)	
`alta_varsta = 17`	Se creează un alt obiect al clasei `int`, i se atribuie valoarea `17`, iar variabila `alta_varsta` va indica acest obiect	
`print(alta_varsta, id(alta_varsta))`	Pe ecran se afișează: `17 140736920964024`	
`varsta = 7.5`	Se creează un obiect al clasei `float`, i se atribuie valoarea `7.5`, iar variabila `varsta` va indica acest obiect. În acest moment, obiectul cu valoarea `16` nu mai este referit de nicio variabilă (el este „*orfan*”). În acest caz, un program denumit *garbage collector* identifică zonele de memorie care nu mai sunt referite de nicio variabilă și eliberează memoria alocată pentru a fi reutilizată.	
`print(varsta, id(varsta))`	Pe ecran se afișează: `7.5 2808498596976`	

4. Expresii

În mod similar cu variabilele, și în cazul expresiilor vom introduce în continuare elemente suplimentare specifice limbajului Python.

Operatori aritmetici

În limbajul Python există, în plus față de cei deja studiați în pseudocod, următorii operatori aritmetici (*Tabelul 8*):

Tabelul 8

Operator	Semnificație	Exemple	Valori afișate

`//` (binar)	Câtul împărțirii întregi	`a = 17`
		`print(a//5)`
		`print(a/5)`
			`3`
			`3.4`

`**` (binar)	Ridicare la putere (exponențiere)	`a = -2`
		`b = 2`
		`print(b**5)`
		`print(b**a)`
			`32`
			`0.25`

Operatori de atribuire

Descoperă în *Tabelul 9* operatorii de atribuire.

Tabelul 9

Operator	Utilizare	Exemple	Explicații
Operatorul de atribuire =	`variabilă = expresie` Se atribuie variabilei din partea stângă a operatorului = valoarea expresiei din partea dreaptă.	`a = 2*5` `print(a)`	Se afișează: `10`

Operatori de atribuire compuși
`op=`
	`variabilă op= expresie`
	(unde `op` poate fi orice operator aritmetic binar) este echivalentă cu:
	`variabilă=variabilă op expresie`
		`a = 5`
		`print(a,id(a))`
		`a **= 2`
		`print(a,id(a))`
		`a += 5`
		`print(a,id(a))`
		`a *= 10`
		`print(a,id(a))`
			Se afișează:
			`5 140735094803000`
			`25 140735094803640`
			`30 140735094803800`
			`300 3027550121424`
			Observați că:
			• `a**=2` e echivalent cu `a=a**2`
			• `a+=5` e echivalent cu `a=a+5`
			• `a*=10` e echivalent cu `a=a*10`
			• După fiecare operație, variabila `a` indică un alt obiect.

Observație

Operatorul de atribuire simplă = poate fi utilizat înlănțuit. De exemplu, `a=b=c=10` (toate cele trei variabile vor indica un obiect cu valoarea `10`).

Operatori relaționali

Operatorii relaționali din limbajul Python sunt: < (mai mic), > (mai mare), <= (mai mic sau egal), >= (mai mare sau egal), == (egal), `!=` (diferit).

Acești operatori compară **valorile** variabilelor și returnează un rezultat logic, după caz.

Pentru a verifica dacă două variabile indică sau nu același obiect (deci pentru a compara **referințele**, nu valorile) au fost introduși în limbajul Python alți doi operatori (*Tabelul 10*):

- `is` returnează `True` dacă ambii operanzi indică același obiect, respectiv `False` în caz contrar;
- `is not` returnează `True` dacă ambii operanzi indică obiecte diferite, respectiv `False` în caz contrar.

Tabelul 10

Exemplu	Valori afișate

`a = 5`
`b = 5`
`print(a is b)`
`b += 1`
`print(a is not b)`
	`True`
	`True`

Identitatea obiectelor poate să depindă de modul intern de gestionare a memoriei în Python.

Prioritatea operatorilor

Descoperă în *Tabelul 11* operatorii studiați până acum în ordinea descrescătoare a priorității.

Tabelul 11

Operatori	Simbol
Operatorul de selecție	`.`
Operatorul de exponențiere	`**`
Operatorii unari semn algebric	`+ -`
Operatorii aritmetici multiplicativi	`* / % //`
Operatorii aritmetici aditivi	`+ -`
Operatorii relaționali	`< > <= >= == != is is not`
Operatorul de negație logică	`not`
Operatorul de conjuncție logică	`and`
Operatorul de disjuncție logică	`or`
Operatorii de atribuire	`= += -= *= /= //= %= **=`

5. Fișă de autoevaluare

1 Variabilele `a` și `b` sunt de tip `<class 'float'>`. Selectează expresiile corespunzătoare acesteia: $\frac{a \cdot b - 2}{5 \cdot b} - \frac{7}{2}$.

a. `a * b - 2 / 5 * b - 7 / 2`
b. `(a * b - 2) / 5 * b - 7 / 2`
c. `(a * b - 2) / (5 * b) - 7 / 2`
d. `(a * b - 2) / 5b - 7 / 2`
e. `((a * b - 2) / 5 * b - 7) / 2`
f. `((a * b - 2) / (5 * b) - 7) / 2`

2 Știind că variabila de tip `<class 'int'>` `n` memorează un număr natural de exact `5` cifre, scrie o expresie care să aibă ca valoare cifra din mijloc a lui `n`.

3 Scrie o expresie care să aibă valoarea `True` dacă și numai dacă numerele întregi nenule memorate în variabilele `x` și `y` au ca divizor comun numărul natural nenul memorat în variabila `d`.

4 Selectează expresiile care au valoarea `True` dacă și numai dacă suma de bani `S` poate fi împărțită în mod egal între doi frați și fiecare frate poate să-și cumpere apoi o înghețată de `P` lei.

a. `S % 2 != 0 and S // 2 >= P`
b. `not (S % 2 !=0 or S // 2 < P)`
c. `S % 2 == 0 AND S // 2 >= P`
d. `S // 2 == 0 and S % 2 >= P`

5 Selectează expresiile care determină inserarea cifrei `7` în fața ultimei cifre a unui număr natural, cu mai mult de două cifre, memorat în variabila de tip `<class 'int'>` `x`.

a. `x = (x // 10 * 10 + 7) * 10 + x % 10`
b. `x = (x % 10 * 10 + 7) * 10 + x // 10`
c. `x = x // 10 + 7 + x % 10`
d. `x = (x // 10 + 7) * 10 + x % 10`

6 Selectează expresiile care au valoarea `True` dacă și numai dacă numărul întreg memorat în variabila de tip `<class 'int'>` `x` **nu** aparține intervalului $(-10,-2) \cup [50,100]$.

a. `(x <= -10) or (x < 50 and x >= -2) or (x > 100)`
b. `(x <= -10) or (x <= 50 and x >= -2) and (x >= 100)`
c. `(x < -10) or (x < 50 and x > -2) and (x > 100)`
d. `(x <= -10) or (x <= 50 and x >= -2) and (x > 100)`

6. Structura alternativă (de decizie)

Sintaxa

```
if expresie_logică_1:
     Instrucțiuni_1
elif expresie_logică_2:
     Instrucțiuni_2
elif expresie_logică_3:
     Instrucțiuni_3
    ...
else:
     Instrucțiuni
```

Semantica

Pas 1: Se evaluează `expresie_logică_1`. Dacă valoarea acestei expresii este `True`, se execută `Instrucțiuni_1`, apoi se iese din structura alternativă; în caz contrar, se trece la pasul următor.

Pas 2: Se evaluează `expresie_logică_2`. Dacă valoarea acestei expresii este `True`, se execută `Instrucțiuni_2`, apoi se iese din structura alternativă; în caz contrar, se trece la pasul următor.

Pas 3: Se evaluează `expresie_logică_3`. Dacă valoarea acestei expresii este `True`, se execută `Instrucțiuni_3`, apoi se iese din structura alternativă; în caz contrar, se trece la pasul următor.

…

Pasul final: Dacă niciuna dintre expresiile evaluate la pașii precedenți nu a avut valoarea `True` și ramura `else` există, se execută `Instrucțiuni`.

Observații

1. Structura alternativă poate conține `0`, `1`, `2` sau mai multe ramuri `elif`.
2. Ramura `else` este opțională (ea poate lipsi dacă nu este necesar să executăm instrucțiuni pentru cazul în care toate expresiile de pe ramurile precedente au avut valoarea `False`).

Indentarea

În limbajul Python indentarea codului este obligatorie.

Indentarea este deplasarea codului spre dreapta, folosind spații sau tab-uri, pentru a indica ce instrucțiuni se execută împreună.

Regulă

Toate liniile care fac parte din același bloc de instrucțiuni se indentează în același mod.

Observații

1. În sintaxa instrucțiunii `if` sunt indicate mai multe blocuri de instrucțiuni (`Instrucțiuni_1`, `Instrucțiuni_2` etc.). Pentru a indica interpretorului Python că aceste instrucțiuni sunt subordonate instrucțiunii `if`, acestea trebuie să fie scrise indentat.
2. Recomandarea PEP 8 este ca pentru indentare să fie folosite 4 spații.

Exemplu:

Următoarea secvență de cod afișează modulul numărului memorat în variabila `a` și este indentată corect:

```
if a >= 0:
    print(a)
else:
    print(-a)
```

Instrucțiunea if „one line”

Sintaxa Python permite și scrierea instrucțiunii `if` în modul „*one line*” (adică toate instrucțiunile dintr-un bloc de instrucțiuni pot fi scrise pe aceeași linie, separate prin caracterul `';'`).

Codul devine mai compact, dar nu întotdeauna este mai lizibil. Este recomandat să utilizăm această sintaxă doar atunci când o ramură a instrucțiunii conține doar instrucțiuni simple și clare (atribuire, citire, afișare). Pentru blocuri de instrucțiuni mai complexe, utilizați indentarea.

Exemple:

1.

```
maxim = a
if maxim < b: maxim = b
```

2.

```
if a % 2 == 0: print("număr par"); a = a // 2
else: print("număr impar"); a = (a + 1) // 2
print(a)
```

7. Probleme rezolvate

Maxim 2

1 Se citesc de la tastatură două numere întregi `x` și `y`. Scrie un program care determină și afișează cea mai mare dintre valorile citite (valoarea maximă).

Soluție

Un algoritm de rezolvare a problemei a fost prezentat în unitatea de învățare „*Principii de elaborare a unui program. Fundamente ale elaborării algoritmilor*”. Să implementăm acest algoritm în Python (*Tabelul 12*):

Tabelul 12

Algoritm în pseudocod	Program în limbajul Python

citește x, y
┌**dacă** x > y **atunci**
│ maxim ← x
│**altfel**
│ maxim ← y
└■
scrie maxim
	x = int(input("x= "))
	y = int(input("y= "))
	if x > y:
	  maxim = x
	else:
	  maxim = y
	print(maxim)

Maxim 3

2 Se citesc de la tastatură trei numere întregi `x`, `y` și `z`. Scrie un program care determină și afișează cea mai mare dintre valorile citite (valoarea maximă).

Soluție

Un algoritm de rezolvare a problemei a fost prezentat în unitatea de învățare „*Principii de elaborare a unui program. Fundamente ale elaborării algoritmilor*”. Să implementăm acest algoritm în Python (*Tabelul 13*):

Tabelul 13

Algoritm în pseudocod	Program în limbajul Python

citește x, y, z
maxim ← x
┌**dacă** y > maxim **atunci**
│ maxim ← y
└■
┌**dacă** z > maxim **atunci**
│ maxim ← z
└■
scrie maxim
	x = int(input("x= "))
	y = int(input("y= "))
	z = int(input("z= "))
	maxim = x
	if y > maxim:
	  maxim = y
	if z > maxim:
	  maxim = z
	print(maxim)

Competențe digitale

3 La examenul de bacalaureat, proba de competențe digitale, punctajul obținut de candidat este convertit într-un calificativ ca în *Tabelul 14*.

Scrie un program în limbajul Python care citește un număr natural reprezentând punctajul obținut de un elev la proba de competențe digitale și afișează calificativul elevului.

Tabelul 14

Interval	Calificativ
[75, 100]	Experimentat
[56, 74]	Avansat
[31, 55]	Mediu
[0, 30]	Începător

Soluție

Vom compara, în ordine, punctajul elevului cu punctajul minim necesar pentru fiecare calificativ (*Tabelul 15*).

Tabelul 15

Algoritm în pseudocod	Program în limbajul Python

citește pct
dacă pct >= 75 **atunci**
  scrie "Experimentat"
altfel
  dacă pct >= 56 **atunci**
  scrie "Avansat"
  altfel
  dacă pct >= 31 **atunci**
  scrie "Mediu"
  altfel
  scrie "Incepator"
	pct = int(input("punctaj= "))
	if pct >= 75:
	  print("Experimentat")
	elif pct >= 56:
	  print("Avansat")
	elif pct >= 31:
	  print("Mediu")
	else:
	  print("Incepator")

Portofoliu digital

Creează un director numit `<nume_elev>_Portofoliu_digital_Python` *unde vei înlocui* `<nume_elev>` *cu numele tău. Adaugă în el câte un folder pentru fiecare unitate de învățare, în care să îți păstrezi problemele rezolvate.*

1 Scrie un program care citește de la tastatură ora și minutul la care a ajuns la școală Marc, respectiv ora și minutul la care a ajuns la școală Ioan și afișează numele copilului care a ajuns primul sau numele ambilor copii, în cazul în care au ajuns simultan.

2 Scrie un program care citește de la tastatură două numere reale `a` și `b`, reprezentând coeficienții unei ecuații de gradul I de forma `ax+b=0` și afișează pe ecran soluția ecuației sau un mesaj în funcție de caz.

3 Scrie un program care citește de la tastatură un număr întreg reprezentând temperatura și afișează mesajul `"Foarte frig"` (dacă temperatura este sub `0°`), `"Răcoros"` (dacă temperatura este între `0°` și `15°`), `"Plăcut"` (dacă temperatura este între `16°` și `30°`), respectiv mesajul `"Caniculă"` (dacă temperatura este peste `30°`).

4 Scrie un program care citește de la tastatură un număr întreg reprezentând valoarea unei achiziții și, în funcție de valoarea acesteia, stabilește procentul de reducere. Dacă valoarea este cel puțin `300` lei, atunci se aplică o reducere de `20%`; pentru o valoare între `100` și `299` lei se aplică o reducere de `10%`, iar pentru o valoare sub `100` lei nu se aplică reduceri. Programul va afișa valoarea reducerii, precum și suma pe care clientul trebuie să o plătească după aplicarea reducerii.

5 Scrie un program care citește de la tastatură două numere întregi, apoi afișează pe ecran următorul text:

```
1.Suma celor două numere
2.Diferența celor două numere
3.Produsul celor două numere
4.Câtul și restul împărțirii primului număr la cel de-al doilea
Alegeți o opțiune (1, 2, 3 sau 4)!
```

Programul citește opțiunea utilizatorului și afișează rezultatul operației selectate sau mesajul "Opțiune inexistentă", în cazul în care utilizatorul a tastat altceva decât 1, 2, 3 sau 4.

Organizarea Portofoliului digital



Accesează manualul digital pentru a afla cum să îți organizezi portofoliul digital.

8. Structura repetitivă

În limbajul Python există două instrucțiuni repetitive:

- instrucțiunea `while`, care se folosește de obicei când repetarea acțiunilor este controlată de o anumită condiție și numărul de repetări nu este obligatoriu cunoscut;
- instrucțiunea `for`, care se folosește atunci când acțiunile se repetă de un număr cunoscut de ori.

În continuare vei descoperi instrucțiunea `while` și probleme rezolvate cu ajutorul acesteia. Apoi, vei descoperi instrucțiunea `for` și probleme rezolvate cu ajutorul acesteia.

A. Instrucțiunea while

Sintaxa

```
while expresie_logică:
      Instrucțiuni
```

Semantica

Pas 1: se evaluează `expresie_logică`;
Pas 2: dacă valoarea expresiei este `False`, se iese din instrucțiunea `while`;
dacă valoarea expresiei este `True`, se execută `Instrucțiuni`, apoi se revine la Pas 1.

Observații

Blocul `Instrucțiuni` este subordonat instrucțiunii `while`, deci trebuie să fie scris indentat.

9. Probleme rezolvate folosind instrucțiunea `while`

Maxim

1 Se citește de la tastatură un număr natural nenul `n`, apoi se citesc succesiv `n` valori întregi. Scrie un algoritm care determină și afișează cea mai mare valoare citită.

Soluție

Un algoritm de rezolvare a acestei probleme a fost prezentat în capitolul 1. Să îl implementăm în Python (*Tabelul 16*):

Tabelul 16

Algoritm în pseudocod	Program în limbajul Python

citește n, maxim
i ← 1
cât timp i < n **execută**
  citește a
  i ← i+1
  dacă maxim < a **atunci**
  maxim ← a
scrie "maxim= ", maxim
	n = int(input("n= "))
	maxim = int(input("Introduceti o valoare= "))
	i = 1
	while i < n:
	  a = int(input("Introduceti o valoare= "))
	  i += 1
	  if maxim < a:
	  maxim = a
	print("maxim= ", maxim)

Economii

2 O persoană economisește lunar o sumă variabilă. Scrie un program care citește suma economisită în fiecare lună și afișează numărul de luni trecute până când a reușit să economisească `10000` lei (*Tabelul 17*).

Tabelul 17

Algoritm în pseudocod	Program în limbajul Python

`nr_luni ← 0; economii ← 0`
`limita ← 10000`
`┌cât timp economii < limita execută`
`│ citește suma`
`│ economii ← economii+suma`
`│ nr_luni ← nr_luni+1`
`└■`
`scrie nr_luni`
	`nr_luni = economii = 0`
	`limita = 10000`
	`while economii < limita:`
	`  suma = int(input("Suma din luna curentă "))`
	`  economii += suma`
	`  nr_luni += 1`
	`print("Numărul de luni necesare= ", nr_luni)`

Să exersăm!

3 Scrie programul în limbajul Python pentru algoritmul de rezolvare a problemei *Secvență*, rezolvată în unitatea de învățare „*Principii de elaborare a unui program. Fundamente ale elaborării algoritmilor*”.

B. Instrucțiunea for

Sintaxă

```
for var in secvență_de_valori:
    Instrucțiuni
```

Semantică

În sintaxa instrucțiunii `for`:

- `secvență_de_valori` reprezintă o colecție de valori care poate fi parcursă (iterabilă);
- `var` este o variabilă care va primi succesiv fiecare valoare din secvență (vom spune că vom parcurge secvența cu ajutorul variabilei `var`).

Pentru fiecare valoare din `secvență_de_valori` atribuită variabilei `var` vor fi executate `Instrucțiuni`.

Observație

Blocul `Instrucțiuni` este subordonat instrucțiunii `for`, deci trebuie să fie scris indentat.

Funcția range()

Când vom studia structuri de date (de exemplu, în clasa a IX-a vom studia clasa `list`), vom învăța cum se parcurge secvența de valori memorată într-o astfel de structură. Pentru început, vom parcurge doar secvențe de valori generate cu ajutorul funcției `range()`.

Funcția `range()` poate fi apelată în trei moduri (*Tabelul 18*):

Tabelul 18

Apel	Explicație	Exemplu

`range(v_max)`	Generează o secvență formată din toate numerele întregi mai mici decât `v_max` (din intervalul `[0,v_max-1]`).	`for i in range(5):`
		`  print(i, end=" ")`
		După execuția acestei instrucțiuni se vor afișa pe ecran valorile: `0 1 2 3 4`

`range(v_min,v_max)`	Generează o secvență formată din numerele întregi din intervalul `[v_min, v_max-1]`.	`for i in range(7, 10):`
		`  print(i, end=" ")`
		După execuția acestei instrucțiuni se vor afișa pe ecran valorile: `7 8 9`

`range(v_min,v_max, pas)`	Generează o secvență de numere întregi care constituie o progresie aritmetică cu primul termen `v_min` și rația `pas`. Termenii vor fi <`v_max` (dacă `pas>0`) sau >`v_max` (dacă `pas<0`).	`for i in range(1, 10, 2):`
		`  print(i, end=" ")`
		Pe ecran se afișează cifrele impare:
		`1 3 5 7 9`
		`for i in range(5, 0, -1):`
		`  print(i, end=" ")`
		Pe ecran se afișează: `5 4 3 2 1`

10. Probleme rezolvate folosind instrucțiunea `for`

Stație meteorologică

1 Se citește un număr natural n, apoi o succesiune de n valori întregi reprezentând temperaturile înregistrate la o stație meteorologică pe parcursul a n zile consecutive. Scrie un program care să determine numărul maxim de zile consecutive pentru care temperaturile au fost negative.

