Lecții suport pentru început de an școlar

9

Informatică

curriculum de specialitate (CS), filiera teoretică, profilul real, specializarea matematică-informatică

clasa a IX-a

SIGMA

Capitolul 1 – Strategii de rezolvare a problemelor. Fundamentele programării
(1.3.; 1.4.; 2.1.; 2.3.; 2.4.; 3.3.; 3.4.; 4.1.; 4.3.; 4.4.; 5.3.; 6.3.)

Capitolul 1 – Strategii de rezolvare a problemelor. Fundamentele programării

Lecția 1 – Etapele de elaborare a unui program: de la analiză și proiectare la testare și depanare

Fișa de identificare a lecției

Element	Conținut
Clasa	a IX-a
Durata	50 de minute
Tipul lecției	Dobândire și aplicare de cunoștințe
Limbaj	Pseudocod și C++
Prerechizite	Operații aritmetice, media aritmetică, comparații și noțiuni elementare despre calculator
Resurse	Calculator, mediu Python, videoproiector, fișă de lucru și tablă
Conținuturi	Date, algoritmi, programe, etapele elaborării unui program, testare și depanare

Competențe urmărite

- Analizarea unei probleme și identificarea datelor necesare rezolvării.
- Descrierea unei succesiuni finite și ordonate de pași pentru rezolvarea problemei.
- Implementarea și verificarea unei soluții simple într-un limbaj de programare.
- Identificarea erorilor și îmbunătățirea soluției pe baza testelor.

Obiective de învățare

La finalul lecției, elevul va fi capabil:

- să explice relația dintre date, algoritm și program;
- să enumere și să descrie etapele elaborării unui program;
- să distingă erorile de sintaxă, de logică și de execuție;
- să proiecteze cazuri de test generale, limită și invalide, în funcție de restricțiile problemei;
- să urmărească evoluția unor variabile și să corecteze o eroare simplă.

Scenariul didactic

Etapă	Timp	Activitatea profesorului	Activitatea elevilor	Dovada învățării
Conectare	5 min	Prezintă un program care rulează, dar calculează greșit media notelor unui elev.	Anticipează cauza și formulează întrebări.	Ipoteză argumentată
Explorare	8 min	Conduce discuția despre informații, date și pași de rezolvare.	Identifică intrările, ieșirile și pașii.	Schemă date-algoritm-rezultat

Explicare	15 min	Clarifică etapele elaborării și tipurile de erori.	Notează, compară și oferă exemple.	Răspunsuri orale
Aplicare	17 min	Ghidează problema mediei notelor.	Analizează, proiectează și testează soluția.	Algoritm și tabel de teste
Verificare	5 min	Aplică evaluarea și oferă feedback.	Răspunde și își evaluează înțelegerea.	Trei răspunsuri scurte

Întrebarea esențială

Întrebare: Cum putem demonstra că un program rezolvă corect problema cerută, nu doar că rulează?

Situația de pornire

Un elev scrie un program care calculează media notelor. Programul pornește și afișează un rezultat, dar pentru notele 8, 9 și 10 afișează 8. Profesorul invită clasa să decidă dacă programul este corect și ce informații sunt necesare pentru a localiza problema.

Predicție: Elevii notează dacă eroarea este în înțelegerea cerinței, în algoritm, în cod sau în datele de test și justifică alegerea.

Informatică, date și algoritmi

Informatica studiază reprezentarea, prelucrarea și transmiterea informației, în special cu ajutorul sistemelor de calcul. Rezolvarea informatică a unei probleme pornește de la date și folosește un algoritm care poate fi implementat într-un program.

Gândirea computațională

Gândirea computațională este o modalitate de abordare a problemelor prin formularea unor soluții care pot fi descrise precis și, atunci când este potrivit, executate de un calculator.