Soluție

Să implementăm în limbajul Python algoritmul proiectat pentru această problemă în unitatea de învățare „*Principii de elaborare a unui program. Fundamente ale elaborării algoritmilor*” (*Tabelul 19*):

Tabelul 19

Algoritm în pseudocod	Program în limbajul Python

citește n
lg ← 0; lgmax ← 0
pentru i ← 1, n **execută**
  citește t
  dacă t < 0 **atunci**
  lg ← lg+1
  altfel
  dacă lg > lgmax **atunci**
  lgmax ← lg
  lg ← 0
dacă lg > lgmax **atunci**
  lgmax ← lg
scrie lgmax
	n = int(input("n="))
	lg = lgmax = 0
	for i **in** range(n):
	  t = int(input("t= "))
	  if t < 0:
	  lg += 1
	  else:
	  if lg > lgmax: lgmax = lg
	  lg = 0
	if lg > lgmax: lgmax = lg
	print(lgmax)

Pătrat de numere

2 Pentru un număr natural n dat se construiește un pătrat de latură n în care vor fi plasate numere după modelul următor (unde n=5):

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```

Scrie un program în limbajul Python (*Tabelul 20*) care să citească un număr natural n și care să afișeze pe ecran un pătrat de latură n construit după modelul de mai sus.

Tabelul 20

Algoritm în pseudocod	Program în limbajul Python

citește n
pentru nr ← 1, n **execută**
  pentru val ← nr, nr*n, nr **execută**
  scrie val, " "
  scrie "\n"
	n = int(input("n="))
	for nr **in** range(1, n + 1):
	  for val **in** range(nr, nr * (n + 1), nr):
	  print(val, end=" ")
	  print()

Termen Fibonacci

3 Fie șirul Fibonacci 1, 1, 2, 3, 5, 8, 13, 21, 34, … .

Primii doi termeni sunt egali cu 1, fiecare termen următor fiind egal cu suma celor doi termeni care îl precedă. Fie n un număr natural citit de la tastatură. Scrie un program care afișează cel de-al n-lea termen din șirul Fibonacci.

Soluție

Observăm că pentru a calcula un termen al șirului Fibonacci avem nevoie de doi termeni precedenți. Vom utiliza trei variabile, f0, f1 și f2. În f0 și f1 vom reține doi termeni Fibonacci consecutivi, iar în f2 vom calcula termenul următor. Variabilele f0 și f1 vor fi inițializate cu 1 (primii doi termeni din șirul Fibonacci). Următorii termeni (de la al treilea până la al n-lea) îi vom calcula succesiv. La fiecare pas i, în f2 vom calcula al i-lea termen din șirul Fibonacci (f2=f1+f0), apoi ne pregătim pentru pasul următor (f0=f1, f1=f2), așa cum observi în *Tabelul 21*:

Tabelul 21

Algoritm în pseudocod	Program în limbajul Python

citește n
f0 ← 1; f1 ← 1
┌**pentru** i ← 3, n **execută**
│ f2 ← f1+f0 //calculez termenul curent
│ f0 ← f1 // mă deplasez în șir
│ f1 ← f2
└■
scrie f1
	n = int(input("n="))
	f0 = f1 = 1
	for i **in** range(3, n + 1):
	  f2 = f1 + f0
	  f0 = f1
	  f1 = f2
	print(f1)

Să exersăm!

4 Modifică algoritmul anterior astfel încât să calculeze cel mai mare termen Fibonacci mai mic sau egal cu n.

11. Instrucțiunile break și continue

Instrucțiunea break determină ieșirea necondiționată din instrucțiunea repetitivă în care apare.

Instrucțiunea continue termină iterația curentă și trece la iterația următoare (*Tabelul 22*).

Tabelul 22

Exemplu	Explicație

suma = 0
while (True):
  a = int(input("a="))
  if a < 0: **break**
  suma += a
print(suma)
	Programul citește de la tastatură valori până la citirea unei valori negative și însumează valorile pozitive citite.
	Observați că instrucțiunea while este controlată de o expresie logică care este permanent adevărată (ciclu infinit). Ieșirea din while se realizează cu ajutorul instrucțiunii break.

for i **in** range(10):
  if i % 2 == 0: **continue**
  print(i)
	Se afișează pe ecran doar cifrele impare. Observați că atunci când i este par, instrucțiunea continue determină trecerea la următoarea iterație (următoarea valoare a lui i) fără a executa următoarele instrucțiuni din blocul de instrucțiuni incluse în for.

12. Comentarii în limbajul Python

O bună practică în programare este comentarea codului (atât pentru elevi, cât și pentru profesioniștii în domeniu). Comentariile sunt texte explicative (ignorate de interpretorul Python), care explică succint ce problemă/subproblemă rezolvă codul respectiv, precum și modul în care este rezolvată. Scopul comentariilor este de a face codul mai ușor de înțeles pentru colegii de echipă sau chiar pentru programatorul însuși, atunci când va reveni asupra codului său pentru a-și clarifica ideile.

În limbajul Python există două modalități de a introduce comentarii în cod:

- comentarii pe o singură linie: încep cu caracterul # și continuă până la sfârșitul liniei;
- comentarii pe mai multe linii: sunt încadrate de două secvențe """ (trei ghilimele).

Recomandări

1. Introduceți comentarii în cod:
   - la începutul programului → ce face programul;
   - la fiecare secvență de instrucțiuni care rezolvă o anumită subproblemă → ce subproblemă este rezolvată;
   - la formule → ce reprezintă formula respectivă.
2. Comentariile să fie clare, scurte, scrise în limbaj natural.
3. Nu abuzați de comentarii! Adică nu introduceți comentarii inutile, acolo unde semnificația codului este evidentă sau care repetă codul scris.

Portofoliu digital

1 ***Consum apă***

Scrie un program care citește o succesiune de valori reale reprezentând consumul zilnic de apă (exceptând ultima valoare care este `-1`) și care rezolvă următoarele cerințe:

- determină cantitatea totală de apă consumată;
- determină media consumului zilnic;
- determină ziua în care consumul de apă a fost maxim (dacă există mai multe zile în care consumul de apă a fost maxim, se va afișa cea mai recentă dintre acestea).

2 ***Steluțe***

Scrie un program care să citească de la tastatură un număr natural `n` și care să afișeze pe ecran un triunghi de steluțe după modelul următor (pentru care `n=4`).

```
   *
  ***
 *****
*******
```

3 ***Pătrat de numere***

Scrie un program care citește un număr natural `n` și care afișează un pătrat de numere după modelul următor (pentru care `n=4`).

```
4 4 4 4
4 3 3 3
4 3 2 2
4 3 2 1
```

Fișă de observare a comportamentului

Evaluează cu sinceritate activitatea desfășurată până acum și scrie simbolul X pentru fiecare enunț în coloana care ți se potrivește (*Tabelul 23*). Folosește această fișă de-a lungul anului școlar, după ce ai parcurs o unitate de învățare.

Tabelul 23

Comportamentul	Întotdeauna	Deseori	Uneori	Niciodată
M-am implicat în rezolvarea sarcinilor.				
Am respectat indicațiile cadrului didactic.				
Am solicitat sprijin/explicații atunci când am întâmpinat dificultăți.				
Am finalizat activitățile începute.				
Mi-am exprimat opiniile în mod adecvat.				

13. Evaluare

Timp de lucru: 50 de minute
Se acordă 10 puncte din oficiu

1 Analizează următoarele scenarii și identifică tipul adecvat pentru fiecare variabilă.

a. o variabilă în care să memorăm numărul de pagini al unei cărți;
b. o variabilă în care să memorăm dacă un utilizator este sau nu logat într-o aplicație;
c. o variabilă în care să memorăm un mesaj de eroare pe care o aplicație trebuie să îl afișeze;
d. o variabilă în care trebuie să memorăm prețul unui produs. *10 p*

2 Selectează variantele care conțin o instrucțiune repetitivă.

a. `break`
b. `for x in range(2,20,2): print(x)`
c. `while i * i < 100: print(i*i); i+=1`
d. `if a>0: print(-a)` *10 p*

3 Ce valori ar putea avea inițial variabila `nota`, care memorează un număr natural, astfel încât după executarea următoarei secvențe de instrucțiuni să se afișeze pe ecran mesajul `Foarte bine`?

```
if nota < 5:
    print("Insuficient")
elif nota < 7:
    print("Bine")
elif nota < 9:
    print("Foarte bine")
else:
    print("Excelent")
```
12 p

4 Precizează ce se afișează pe ecran după executarea următoarei secvențe de instrucțiuni.

```
p = 1
for i in range(8):
    print(p, end=' ')
    p *= 2
```
10 p

5 Precizează ce valoare inițială ar trebui să aibă variabila `n` astfel încât după execuția următoarei instrucțiuni să fie afișate pe ecran `8` caractere `#`.

```
while n > 2:
      print("##", end="")
      n -= 1
```
13 p

6 Completează spațiile lacunare din următoarea secvență de instrucțiuni, astfel încât, după executarea acesteia, pe ecran să se afișeze valorile alăturate.

```
for i in range(5):
    for j in range(5):
        …
```

```
1 2 3 4 5
10 9 8 7 6
11 12 13 14 15
20 19 18 17 16
21 22 23 24 25
```
15 p

7 La o firmă s-a scos la concurs un post la secretariat pentru care au aplicat `N` persoane, numerotate de la `1` la `N`, în ordinea în care au depus aplicațiile. Persoanele sunt invitate la interviu în ordinea de la `1` la `N` și în urma interviului, pentru fiecare persoană s-a calculat un scor.

Directorul firmei stabilește următoarele condiții pentru angajare:

- primele trei persoane nu sunt angajate;
- va fi angajată prima persoană care are scorul strict mai mare decât scorurile tuturor persoanelor intervievate înaintea sa;
- dacă nu există o persoană care să îndeplinească condiția 2, atunci va fi angajată ultima persoană intervievată.

Scrie un program care, pe baza scorurilor obținute la interviu de cele `N` persoane, determină persoana care va ocupa postul. De exemplu, dacă `N=6` și scorurile obținute sunt `10`, `25`, `13`, `10`, `29`, `40`, atunci va fi angajată persoana `5`. *20 p*

1. Prelucrarea cifrelor unui număr natural

Sumă cifre

1 Se citește de la tastatură un număr natural n. Să se calculeze suma cifrelor lui n.

Soluție

Ultima cifră din număr (cifra unităților) se obține ușor: `n % 10`. După ce utilizăm ultima cifră (o adăugăm la sumă), nu mai avem nevoie de ea și o eliminăm din număr (`n = n // 10`). Cifra zecilor a devenit acum ultima cifră. Repetăm procedeul, cât timp numărul mai are cifre.

Model de implementare 1

```
n = int(input("n=")) #1
s = 0     #2
while n > 0:    #3
      ultima_cifra = n % 10 #4
      s += ultima_cifra     #5
      n //= 10   #6
print("Suma cifrelor este ", s) #7
```

Urmărește în manualul digital execuția acestui algoritm pentru n=375.

Observație

Inițializarea unei variabile în care trebuie să calculăm o sumă se face întotdeauna cu 0 (elementul neutru la adunare). În mod similar, inițializarea unei variabile în care trebuie să calculăm un produs se face întotdeauna cu 1 (elementul neutru la înmulțire).

Să exersăm!

2 Modifică algoritmul astfel încât să determine numărul de cifre ale lui n.

Oglinditul unui număr natural

3 Se citește de la tastatură un număr natural n. Să se calculeze oglinditul lui n (numărul scris cu cifrele lui n în ordine inversă).

Soluție

Vom extrage succesiv cifrele lui n, ca în programul precedent, dar în loc să adunăm cifrele vom construi un alt număr (oglinditul), alipind fiecare nouă cifră extrasă la finalul oglindițului.

Pentru a adăuga o cifră la dreapta unui număr (la final) trebuie să îi „facem loc". Noua cifră va fi cifra unităților, prin urmare cifra care reprezenta înainte de alipire cifra unităților trebuie să devină cifra zecilor, cifra zecilor trebuie să devină cifra sutelor etc. (*Tabelul 1*)

Tabelul 1

n		n_k	...	n_3	n_2	n_1	n_0
		10^k		10^3	10^2	10^1	10^0
n*10	n_k	...	n_3	n_2	n_1	n_0	0
	10^{k+1}	10^k		10^3	10^2	10^1	10^0

Model de implementare 2

```
n = int(input("n="))
oglindit = 0
while n > 0:                                  #cât timp n mai are cifre
      ultima_cifra = n % 10                   #determinăm ultima cifră
      oglindit = oglindit * 10 + ultima_cifra #alipim ultima_cifră la dreapta
      n //= 10                                #eliminăm ultima cifră a lui n
print("oglinditul este ", oglindit)
```

4 Modifică algoritmul astfel încât să verifice dacă numărul n este palindrom. Se numește *palindrom* un număr care citit de la stânga la dreapta este același cu numărul citit de la dreapta la stânga. De exemplu, 121, 14541 sunt numere palindrom.

Construire număr

5 Se citește de la tastatură un număr natural n. Să se construiască un număr m care să conțină la început cifrele impare ale lui n (în ordinea în care acestea apar în n), iar la final cifrele pare ale lui n (în ordinea inversă a apariției lor în n).

De exemplu, dacă n=231485, numărul m va fi 315842.

Soluție

Vom extrage succesiv cifrele lui n și vom verifica paritatea fiecărei cifre. Cifrele pare le vom alipi la dreapta numărului m, așa cum am procedat în programul precedent. Cifrele impare trebuie să le alipim la stânga numărului m, pentru a păstra ordinea acestora. Pentru aceasta va trebui să înmulțim cifra cu puterea lui 10 corespunzătoare poziției sale în m. Prin urmare, în variabila poz vom reține poziția pe care trebuie să fie plasată o cifră în m pentru a fi prima cifră a lui m, iar în variabila p10 vom reține 10^{poz}.

Model de implementare 3

```
n = int(input("n="))
m = poz = 0; p10 = 1
while n > 0:                                  #cât timp n mai are cifre
      ultima_cifra = n % 10                   #determinăm ultima cifră
      if ultima_cifra%2 == 0:                 #cifra pară, o alipim la dreapta
         m = m * 10 + ultima_cifra
      else:                                   #cifră impară o alipim la stânga
         m = m + ultima_cifra * p10
      n //= 10                                #eliminăm ultima cifră a lui n
      poz += 1; p10 *= 10                     #actualizăm poz și p10
print(m)
```

6 În limbajul Python există un operator pentru ridicarea la putere, deci am fi putut calcula la fiecare pas p10 = 10 ** poz. În algoritmul anterior, am calculat însă puterea înmulțind la fiecare pas p10 cu 10, ceea ce este mai eficient. Poți justifica eficiența?

7 Modifică algoritmul precedent astfel încât să construiască două numere: numărul a care va conține cifrele pare în ordinea apariției lor în n, respectiv numărul b care va conține cifrele impare în ordinea inversă a apariției lor în n.

Eliminarea unei cifre

8 Se citesc de la tastatură un număr natural n și o poziție poz. Să se elimine din n cifra de pe poziția poz, considerând că pozițiile cifrelor sunt numerotate începând cu 0, de la dreapta la stânga. De exemplu, dacă n=320845 și poz=3, după eliminare n va fi 32845.

Soluție

O primă idee ar fi să extragem succesiv cifrele lui n și să construim un alt număr în care să plasăm, în ordine, toate cifrele lui n, exceptând cifra de pe poziția poz, apoi să copiem numărul obținut în n.

Este o soluție corectă, dar asemănătoare din punct de vedere algoritmic cu cea utilizată pentru rezolvarea problemei precedente, din acest motiv o propunem ca exercițiu.

O idee mai interesantă ar fi să eliminăm direct din n cifra de pe poziția poz.

Tabelul 2

	prefix			cifra	sufix				
n	n_k	...	n_{poz+1}	n_{poz}	n_{poz-1}	...	n_2	n_1	n_0
	10^k		10^{poz+1}	10^{poz}	10^{poz-1}		10^2	10^1	10^0

După eliminare, n va trebui să fie format din prefix concatenat cu sufix (*Tabelul 2*).

Pentru a rezolva problema, să răspundem la următoarele întrebări (care reprezintă subprobleme ale problemei date):

Întrebarea 1: Cum obținem sufixul format din ultimele `poz` cifre? Determinăm restul împărțirii lui `n` la 10^{poz}!

Întrebarea 2: Cum obținem prefixul format din cifrele de la începutul numărului până la poziția `poz+1` (inclusiv)? Determinăm câtul împărțirii lui `n` la 10^{poz+1}!

Întrebarea 3: Cum concatenăm prefixul cu sufixul? Înmulțim `prefix` cu 10^{poz} și adunăm `sufix`.

Model de implementare 4

```
n = int(input("n="))
poz = int(input("poz="))
p10 = 10 ** poz
sufix = n % p10
prefix = n // (10 * p10)
n = prefix * p10 + sufix
print(n)
```

9 Ce va afișa programul precedent dacă poziția citită nu este poziția unei cifre din număr?

Inserarea unei cifre

10 Se citesc de la tastatură un număr natural `n`, o poziție `poz` și o cifră `c`. Să se insereze cifra `c` pe poziția `poz` în `n`, considerând că pozițiile cifrelor sunt numerotate începând cu `0`, de la dreapta la stânga.

De exemplu, dacă `n=32845`, `poz=3` și `c=0`, după inserare `n` va fi `320845`.

Soluție

Și în cazul acestei probleme, am putea extrage succesiv cifrele lui `n` și construi un alt număr în care să plasăm cifrele în ordinea din `n`. Când ajungem la cifra de pe poziția `poz`, plasăm mai întâi în numărul rezultat cifra `c`, apoi cifra care exista inițial în `n` pe poziția `poz`. Și în acest caz soluția este similară din punct de vedere algoritmic cu cea prezentată la problema *Construire număr*, din acest motiv implementarea o lăsăm ca exercițiu.

O abordare mai interesantă ar fi să inserăm direct în `n` cifra `c` pe poziția `poz` (*Tabelul 3*).

Tabelul 3

	prefix				sufix				
n	n_k	...	n_{poz+1}	n_{poz}	n_{poz-1}	...	n_2	n_1	n_0
	10^k		10^{poz+1}	10^{poz}	10^{poz-1}		10^2	10^1	10^0

După inserare, `n` va fi de forma `prefix c sufix` (*Tabelul 4*):

Tabelul 4

	prefix				cifra	sufix				
n	n_k	...	n_{poz+1}	n_{poz}	c	n_{poz-1}	...	n_2	n_1	n_0
	10^{k+1}		10^{poz+2}	10^{poz+1}	10^{poz}	10^{poz-1}		10^2	10^1	10^0

Din nou, pentru rezolvarea problemei, vom identifica subproblemele acesteia și le vom rezolva, răspunzând la următoarele trei întrebări.

Întrebarea 1: Cum obținem sufixul format din ultimele `poz` cifre? Determinăm restul împărțirii lui `n` la 10^{poz}!

Întrebarea 2: Cum obținem prefixul format din cifrele de la începutul numărului până la poziția `poz` (inclusiv)? Determinăm câtul împărțirii lui `n` la 10^{poz}!

Model de implementare 5

```
n = int(input("n="))
poz = int(input("poz="))
c = int(input("c="))
p10 = 10 ** poz
sufix = n % p10
prefix = n // p10
n = (prefix * 10 + c) * p10 + sufix
print(n)
```

Întrebarea 3: Cum concatenăm `prefix` cu `c` și apoi cu `sufix`? Înmulțim `prefix` cu `10` și adunăm `c` (am concatenat `prefix` cu `c`), apoi înmulțim rezultatul cu 10^{poz} și adunăm `sufix` (astfel am concatenat și `sufix`).

11 Ce va afișa programul precedent, dacă poziția citită nu este poziția unei cifre din număr?

2. Fișă de autoevaluare

1 Selectează secvențele de instrucțiuni care atribuie variabilei întregi `u` valoarea primei cifre a numărului natural memorat în variabila `x`.

a.
```
u = x // 10
```

b.
```
u = x % 10
```

c.
```
u = x
while u > 9: u = u % 10
```

d.
```
u = x
while u > 9: u = u//10
```

2 Următorul program trebuie să determine cifra maximă din numărul natural `n`. Corectează erorile.

```
cifra_maxima = 9
while n > 0:
     ultima_cifra = n % 10
     if ultimacifra > cifra_maxima:
         ultima_cifra = cifra_maxima
n //= 10
print(cifra_maxima)
```

3 Se consideră următorul program:

```
n = int(input("Introduceți un număr natural: "))
c = int(input("Introduceți o cifră zecimală: "))
p = 1
while n >= p:
      if n // p % 10 == c:
          n = n // (p * 10) * p + n % 10
      p = p * 10
print(n)
```

a. Urmărește execuția pas cu pas a acestui program pentru cazul în care se citesc valorile `32425 2`. Ce rezultat va fi afișat pe ecran?

b. Scrie rezultatul afișat pe ecran dacă se citesc valorile `322245227 2`.

c. Modifică programul astfel încât să se afișeze numărul obținut prin eliminarea din `n` a tuturor aparițiilor cifrei `c`.

Portofoliu digital

1 Într-un laborator, fiecare substanță chimică este identificată printr-un cod numeric. Se cunosc codul unei substanțe, notat cu `n`, și o cifră `c`, care corespunde unui anumit element din compoziția substanței. Scrie un program care să determine numărul de apariții ale cifrei `c` în numărul `n`.

2 În timpul unui experiment de fizică, un elev notează succesiv valorile măsurătorilor obținute. Valorile sunt numere naturale, iar introducerea lor se oprește atunci când se citește valoarea `-1`. Scrieți un program care să determine de câte ori apare cifra `0` în scrierea tuturor valorilor notate.

3 Într-un laborator de biologie, probele sunt etichetate cu numere naturale. Pentru a respecta o anumită regulă de clasificare, eticheta unei probe trebuie să conțină cel puțin o cifră pară. Scrieți un program care să verifice dacă un număr natural `n` respectă această condiție. Programul va afișa mesajul `DA` sau `NU`, după caz.

4 În cadrul unui experiment, mai multe probe sunt identificate prin coduri numerice. Codurile sunt citite succesiv, iar introducerea lor se oprește la întâlnirea valorii `0`. Scrie un program care să verifice dacă fiecare cod citit conține exact trei cifre impare. Programul va afișa mesajul `DA` dacă toate codurile respectă cerința sau `NU` în caz contrar.

5 Un instrument de măsură înregistrează date sub forma unui număr natural n. În acest sistem, cifrele pare reprezintă semnale utile și trebuie amplificate, în timp ce cifrele impare rămân neschimbate. Scrie un program care dublează fiecare cifră pară din valoarea înregistrată, păstrând ordinea cifrelor. De exemplu, dacă n=14578, programul va afișa 1445788.

3. Prelucrarea divizorilor numerelor naturale

Fie a și b două numere naturale nenule. Spunem că a este ***divizor*** al lui b dacă restul împărțirii lui b la a este 0. Spunem că a este ***multiplu*** al lui b dacă restul împărțirii lui a la b este 0.

Divizori

1 Dat fiind n>0 un număr natural, să se determine toți divizorii naturali ai lui n.

Soluție

În primul rând, observăm că orice număr natural n are ca divizori pe 1 și pe n (aceștia se numesc divizori improprii). În afară de divizorii improprii, cel mai mic divizor ar putea fi 2, iar cel mai mare n/2. Prin urmare, pentru a afla divizorii proprii ai lui n, vom parcurge toate numerele naturale din intervalul [2, n/2] și vom testa pentru fiecare număr natural d din acest interval dacă divide sau nu pe n (adică dacă restul împărțirii lui n la d este 0).

Model de implementare 6

```
n = int(input("n="))
print(1, end=' ')
if n > 1:
   for d in range(2, n // 2 + 1) :      # parcurg intervalul [2,n/2]
       if n % d == 0:                   # d divide n?
          print(d, end=' ')             # d este divizor, îl scriu
   print(n)
```

Complexitatea timp a algoritmului

Algoritmul efectuează n/2-1 verificări de divizibilitate.

Număr prim

2 Fie n un număr natural, citit de la tastatură. Scrie un program care verifică dacă numărul n este prim.

Soluție

Pentru a testa dacă numărul n este prim, vom verifica dacă are sau nu divizori proprii (divizori diferiți de 1 și de n). Pentru aceasta vom parcurge intervalul numerelor naturale de la 2 la $\sqrt{n}$, verificând pentru fiecare număr dacă este sau nu divizor al lui n.

Pentru a reține rezultatul testului, vom utiliza o variabilă logică denumită prim cu semnificația „prim este True, dacă n este prim, și False, dacă n nu este prim". Inițial, variabilei prim îi atribuim valoarea True („prezumția de nevinovăție"). Pe parcursul verificării, dacă găsim un divizor, variabilei prim îi atribuim valoarea False („dovada" că n nu este prim a fost găsită). Verificarea continuă cât timp avem *unde* căuta (nu am ajuns la $\sqrt{n}$) și avem *de ce* căuta (nu am găsit încă „dovada vinovăției" – un divizor al lui n).

Model de implementare 7

```
n = int(input("n="))
if n < 2:
    prim = False
else:
    prim = True
    d = 2
    while d * d <= n and prim:
        if n % d == 0: prim = False
        else: d += 1
if prim: print("DA")
else: print("NU")
```

Să exersăm!

3 Urmărește execuția pas cu pas a acestui program pentru n=13, n=8 și n=49.

4 Modifică programul astfel încât să afișeze toate numerele naturale prime ≤ n.

Descompunere în factori primi

5 Fie n un număr natural (n>1), citit de la tastatură. Scrie un program care să afișeze descompunerea numărului natural n în factori primi. De exemplu, pentru n=720, programul va afișa:

```
2 4
3 2
5 1
```

Acest fapt înseamnă că divizorii primi sunt 2, 3 și 5, având puterile în descompunerea în factori primi 4, 2 și, respectiv, 1.



Soluție

Vom aplica algoritmul învățat la matematică: considerăm toate numerele naturale începând cu 2. Pentru fiecare număr, verificăm dacă este divizor al lui n. În caz afirmativ, calculăm puterea la care apare acest divizor în descompunerea în factori primi a lui n, împărțind succesiv pe n la divizor și calculând numărul de împărțiri efectuate, după care afișăm atât divizorul găsit, cât și puterea. Procedeul se repetă cât timp n mai are divizori (n>1).

Model de implementare 8

```
n = int(input("n="))
d = 2
while n > 1:
      p = 0
      while n % d == 0:
          n //= d
          p += 1
      if p > 0: print(d, p)
      d += 1
```

Complexitatea timp a algoritmului

În cazul cel mai defavorabil (de exemplu, când n este prim) acest algoritm execută n operații.

Să exersăm!



6 Urmărește execuția pas cu pas a acestui program pentru n=2500.

7 Nu am testat nicăieri dacă d este prim. Nu este necesar, deoarece datorită faptului că atunci când găsim un divizor al lui n îl împărțim pe n la divizor ori de câte ori este posibil, orice divizor nou găsit este prim. Demonstrează această afirmație prin reducere la absurd.



Algoritmul poate fi optimizat. De exemplu, observăm că atunci când n este prim (adică are un singur divizor >1, el însuși) n va deveni 1 abia când va fi împărțit la n (adică d va parcurge intervalul [2, n]). Vom modifica algoritmul astfel încât d să parcurgă intervalul $[2, \sqrt{n}]$. Dar în acest caz este posibil să nu găsim toți divizorii. Dacă la final n>1, deducem că în urma împărțirilor, n are ca valoare un număr prim la puterea întâi, ultimul factor prim din descompunere.

Model de implementare 9

```
n = int(input("n="))
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)
```

8 Modifică programul astfel încât să afișeze numai cel mai mare divizor prim.

9 Modifică programul astfel încât să calculeze numărul total de divizori ai lui n.

Cel mai mare divizor comun

10 Fie n și m două numere naturale nenule citite de la tastatură. Scrie un program care să calculeze și să afișeze c.m.m.d.c. (n, m).



Soluție

Vom aplica algoritmul lui *Euclid*. Cât timp m≠0, împărțim pe n la m și calculăm restul; la pasul următor, împărțitorul devine deîmpărțit, iar restul devine împărțitor. Ultimul rest≠0 este c.m.m.d.c.

De exemplu, pentru `n=495` și `m=720`, *cel mai mare divizor comun* prin algoritmul lui Euclid se obține astfel:

Deîmpărțit	Împărțitor	Rest
495 =	720 · 0 +	495
720 =	495 · 1 +	225
495 =	225 · 2 +	45
225 =	45 · 5 +	0 ⇒ c.m.m.d.c.(495,720)=45

Model de implementare 10

```
n = int(input("n="))
m = int(input("m="))
if n == 0 and m == 0: print("Nu exista solutie")
else:
    while m > 0:
        rest = n % m                    # calculez restul
        n = m                           # impartitorul devine deimpartit
        m = rest                        # restul devine impartitor
    print("cmmdc=", n)                  # ultimul rest diferit de 0
```

11 Urmărește execuția pas cu pas a acestui program pentru `n=121` și `m=4950`.

12 Modifică programul astfel încât să afișeze etapele de determinare a c.m.m.d.c. ca în exemplu.

13 Modifică acest program astfel încât să simplifice fracția `n/m` și să afișeze fracția ireductibilă obținută.

14 Modifică acest program astfel încât să verifice dacă numerele `n` și `m` sunt prime între ele.

15 Modifică acest program astfel încât să calculeze c.m.m.m.c. `(n,m)` (cel mai mic multiplu comun al numerelor `n` și `m`).

Investigație

16 Realizează o investigație online pentru a descoperi cum reușea Euclid să găsească cel mai mare divizor comun dintre două numere fără calculator. Descoperă câteva domenii în care un algoritm inventat acum peste 2300 de ani este folosit și astăzi.

Următorii pași te vor ajuta în realizarea investigației tale:

1. **Formularea întrebării de investigat** (o soluție ar putea fi: „Cum proceda Euclid pentru a determina cel mai mare divizor comun a două numere?”)
2. **Planificarea investigației**
   Caută informații din surse sigure și notează ideile principale despre Euclid și metoda utilizată de acesta pentru calculul celui mai mare divizor comun, metodă care a condus la algoritmul care îi poartă numele.
3. **Colectarea datelor**
   Colectează informații relevante despre Euclid și metoda sa de calcul pentru cel mai mare divizor comun, precum și imagini care să ilustreze modul de lucru.
4. **Analiza datelor**
   Analizează informațiile descoperite și compară metoda folosită de Euclid cu algoritmul studiat de tine la școală. Poți identifica domenii în care și astăzi este utilizat acest algoritm?
5. **Prezentarea rezultatelor**
   Sintetizează toate aceste informații într-un document Word, indicând sursele de informare utilizate, precum și criteriile pe care le-ai utilizat pentru a evalua credibilitatea acestora.

Algoritmul lui Euclid cu scăderi repetate

17 O variantă de implementare a algoritmului lui Euclid este de a simula împărțirile cu scăderi repetate.

Model de implementare 11

```
n = int(input("n="))
m = int(input("m="))
if n == 0 and m == 0: print("Nu exista solutie")
else:
    while n != m:
        if n > m: n -= m
        else: m -= n
    print("cmmdc=", n)
```

Observație

În mod evident, această variantă de implementare este ineficientă – în loc să facem o iterație cu o singură împărțire (`n%m`), vom avea `n//m` iterații cu câte o scădere.

4. Probleme rezolvate

Existență

1 Se citesc de la tastatură două numere naturale nenule `n` și `p`, apoi se citesc succesiv `n` valori întregi. Scrie un algoritm care să verifice dacă printre cele `n` valori citite există multipli ai lui `p`.

Soluție

Vom citi succesiv cele `n` valori întregi în aceeași variabilă `a`. După fiecare citire vom verifica dacă valoarea citită este divizibilă cu `p` și, în caz afirmativ, vom reține faptul că există un multiplu al lui `p`. Rezultatul verificării îl vom reține în variabila `exista`. Inițial valoarea acestei variabile este `False` (deoarece încă nu am descoperit niciun multiplu al lui `p`). Dacă pe parcursul citirii vom descoperi un multiplu al lui `p`, variabila `exista` va primi valoarea `True`.

Model de implementare 12

```
n = int(input("n="))
p = int(input("p="))
exista = False
for i in range(n):
    a = int(input("a="))
    if a % p == 0:   # am identificat un multiplu
        exista = True; break
if exista: print("Exista un multiplu al lui ", p)
else: print("Nu exista un multiplu al lui ", p)
```

Cel mai mare divizor comun a n numere naturale

2 Se citește `n`, un număr natural nenul, apoi se citesc `n` valori naturale nenule. Scrie un program care să calculeze cel mai mare divizor comun al celor `n` numere citite.

Soluție

Operația c.m.m.d.c. este asociativă. Cu alte cuvinte, pentru a calcula c.m.m.d.c. a `n` numere, vom calcula c.m.m.d.c. dintre primul și al doilea, apoi c.m.m.d.c. dintre rezultat și al treilea ș.a.m.d.

Model de implementare 13

```
n = int(input("n="))
d = int(input("d="))
for i in range(n - 1):
    a = int(input("a="))
    while a > 0:
        rest = d % a        # calculez restul
        d = a               # impartitorul devine deimpartit
        a = rest            # restul devine impartitor
print("cmmdc=", d)
```

Pătrat perfect

3 Un inginer lucrează la proiectarea unor plăci pătrate pentru acoperirea unei suprafețe de arie `n` cm² (`n>1`). Pentru ca plăcile să se potrivească perfect, aria totală trebuie să fie un pătrat perfect, adică să poată fi scrisă ca aria unui pătrat cu lungimea laturii număr natural. Pentru a ajusta suprafața inițială, inginerul poate mări aria prin înmulțirea cu un număr natural `k`. Scrie un program care să determine cel mai mic număr natural nenul `k`, astfel încât produsul `n*k` să fie un pătrat perfect. De exemplu, dacă `n=12`, atunci `k=3`, deoarece `n*k=36=6²`.

Soluție

Pentru ca un număr să fie pătrat perfect, este necesar ca, în descompunerea sa în factori primi, fiecare factor prim să apară la o putere pară. De aceea, vom descompune numărul `n` în factori primi și pentru fiecare factor prim care apare cu un exponent impar, vom înmulți valoarea lui `k` cu acel factor prim. În acest mod, prin înmulțirea lui `n` cu `k`, toți exponenții din descompunerea în factori primi vor deveni pari, iar produsul `n*k` va fi un pătrat perfect.

Model de implementare 14

```
n = int(input("n="))
d = 2; k = 1
while d * d <= n:
      p = 0
      while n % d == 0:
          n //= d
          p += 1
      if p % 2 == 1: k *= d
      d += 1
if n > 1: k *= n
print(k)
```

5. Fișă de autoevaluare

1 Selectează secvențele de instrucțiuni care afișează cel mai mare divizor propriu al numărului natural neprim memorat în variabila `x`.

a.

```
d = 2
while x % d == 0:
      d = d + 1
print(x // d)
```

b.

```
d = 2
while x % d != 0:
      d = d + 1
print(x // d)
```

c.

```
d = 2
while d * d <= x:
      if x % d == 0: rez = d
      d += 1
print(rez)
```

d.

```
d = x // 2
while x % d != 0:
      d = d - 1
print(d)
```

2 Se consideră următorul program:

```
n = int(input("Un numar natural > 1:"))
x = 2
while n > 1:
      while n % x == 0:
          n = n // x
      x = x + 1
print(x - 1)
```

a. Scrie valoarea afișată dacă se citește numărul 720.

b. Dă exemple de trei valori distincte ce pot fi citite, astfel încât programul să afișeze o valoare egală cu cea citită.

c. Formulează în limbaj natural enunțul problemei rezolvate de acest program.

3 Se consideră următorul program:

```
a = int(input("introduceti un numar natural nenul a="))
b = int(input("introduceti un numar natural nenul b="))
if a < b: c = a
else: c = b
d = 2
while d <= c:
     if a % d == 0 and b % d == 0:
        print(d)
     d = d + 1
```

a. Scrie ce se afișează în urma executării algoritmului dacă se citesc, în această ordine, numerele 70 și 308.

b. Dacă pentru variabila a se citește 363, scrie trei numere naturale nenule distincte care pot fi citite pentru variabila b astfel încât, pentru fiecare dintre acestea, în urma executării algoritmului, să se afișeze 11 121.

c. Scrie un program cât mai eficient, echivalent cu cel dat.

Portofoliu digital

1 ***Exponent***

Fie n un număr natural și p un număr prim. Scrie un program care să determine cel mai mare număr natural k, astfel încât p^k divide n.

2 ***Număr perfect***

Un număr este *perfect* dacă el este egal cu suma divizorilor săi (exclusiv el însuși). Scrie un program care să testeze dacă un număr natural dat este perfect.

3 ***Numere prietene***

Două numere naturale a și b se numesc *prietene* dacă a este egal cu suma divizorilor lui b (exclusiv b), iar b este egal cu suma divizorilor lui a (exclusiv a). De exemplu, a=220 și b=284 sunt prietene. Scrie un program care să determine primele trei perechi de numere prietene cu a<b.

4 ***Prime vecine***

Fie x un număr natural, x>2. Scrie un program care să determine cel mai mare număr prim mai mic decât x și cel mai mic număr prim mai mare decât x.

5 ***Număr divizori primi***

Se citește de la tastatură un număr natural nenul n, apoi se citesc succesiv n numere naturale > 1. Scrie un program care să verifice dacă oricare număr dintre cele n citite are un număr impar de divizori primi.

AUTOEVALUARE

Reflectați asupra activității de până acum, apoi completați următoarele enunțuri.

a. Prin rezolvarea exercițiilor am învățat: ....

b. În rezolvarea exercițiilor am avut dificultăți la .... .

c. Cred că mi-aș îmbunătăți performanța dacă .... .

d. Cel mai bine m-am descurcat .... .

e. Am înțeles legătura dintre teorie și practică, atunci când ...

Pentru unitățile de învățare parcurse, dar și pentru cele ce vor urma, folosește această grilă de autoevaluare pentru a-ți analiza progresul și dificultățile de învățare.

6. Evaluare

Timp de lucru: 50 de minute
Se acordă 10 puncte din oficiu

1 La executarea următoarei secvențe de instrucțiuni se vor afișa pe ecran:

```
n=25
for i in range(1, n+1):
    if n % i == 0:
        print(i, end=' ')
```

a. numerele pare până la 25;
b. divizorii lui 25;
c. multiplii lui 25;
d. cifrele lui 25. *14 p*

2 Care dintre următoarele secvențe de instrucțiuni elimină zerourile de la finalul numărului natural nenul memorat în variabila n?

a.
```
while n > 0:
    if n % 10 == 0:
        n //= 10
```

b.
```
while n % 10 == 0:
    n //= 10
```

c.
```
while n // 10 == 0:
    n %= 10
```

d.
```
while n > 0:
    if n % 10 == 0:
        n //= 10
    else:
        break
```
14 p

3 Care este cel mai mare număr de două cifre care ar putea fi atribuit variabilei n, astfel încât după executarea următoarei secvențe de instrucțiuni valoarea afișată să fie 1?

```
while n % 2 == 0: n //= 2
while n % 3 == 0: n //= 3
while n % 5 == 0: n //= 5
print(n)
```
17 p

4 Două module ale unui sistem informatic generează independent două coduri numerice de identificare, a și b. Pentru a obține un cod final cât mai mare, inginerul poate concatena cele două numere în oricare dintre cele două moduri posibile. Scrie un program care să construiască cel mai mare număr natural care se poate obține concatenând cele două numere. De exemplu, pentru a=7, b=1234 rezultatul va fi 71234; pentru a=13 și b=166, rezultatul va fi 16613. *20 p*

5 A venit primăvara. Ana și Bogdan au decis să dea copacii de pe marginea șoselei cu var. Fiind pasionați atât de natură, cât și de matematică, ei au decis să procedeze în modul următor:

- mai întâi au numerotat copacii de la 1 la n, începând cu cel de la intrarea în satul lor;
- Ana are o cutie de var alb și văruiește copacii cu numerele p, 2p, 3p etc.;
- Bogdan are o cutie de var roșu și văruiește copacii cu numerele q, 2q, 3q etc.

Astfel, când vor termina de văruit, unii copaci vor fi văruiți cu alb, alții cu roșu, alții cu roz (cei văruiți atât cu alb, cât și cu roșu) și unii vor rămâne nevăruiți.

a. Cunoscând numerele n, p și q ($1<n\leq10^9$, $1\leq p, q\leq10^4$), scrie un program eficient care să afișeze: câți copaci rămân nevăruiți; câți copaci sunt văruiți cu alb; câți copaci sunt văruiți cu roșu și câți copaci sunt văruiți cu roz.