Componentă	Întrebare de lucru	Exemplu
Descompunerea	În ce subprobleme mai mici putem împărți problema?	Citirea notelor, calculul sumei, calculul mediei, decizia
Recunoașterea modelelor și generalizarea	Ce soluție cunoscută se poate adapta?	Media unei liste de valori
Abstractizarea	Ce informații sunt esențiale?	Numărul notelor și valorile lor
Algoritmizarea	Care este ordinea exactă a pașilor?	Citire, calcul, comparație, afișare
Evaluarea	Este soluția corectă și eficientă?	Testare cu valori obișnuite și limită

Datele

O dată este o reprezentare asupra căreia un algoritm poate opera. Într-un program, datele sunt caracterizate prin nume, tip și valoare. Datele elementare includ, de exemplu, numerele, valorile logice și caracterele. Datele structurate grupează mai multe valori, ca într-o listă sau într-un tablou.

- După rol: date de intrare, date intermediare sau de lucru și date de ieșire.
- După posibilitatea modificării: variabile și constante.
- În calculator, datele sunt reprezentate digital, prin secvențe de biți.

Algoritmul și programul

Un algoritm este o succesiune finită, clară și ordonată de pași care transformă datele de intrare în rezultate. Pentru aceeași clasă de probleme pot exista algoritmi diferiți, iar aceștia pot fi comparați după corectitudine, claritate, timp de execuție și memorie utilizată.

Programul este implementarea unui algoritm într-un limbaj de programare. El trebuie să respecte sintaxa limbajului și să realizeze corect prelucrarea stabilită prin algoritm. Faptul că un program rulează nu garantează că rezultatele sale sunt corecte.

Etapele elaborării unui program

Etapă	Întrebarea principală	Rezultat
Analiza problemei	Ce se cunoaște, ce se cere și ce restricții există?	Intrări, ieșiri, relații și condiții
Proiectarea soluției	Cum transformăm datele în rezultatul dorit?	Algoritm, pseudocod sau diagramă
Implementarea	Cum exprimăm algoritmul în limbajul ales?	Cod sursă clar și lizibil
Testarea	Pentru ce date verificăm și ce rezultat așteptăm?	Cazuri de test și rezultate comparate
Depanarea	Unde apare cauza rezultatului greșit?	Eroare localizată și corectată
Analiza și optimizarea	Putem reduce timpul sau memoria fără să pierdem corectitudinea?	Soluție evaluată și, dacă este necesar, îmbunătățită

În practică, etapele nu sunt complet liniare. Un test nereușit poate impune revenirea la implementare, proiectare sau chiar la analiza cerinței.

Testarea și depanarea

Proiectarea testelor

Un test conține date de intrare, un rezultat așteptat stabilit înaintea rulării și rezultatul obținut. Testele se aleg din cerință și din restricții; o valoare negativă este invalidă numai dacă problema o interzice.

Categorie	Scop	Exemplu pentru note
Caz general	Verifică situația obișnuită	n = 3, note 8, 9, 10
Caz limită	Verifică valorile de la marginea domeniului	notele 1 și 10; media exact 5
Caz special	Verifică o configurație aparte, dar permisă	o singură notă; toate notele egale
Date invalide	Verifică respectarea restricțiilor	n = 0; notă 11; text în loc de număr

Tipuri de erori

Tip	Manifestare	Exemplu
Eroare de sintaxă	Codul nu respectă regulile limbajului și nu poate fi interpretat corect.	Lipsa semnului ; după o instrucțiune C++
Eroare de logică	Programul rulează, dar produce rezultate greșite.	Împărțirea sumei la n - 1 în loc de n
Eroare de execuție	Programul se oprește în timpul rulării.	Împărțire la zero când n = 0

Urmărirea variabilelor

- Realizează manual un tabel cu valorile variabilelor după fiecare pas important.
- Folosește temporar instrucțiuni de afișare pentru a observa valorile intermediare.
- Rulează pas cu pas cu ajutorul unui depanator, dacă mediul îl oferă.
- Corectează o singură cauză, apoi repetă toate testele relevante.