De exemplu, pentru n=30, p=4, q=6 se vor afișa valorile 20 5 3 2. *15 p*

b. Descrie algoritmul în limbaj natural, justificând eficiența acestuia. *10 p*

1. Utilitatea subprogramelor în rezolvarea problemelor complexe

În realizarea aplicațiilor informatice complexe, o problemă mare nu poate fi abordată eficient ca un întreg. Ea trebuie descompusă în subprobleme mai mici, mai ușor de înțeles, de rezolvat și de testat. Fiecărei subprobleme îi corespunde un *subprogram* (funcție sau procedură), care se ocupă de o sarcină bine definită.

Această organizare este esențială mai ales atunci când mai mulți programatori lucrează la aceeași aplicație. Fiecare poate implementa un modul diferit, iar la final toate modulele sunt reunite într-un program complet. Pentru ca acest lucru să fie posibil, se stabilește dinainte cum comunică subprogramele între ele, adică ce date primesc și ce rezultate returnează.

În plus, în aproape orice aplicație apar operații care se repetă. De exemplu, calcularea unei medii, verificarea unei condiții sau afișarea unui mesaj. Dacă aceste operații ar fi scrise de fiecare dată de la zero, programul ar deveni lung, greu de înțeles și dificil de modificat. Folosind subprograme, o astfel de operație se scrie o singură dată și poate fi reutilizată ori de câte ori este nevoie.

De ce sunt subprogramele atât de importante?

Folosirea subprogramelor:

- face programul mai ușor de înțeles;
- permite lucrul în echipă;
- reduce repetarea codului;
- ușurează corectarea și modificarea programului;
- crește fiabilitatea aplicației.

Prin urmare, subprogramele sunt fundamentale pentru organizarea, claritatea și eficiența oricărei aplicații informatice, mai ales atunci când aceasta rezolvă probleme reale, complexe.

2. Subprograme în limbajul Python

În limbajul Python subprogramele sunt denumite funcții.

O ***funcție*** este o secvență de instrucțiuni care primește (opțional) date de intrare, execută anumite operații și returnează (opțional) un rezultat.

În limbajul Python există:

- funcții definite de utilizator;
- funcții predefinite – de exemplu, funcțiile `print()`, `input()`, `type()`, `id()` și funcțiile de conversie a tipului `int()`, `float()` etc., pe care le-am utilizat deja; alte funcții frecvent utilizate: `abs()` – returnează modulul unui număr; `round()` – returnează un număr real rotunjit la un număr specificat de zecimale.
- funcții din biblioteci – de exemplu, `math.sqrt()`.

Definirea unei funcții în limbajul Python

Sintaxa:

```
def nume_funcție(lista_parametri):
    Instrucțiuni
```

Semantica:

- **`def`** – cuvânt-cheie care indică începutul unei definiții de funcții;
- `nume_funcție` – numele funcției; respectă aceleași reguli sintactice ca numele de variabile și se recomandă utilizarea *snake-case*: numele vor fi scrise cu litere mici, iar în cazul în care numele este format din mai multe cuvinte, acestea sunt separate prin caracterul '_'; de exemplu, `calcul_medie()`, `determinare_nr_elevi()`;

- `lista_parametri` – o succesiune de parametri separați prin virgulă (este opțională);
- `Instrucțiuni` – reprezintă instrucțiunile executate când funcția este apelată.

Parametrii unei funcții constituie o interfață prin intermediul căreia funcția comunică cu exteriorul.

Parametrii sunt niște denumiri fictive, care desemnează date primite de funcție din exterior și sunt utilizați pentru a descrie prelucrările efectuate de funcție.

Instrucțiunea return

Sintaxa:

```
return expresie_rezultat_1, expresie_rezultat_2, …, expresie_rezultat_N
```

Semantica:

Instrucțiunea `return` poate apărea oriunde în blocul funcției și are ca efect terminarea execuției funcției și transmiterea rezultatelor către programul care a apelat funcția.

O funcție poate returna mai multe rezultate. Dacă nu utilizăm instrucțiunea `return` în blocul de instrucțiuni al funcției, atunci funcția returnează implicit valoarea `None`.

Apelarea unei funcții în limbajul Python

Sintaxa:

```
nume_funcție(listă_argumente)
```

Semantica:

Argumentele sunt valori concrete care sunt furnizate funcției la apel.

La apel, fiecare parametru va fi înlocuit de un argument și se execută instrucțiunile din funcție pentru valorile argumentelor specificate la apel.

În cazul transferului pozițional, argumentele înlocuiesc în ordine parametrii (primul argument înlocuiește primul parametru, al doilea argument înlocuiește al doilea parametru etc.), așa cum observați în *Tabelul 1*.

Tabelul 1

Exemplu	Apel	Efect apel
`def salut():` `  print("Bună!")`	`salut()`	Pe ecran va apărea mesajul `Bună!` Funcția nu are parametri.
`def salut(nume):` `  print("Bună,", nume)`	`salut("Ana")`	Pe ecran va apărea mesajul `Bună, Ana` Funcția are un parametru denumit `nume`. La apel, parametrul a fost înlocuit de argumentul `"Ana"`.
`def suma(a,b):` `  return a+b`	`x=10` `s=suma(x,5)`	Se atribuie variabilei `s` valoarea returnată de apelul funcției `suma` cu argumentele `x` și `5` (`15`).

`def schimba(x, y):`
`  return y, x`
	`a=3`
	`b=7`
	`a,b=schimba(a, b)`
		Funcția returnează două rezultate (valorile interschimbate ale celor doi parametri).
		După apel, `a` va avea valoarea `7` și `b` valoarea `3`.

Transferul argumentelor la apel

1 Să analizăm în *Tabelul 2* o altă variantă de implementare a funcției `schimba()`:

Tabelul 2

Exemplu definire	Apel	Efect apel

`def schimba_1(x, y):`
`  aux=x`
`  x=y`
`  y=aux`
`  print("In functie:",x,y)`
	`a=3`
	`b=7`
	`schimba_1(a,b)`
	`print("Dupa apel:", a,b)`
		Se afișează:
		`In functie: 7 3`
		`Dupa apel: 3 7`
		Deci, valorile argumentelor nu au fost interschimbate.

Pentru a înțelege de ce această variantă nu este corectă, trebuie să înțelegem cum se realizează transferul parametrilor în Python.

În Python, totul este un obiect. Mecanismul de transfer care se utilizează pentru un parametru depinde de tipul obiectului.

Există două mecanisme de transfer al parametrilor, deoarece există două categorii de obiecte:

1. **Obiecte imutabile**
   Un obiect este **imutabil** dacă, odată creat, conținutul său nu mai poate fi modificat. Orice operație de modificare a unui astfel de obiect creează o instanță nouă (un obiect nou în memorie cu noua valoare). De exemplu, toate obiectele pe care le-am utilizat până acum sunt imutabile (`int`, `float`, `bool`, `str`).
2. **Obiecte mutabile**
   Un obiect este **mutabil** dacă îi poți schimba conținutul fără a-i schimba identitatea (identificatorul unic). De exemplu, în continuare vom studia clasa `list`. Obiectele clasei `list` sunt mutabile (când modificăm o listă, nu se creează o listă nouă, ci este actualizată lista existentă).

Mecanismul de transfer al parametrilor:

- pentru obiectele **imutabile** transfer prin **referințe imutabile**;
- pentru obiectele **mutabile** transfer prin **referințe mutabile**.

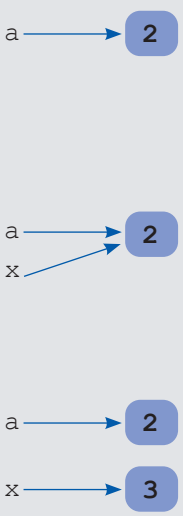
Transfer prin referințe imutabile

În cazul în care argumentul este un obiect imutabil, funcția primește o referință către acesta. Orice modificare pe care o efectuăm în funcție asupra unui argument imutabil creează un nou obiect, existent doar în funcție. După terminarea execuției funcției, obiectul original (care a fost transmis ca argument) nu va fi modificat (*Tabelul 3*).

Tabelul 3

Exemplu	Ecran	Explicație

`def inc(x):`
`  print(id(x)) #2`
`  x = x + 1`
`  print(id(x)) #3`
 	
`a = 2`
`print(id(a)) #1`
`inc(a)`
`print(id(a)) #4`
	`140722298177976`
	`140722298177976`
	`140722298178008`
	`140722298177976`
		Se creează un obiect al clasei `int` denumit `a`, apoi se afișează ID-ul acestuia (`140722298177976`).
		Se apelează funcția `inc` cu argumentul `a` (obiect imutabil). În funcția `inc` afișăm ID-ul obiectului primit ca argument și observăm că parametrul `x` a primit o referință (imutabilă) către obiectul `a` (se afișează tot `140722298177976`).
		Se incrementează variabila `x`, deci se va crea un nou obiect cu valoarea `3`. Afișăm pe ecran ID-ul obiectului `x` și observăm că este diferit (`140722298178008`).
		La terminarea execuției funcției, afișăm din nou ID-ul obiectului `a` și observăm că se afișează același ID (`140722298177976`), deci `a` indică același obiect.



Transfer prin referințe mutabile

În cazul în care argumentul este un obiect mutabil, funcția primește adresa de memorie a acestuia. În acest caz, modificarea unui argument în funcție schimbă direct variabila originală (cunoscând adresa la care este memorată variabila, funcția acționează direct asupra acestei zone de memorie).

Deci, de ce nu funcționează schimba_1()?

Argumentele sunt obiecte ale clasei `int`, deci imutabile. Transferul fiind realizat prin referințe imutabile, la terminarea execuției funcției, valorile originale ale argumentelor rămân nemodificate.

Variabile globale și variabile locale

O ***variabilă globală*** este o variabilă definită la nivel de fișier (modul) și poate fi accesată în întregul modul (exceptând cazurile de omonimie).

O ***variabilă locală*** este o variabilă definită într-o funcție și poate fi accesată doar în cadrul funcției respective (*Tabelul 4*).

Tabelul 4

Exemplu	Ecran	Explicații

`a = 7`
`def scrie():`
`  print(a,id(a))`
`print(a, id(a))`
`scrie()`
`print(a, id(a))`
	`7 140715881220728`
	`7 140715881220728`
	`7 140715881220728`
		`a` este o variabilă globală, pe care am accesat-o atât în cadrul funcției `scrie()`, cât și la nivel de modul. Identificatorul unic demonstrează că ne referim la aceeași variabilă atât în funcție, cât și la nivel de modul.

3. Biblioteci

O ***bibliotecă*** este o colecție de module care conțin funcții, clase și variabile gata scrise, pe care le putem folosi în alte aplicații.

În activitatea de programare există anumite operații care se execută frecvent. Pentru aceste operații au fost scrise funcții, care au fost grupate în biblioteci, pe categorii. Fără biblioteci, ar trebui să scriem tot codul de la zero. Practic, bibliotecile ne ajută să:

- economisim timp, fiindcă utilizăm cod gata scris;
- reducem numărul de erori, deoarece codul din biblioteci este corect, a fost deja testat;
- scriem cod mai lizibil;
- facem cu ușurință aplicații mai complexe; de exemplu, aplicații care utilizează inteligența artificială, jocuri, aplicații *web* etc.

În limbajul Python se pot utiliza trei tipuri de biblioteci:

- **Biblioteca standard** – aceasta este instalată automat când se instalează limbajul Python.
  Biblioteca standard conține module, cum ar fi: `math` (calcule matematice), `datetime` (date și timp), `random` (numere aleatoare), `os` (lucrul cu fișiere), `sys` (informații despre sistem), `turtle` (grafică 2D educațională), `tkinter` (interfețe grafice) etc.
- **Biblioteci externe** – acestea sunt create de alți programatori și trebuie instalate:
  `pip install <nume_bibliotecă>`
  Exemple: `numpy` (calcule matematice rapide), `pygame` (jocuri), `matplotlib` (grafice)
- **Biblioteci create de utilizator**

Utilizarea bibliotecilor

Pentru a utiliza o bibliotecă, aceasta trebuie să fie importată.

Varianta 1: Importăm toată biblioteca, așa cum vei descoperi în *Tabelul 5*.

```
import nume_bibliotecă
```

Dacă doriți, la importul bibliotecii puteți să îi asociați și un alias (un nume semnificativ, mai scurt):

```
import nume_bibliotecă as alias
```

Pentru a apela o funcție dintr-o bibliotecă trebuie să specificăm și biblioteca din care face parte funcția sub forma: `nume_bibliotecă.nume_funcție` sau `alias.nume_funcție`.

Tabelul 5

Exemplu	Explicație

`import math`
`print(math.sqrt(25))`
`print(math.ceil(3.75))`
`print(math.floor(3.75))`
	Am importat biblioteca `math`, apoi am apelat următoarele funcții:
	• `sqrt(25)` extrage radicalul din `25`, deci se va afișa `5.0`;
	• `ceil(x)` returnează partea întreagă superioară (cel mai mic număr întreg ≥`x`), deci se va afișa `4`;
	• `floor(x)` returnează partea întreagă inferioară (cel mai mare număr întreg ≤`x`), deci se va afișa `3`.

`import random` `print(random.randint(1,6))`	Am importat biblioteca `random`, apoi am generat un număr aleator cuprins între `1` și `6` (ca și cum am da cu zarul).

Varianta 2: Importăm doar anumite componente ale bibliotecii, așa cum vei descoperi în *Tabelul 6.*

`from nume_bibliotecă import nume_componenta_1, nume_componenta_2, ...`

Tabelul 6

Exemplu	Explicație

`from math import sqrt, pi`
`print(sqrt(25))`
`R = 10`
`print(pi * R * R)`
	Am importat din biblioteca `math` funcția `sqrt()` și constanta `pi`.
	Funcția importată este apelată fără a mai specifica biblioteca din care face parte. Similar, utilizăm constanta `pi` pentru a calcula aria unui cerc de rază `R`, fără a specifica biblioteca în care este definită.

4. Cum ne documentăm?

În programare nu este important să reții totul, pentru că informația este prea multă. Este important să știi unde să cauți informația și cum să o folosești.

Documentația este un manual oficial al unui limbaj de programare.

Documentația oficială a limbajului Python se găsește la adresa https://docs.python.org/.

Sfaturi practice

- Dacă vrei să găsești rapid un modul sau o funcție, folosește bara de căutare.
- Documentația oficială este referința completă pentru Python, dar uneori poate părea prea tehnică. Pentru explicații mai simple sau exemple, poți folosi tutoriale sau site-uri dedicate ca *Real Python*, *W3Schools*, *Tutorialspoint* etc.

5. Arhitectura unei aplicații Python

O aplicație Python are o structură ierarhică. Principalele elemente ale unei aplicații sunt:

- **Modul** – un fișier care conține cod Python și are extensia `.py`;
- **Pachet** (*Package*) – un dosar care conține mai multe module;
- **Proiect** – conține toate dosarele și fișierele aplicației.

Punctul de intrare în aplicație este, de regulă, modulul `main.py`. Codul din orice modul poate fi executat independent sau poate fi utilizat în alte module (dacă este importat).

6. Probleme rezolvate

Funcție de testare a primalității

1 Scrie o funcție care să primească un singur parametru (numărul natural `n`) și să verifice dacă `n` este prim. Funcția va returna `True` sau `False` după caz. Salvează funcția în fișierul `divizibilitate.py`.

Soluție

Verificarea proprietății de număr prim am studiat-o în unitatea de învățare „*Prelucrări ale datelor numerice*”. Să construim funcția corespunzătoare.

Model de implementare 1

```
def este_prim(n):
    if n < 2: return False
    d = 2
    while d * d <= n:
        if n % d == 0: return False
        d += 1
    return True
```

Superprim

2 Un număr se numește *superprim* dacă fiecare prefix al său este număr prim. De exemplu, numărul `317` este superprim, deoarece prefixele `3`, `31` și `317` sunt toate prime. Scrie o funcție care primește ca parametru un număr natural `n` și verifică dacă acesta este superprim. Funcția va returna `True` sau `False`, după caz. Funcția va apela funcția `este_prim()`, salvată în modulul `divizibilitate.py`.

Soluție

Vom importa din modulul `divizibilitate.py` funcția `este_prim()` și vom verifica succesiv dacă fiecare prefix al lui `n` este prim, apelând această funcție.

Model de implementare 2

```
from divizibilitate import este_prim
def superprim(n):
    while n > 0:
        if not este_prim(n): return False
        n//=10
    return True
```

7. Fișă de autoevaluare

1 Alege afirmațiile adevărate:

a. Parametrii unei funcții sunt opționali.

b. Utilizarea instrucțiunii `return` într-o funcție oprește imediat execuția funcției.

c. Funcțiile Python nu pot avea mai mulți parametri.

d. Nu poți apela aceeași funcție de mai multe ori în același program.

e. O funcție poate returna mai multe valori.

f. O variabilă definită într-o funcție se numește variabilă locală și poate fi accesată doar în funcție.

g. Funcțiile ajută la evitarea duplicării codului și fac programele mai clare.

2 Scrie o funcție care returnează pătratul unui număr primit ca parametru.

3 Selectează funcțiile care returnează cele mai mari două numere pare mai mici sau egale cu `n`.

a.
```
def pare(n):
    for i in range (0,n,2):
        a=i
    return a-2, a
```

b.
```
def pare(n):
    if n % 2 == 0:
       return n-2
       return n
    return n-3
    return n-1
```

c.
```
def pare(n):
    if n % 2 == 0:
       return n-2, n
    return n-3, n-1
```

d.
```
def pare(n):
    b=2
    while b + 2 <= n:
        b += 2
    return b-2, b
```

4 Selectează funcțiile care convertesc măsura unui unghi din grade în radiani (180° = π radiani).

a.
```
import math
def grade_in_radiani(grade):
    radiani = grade * math.pi / 180
    return radiani
```

b.
```
import math
def grade_in_radiani(grade):
    radiani = grade * 180/ math.pi
    return radiani
```

c.
```
import math
def grade_in_radiani(grade):
    radiani = grade * pi / 180
    return radiani
```

d.
```
from math import pi
def grade_in_radiani(grade):
    radiani = grade * pi / 180
    return radiani
```

AUTOEVALUARE

Lista mea de verificare	Da/ Nu
Am identificat corect afirmațiile adevărate despre funcții și parametri?	
Am scris corect o funcție care primește parametri și returnează un rezultat?	
Am analizat corect funcțiile propuse și am selectat variantele care rezolvă cerința?	
Am înțeles rolul instrucțiunii return și al variabilelor locale într-o funcție?	

Portofoliu digital

1 La un eveniment participă `n` (`n>1`) invitați. Scrie o funcție care determină numărul minim de mese necesare, astfel încât invitații să poată fi așezați în mod egal la mese, cu condiția ca la fiecare masă să stea cel puțin `4` și cel mult `20` de invitați. Dacă nu există o astfel de repartizare posibilă, funcția va returna valoarea `0`.

2 Se consideră șirul de numere `1, -1, 2, -2, 3, -3, 4, -4, …`, în care pozițiile termenilor în șir sunt numerotate începând cu `1`. Determină regula de formare a șirului și scrie o funcție care primește ca parametru un număr natural `n` (`n>0`) și returnează termenul aflat pe poziția `n` în șir.

3 Un pedometru digital este o aplicație care înregistrează, pe parcursul unei zile, numărul total de pași efectuați de un utilizator, precum și durata totală a deplasării, exprimată în minute. La instalarea aplicației, utilizatorul introduce lungimea pasului, exprimată în centimetri, și greutatea corporală, exprimată în kilograme.

Scrie o funcție care primește ca parametri: numărul de pași efectuați, durata deplasării (în minute), lungimea pasului (în centimetri), greutatea utilizatorului (în kilograme) și care returnează:

- distanța totală parcursă, exprimată în kilometri;
- consumul caloric estimat.

Consumul caloric se calculează aproximativ ca produsul dintre:

- durata deplasării exprimată în ore,
- greutatea utilizatorului exprimată în kilograme,
- un coeficient numit MET (*Metabolic Equivalent of Task*).

Coeficientul MET exprimă intensitatea efortului fizic și este determinat experimental, valorile sale fiind stabilite conform *Tabelului 7*, în funcție de tipul de activitate.