Exemplu rezolvat: media notelor

Cerință: Mihai are n note la disciplina Informatică. Determinați media aritmetică a notelor și afișați dacă a promovat. Considerăm că promovarea are loc pentru o medie mai mare sau egală cu 5.

Analiza problemei

Categorie	Conținut
Date de intrare	n, număr natural nenul; n note reale din intervalul [1, 10]
Date de ieșire	media notelor și mesajul PROMOVAT sau NEPROMOVAT
Date de lucru	suma notelor și nota citită la pasul curent
Relații	media = suma / n; promovat dacă media >= 5
Restricții	n >= 1 și fiecare notă aparține intervalului [1, 10]

Proiectarea algoritmului

1. Citește numărul n.
2. Inițializează suma cu 0.
3. Repetă de n ori: citește o notă și adaug-o la sumă.
4. Calculează media ca suma împărțită la n.
5. Afișează media.
6. Dacă media este cel puțin 5, afișează PROMOVAT; altfel, afișează NEPROMOVAT.

Pseudocod

```
citește n
suma ← 0
pentru i ← 1, n execută
   citește nota
   suma ← suma + nota
media ← suma / n
scrie media
dacă media ≥ 5 atunci
   scrie „PROMOVAT”
altfel
   scrie „NEPROMOVAT”
      else:
         print('NEPROMOVAT')
```

Tabel de teste

Nr.	Date de intrare	Tip	Rezultat așteptat
1	n = 3; 8, 9, 10	general	media 9,00; PROMOVAT
2	n = 2; 4, 6	limită	media 5,00; PROMOVAT
3	n = 1; 4	special	media 4,00; NEPROMOVAT
4	n = 3; 10, 10, 10	special	media 10,00; PROMOVAT
5	n = 0	invalid	Număr invalid de note
6	n = 2; 8, 11	invalid	Mesaj privind note în afara intervalului

Complexitate

Algoritmul parcurge fiecare notă o singură dată. Timpul de execuție este proporțional cu n, adică O(n), iar memoria suplimentară utilizată este constantă, O(1), deoarece notele nu sunt păstrate într-o listă.

Activități și exerciții pentru elevi

Activitate ghidată

Sarcină: Pentru notele 6, 4 și 8, completează un tabel cu valorile variabilelor i, nota și suma după fiecare repetare. Calculează media și stabilește mesajul afișat.

Criterii de reușită: suma este actualizată corect la fiecare pas; media este calculată după citirea tuturor notelor; condiția de promovare este aplicată rezultatului calculat.

Exerciții

1. Ordonează etapele: implementare, analiză, depanare, proiectare, testare și optimizare. Explică de ce testarea precedă depanarea.
2. Propune câte un caz general, limită și invalid pentru o problemă care citește vârsta unei persoane din intervalul [0, 120].
3. Modifică algoritmul mediei astfel încât să afișeze și numărul notelor mai mici decât 5.

Diferențiere

- Sprijin: elevul primește algoritmul cu doi pași lipsă și un tabel de urmărire parțial completat.
- Nivel de bază: elevul identifică intrările, ieșirile și tipurile de erori.
- Aprofundare: elevul compară două implementări și justifică diferențele de memorie.

Temă pentru acasă

Alege o problemă simplă din viața cotidiană, precizează datele de intrare și de ieșire, descrie algoritmul în 5-8 pași și propune trei teste: general, limită și invalid sau special.

Evaluare

1. Care este diferența dintre algoritm și program?
2. De ce un program care rulează poate fi totuși incorect?
3. Scrie un test limită pentru problema mediei notelor și precizează rezultatul așteptat.

Repere pentru profesor

Sugestii metodice

- Porniți de la un rezultat greșit produs de un program care rulează; situația separă rapid noțiunea de executare de cea de corectitudine.
- Solicitați elevilor să precizeze rezultatul așteptat înainte de rulare. Altfel, testarea riscă să devină simplă observare.
- Insistați asupra faptului că validitatea unei valori depinde de restricțiile problemei.
- Folosiți tabelul de urmărire înaintea depanatorului software, pentru ca elevii să înțeleagă schimbarea stării programului.
- Încurajați explicațiile despre motivul alegerii testelor, nu doar enumerarea unor valori.

Întrebări de ghidare

- Ce informații din enunț devin date de intrare?
- Cum știm ce rezultat ar trebui să obținem?
- Care este prima valoare pentru care soluția ar putea eșua?
- Dacă programul rulează, ce tip de eroare mai poate exista?
- După corectare, ce teste trebuie repetate?

Soluții orientative

Exercițiu	Răspuns orientativ
1	Analiză, proiectare, implementare, testare, depanare, optimizare. Depanarea folosește simptomele observate la testare pentru a localiza eroarea.
2	Exemple: 25 general; 0 și 120 limită; -1 sau 121 invalid.
3	Se inițializează un contor cu 0 și se mărește atunci când nota < 5.

Lecția 2 – Reprezentarea algoritmilor: scheme logice, pseudocod, limbaje de programare

Fișa de identificare a lecției

Element	Conținut
Clasa	a IX-a
Durata recomandată	Două ore; conținutul poate fi împărțit în reprezentări și implementare
Tipul lecției	Dobândire, comparare și aplicare de cunoștințe
Limbaj	Pseudocod și C++
Prerechizite	Noțiunile de dată, algoritm, program, condiție și repetare
Resurse	Calculator, compilator C++, videoproiector, fișă de lucru
Conținuturi	Scheme logice, pseudocod, structuri de control și elemente de C++

Competențe urmărite

- Reprezentarea unui algoritm prin forme grafice și textuale adecvate.
- Recunoașterea și utilizarea structurilor secvențială, alternativă și repetitivă.
- Transpunerea unei soluții simple din pseudocod în C++.
- Compararea reprezentărilor după claritate, precizie și posibilitatea execuției.

Obiective de învățare

La finalul lecției, elevul va fi capabil:

- să identifice rolul simbolurilor principale dintr-o schemă logică;
- să descrie cele trei structuri fundamentale de control;
- să scrie pseudocod pentru algoritmi simpli;
- să recunoască elementele de bază ale unui program C++;
- să transpună aceeași soluție în schemă logică, pseudocod și C++;
- să aleagă reprezentarea potrivită unei etape de lucru.

Scenariul didactic

Etapă	Timp	Activitatea profesorului	Activitatea elevilor	Dovada învățării
Conectare	8 min	Prezintă trei reprezentări ale aceleiași soluții.	Identifică asemănările și diferențele.	Ipoteză de corespondență
Explorare	20 min	Introduce simbolurile și structurile de control.	Urmărește fluxul și verbalizează pașii.	Traseu explicat
Explicare	22 min	Clarifică pseudocodul și elementele C++.	Compară sintaxa și rolul instrucțiunilor.	Tabel de corespondențe
Aplicare	40 min	Propune probleme pentru cele trei structuri.	Proiectează și transpune soluțiile.	Scheme, pseudocod și cod
Verificare	10 min	Aplică evaluarea.	Răspunde și își evaluează progresul.	Răspunsuri scurte

Întrebarea esențială

Întrebare: Cum putem exprima aceeași idee algoritmică în forme diferite fără să-i schimbăm sensul?

Situația de pornire

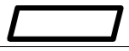
Un coleg primește o schemă logică, altul un algoritm în pseudocod, iar un al treilea un program C++. Toate ar trebui să calculeze suma a două numere. Elevii trebuie să decidă ce elemente corespund între cele trei reprezentări și care dintre ele poate fi executată direct de calculator.

Trei moduri de reprezentare

Un algoritm poate fi reprezentat în mai multe forme. Alegerea depinde de scop: înțelegerea vizuală a fluxului, proiectarea independentă de un limbaj sau executarea efectivă de către calculator.