Tabelul 7

Activitate	MET
Stat în repaus	1.0
Mers foarte lent (2 km/h)	2.0
Mers normal (4 km/h)	3.3
Mers alert (5–6 km/h)	4.5
Alergare (viteza > 6 km/h)	10

4 Un număr natural se numește *extraprim* dacă este număr prim și dacă, prin eliminarea oricăreia dintre cifrele sale, se obține tot un număr prim. De exemplu, numărul `317` este extraprim, deoarece este prim, iar prin eliminarea unei cifre se obțin numerele `17`, `37` și `31`, toate prime. Scrie o funcție care verifică dacă un număr dat este extraprim. În implementare, funcția poate apela funcția `este_prim()`, definită anterior și salvată în modulul `divizibilitate.py`.

8. Evaluare

Timp de lucru: 50 de minute
Se acordă 10 puncte din oficiu

1 Selectează variantele care reprezintă un avantaj al utilizării subprogramelor.

a. permite lucrul în echipă;
b. reduce repetarea codului;
c. reduce întotdeauna timpul de execuție al programului;
d. reduce întotdeauna cantitatea de memorie utilizată de program;
e. ușurează corectarea și modificarea programului;
f. elimină complet erorile din program

10 p

2 Să considerăm următoarea funcție:

```
def ultima_cifra(n):
    return n % 10
```

Selectează care dintre următoarele variante conțin un apel corect al funcției `ultima_cifra()` și afișează ultima cifră a numărului `2026`.

a. `print(ultima_cifra(2026))`
b. `x=2026; a=ultima_cifra(x); print(a)`
c. `x=2026; ultima_cifra(x,c); print(c)`
d. `n=2026; print(ultima_cifra())`

10 p

3 Ce se va afișa pe ecran după executarea următoarei secvențe de instrucțiuni?

```
def f(n, x):
    n += x
    print(n, end=" ")
n = 10
f(n, 13)
print(n)
```

a. `23 23`
b. `10 10`
c. `23 10`
d. Nimic, deoarece generează eroare la execuție.

10 p

4 Stabilește valoarea de adevăr a următoarei propoziții: „*O variabilă locală este definită în cadrul unei funcții și poate fi accesată atât în interiorul funcției, cât și în exteriorul acesteia*”. *10 p*

5 Dă exemplu de două numere distincte de trei cifre care ar putea fi citite pentru variabila `x`, astfel încât la executarea următoarei secvențe de instrucțiuni să se afișeze mesajul `DA`.

```
import math
x=int(input("x= "))
r = int(math.sqrt(x))
if (r * r == x): print("DA")
else: print("NU")
```

15 p

6 Ce va afișa pe ecran următoarea secvență de instrucțiuni?

```
import math
x=6.7
print(round(x,0))
print(math.floor(x))
print(math.ceil(x))
print(int(x))
```

15 p

7 Proprietarul unui magazin dorește să automatizeze calculul prețului final pentru produsele sale. Pentru aceasta, el îți solicită un program care să conțină două funcții:

a. `aplica_reducere(pret, procent)`: primește prețul de listă al unui produs și un procent de reducere și returnează prețul produsului după aplicarea reducerii;
b. `calcul_pret_final(pret_redus, tva)`: primește prețul după reducere și un procent reprezentând TVA-ul (valoarea implicită a acestui parametru fiind 21%) și returnează prețul final pe care îl va plăti clientul.

Programul citește de la tastatură prețul unui produs, procentul de reducere și TVA-ul, apoi afișează prețul final al produsului apelând succesiv cele două funcții. *20 p*

1. Ce este un fișier text?

Până acum, în programele pe care le-am scris, datele au fost introduse direct de la tastatură, iar rezultatele au fost afișate pe ecran. Evident, acest mod de lucru este util pentru exerciții simple, dar în viața reală nu este aplicabil. Cum ar fi ca de fiecare dată când trebuie să calculăm salariile angajaților de la firmă să fie necesar să introducem de la tastatură toate datele necesare? Sau dacă toate notele, absențele, mediile, datele personale ale elevilor ar dispărea din catalogul electronic odată cu închiderea calculatorului?

Datele introduse de la tastatură se stochează în memoria RAM a calculatorului, deci după terminarea execuției programului acestea se vor pierde, după cum se pierd și rezultatele care se afișează pe ecran.

Datele pe care trebuie să le stocăm și pe care nu dorim să le pierdem la închiderea calculatorului trebuie să fie stocate în memoria externă (hard-disc, SSD, memory-stick etc.) sub formă de fișiere.

Lucrul cu fișiere este necesar deoarece permite stocarea permanentă a datelor, astfel încât acestea să poată fi utilizate și după încheierea execuției programului.

Din punctul de vedere al conținutului, fișierele pot fi clasificate în două mari categorii:

- fișiere *binare* – conțin date stocate în format binar, care nu pot fi „citite” direct de către oameni, ci trebuie să fie prelucrate cu programe specifice;
- fișiere *text* – conțin caractere „lizibile” pentru oameni, chiar dacă acestea sunt codificate și stocate tot în format binar.

> Un ***fișier text*** este un fișier care conține caractere structurate pe linii.
> Marcajul de sfârșit de linie este caracterul *newline* (`'\n'`).

Pentru codificarea caracterelor se pot utiliza diferite coduri, cele mai frecvent utilizate fiind codul ASCII (*American Standard Code for Information Interchange*) și UTF-8 (*Unicode Transformation Format*), ambele asociind fiecărui caracter dintr-o mulțime predefinită un cod numeric.

2. Prelucrarea fișierelor text în Python

Pentru a prelucra un fișier text, acesta trebuie să fie deschis. La deschidere se precizează modul în care îl vom prelucra. Din punctul de vedere al operațiilor permise cu fișiere text, putem clasifica fișierele în două mari categorii:

- *fișiere de intrare* – sunt fișiere din care vom citi date (deci singura operație permisă este operația de citire);
- *fișiere de ieșire* – sunt fișiere în care scriem date (deci singura operație permisă este operația de scriere).

Când am încheiat prelucrările, fișierul va fi închis, pentru a ne asigura că datele au fost salvate integral.

Schematic, lucrul cu fișiere text este ilustrat în *Diagrama 1*.

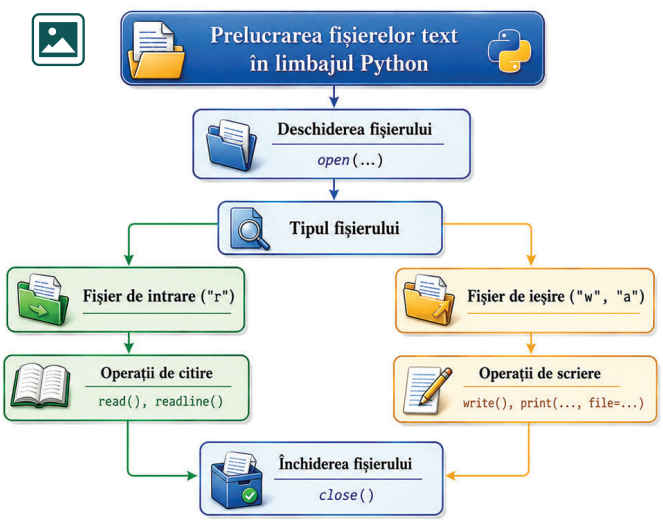


Diagrama 1

Deschiderea fișierelor

Sintaxa

```
variabila_fișier = open("specificator_de_fișier", "mod_deschidere")
```

Semantica

- `variabila_fișier` este un obiect al clasei predefinite `TextIOWrapper`, cu ajutorul căruia putem efectua operații asupra fișierului;
- `specificator_de_fișier` reprezintă fișierul fizic, existent pe suport extern, care urmează să fie prelucrat; dacă fișierul fizic se află în același director (*folder*) cu programul, este suficient să specificăm numele fișierului și extensia sub forma `nume_fișier.extensie`; dacă fișierul fizic se află într-un alt director, trebuie să specificăm și calea către directorul în care se află fișierul (succesiunea directoarelor până la directorul în care se află fișierul, directoarele fiind separate prin caracterul `'/'`);
- `mod_deschidere` este indicat de un șir de caractere; cele mai frecvente moduri de deschidere sunt:
  - `"r"` – fișierul trebuie să existe și este deschis pentru operații de citire (*read*);
  - `"w"` – se creează fișierul fizic specificat și este deschis pentru operații de scriere (*write*); dacă există deja un fișier cu același nume, conținutul acestuia va fi șters;
  - `"a"` – dacă fișierul există, atunci acesta este deschis pentru operații de scriere și datele scrise vor fi adăugate la sfârșitul fișierului fără a șterge conținutul existent (*append*); dacă fișierul nu există, el va fi creat.

În *Tabelul 1* sunt exemplificate cele 3 moduri de deschidere ale unui fișier prezentate anterior.

Tabelul 1

Exemplu	Explicație
`fin = open("exemplu.in", "r")`	Este deschis pentru citire fișierul fizic `exemplu.in` aflat în directorul curent (directorul în care se află programul) și se creează o variabilă denumită `fin`, care este un obiect al clasei `TextIOWrapper`.
`fout = open("exemplu.out", "w")`	Se creează în directorul curent fișierul denumit `exemplu.out` și se creează o variabilă denumită `fout`, care este un obiect al clasei `TextIOWrapper`.
`fa = open("teme/info/exemplu.txt", "a")`	Se deschide pentru scriere la final (adăugare) fișierul `exemplu.txt` aflat în subdirectorul `info`, care este un subdirector al directorului `teme` din directorul curent.

Analogie

Ne putem imagina că la deschiderea unui fișier se creează o „conductă” prin care variabila fișier comunică cu fișierul fizic. Dacă deschid fișierul în modul `"r"`, prin conductă vor fi transferate caractere din fișier către memoria alocată variabilelor care se citesc. Dacă deschid fișierul în modul `"w"` sau `"a"`, prin conductă vor fi transferate date din memoria calculatorului către fișier, convertite automat în caractere.

Excepții

Dacă încercăm să deschidem pentru citire un fișier care nu există pe disc, Python va genera o excepție de tip `FileNotFoundError`, de exemplu:

```
Traceback (most recent call last):
  File "exemplu_program.py", line 1, in <module>
fin = open("exemplu.in", "r")
FileNotFoundError: [Errno 2] No such file or directory: 'exemplu.in'
```

O ***excepție*** reprezintă o situație de eroare care apare în timpul executării unui program și care întrerupe fluxul normal al instrucțiunilor.

Atunci când Python întâlnește o operație imposibil de realizat (de exemplu, împărțirea la zero, accesarea unui indice inexistent sau compararea unor tipuri incompatibile), generează o excepție, semnalând faptul că a apărut o problemă. Dacă excepția nu este tratată, programul se oprește și afișează un mesaj de eroare (*traceback*), care oferă informații despre cauza și locul apariției problemei. Python permite gestionarea excepțiilor prin mecanisme dedicate, astfel încât programul să poată continua execuția sau să reacționeze controlat la situații neprevăzute, sporind robustețea și fiabilitatea aplicațiilor.

Închiderea fișierelor

După ce am terminat de lucrat cu un fișier, este important să-l închidem. În Python, închiderea se face apelând metoda `close()`, astfel `variabila_fișier.close()`.

Închiderea fișierului finalizează toate operațiile de citire sau scriere și garantează că toate informațiile adăugate sau modificate sunt salvate corect pe disc. Dacă un fișier rămâne deschis, datele s-ar putea pierde, iar fișierul ar putea fi blocat, împiedicând alte programe sau alte părți ale aceluiași program să-l folosească.

Folosirea blocului `with` pentru fișiere

În Python, o metodă sigură și simplă de a lucra cu fișiere este folosirea blocului `with`. Atunci când deschidem un fișier cu `with`, acesta se închide automat la finalul blocului, chiar dacă apare o eroare în timpul citirii sau scrierii. Astfel, nu mai trebuie să apelăm manual metoda `close()`.

Sintaxa

```
with open("specificator_fișier", "mod_deschidere") as variabilă_fișier:
     Instrucțiuni
```

Semantica

Este deschis un fișier care este disponibil doar în interiorul blocului `Instrucțiuni` din `with`, iar la finalul blocului `Instrucțiuni`, fișierul este închis automat, fără a fi nevoie de apelul explicit al metodei `close()`.

Citirea datelor din fișier

Pentru citirea datelor se pot utiliza două metode ale clasei `TextIOWrapper`:

- `read()` – citește tot conținutul fișierului ca un singur șir de caractere;
- `readline()` – citește linia curentă din fișier, tot într-un șir de caractere.

Chiar dacă fișierul conține valori numerice, metodele `read` și `readline` citesc în șiruri de caractere, care, ulterior, pentru a fi prelucrate pot fi convertite în valori numerice.

3. Probleme rezolvate

A. Citirea unui șir de n numere scrise pe linii diferite

1 În fișierul `exemplu.in` se află pe prima linie numărul natural `n`, apoi un șir de `n` valori întregi, câte un număr pe o linie. Secvența următoare de instrucțiuni va citi `n` și valorile din șir și va determina suma lor:

Model de implementare 1

```
suma = 0
with open("exemplu.in", "r") as fin:
    n = int(fin.readline())        #citesc numărul de valori din șir
    for i in range(n):
        linie = fin.readline()     #citesc o linie
        numar = int(linie)         #convertesc din șir în număr
        suma += numar              #adun la sumă
```

B. Citirea unui șir de numere scrise pe linii diferite până la sfârșitul fișierului

2 Un fișier text (obiect de tip `TextIOWrapper`) este iterabil, adică poate fi parcurs linie cu linie, într-o instrucțiune `for`, fără a mai utiliza `readline()`. Secvența următoare de instrucțiuni reprezintă o modalitate de a citi un număr necunoscut de valori scrise pe linii diferite în fișier:

Model de implementare 2

```
suma = 0
with open("exemplu.in", "r") as fin:
    for linie in fin:
        suma += int(linie)
```

Analogie

Când citim date dintr-un fișier, imaginează-ți un deget care indică unde am ajuns în text. Ori de câte ori citim din fișier, degetul se deplasează și va fi plasat după ultimul caracter citit. Astfel, următoarea citire va începe de unde a rămas poziționat „degetul", nu de la început!

Scrierea datelor în fișier

Pentru scrierea datelor se poate utiliza metoda `write()` a clasei `TextIOWrapper`. Această metodă acceptă ca argument un șir de caractere care va fi afișat în fișier. Dacă dorim să afișăm date numerice în fișier, va trebui să le convertim în șiruri de caractere, utilizând funcția de conversie `str()`. Dacă încercăm să scriem direct un număr întreg într-un fișier, fără a-l converti în șir de caractere, Python va genera o eroare.

În mod asemănător cu funcția `print()`, metoda `write()` poate accepta ca argument un șir de caractere formatat (șir de caractere precedat de litera `f`).

Afișarea datelor în fișier se poate realiza și cu funcția predefinită `print()` pe care am utilizat-o pentru afișarea pe ecran, dar în acest caz trebuie să specificăm ca argument al funcției: `file=variabilă_fișier`. *Tabelul 2* ilustrează modalități de afișare a datelor în fișier.

Tabelul 2

Exemplu	Explicație

`fout = open("exemplu.out","w")`
`for i in range (10):`
`  fout.write(str(i*i))`
`  fout.write('\n')`
	În fișierul `exemplu.out` vor fi afișate primele 10 pătrate perfecte, câte un număr pe o linie. Funcția `str()` convertește o valoare numerică în șir de caractere.

`fout = open("exemplu.out","w")`
`for i in range (10):`
`  fout.write(f"{i*i}, ")`
	În fișierul `exemplu.out` vor fi afișate cele mai mici `10` pătrate perfecte, pe o linie, fiecare număr fiind urmat de `", "`.

`fout = open("exemplu.out","w")`
`for i in range (10):`
`  fout.write(i*i)`
	Se obține eroare, deoarece metoda `write()` admite ca argument doar un șir de caractere:
	`Traceback (most recent call last):`
	`  File "E:\invat_python\exersez.py", line 3, in <module>`
	`  fout.write(i*i)`
	`TypeError: write() argument must be str, not int`

`fout = open("exemplu.out","w")`
`for i in range (10):`
`  print(i*i,end=", ", file= fout)`
	În fișierul `exemplu.out` vor fi afișate primele `10` pătrate perfecte, pe o singură linie, fiecare număr fiind urmat de `", "`.

`fout = open("exemplu.out","a")`
`for i in range (10, 20):`
`  fout.write(f"{i*i}\n")`
	În fișierul `exemplu.out` vor fi adăugate următoarele `10` pătrate perfecte, fiecare număr afișat fiind urmat de `'\n'`.

4. Fișă de autoevaluare

1 Se consideră următoarea secvență de instrucțiuni. Selectează afirmația corectă.

```
f = open("date.txt", "r")
f.write("Salut")
f.close()
```

a. Secvența este eronată pentru că fișierul nu este închis.
b. Secvența este eronată pentru că fișierul este deschis într-un mod greșit.
c. Secvența este eronată pentru că metoda `write` nu poate afișa șiruri de caractere.
d. Secvența este corectă.

2 Se consideră următoarea secvență de instrucțiuni. Selectează afirmația corectă.

```
f = open("date.txt", "w")
f.write(25)
f.close()
```

a. Se scrie la sfârșitul fișierului numărul `25`.
b. Se scrie în fișier șirul de caractere `"25"`.
c. Va apărea o eroare la execuție.
d. Se creează un fișier denumit `date.txt` și se scrie în fișier numărul `25`.

3 Se consideră următoarea secvență de instrucțiuni. Selectează afirmația corectă.

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

a. Citește prima linie din fișier și o afișează pe ecran.
b. Citește fișierul linie cu linie și afișează pe ecran fiecare linie.
c. Citește tot conținutul fișierului într-un șir de caractere pe care îl afișează pe ecran.
d. Va apărea o eroare la execuție.

4 Ce se întâmplă dacă deschidem un fișier în modul `"w"` și fișierul există deja?

a. Se adaugă date la final.
b. Se produce o eroare la execuție.
c. Se șterge conținutul fișierului.
d. Fișierul va deveni fișier de intrare, care permite doar operații de citire.

Portofoliu digital

1 Elevii unei școli au fost împărțiți în trei echipe:

- echipa R, ai cărei membri poartă tricouri roșii inscripționate cu numărul `1`;
- echipa G, ai cărei membri poartă tricouri verzi inscripționate cu numărul `2`;
- echipa B, ai cărei membri poartă tricouri albastre inscripționate cu numărul `3`.

Elevii s-au așezat într-un rând, într-o ordine arbitrară.

Fișierul de intrare `elevi.in` conține pe prima linie numerele inscripționate pe tricourile elevilor, în ordinea în care aceștia s-au așezat în rând, valorile fiind separate prin spații.

Scrie un program care să afișeze în fișierul de ieșire `elevi.out` numerele de pe tricourile elevilor, ordonate crescător, separate prin câte un spațiu.

2 Fișierul de intrare `fractii.in` conține fracții, pentru fiecare fracție fiind specificat numărătorul și numitorul pe linii separate.

Scrie un program care, pentru fiecare fracție din fișierul de intrare:

- determină forma ireductibilă a fracției (prin simplificare),
- afișează rezultatul în fișierul de ieșire `fractii.out`, scriind numărătorul și numitorul pe aceeași linie, separați printr-un spațiu.