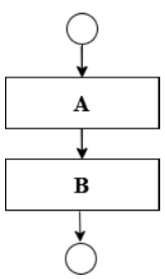
Reprezentare	Caracteristici	Avantaj principal	Limită
Schema logică	Folosește simboluri și săgeți pentru flux.	Face vizibile ramificațiile și repetările.	Devine greu de urmărit pentru algoritmi mari.
Pseudocodul	Folosește instrucțiuni convenționale, apropiate de limbajul natural.	Este clar și independent de limbajul de programare.	Nu poate fi executat direct.
C++	Folosește vocabular și sintaxă strictă.	Programul poate fi compilat, executat, testat și reutilizat.	Detaliile de sintaxă pot ascunde inițial ideea algoritmică.

Simbolurile principale ale schemei logice

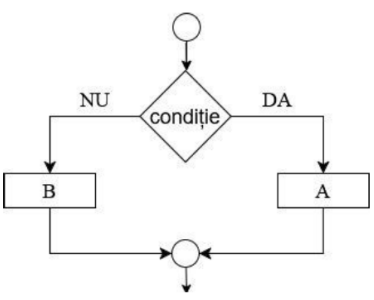
	***Linie de flux***: realizează legătura între blocuri, indicând ordinea de executare a pașilor
	***Bloc de calcul***: este folosit pentru efectuarea diverselor operații și atribuiri
	***Bloc de subprogram***: reprezintă un apel al subprogramului care are o schemă logică separată
	***Bloc de intrare/ieșire***: reprezintă operația de citire/scriere a datelor
	***Bloc de decizie***: reprezintă o condiție care are ca rezultat două ieșiri (Da/Nu sau Adevărat/Fals)
	***Bloc terminal***: reprezintă momentul de început (***START***) sau de final (***STOP***).
	***Bloc conector***: este folosit pentru a conecta două sau mai multe puncte ale schemei

Structurile fundamentale de control

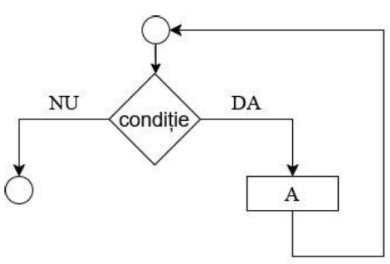
Orice algoritm poate fi construit folosind structura secvențială, alternativă și repetitivă.



structura liniară



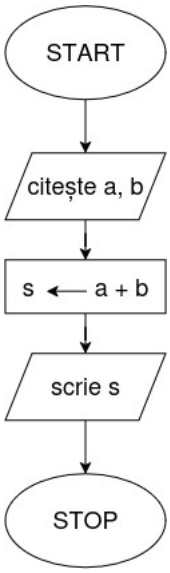
structura alternativă



structura repetitivă

Structura secvențială

Exemplu: se citesc două numere a și b, se calculează suma lor, apoi se afișează rezultatul. Fiecare instrucțiune se execută o singură dată.



Pseudocod

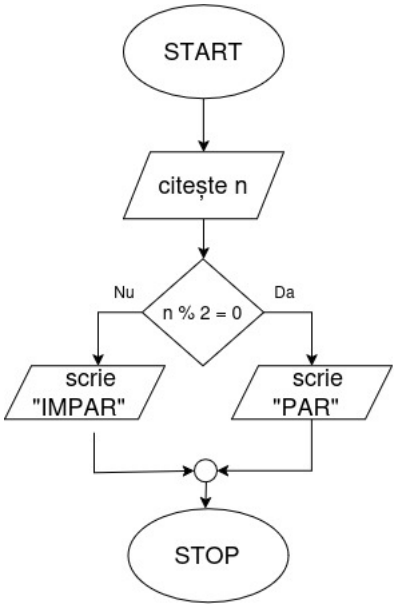
```
citește a, b
s ← a + b
scrie s
```

C++

```
int a, b;
cin >> a >> b;
s = a + b;
cout << s;
```

Structura alternativă

Structura alternativă selectează instrucțiunea sau grupul de instrucțiuni care se execută în funcție de valoarea unei condiții. Condiția are valoarea adevărat sau fals.