5. Evaluare

Timp de lucru: 50 de minute
Se acordă 10 puncte din oficiu

1 Selectează varianta care reprezintă un avantaj al utilizării fișierelor text pentru citirea și afișarea datelor.

a. Fișierele text sunt executate automat de Python fără a fi nevoie de cod suplimentar.
b. Fișierele text oferă performanță mai mare decât toate celelalte tipuri de fișiere.
c. Lucrul cu fișiere permite stocarea permanentă a datelor, astfel încât acestea să poată fi folosite și după încheierea execuției programului.
d. Utilizarea fișierelor text prezintă avantaje doar în cazul în care prelucrăm date de tip șir de caractere. *10 p*

2 Următoarea secvență de instrucțiuni trebuie să scrie numărul `13` în fișierul `date.out`. Completează zona lacunară astfel încât să eviți o eroare de tip (*TypeError*).

```
f = open("date.out", "w")
f.write(...(13))
f.close()
```
10 p

3 Selectează variantele care descriu rolul utilizării instrucțiunii `with` atunci când lucrăm cu fișiere în Python.

a. Asigură închiderea automată a fișierului după terminarea blocului de cod, chiar dacă apare o excepție.
b. Permite citirea fișierului fără a fi necesară deschiderea acestuia anterior.
c. Optimizează viteza de acces la fișiere de cel puțin două ori față de funcția `open()`.
d. Afișează automat conținutul fișierului pe ecran după închidere. *10 p*

4 Fișierul de intrare `date.in` conține valori întregi, câte o valoare pe o linie. Completează spațiile lacunare din secvența următoare de instrucțiuni, astfel încât să citească valorile din fișierul `date.in` și să creeze un nou fișier, denumit `date.out`, în care să fie scrise valorile pare din fișierul `date.in`, pe aceeași linie, urmate de câte un spațiu.

```
fin = open("date.in", "r")
fout = open("date.out", ...)
for linie in fin:
    nr = ...
    if nr % 2 == 0:
        ...
fin.close(); fout.close()
```
20 p

5 La sfârșitul zilei, o casă de marcat dintr-un magazin exportă valorile tuturor bonurilor fiscale emise (exprimate în RON) sub forma unui șir de numere întregi, fiecare număr fiind scris pe o linie într-un fișier denumit `incasari.txt`. Completează spațiile lacunare din următoarea secvență de instrucțiuni, astfel încât aceasta să citească valorile din fișier și să afișeze pe ecran numărul de vânzări „mari” (vânzări pentru care suma încasată depășește 100 RON).

```
vanzari_mari = 0
with ... as f:
    for linie in ...:
        valoare = ...
        if valoare > 100:
            vanzari_mari += 1
print("Număr vânzări peste 100 RON:", vanzari_mari)
```
20 p

6 Un biolog studiază evoluția unei culturi de bacterii în laborator pe parcursul mai multor zile. Un senzor automat măsoară zilnic rata de multiplicare a bacteriilor (de câte ori a crescut numărul lor față de ziua precedentă) și salvează aceste numere întregi într-un fișier text denumit `numere.in`, scriind câte o valoare pe fiecare linie.

Din cauza unor fluctuații de tensiune, senzorul dă uneori erori și înregistrează valori negative sau zero (care nu au sens biologic în acest experiment și reprezintă zile compromise).

Scrie un program care să citească datele din fișierul `numere.in` și să calculeze factorul total de creștere al culturii pe durata experimentului. Acest factor reprezintă produsul tuturor ratelor de multiplicare valide. Dacă nu au fost înregistrate date valide, programul va afișa mesajul „`Nu au fost înregistrate date valide`”. *20 p*

1. Modelul conceptual liniar – listă

În rezolvarea multor probleme din lumea reală, eficiența unui algoritm depinde în mod esențial de modul în care datele sunt reprezentate și organizate. Alegerea unei structuri de date adecvate influențează atât simplitatea soluției, cât și performanța acesteia.

O ***structură de date*** reprezintă o colecție de date organizate după anumite reguli, care depind de tipul structurii.

În funcție de modul de organizare a datelor, anumite operații se pot realiza mai eficient sau mai puțin eficient. Din acest motiv, în etapa de analiză a problemei, vom stabili care este modul cel mai adecvat de organizare a datelor, în funcție de prelucrările pe care urmează să le efectuăm asupra datelor pentru rezolvarea problemei. Cu alte cuvinte, nu există o structură de date „perfectă", alegem structura de date potrivită pentru un anumit scop.

O ***listă*** este o structură de date liniară în care elementele se succed într-o anumită ordine. Fiecare element are un succesor unic cu excepția ultimului element și un predecesor unic cu excepția primului element al listei.

Fiecărui element al listei îi corespunde o poziție precisă în cadrul succesiunii, poziție identificată printr-un număr întreg denumit *indice* sau *index*. Numerotarea elementelor se realizează începând cu valoarea `0` pentru primul element și continuă în ordine crescătoare, conform succesiunii elementelor din listă.

Operațiile elementare care pot fi efectuate asupra unei liste sunt:

- *parcurgerea*: constă în accesarea secvențială a elementelor listei;
- *inserarea* unui element: se poate realiza la începutul listei, la sfârșitul acesteia sau într-o poziție intermediară;
- *eliminarea* unui element: constă în ștergerea unui element existent din listă.

Prin efectuarea operațiilor de inserare și eliminare, numărul de elemente din listă se poate modifica, însă rămâne întotdeauna finit și este limitat de memoria disponibilă.

Numărul de elemente din listă este denumit ***lungimea*** sau ***dimensiunea*** listei.

În ciuda simplității structurale, această structură de date se remarcă printr-o versatilitate ridicată, oferind soluții flexibile și eficiente pentru organizarea și gestionarea datelor.

Clasificarea listelor

În funcție de modul de acces la elementele unei liste și restricțiile referitoare la operațiile de inserare și ștergere, listele se clasifică în felul următor:

- *listele generale* – structuri de date liniare în care operațiile de inserare și eliminare a elementelor se pot realiza în orice poziție a listei: la început, la sfârșit sau într-o poziție intermediară fără a impune restricții asupra poziției de efectuare a acestor operații.
- *listele restrictive* – operațiile de inserare și ștergere a unui element sunt restricționate, ele fiind posibile doar la extremități; de exemplu, stiva este o structură de date pentru care atât operația de inserare, cât și cea de extragere se efectuează la un singur capăt al structurii sau coada care este o structură de date pentru care operația de inserare se efectuează la una dintre extremități, iar extragerea la cealaltă extremitate.

În funcție de relația de ordine dintre valorile elementelor, listele generale se clasifică în:

- *liste neordonate* – liste în care nu este impusă nicio relație de ordine între valorile elementelor. Ordinea elementelor reflectă exclusiv succesiunea inserărilor sau a operațiilor efectuate.
- *liste ordonate* – liste în care elementele sunt dispuse conform unei relații de ordine bine definite (de exemplu, ordine crescătoare sau descrescătoare). Inserarea unui nou element se realizează astfel încât relația de ordine a listei să fie menținută.

2. Fișă de autoevaluare

1 O listă conține, în ordine, următoarele elemente: [2, 4, 3, 6, 5, 9, 12, 14, 17]. Identifică succesorul și predecesorul lui 3 dacă din listă s-au șters toate numerele pare.

2 Se citesc de la tastatură 10 numere naturale, care sunt adăugate într-o listă, inițial vidă, după regulile următoare:

- primul număr citit este introdus în listă;
- dacă numărul citit are cel puțin două cifre se introduce la începutul listei, în caz contrar se introduce la finalul listei;
- dacă valoarea citită se regăsește deja în listă, atunci ea nu mai este adăugată.

Determină configurația finală a listei dacă de la tastatură se citesc succesiv numerele 5, 12, 1003, 20, 1003, 7201, 8, 15, 8, 7.

3 Se consideră o listă inițial vidă, în care sunt inserate, în această ordine, valorile 20, 40, 60 și 80. Sunt definite următoarele operații asupra listei:

- `Del(poz)` – șterge elementul aflat pe poziția `poz` în listă;
- `Ad(x, poz)` – inserează valoarea `x` pe poziția `poz` în listă.

Scrie o succesiune de operații astfel încât, în urma executării acestora, lista să conțină elementele în ordine inversă, respectiv: 80, 60, 40, 20.

3. Clasa `list` din Python

În acest capitol sunt prezentate aspectele esențiale privind utilizarea structurii de date *listă* în limbajul de programare Python, prin intermediul implementării sale predefinite. În Python, lista nu reprezintă doar un concept teoretic, ci este concretizată sub forma unei clase predefinite, denumită `list`, care oferă suport complet pentru gestionarea colecțiilor de elemente. Această implementare permite crearea, stocarea și prelucrarea datelor fără ca programatorul să fie nevoit să implementeze structura de la zero.

Clasa `list` include un set extins de metode pentru realizarea operațiilor uzuale asupra colecțiilor de date, cum ar fi adăugarea și eliminarea elementelor, căutarea unei valori, determinarea numărului de apariții ale unui element, copierea listei sau sortarea acesteia. Prin utilizarea acestor metode, împreună cu operatorii specifici limbajului, manipularea listelor se realizează într-un mod eficient și concis.

Caracteristicile listelor în Python

Listele din Python au câteva caracteristici esențiale care le fac extrem de versatile:

1. **Indexate:** Elementele unei liste sunt memorate într-o ordine bine definită și pot fi accesate prin intermediul unui index numeric, începând de la 0, fiecare element având un predecesor și un succesor, cu excepția primului și ultimului.
2. **Modificabile:** Putem schimba elementul de pe o anumită poziție, putem adăuga sau elimina elemente dintr-o listă.
3. **Eterogene:** Listele din Python pot stoca elemente de tipuri diferite.
4. **Dinamice:** Dimensiunea unei liste se poate modifica în timpul rulării programului, permițând adăugarea, inserarea sau eliminarea elementelor după nevoie.
5. **Iterabile:** Listele sunt iterabile.

O structură de date se numește **iterabilă** dacă elementele sale pot fi parcurse secvențial în cadrul unei instrucțiuni `for`.

Definirea listelor în Python

Pentru definirea unei liste în Python avem următoarele variante:

1. prin plasarea elementelor între paranteze drepte `[ ]`, separate prin virgule.

Exemple:

```
numere = [1, 2, 3, 4, 5]                # Listă cu numere
date = [1, True, 2.5, "Informatica"]    # Listă mixtă
```

Listele pot fi vide la început și se pot completa ulterior. Pentru a defini o listă vidă putem folosi următoarea sintaxă: `lista = []`

2. folosind funcția de conversie `list()` care poate transforma și alte tipuri de date iterabile (spre exemplu, rezultatul lui `range`) într-o listă.

```
# Crearea unei liste vide cu list()
lista1 = list()
# Crearea unei liste folosind range
lista2 = list(range(5))      # [0, 1, 2, 3, 4]
# Crearea unei liste cu numere de la 2 la 6
lista3 = list(range(2, 7))   # [2, 3, 4, 5, 6]
```

4. Operatori specifici pentru liste

În capitolele anterioare am învățat că Python folosește operatori pentru a efectua operații cu numere, cum ar fi adunarea, scăderea sau înmulțirea. Operatorii nu sunt limitați doar la numere; ei pot fi folosiți și cu alte tipuri de date, cum ar fi listele. Operatorii specifici listelor ne permit să efectuăm operații rapide și intuitive, precum combinarea a două liste, verificarea dacă un element se află într-o listă sau repetarea elementelor. Astfel, putem scrie programe clare și scurte, fără a fi nevoie să folosim cod complex pentru sarcini uzuale.

Accesarea elementelor dintr-o listă – operatorul de indexare

Fiecare element al unei liste are asociat un *index*, care indică poziția exactă a elementului în cadrul listei. Pozițiile elementelor sunt numerotate începând de la `0` pentru primul element și continuă succesiv până la valoarea egală cu numărul total de elemente din listă minus `1`.

Pentru a accesa un element din listă folosim următoarea sintaxă:

```
nume_lista[index]
```

Această operație se numește *indexare*, iar operatorul de indexare este paranteza dreaptă `[ ]` care conține indexul elementului dorit. Practic, citim sintaxa astfel: „din lista `nume_lista`, accesează elementul aflat la poziția indicată între parantezele drepte”.

Este important să facem diferența între folosirea parantezelor drepte pentru crearea listei și operatorul de indexare:

- Crearea listei (parantezele drepte `[ ]` folosite atunci când definim lista) servește pentru a forma lista și a introduce elemente în ea. De exemplu:
  ```
  numere = [10, 20, 30, 40]   # creează lista cu aceste elemente
  ```
- Operatorul de indexare (tot `[ ]`, dar folosit după ce lista a fost creată) servește pentru a accesa un element aflat la o anumită poziție din listă. De exemplu:
  ```
  print(numere[2]) # accesăm elementul de pe poziția 2, adică 30
  numere[0] = 15   # modificăm primul element
  ```

Operatorii de apartenență și non-apartenență

După ce am învățat să accesăm elementele unei liste folosind operatorul de indexare, următorul operator important este cel care verifică dacă un element se află sau nu într-o listă. În Python, acest lucru se face cu operatorii `in` și `not in`, așa cum observi în *Tabelul 1*.

- Operatorul `in` verifică dacă un element există în listă.
- Operatorul `not in` verifică dacă un element nu se află în listă.

Acești operatori returnează valoarea `True` dacă este adevărată condiția și `False` dacă este falsă.

Tabelul 1

Exemplu	Ecran	Explicații
`numere = [5, 10, 15, 20, 25]`		
`print(10 in numere)`	`True`	`10` se află în listă
`print(30 in numere)`	`False`	`30` nu se află în listă și **`in`** întoarce `False`
`print(30 not in numere)`	`True`	`30` nu se află în listă și **`not in`** întoarce `True`
`print(5 not in numere)`	`False`	`5` se află în listă și **`not in`** întoarce `False`

Observație

Acești operatori nu modifică lista, ci doar verifică dacă există sau nu elementul specificat.

Operatorul de concatenare

Concatenarea a două liste reprezintă operația prin care elementele unei liste sunt adăugate la sfârșitul celeilalte, rezultând o listă nouă care conține toate elementele celor două liste, în aceeași ordine. *Tabelul 2* conține exemple de utilizare a operatorului de concatenare. Operatorul folosit pentru această operație este `+`.

Tabelul 2

Exemplu	Ecran	Explicații

`lista1 = [1, 2]`
`lista2 = [4]`
`lista3 = lista1 + lista2`
`print(lista3)`
`lista1 += lista3`
`print(lista1)`
`print(lista3)`
	`[1, 2, 4]`
	`[1, 2, 1, 2, 4]`
	`[1, 2, 4]`
		Operatorul `+` concatenează listele și creează `lista3` (nu modifică `lista1` sau `lista2`).
		
		`+=` extinde `lista1` adăugând toate elementele din `lista3`, modificând `lista1` direct, dar fără să afecteze `lista3`

Operatorul de multiplicare

Multiplicarea unei liste reprezintă operația prin care elementele unei liste sunt repetate de un anumit număr de ori, rezultând o listă nouă care conține aceleași elemente în ordinea originală, repetate succesiv. Operatorul folosit pentru această operație este `*`. *Tabelul 3* conține exemple de utilizare a operatorului de multiplicare.

Tabelul 3

Exemplu	Ecran	Explicații

`numere = [1, 2]`
`numere_repetate = numere * 3`
`print(numere_repetate)`
	`[1, 2, 1, 2, 1, 2]`	Sunt repetate de `3` ori elementele din lista `numere`.

Operatorul `*` este util atunci când vrem să generăm rapid liste repetitive sau să pregătim o listă cu valori duplicate, fără a le scrie manual.

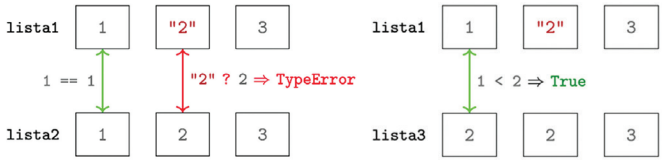
Operatorii relaționali pentru compararea listelor

Listele din Python pot fi comparate între ele folosind operatori de comparare, la fel cum comparăm numere sau șiruri de caractere. Compararea cu operatorul `==` se face element cu element, de la stânga la dreapta, și se oprește la prima pereche de valori diferite sau atunci când cel puțin una dintre liste este parcursă complet. Similar pentru operatorul `!=`, care este opusul operatorului `==`.

Operatorii `<`, `<=`, `>`, `>=` compară elementele în ordine lexicografică: lista `a` este considerată mai mică decât lista `b` (`a < b`) dacă există o poziție `k>=0`, astfel încât `a[i]==b[i]` pentru orice `0<=i<k` și `a[k]<b[k]` sau dacă `a` este prefix al lui `b`.

Listele se pot compara folosind acești operatori dacă elementele de pe poziții identice sunt de același tip. În caz contrar, va fi generată o excepție de tipul `TypeError` (*Imaginea 1*).

```
lista1 = [1, "2", 3]
lista2 = [1, 2, 3]
print(lista1 == lista2) # False
lista3 = [2, 2, 3]
print(lista1 < lista3) # True
```



Imaginea 1

Dacă încercăm să aplicăm <= pentru `lista1` și `lista2`, pentru prima pereche (`1` și `1`) elementele sunt egale, astfel că este continuată comparația. La a doua pereche, comparăm valorile `"2"` (șir de caractere) și `2` (număr întreg). Deoarece în Python nu este permisă compararea directă între tipuri diferite, precum `str` și `int`, evaluarea expresiei se întrerupe și este generată o excepție de tip `TypeError`.

5. Metodele clasei `list`

Pe lângă operatorii deja învățați, listele din Python au metode predefinite care permit efectuarea unor operații uzuale precum: adăugarea unui element, ștergerea unui element, căutarea, copierea și sortarea elementelor.

Adăugarea unui element

Metoda `append()` adaugă un element la finalul listei (*Tabelul 4*).

Tabelul 4

Exemplu	Ecran	Explicații

`numere = [10, 20, 30]`
`numere.append(40)`
`print(numere)`
	`[10, 20, 30, 40]`	Este adăugată valoarea `40` la finalul listei.

Observație

Lista originală se modifică. `append()` nu creează o listă nouă, ci adaugă elementul în lista existentă.

Inserarea unui element

Metoda `insert(index, element)` adaugă un element la o poziție specifică. Primul parametru (`index`) specifică poziția la care dorim să adăugăm, iar al doilea parametru (`element`) reprezintă elementul pe care dorim să-l adăugăm în listă (*Tabelul 5*).

Tabelul 5

Exemplu	Ecran	Explicații

`numere = [10, 20, 30]`
`numere.insert(1, 15)`
`print(numere)`
	`[10, 15, 20, 30]`	Inserează valoarea `15` pe poziția `1`, mutând elementele existente inițial începând cu această poziție la dreapta.

Observație

Toate elementele care se află după poziția la care am făcut inserarea sunt mutate cu o poziție mai la dreapta.

Este important de remarcat că metoda `insert()` permite ca argumentul pentru `index` să fie mai mare decât dimensiunea listei. În acest caz, Python nu va genera o excepție, ci va interpreta poziția ca fiind sfârșitul listei și va adăuga elementul la final.

De exemplu, dacă variabila `numere` va indica lista `[1, 2, 3, 4]` și apelăm `numere.insert(10, 5)`, argumentul `10` depășește lungimea listei, iar în acest caz elementul `5` va fi plasat după ultimul element, rezultând lista `[1, 2, 3, 4, 5]`.

Ștergerea unui element

Metoda `remove()` șterge prima apariție a unui element cu valoarea specificată și poate fi folosită ca în *Tabelul 6*.

Tabelul 6

Exemplu	Ecran	Explicații

`numere = [10, 20, 15, 20, 30]`
`numere.remove(20)`
`print(numere)`
	`[10, 15, 20, 30]`	Șterge prima apariție a elementului `20` din lista `numere`.

Observație

Dacă elementul pe care încercăm să-l ștergem nu există în listă, va fi generată o excepție de tip `ValueError`.

Ștergerea elementului de la o anumită poziție

Metoda `pop()` șterge elementul de la o poziție specificată și returnează valoarea ștearsă. Dacă nu se specifică poziția, elimină ultimul element, așa cum observi în *Tabelul 7*.

Tabelul 7

Exemplu	Ecran	Explicații

`numere = [10, 20, 30, 40]`
`ultim = numere.pop()`
`print(ultim)`
`print(numere)`
`elem = numere.pop(1)`
`print(elem, numere)`
	`40`
	`[10, 20, 30]`
	`20 [10, 30]`
		Primul apel al metodei `pop` șterge ultimul element din listă și îl returnează.
		
		Al doilea apel al metodei `pop` elimină și returnează elementul de pe poziția `1` din listă.

Dacă sunt utilizate exclusiv aceste două metode — `append()` pentru inserare și `pop()` pentru extragere — lista funcționează ca o structură restrictivă, în care toate operațiile se realizează la același capăt. În aceste condiții, comportamentul listei corespunde pe deplin modului de funcționare al unei stive, conform principiului LIFO („ultimul introdus, primul eliminat”).

Observație

Dacă încercăm să folosim `pop()` cu un index care depășește lungimea listei, Python va genera o excepție de tipul `IndexError` (*Tabelul 8*).

Tabelul 8

Exemplu	Ecran	Explicații

`numere = [10,20,30,40]`
`elem = numere.pop(4)`
	`elem = numere.pop(4)`
	`^^^^^^^^^^^^`
	`IndexError: pop index out of range`
		Lista are `4` elemente, iar ultimul element are indexul `3`. Încercăm să eliminăm elementul de la un index inexistent și este generată excepție.

Căutarea unui element

Metoda `index()` returnează poziția primei apariții a unui element într-o listă (*Tabelul 9*).

Tabelul 9

Exemplu	Ecran	Explicații

`numere = [10, 15, 20, 15, 40]`
`poz = numere.index(15)`
`print(poz)`
	`1`	Elementul `15` apare prima dată în listă pe poziția `1`.

Observație

Dacă elementul nu există în listă, Python va genera o excepție de tipul `ValueError`.

Determinarea numărului de apariții

Metoda `count()` numără de câte ori apare un element într-o listă (*Tabelul 10*).

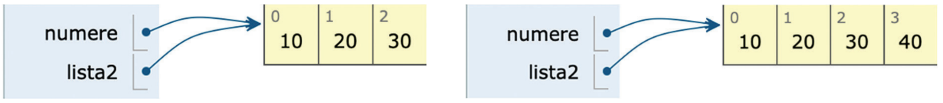
Tabelul 10

Exemplu	Ecran	Explicații
`numere = [10, 15, 20, 15, 25, 30]` `print(numere.count(15))`	`2`	Elementul `15` apare de două ori în listă.

Copierea listei

Uneori este nevoie să prelucrăm o listă, dar să păstrăm și varianta originală neschimbată. Dacă facem doar o atribuire de tipul `lista2 = numere`, nu obținem o copie independentă, ci un alias. Cu alte cuvinte, ambele variabile indică aceeași listă și orice modificare făcută pentru `lista2` va afecta și lista originală.

Observă exemplul din *Imaginea 2* și explicațiile din *Tabelul 11*.



Imaginea 2

Tabelul 11

Exemplu	Ecran	Explicații

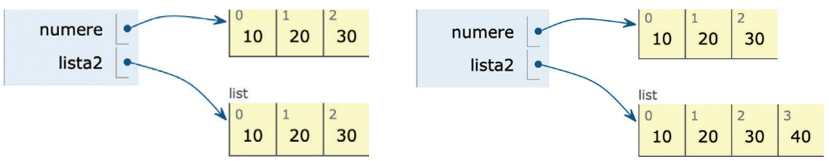
`numere = [10, 20, 30]`
`lista2 = numere`
`lista2.append(40)`
`print(numere)`
	`[10, 20, 30, 40]`	`lista2` referă aceeași zonă de memorie ca `numere`. Astfel, lista originală a fost modificată.

Pentru a evita acest lucru, vom folosi metoda `copy()`, care creează o copie independentă a listei, așa cum descoperi în *Imaginea 3* și în explicațiile din *Tabelul 12*.

Tabelul 12

Exemplu	Ecran	Explicații

`numere = [10, 20, 30]`
`lista2 = numere.copy()`
`lista2.append(40)`
`print(numere)`
`print(lista2)`
	`[10, 20, 30]`
	`[10, 20, 30, 40]`
		`lista2` este o copie independentă a listei `numere`, așa că adăugarea lui `40` afectează doar `lista2`, nu și `numere`.



Imaginea 3

Sortarea elementelor

Metoda `sort()` ordonează elementele listei în ordine crescătoare dacă este folosită în varianta implicită. Dacă vrem să obținem o sortare descrescătoare, folosim `reverse=True` (*Tabelul 13*).

Tabelul 13

Exemplu	Ecran	Explicații

`numere = [15, 10, 30, 15]`
`numere.sort()`
`print(numere)`
`numere.sort(reverse=True)`
`print(numere)`
	`[10, 15, 15, 30]`
	`[30, 15, 15, 10]`
		Lista este sortată mai întâi crescător și apoi descrescător.

Metoda `sort()` modifică lista originală. Dacă vrem să obținem o nouă listă sortată, fără a modifica lista inițială, putem folosi `sorted` care returnează o listă nouă (*Tabelul 14*).

Tabelul 14

Exemplu	Ecran	Explicații

`numere = [5, 1, 3, 1]`
`sortate = sorted(numere)`
`print(numere, sortate)`
	`[5, 1, 3, 1] [1, 1, 3, 5]`	`sorted` creează o listă nouă sortată, iar lista originală rămâne neschimbată.

6. Parcurgerea listelor

Parcurgerea unei liste reprezintă procesul prin care accesăm și prelucrăm succesiv elementele sale. Datorită faptului că listele din Python sunt iterabile, putem folosi diverse mecanisme pentru a explora elementele în mod eficient, fără a fi necesar să specificăm lungimea sau indexul fiecărui element.

Cel mai uzual mod de a parcurge o listă este utilizarea instrucțiunii `for`. Sintaxa generală este:

```
for element in nume_lista:
    Instrucțiuni
```

Această structură repetitivă preia, pe rând, fiecare element al listei și îl atribuie variabilei `element`, care poate fi utilizată în blocul de instrucțiuni care se execută în `for`.

De exemplu, pentru a calcula suma elementelor unei liste de numere întregi, se poate utiliza următoarea secvență:

Model de implementare 1

```
numere = [1, 2, -3, 4, 5, 6]
suma = 0
for x in numere:
    suma += x
print(suma) # 15
```

Instrucțiunea `for` parcurge lista element cu element, stochează fiecare valoare în variabila `x`, iar aceasta este adăugată succesiv la variabila `suma`.

Observație

În cazul listelor de valori numerice, aceeași operație poate fi realizată mai concis prin utilizarea funcției predefinite `sum()`. De exemplu, instrucțiunea `print(sum(numere))` va afișa pe ecran suma valorilor numerice din lista `numere`.

Există situații în care este necesară cunoașterea poziției elementelor din listă, nu doar a valorii acestora. În astfel de cazuri, se poate utiliza funcția `range()` pentru a parcurge lista prin intermediul indicilor. De exemplu, următoarea secvență de cod calculează suma elementelor aflate pe poziții pare:

Model de implementare 2

```
numere = [1, 2, -3, 4, 5, 6]
suma = 0
for poz in range(len(numere)):
    if poz % 2 == 0:
        suma += numere[poz]
print(suma) # => 1 + (-3) + 5 = 3
```

Funcția `len()` returnează numărul de elemente din lista transmisă la apel. O listă poate fi parcursă și cu ajutorul instrucțiunii `while`, prin utilizarea unui index care este incrementat manual. Această metodă este însă mai puțin folosită, deoarece necesită gestionarea explicită a condiției de oprire și a variabilei de control.

Model de implementare 3

```
numere = [10, 20, 30, 40]
i = 0
n = len(numere)
while i < n:
    print(numere[i])
    i += 1
```

7. Fișă de autoevaluare

1 Completează spațiile lacunare din următoarea secvență de cod astfel încât, după rularea acesteia, dacă primele două elemente din lista `numere` sunt egale, atunci al doilea element din listă este înlocuit cu dublul primului element din listă.

```
numere = [1, 1, 3, 4, 5]
if ... :
    numere... = ...
```

2 Scrie o instrucțiune care folosește operatorul `*` pentru a construi lista `[1,2,1,2,1,2,1,2]`.

3 Selectează metoda care poate fi folosită pentru a determina numărul de apariții ale unui element într-o listă.

a. `index` **b.** `count` **c.** `len` **d.** `pop`

4 Determină rezultatul produs de rularea următoarei secvențe de cod:

```
numere = [10, 15, 20, 15, 25]
numere.insert(numere.index(15), 16)
print(numere)
```

5 Selectează secvența de instrucțiuni care va genera o excepție atunci când este executată.

a.
```
numere = [10, 20, 30]
numere.append(40)
print(numere)
```

b.
```
numere = [5, 10, 15]
numere.remove(20)
print(numere)
```

c.
```
numere = [1, 2, 3]
numere.insert(5, 10)
print(numere)
```

d.
```
numere = [3, 1, 2]
numere.sort()
print(numere)
```

6 Completează următoarea secvență de cod pentru a afișa toate elementele listei `numere` pe aceeași linie, fiecare număr fiind urmat de un caracter #:

```
numere = [4, 7, 2, 9]
for ... in ...:
    print(...)
```

7 Alege valoarea de adevăr pentru fiecare dintre următoarele afirmații:

A/ F Instrucțiunea `for` poate fi folosită pentru a parcurge elementele unei liste.

A/ F Parcurgerea unei liste se poate face doar folosind indici.

A/ F Parcurgerea unei liste nu poate fi realizată cu ajutorul instrucțiunii `while`.

A/ F Media aritmetică a elementelor unei liste poate fi calculată cu o singură parcurgere a listei.

Portofoliu digital

1 ***Note***

Un elev a susținut mai multe teste în anul școlar curent, iar notele obținute sunt memorate în fișierul `note.in`. Scrie un program care să citească notele din fișier și să le introducă într-o listă, apoi să gestioneze lista notelor astfel:

a. dacă nota `10` nu se află în lista de note, să o adauge la final;

b. să determine care este cea mai mică notă pe care a luat-o elevul;

c. să sorteze crescător lista notelor și să afișeze notele în ordine crescătoare în fișierul `note.out` pe o singură linie și separate prin câte un spațiu.

2 ***Concurs***

La un concurs școlar participă elevi identificați prin numere naturale, corespunzătoare legitimațiilor de concurs. Acest concurs are două probe: Matematică și Informatică. Lista elevilor înscriși la proba de Matematică se află în fișierul `mate.in`, pe o singură linie, numerele fiind separate prin spații. Lista elevilor înscriși la proba de Informatică se află în fișierul `info.in`, în același format. Scrie un program care să rezolve următoarele cerințe:

a. să citească cele două liste și să construiască o listă finală cu toți participanții, fără duplicate (dacă un elev participă la ambele probe, va apărea o singură dată în lista finală);

b. să verifice dacă un participant cu un număr citit de la tastatură a participat la proba de Informatică;

c. să identifice și să afișeze numărul de elevi care au participat la ambele probe.

3 ***Stație meteo***

O stație meteo înregistrează temperaturile medii zilnice (numere reale, în grade Celsius) pentru o perioadă de 30 de zile. Aceste date sunt memorate în lista `temperaturi`. Implementează un program care gestionează și prelucrează lista de `temperaturi` pe baza următoarelor subprograme:

a. `adauga_temperatura(temperaturi, grade)` – dacă lista nu conține deja 30 de elemente, adaugă `grade` la finalul listei `temperaturi`. Dacă lista conține deja 30 de elemente, șterge primul element din listă și apoi adaugă `grade` la finalul listei.

b. `medie_temperaturi(temperaturi)` – calculează media temperaturilor din lista `temperaturi`.

c. `temperaturi_peste_medie(temperaturi)` – construiește o listă nouă care conține doar temperaturile mai mari decât media și o returnează. Lista `temperaturi` nu trebuie modificată.

8. Metode de generare a elementelor unei liste

Generarea unei liste pe baza unor proprietăți date. Determinare element median

Se consideră un număr natural `n`. Să se determine elementul situat la mijlocul listei formate din toate numerele naturale mai mici decât `n` care sunt divizibile cu `7` și nu sunt divizibile cu `5` (element median). În cazul în care lista obținută are un număr par de elemente, se va determina și afișa perechea de elemente situate la mijlocul listei. De exemplu, dacă `n=50` lista va fi `[7, 14, 21, 28, 42, 49]`, iar perechea din mijloc este `21 28`.

Soluție

Pentru identificarea elementului median sau a perechii centrale, se va genera mai întâi lista valorilor care respectă condițiile impuse. Se parcurge intervalul `[1, n)`, iar fiecare valoare care este divizibilă cu `7` și nedivizibilă cu `5` este adăugată într-o listă inițial vidă. După construirea listei, se determină poziția (sau pozițiile) corespunzătoare elementului median, în funcție de lungimea listei.

Model de implementare 4

```
n = int(input("n = "))
lista = []
for x in range(1, n):
    if x % 7 == 0 and x % 5 != 0:
        lista.append(x)
lungime = len(lista)
if lungime == 0:
    print("Lista este vida.")
else:
    print("Lista:", lista)
    poz_mijloc = lungime//2
    if lungime % 2:
        print("Elementul din mijloc:", lista[poz_mijloc])
    else:
        print("Perechea din mijloc:", lista[poz_mijloc-1], lista[poz_mijloc])
```

Generarea unei liste pe baza unor expresii matematice. Antrenament progresiv

Un atlet urmează un program de antrenament pentru maraton pe durata a `n` zile. În prima zi, acesta aleargă o distanță de `x` metri, iar în fiecare zi următoare distanța parcursă crește cu `y` metri față de ziua precedentă. Antrenorul a calculat că atletul consumă `z` calorii pentru fiecare metru alergat.

Să se determine numărul minim de zile de antrenament consecutive în care atletul a consumat, în total, cel puțin `k` calorii. De asemenea, se vor afișa, în ordine crescătoare, valorile corespunzătoare consumului caloric zilnic pentru zilele de antrenament respective.

De exemplu, pentru `n=4`, `x=2`, `y=3`, `z=1`, `k=16` în ultimele două zile au fost consumate `8` și `11` calorii, deci `2` este numărul minim de zile consecutive pentru care consumul caloric total a depășit `16` calorii. Șirul caloriilor consumate a fost `2 5 8 11`, deci se va afișa `8 11`.

Soluție

În ziua `i` (`1 ≤ i ≤ n`) atletul a alergat `dist[i]=x+(i-1)*y` metri, iar numărul de calorii consumate în aceeași zi este: `calorii[i]=dist[i]*z`.

Pentru a construi lista care conține consumul de calorii din fiecare dintre cele `n` zile definim o funcție denumită `alergare`. Funcția construiește o listă locală denumită `calorii`, pe care o returnează ca rezultat. Observăm că această listă este ordonată strict crescător, deoarece distanța parcursă zilnic crește în mod constant. Pentru determinarea numărului minim de zile consecutive în care suma caloriilor consumate este cel puțin `k`, se parcurge lista în ordine inversă, începând cu ultima zi de antrenament, și se însumează valorile până când suma cumulată atinge sau depășește pragul `k`. Zilele astfel selectate corespund celor mai mari valori calorice zilnice, asigurând obținerea numărului minim de zile cerut. În final, valorile calorice sunt afișate în ordine crescătoare.

Model de implementare 5

```
def alergare(n, x, y, z):
    calorii=[]
    for i in range(n):
        dist = x + i*y
        calorii.append(dist*z)
    return calorii

n = int(input("n= "))
x = int(input("x= "))
y = int(input("y= "))
z = int(input("z= "))
k = int(input("k= "))
calorii=alergare(n, x, y, z)
i = n-1
sum = 0
while i >= 0 and sum < k:
    sum += calorii[i]
    i -= 1
if (sum < k):
    print(f"Nu a alergat suficient pentru a consuma in total {k} calorii")
else:
    print(n - i - 1)
    for i in range(i + 1, n):
        print(calorii[i], end=" ")
```

Funcții cu parametri de tip list

Listele sunt obiecte **mutabile**. Atunci când o listă este transmisă unei funcții ca parametru, se transmite o referință mutabilă, care indică zona de memorie alocată listei. Prin urmare, orice modificare realizată asupra listei în interiorul funcției afectează direct lista originală.

9. Fișă de autoevaluare

1 Ce valori vor fi afișate după execuția următoarei secvențe de instrucțiuni?

```
lista = []
for i in range(5):
    lista.append((i+1)*10)
for i in range(1, 5):
        lista[i] -= lista[i-1]
print(lista)
```

a. `10 20 30 40 40`;
b. `0 10 20 30 40`;
c. `10 10 10 10 10`;
d. `10 10 20 20 30`.

2 Ce valori vor fi afișate după execuția următoarei secvențe de instrucțiuni?

```
lista_nr = []
for i in range(10):
    lista_nr.append(i*i + i)
i = 0
while i < len(lista_nr):
    if lista_nr[i] % 10 == 0:
        lista_nr.remove(lista_nr[i])
    else:
        i += 1
print(lista_nr)
```

3 Completează spațiile lacunare din următoarea secvență de instrucțiuni, astfel încât, după executarea acesteia, pe ecran să se afișeze: `[7, 27, 47, 67, 87, 107, 127, 147, 167, 187]`.

```
lista_nr = []
for i in range(10):
    lista_nr.append( … )
print(lista_nr)
```

Portofoliu digital

1 ***Validări bilete***

Un sistem de transport public înregistrează, în ordine, numărul de validări făcute la fiecare poartă de acces dintr-o stație, pe parcursul unei zile. Lista conține numărul de persoane care au trecut prin fiecare poartă, în ordinea instalării porților. Se dorește analiza doar a porților aflate la indici impari (`1`, `3`, `5`, … — considerate porți de ieșire). Dintre acestea, se iau în calcul doar porțile prin care a trecut un număr par de persoane (un flux par fiind considerat „echilibrat”). Scrie o funcție care primește lista cu numărul de validări de la fiecare poartă și calculează numărul total de persoane care au trecut prin porțile de ieșire cu flux par. De exemplu, pentru lista `[3,4,6,8,7,3]` funcția va returna valoarea `12`.

2 ***Senzori de trafic***

Un oraș folosește senzori pentru a măsura viteza medie a mașinilor pe un bulevard la intervale regulate. Lista conține valorile măsurate în km/h. Uneori, senzorul nu funcționează și înregistrează valoarea `0`, care înseamnă „date lipsă”, nu viteză reală. Pentru analiză statistică, valorile lipsă trebuie înlocuite cu viteza medie reală a traficului (calculată doar din măsurătorile valide). Scrie o funcție care primește lista de măsurători și înlocuiește fiecare `0` cu media aritmetică a valorilor nenule din listă.

3 ***Recorduri zilnice de vânzări***

O firmă înregistrează în fiecare zi suma vânzărilor zilnice. Managerul vrea să vadă evoluția „celui mai bun rezultat de până acum”. Pentru fiecare zi, dorește să știe care este cea mai mare vânzare zilnică înregistrată până în acea zi. Scrie o funcție denumită `record` care citește valorile zilnice din fișierul `vanzari.in` și care construiește o listă unde elementul de pe poziția `i` reprezintă cea mai mare vânzare dintre primele `i + 1` zile. Funcția va returna lista construită.

10. Metode de sortare a elementelor unei liste

Definirea problemei: *Se consideră o listă de valori numerice. Să se ordoneze crescător elementele listei.*

În lumea reală datele nu apar într-o anumită ordine, iar informația dezorganizată este aproape inutilă. Ordonarea datelor după anumite criterii nu este o preferință estetică, ci o necesitate pentru a eficientiza prelucrarea acestora.

Imaginează-ți cum ar fi să încerci să găsești un cuvânt într-un dicționar care nu este ordonat sau un număr de telefon într-o carte de telefon amestecată aleatoriu sau o carte într-o bibliotecă în care cărțile nu sunt aranjate după niciun criteriu. Ar fi un coșmar!

Utilizăm sortări foarte frecvent, atât în viața de zi cu zi, cât și în activitatea de programare. De exemplu, când căutăm un produs într-un magazin online, probabil vom ordona după preț, după rating sau după procentul de reducere. La fel, când căutăm informații folosind un motor de căutare, lista uriașă de rezultate găsite va fi ordonată după relevanța acestora.

Pentru început vom studia câțiva algoritmi de sortare simpli. Pe măsură ce vom avansa în studiul informaticii vom studia și algoritmi mai complecși, dar mai eficienți.

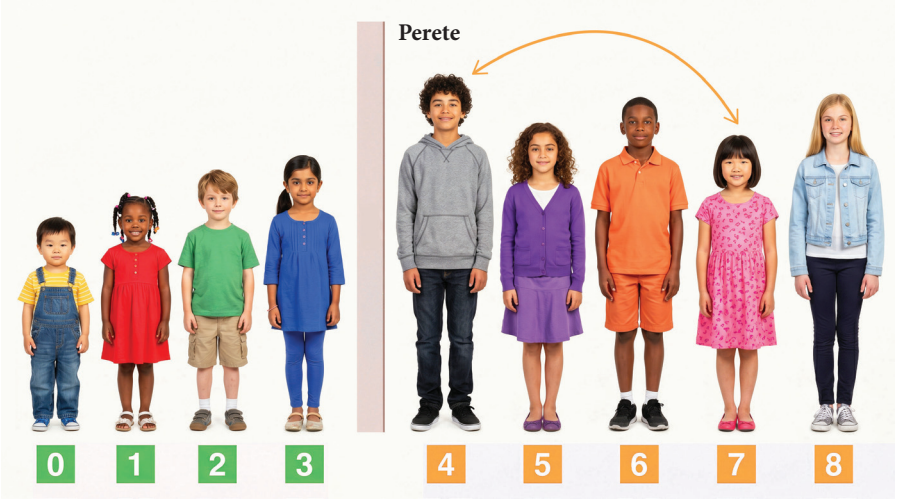
Sortarea prin selecție

Sortarea prin selecție este un algoritm intuitiv, deoarece oglindește modul în care noi, oamenii, am proceda atunci când trebuie să sortăm niște obiecte după un anumit criteriu.