Pseudocod

```
citește n
dacă n mod 2 = 0 atunci
   scrie „PAR”
altfel
   scrie „IMPAR”
```

C++

```
int n;
cin >> n;
if (n % 2 == 0)
   cout << "PAR";
else
   cout << "IMPAR";
```

Testare

Intrare	Ramură	Ieșire
8	condiție adevărată	PAR
5	condiție falsă	IMPAR
0	condiție adevărată	PAR

Structurile repetitive

Structura repetitivă execută de mai multe ori un corp de instrucțiuni. Repetarea poate fi controlată de o condiție sau de un număr cunoscut de pași.

Formă	Locul testului	Număr minim de execuții	Utilizare
Cu testare inițială	Înaintea corpului	0	Corpul se execută numai dacă expresia este adevărată.
Cu testare finală	După corp	1	Corpul trebuie executat cel puțin o dată.
Cu număr cunoscut de pași	Control la început	Stabilit de interval	Numărul repetărilor este cunoscut sau calculabil.

Repetare cu testare inițială

Exemplu: se citesc numere până la apariția valorii 0 și se calculează suma lor.

Pseudocod pentru repetări

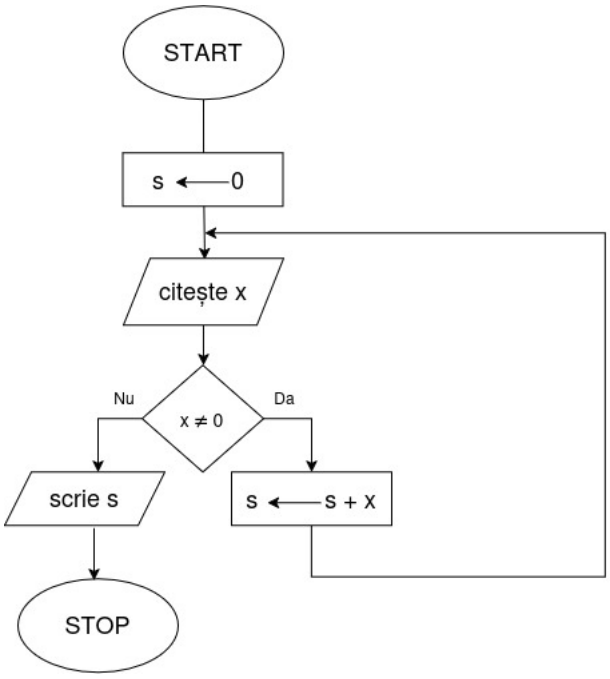
Testare inițială
```
s ← 0
citește x
cât timp x ≠ 0 execută
   s ← s + x
   citește x
scrie s
```

Testare finală
```
repetă
   citește x
   s ← s + x
până când x=0
```

În prima formă, corpul poate să nu se execute. În forma cu testare finală, corpul se execută cel puțin o dată.

Număr cunoscut de pași

Exemplu: dintre valorile înscrise pe n bile se determină valoarea maximă. Prima valoare trebuie folosită pentru inițializare atunci când domeniul nu garantează că valorile sunt pozitive.



Pseudocod

```
citește n, x
maxim ← x
pentru i ← 2, n execută
   citește x
   dacă x > maxim atunci
      maxim ← x
scrie maxim
```

C++

```
int n, x;
cin >> n >> x;
maxim = x;
for(int i = 2; i <= n; i++)
{
      cin >> x;
      if (maxim < x) maxim = x;
}
cout << maxim;
```

Elemente introductive de C++

Un limbaj de programare are vocabular și reguli de scriere. Un mediu de dezvoltare oferă editarea codului, compilarea, rularea și depanarea. Compilatorul traduce codul sursă și semnalează erorile de sintaxă detectate.

Element	Rol	Exemplu
Identificator	Nume ales pentru o dată sau o funcție.	suma, nr, main
Cuvânt cheie	Cuvânt rezervat de limbaj.	int, if, else, while, for
Separator și delimitator	Separă sau grupează elementele.	; , ( ) { }
Comentariu	Explică intenția codului și este ignorat la execuție.	// calculează suma
Operator	Construiește expresii.	+ - * / % < == &&

Tipuri elementare de date

Tip	Categorie	Exemplu
int	număr întreg	-12, 0, 27
long long	număr întreg cu domeniu mai larg	10000000000
float, double	număr real	3.14
char	un caracter	'A'
bool	valoare logică	true, false

Un tip stabilește valorile posibile și modul de reprezentare în memorie. Un șir de caractere se poate reprezenta prin tipul string, care nu este același lucru cu tipul char.

Corespondențe între pseudocod și C++

Pseudocod	C++	Observație
citește x	cin >> x;	x trebuie declarat
scrie x	cout << x;	se poate afișa și text
x ← expresie	x = expresie;	= este atribuire
dacă ... atunci ... altfel	if (...) { ... } else { ... }	condiția este între paranteze
cât timp ... execută	while (...) { ... }	testare inițială
repetă ... până când	do { ... } while (...);	În C++ condiția lui while exprimă continuarea.
pentru i ← 1, n	for (int i=1; i<=n; i++)	control explicit al contorului

Exemplu integrat

Cerință: Se citesc n numere naturale. Determinați suma numerelor care au exact două cifre.

Pseudocod

```
citește n
s ← 0
pentru i ← 1, n execută
   citește x
   dacă 10 ≤ x și x ≤ 99 atunci
      s ← s + x
scrie s
```

C++

```
int n, x, s = 0;
cin >> n;
for (int i = 1; i <= n; i++) {
   cin >> x;
   if (x >= 10 && x <= 99)
      s += x;
}
cout << s;
```

Activități și exerciții pentru elevi

Activitate ghidată

Sarcină: Asociază fiecare bloc din schema structurii alternative cu instrucțiunea corespunzătoare din pseudocod și C++. Explică traseul pentru n = 7 și n = 12.

Criterii de reușită: sunt identificate intrarea, condiția, cele două ramuri și ieșirea; rezultatele sunt justificate prin valoarea condiției.

Exerciții

1. Precizează simbolul potrivit pentru citire, calcul, decizie și terminarea algoritmului.
2. Desenează schema logică și scrie pseudocodul pentru calculul ariei unui dreptunghi.
3. Scrie pseudocodul care afișează maximul dintre două numere distincte.
4. Transformă o repetare de n pași într-o repetare cu testare inițială și contor explicit.
5. Corectează condiția care ar trebui să recunoască numerele cu două cifre: $x > 10$ și $x < 99$.
6. Provocare: proiectează algoritmul care determină maximul și numărul aparițiilor sale într-un șir de n valori.

Diferențiere

- Sprijin: simbolurile și instrucțiunile sunt oferite pe cartonașe pentru asociere.
- Nivel de bază: elevul folosește câte o singură structură de control.
- Aprofundare: elevul combină repetarea cu alternativa și justifică inițializarea.

Evaluare

1. Care reprezentare poate fi executată direct și în ce condiții?
2. De ce o repetare cu testare finală execută corpul cel puțin o dată?
3. Scrie echivalentul C++ pentru instrucțiunea pseudocod $x \leftarrow x + 1$.

Repere pentru profesor

Sugestii metodice

- Prezentați aceeași problemă în toate cele trei reprezentări; corespondența este mai importantă decât memorarea izolată a sintaxei.
- Cereți elevilor să verbalizeze traseul din schemă pentru valori concrete.
- Subliniați diferența dintre condiția de continuare și condiția de oprire la repetările cu testare finală.
- Evitați inițializarea maximului cu 0 când domeniul poate conține valori negative; folosiți prima valoare.