Pentru a înțelege mai bine modul de funcționare a acestei metode, să o analizăm într-un context real.

Imaginează-ți o grupă de `9` elevi la ora de sport, așezați în linie, care trebuie ordonați de la cel mai scund la cel mai înalt. Antrenorul parcurge toată linia și îl caută pe cel mai scund. După ce l-a găsit, acesta face schimb de locuri cu primul elev din linie. Acum, primul loc este „ocupat” și corect sortat. Apoi antrenorul va afla cel mai scund elev începând de la al doilea mai departe și îl va interschimba cu elevul de pe poziția a doua. Procedeul se repetă până când toți elevii sunt ordonați după înălțime.

Metoda utilizată de antrenor este sortarea prin selecția minimului.



Imaginea 4

Strategia pe care se bazează algoritmul pornește de la observația că pe parcursul algoritmului, elementele listei sunt divizate în două zone, una ordonată, conținând elementele de pe primele poziții, iar cealaltă neordonată, conținând restul elementelor. Considerăm că ele sunt despărțite de un „perete” imaginar (*Imaginea 4*).

Algoritmul de selecție efectuează `n-1` pași succesivi care conțin operațiile:

1. identificarea celui mai mic element al zonei neordonate;
2. interschimbarea elementului minim determinat cu primul element al zonei neordonate.

După această selecție (identificare a elementului minim) și interschimbare, „peretele” despărțitor se va muta peste un element spre dreapta. În felul acesta se mărește cu 1 numărul de elemente al zonei ordonate și se micșorează cu 1 numărul de elemente al zonei neordonate.

De câte ori se efectuează această deplasare, spunem că s-a încheiat un pas complet al sortării. Dacă avem n elemente în listă, vor exista n-1 astfel de pași.

Să exersăm!

1 Urmărește în manualul digital execuția acestui algoritm pentru lista [23,78,45,18,32,56].

Funcția corespunzătoare algoritmului de sortare prin selecția minimului

Model de implementare 6

```
def sortare_selectie_minim(lista_numere):
    n = len(lista_numere)
    for pas in range(n - 1):
        poz_minim = pas
        for i in range(pas + 1, n):
            if lista_numere[poz_minim] > lista_numere[i]:
                poz_minim = i
        if poz_minim != pas:
            aux = lista_numere[pas]
            lista_numere[pas] = lista_numere[poz_minim]
            lista_numere[poz_minim] = aux
```

2 Realizează sortarea crescătoare a elementelor unei liste prin selecția maximului.

Sortarea pe baza „listei de frecvențe”

Imaginează-ți că ești managerul unui depozit de haine. Ai primit un camion cu 10000 de tricouri, dar acestea au doar 5 mărimi posibile: 36, 37, 38, 39 și 40. Tricourile vin amestecate și tu ar trebui să le ai ordonate după mărimi. Dacă ai folosi una dintre metodele precedente, ar trebui să compari tricourile între ele de milioane de ori. Este ineficient!

O idee ingenioasă ar fi să pregătești 5 containere, câte unul pentru fiecare mărime, apoi iei fiecare tricou și îl plasezi în containerul corespunzător mărimii sale. La final scoți toate tricourile din cutia 36, apoi pe cele din 37 etc.

Ideea aceasta poate fi generalizată pentru sortarea unui număr mare de numere naturale situate în intervalul [0, VMAX), unde VMAX este o valoare cunoscută, rezonabil de mică (astfel încât să fie posibilă și convenabilă memorarea unei liste cu VMAX elemente). În listă elementul de pe poziția x reprezintă numărul de apariții ale valorii x în șirul de elemente care trebuie să fie ordonat.

Pașii algoritmului de sortare pe baza listei de frecvențe

1. *Inițializare*: construim lista nr cu VMAX elemente egale cu 0.

2. *Numărare*: citim succesiv elementele din șir și le contorizăm; mai exact, când citim valoarea x, vom incrementa nr[x], deoarece am identificat o nouă apariție a acestei valori.

3. *Ordonare*: parcurgem lista nr și plasăm în lista ordonată fiecare valoare x de nr[x] ori.

De exemplu, să presupunem că VMAX=10 și că fișierul de intrare date.in conține valorile 2 5 6 2 3 8 8 4 5 6 2 4, câte o valoare pe o linie. După citirea elementelor și contorizarea acestora, lista nr va avea următorul conținut prezentat în *Imaginea 5*.

0	1	2	3	4	5	6	7	8	9
0	0	3	1	2	2	2	0	2	0

Imaginea 5

Funcția de sortare pe baza listei de frecvențe

Model de implementare 7

```
def sortare_lista_frecvente(VMAX):
    nr = [0] * VMAX
    with open("date.in", "r") as fin:
        for linie in fin:
            x = int(linie)
            nr[x] += 1
    lista_sortata = []
    for x in range(VMAX):
        for i in range(nr[x]):
            lista_sortata.append(x)
    return lista_sortata
```

Să exersăm!

3 Modifică algoritmul astfel încât să sorteze valorile din șir în ordine descrescătoare.

4 Urmărește în manualul digital execuția acestui algoritm de sortare pentru șirul `[2 5 6 2 3 8 8 4]`.

11. Fișă de autoevaluare

1 Considerăm lista `L=[5,7,3,9,4,2]`. Selectează numărul de comparații între elementele listei care se vor efectua în cazul sortării prin metoda selecției minimului.

a. `8` **b.** `15` **c.** `10` **d.** `12`

2 În fiecare dintre situațiile de mai jos este necesar să realizăm o sortare. Selectează pentru care dintre situațiile de mai jos este indicată sortarea pe baza listei de frecvențe.

a. vârstele tuturor oamenilor dintr-un oraș;

b. codurile numerice personale (CNP) ale cetățenilor români;

c. bancnote de `1`, `5`, `10`, `20`, `100` RON;

d. tranzacțiile bancare de la o bancă, fiecare tranzacție fiind identificată printr-un cod intern unic de maximum `20` de cifre.

3 Se consideră următorul program:

```
x = int(input("x="))
L = []
while x:
    L.append(x % 10)
    x = x // 10
for i in range(len(L)): print(L[i], end="")
```

a. Ce se va afișa pentru `x=20341`?

b. Dă exemplu de o valoare pentru `x` pentru care se va afișa un șir de cifre identic cu cel citit.

c. Modifică programul astfel încât să afișeze cel mai mare număr ce se poate forma cu cifrele lui `x`.

Portofoliu digital

1 ***Sortare MinMax***

Scrie un program care sortează elementele situate între minimul și maximul unei liste de numere distincte și returnează lista obținută. Dacă minimul și maximul se află pe indici consecutivi, se va afișa mesajul „`Nu se efectuează sortarea`”.

De exemplu, pentru lista `[14,-13,23,1,120,1000,21]` se va returna: `[14,-13,1,23,120,1000,21]`.

2 ***Radar***

Ești programator pentru Poliția Rutieră. Un radar inteligent a înregistrat pe linii separate în fișierul `radar.txt` vitezele tuturor mașinilor care au trecut printr-o zonă cu limită de `50` km/h. Pentru a genera amenzile este necesară o funcție care să construiască o listă în care sunt memorate doar vitezele mai mari de `50` km/h în ordine crescătoare.

3 ***Muzică***

O aplicație de muzică salvează ID-ul melodiei de fiecare dată când este redată, ID-urile fiind numere naturale mai mici decât `1000`. De exemplu, `melodii=[7,12,7,3,12,7,5,3,12,12,5]`. Scrie un program care afișează lista melodiilor ordonată descrescător după popularitate (cele mai ascultate melodii vor fi primele).

Evaluarea Portofoliului digital

Accesează manualul digital pentru a afla cum să îți evaluezi portofoliul digital din acest an școlar.

12. Proiect: Decoderul Astro-Sort

Satelitul Astro-Link a trimis un pachet de date vitale către Pământ, dar din cauza unei furtuni magnetice, numerele au ajuns amestecate și „murdărite” de interferențe cosmice. Tu ești Ofițerul de Comunicații: misiunea ta este să preiei aceste numere, să le cureți și să le organizezi într-o listă perfectă pentru a descifra mesajul ascuns.

Un pachet de date este valid și poate fi decodat doar dacă:

1. Dimensiune optimă: Lista conține exact `6` coduri (numere).
2. Filtru de siguranță: Toate numerele din listă sunt mai mici sau egale cu `1000`.
3. Sincronizare: Lista este sortată crescător.
4. Codul de mijloc: Al patrulea element din lista sortată (indexul `3`) este un număr par.

Doar dacă toate cele patru condiții sunt îndeplinite, sistemul afișează mesajul secret.

Interacțiunea cu utilizatorul

Programul rulează un meniu interactiv unde poți gestiona lista de date:

- Adaugă cod: Cere un număr și îl adaugă în listă.
- Șterge cod: Elimină un număr greșit (introdus din eroare).
- Sortează datele: Ordonează toate numerele din listă.
- Decodează: Verifică cele patru condiții de mai sus și afișează rezultatul verificărilor.

Dacă toate condițiile sunt îndeplinite, afișează mesajul: „Conexiune stabilită: Bun venit, Căpitanule!”. Altfel, indică erorile găsite pentru a fi remediate.

Implementează această aplicație, lucrând în echipă cu mai mulți colegi. Prezentați aplicația colegilor de clasă, apoi invitați-i să o testeze și să raporteze eventualele erori identificate.

Etape în dezvoltarea aplicației

Etapa I. Dezbatere – Proiectarea soluției

Realizați o dezbatere în cadrul echipei și stabiliți principalele repere referitoare la proiectarea aplicației.

A. Analiza problemei

- Care este scopul aplicației?
- Ce date sunt generate de calculator?
- Ce date sunt introduse de utilizator?
- Identificarea modului de interacțiune cu utilizatorul

B. Identificarea subproblemelor și stabilirea funcțiilor necesare

Pentru fiecare funcție specificați:

- numele funcției,
- parametrii funcției și semnificația acestora,
- descrierea subproblemei rezolvate de funcție și semnificația rezultatului returnat de aceasta, dacă este cazul.

De exemplu:

- `adauga_cod(Lista, Cod)`
- primește ca parametri lista și codul care trebuie adăugat în listă
- verifică dacă numărul de elemente din listă este mai mic decât 6 și dacă da, adaugă codul în listă
- returnează lista obținută după adăugare

Etapa a II-a. Stabilirea rolurilor în cadrul echipei

Stabiliți rolurile în cadrul echipei. Puteți decide ca, pe parcursul dezvoltării aplicației, același elev să dețină mai multe roluri.

- *Coordonator de echipă*: organizează activitățile echipei, se asigură că toți membrii își îndeplinesc la timp sarcinile de lucru, comunică cu profesorul
- *Programator* – implementează aplicația (funcțiile proiectate)
- *Programator* – proiectează și implementează interfața
- *Tester* – testează aplicația, identifică și raportează erori
- *Documentarist* – scrie documentația aplicației, concepe strategia și materialele de prezentare

Etapa a III-a. Investigație – documentare pentru grafică în Python

Realizați o investigație online în scopul identificării principalelor elemente necesare pentru implementarea interfeței aplicației, folosind Python.

Etapa a IV-a. Dezvoltarea aplicației

Etapa a V-a. Testare și îmbunătățire

Concepeți diferite scenarii de testare a aplicației și corectați diferitele erori detectate.

Etapa a VI-a. Pregătirea și susținerea prezentării aplicației

Cum știi că ai reușit?

Te vei documenta din surse sigure și vei selecta informațiile relevante pentru tema dată. Vei analiza cerințele și vei implementa soluția cea mai potrivită. Vei testa funcționarea soluției tale și vei corecta eventualele erori. La final, te vei asigura că proiectul tău este complet.

Fișă de observare a comportamentului și a modului de lucru în echipă

Utilizează următoarea listă de verificare pentru a monitoriza cât de bine lucrezi în cadrul echipei de proiect. Pentru fiecare afirmație, completează X în coloana corespunzătoare (Acord, Parțial, Deloc) din Tabelul 15:

Tabelul 15

Comportamentul	Acord	Parțial	Deloc
Am participat activ la toate activitățile echipei.			
Mi-am asumat rolul de lider într-un mod corespunzător, când a fost necesar, pentru a mă asigura că realizăm o aplicație funcțională.			
I-am ascultat cu atenție pe ceilalți membri ai echipei.			
Am oferit echipei mele feedback constructiv.			
Mi-am ajutat colegii de echipă atunci când aceștia au avut nevoie de mine.			
Am finalizat la timp sarcinile desemnate.			
I-am tratat cu respect pe membrii echipei mele, chiar și când nu eram de acord cu ei.			
Am păstrat o atitudine pozitivă față de proiect, chiar și atunci când au apărut provocări.			

Observ!

Accesează manualul digital pentru a completa *Fișa de observare a comportamentului*.

13. Evaluare

Timp de lucru: 50 de minute
Se acordă 10 puncte din oficiu

1 Se consideră lista `L=[1,3,7]`. Executarea instrucțiunii `L.append(7)`:

a. elimină valoarea `7` de la finalul listei `L`;
b. inserează valoarea `7` la începutul listei `L`;
c. adaugă valoarea `7` la finalul listei `L`;
d. generează eroare la execuție. *12 p*

2 La executarea următoarei secvențe de instrucțiuni se va afișa pe ecran valoarea:

```
L=[2,4,6,8,7,12,13]
s=0
for x in L:
    if x % 4 == 0:
        s += x
print(s)
```

a. `12`;
b. `32`;
c. `8`;
d. `24`.

13 p

3 Se consideră lista `nr=[2,0,1,3,0,2]`, unde `nr[x]`=numărul de apariții ale valorii `x` într-o listă de numere naturale `L`. Ce valoare va fi plasată în `L` pe poziția `5`, după ordonarea crescătoare a acesteia?

a. `2`; **b.** `3`; **c.** `4`; **d.** `5`. *12 p*

4 Ce se va afișa pe ecran după executarea următoarei secvențe de instrucțiuni dacă se citește valoarea `715329`?

```
x = int(input())
L = []
i = 0
while x:
    i += 1
    if i % 2 == 0:
        L.insert(0, x % 10)
    else:
        L.insert(i, x % 10)
    x //= 10
print(L)
```

13 p

5 O firmă de curierat primește `n` colete, fiecare având un cod numeric de identificare.

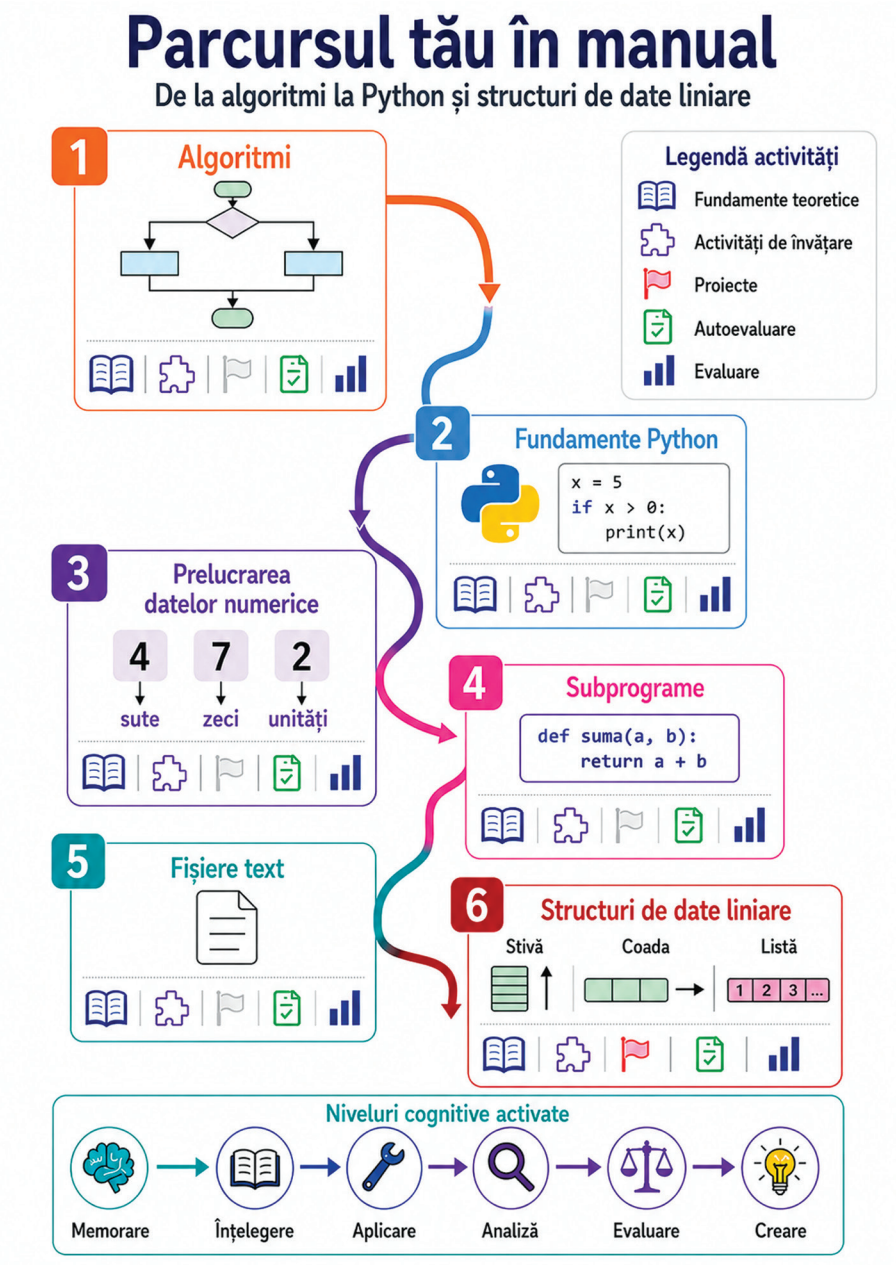
1. Fișierul text `colete.in` conține pe prima linie numărul natural `n`, iar pe liniile următoare `n` numere naturale nenule reprezentând codurile celor `n` colete, câte un cod pe o linie. Scrie o funcție denumită `citire` care să citească codurile și să le memoreze într-o listă `L`, transmisă ca parametru. *5 p*
2. Scrie o funcție denumită `control` care să calculeze cifra de control a unui număr natural primit ca parametru. Cifra de control a unui număr natural `x` se obține însumând cifrele numărului `x`. Dacă suma obținută este o singură cifră, aceasta este cifra de control a lui `x`. În caz contrar, repetăm procedeul pentru suma cifrelor, până când suma obținută este un număr de o singură cifră. De exemplu, pentru `x=99937`, suma=`9+9+9+3+7=37`. Suma are două cifre, continuăm adunările `3+7=10`; noua sumă are două cifre, deci continuăm adunările `1+0=1`, prin urmare `1` este cifra de control. *10 p*
3. Apelează funcția `control` pentru a determina cifrele de control ale codurilor memorate în lista `L`; cifrele de control obținute vor fi memorate într-o listă `C`. *5 p*
4. Scrie o funcție care să sorteze descrescător cifrele de control memorate în lista `C` pentru a prioritiza la livrare coletele în funcție de cifrele de control. Alege o metodă de sortare eficientă și justifică alegerea. *10 p*
5. Scrie o funcție care să afișeze elementele listei `C` sortate în fișierul `colete.out`. Cifrele vor fi scrise pe aceeași linie, fără a fi separate prin spații. *5 p*
6. Apelează funcțiile definite, pentru a rezolva cerințele. *5 p*

Observ!

Accesează manualul digital pentru a descoperi soluțiile exercițiilor și problemelor din fișele de evaluare.

Recomandări resurse WEB

- pbInfo – https://www.pbinfo.ro/
- PEP 8 – Style Guide for Python Code – https://peps.python.org/pep-0008/
- Python 3. Machine learning pentru elevii de liceu – https://www.pyml.ro/
- Python Documentation – https://docs.python.org/3/
- Python Tutor – https://pythontutor.com/
- Real Python – https://realpython.com/
- Think Python – https://greenteapress.com/thinkpython/thinkpython.pdf
- W3 Schools Python Tutorial – https://www.w3schools.com/python/default.asp



intuitext
grup SOFTWIN

INFORMATICĂ

CLASA A IX-A

www.intuitext.ro