Soluții orientative

Exercițiu	Răspuns orientativ
1	Citire: paralelogram; calcul: dreptunghi; decizie: romb; terminare: oval.
2	citește L, l; $A \leftarrow L \times l$; scrie A.
3	dacă a > b atunci scrie a, altfel scrie b.
4	$i \leftarrow 1$; cât timp $i \le n$ execută corpul și apoi $i \leftarrow i + 1$.
5	Condiția corectă, incluzând limitele: $x \ge 10$ și $x \le 99$.
6	Se inițializează maximul cu prima valoare și contorul cu 1; la o valoare mai mare se actualizează maximul și se resetează contorul, iar la egalitate contorul crește.

Lecția 3 – Arhitectura unui program Python. Variabile, tipuri de date, expresii și operatori

Fișa de identificare a lecției

Element	Conținut
Clasa	a IX-a
Durata recomandată	Două sau trei ore, în funcție de experiența elevilor
Tipul lecției	Dobândire și aplicare de cunoștințe
Limbaj și mediu	Python 3; IDLE sau alt mediu compatibil
Prerechizite	Date, algoritmi, programe, intrare, ieșire și expresii matematice
Resurse	Calculator, interpretor Python, videoproiector și fișă de lucru
Conținuturi	Structura programului Python, identificatori, variabile, tipuri, conversii, expresii, operatori și funcții

Competențe urmărite

- Utilizarea unui mediu Python pentru scrierea și rularea programelor simple.
- Declararea prin atribuire și folosirea variabilelor în expresii.
- Citirea, conversia și afișarea datelor.
- Evaluarea expresiilor folosind operatori aritmetici, relaționali și logici.
- Testarea unor programe scurte și explicarea rezultatelor.

Obiective de învățare

La finalul lecției, elevul va fi capabil:

- să descrie modul de executare a unui program Python;
- să respecte regulile de formare a identificatorilor și de indentare;
- să distingă tipurile int, float, complex, bool, str și list;
- să utilizeze input, print, conversii și șiruri formatate;
- să aplice operatorii și precedența lor în expresii;
- să folosească funcții numerice uzuale și modulul math.

Scenariul didactic

Etapă	Timp	Activitatea profesorului	Activitatea elevilor	Dovada învățării
Conectare	8 min	Rulează două fragmente de cod asemănătoare, cu rezultate diferite.	Anticipează tipurile datelor și rezultatele.	Predicție justificată
Explorare	25 min	Utilizează consola, fișierul și funcțiile de intrare-ieșire.	Scrie, salvează și rulează un program.	Fișier Python funcțional
Explicare	30 min	Clarifică variabilele, tipurile și conversiile.	Testează type și transformări explicite.	Tabel de observații

Aplicare	65 min	Propune sarcini pentru operatori și cifrele unui număr.	Rezolvă și testează programe scurte.	Cod și tabel de teste
Verificare	12 min	Aplică evaluarea și oferă feedback.	Răspunde și își evaluează progresul.	Răspunsuri scurte

Întrebarea esențială

Întrebare: Cum influențează tipul datelor și operatorii rezultatul unei expresii Python?

Situația de pornire

```
a = input('a = ')
b = input('b = ')
print(a + b)
```

Pentru valorile 2 și 3, programul afișează 23, nu 5. Elevii explică rezultatul și propun modificarea necesară.

De la consolă la program

Python este un limbaj de nivel înalt. Implementarea uzuală execută programul prin intermediul unui interpretor: codul sursă este analizat și transformat într-o formă internă, apoi instrucțiunile sunt executate. Pentru început este suficient să reținem că programul nu este livrat elevului ca un executabil obținut printr-un pas manual de compilare, ca în fluxul introductiv folosit adesea pentru C++.

Mod de lucru	Utilizare	Caracteristică
Consola interactivă	Teste rapide și expresii scurte.	Instrucțiunile se introduc după marcajul >>>.
Fișier cu extensia py	Programe care se salvează și se rulează repetat.	Conține instrucțiuni scrise în ordinea execuției.
Notebook online	Experimentare în celule, fără instalare locală.	Necesită de regulă browser și acces la serviciu.

Flux de lucru în IDLE

1. Deschide IDLE și selectează File, apoi New File.
2. Scrie programul în fereastra editorului.
3. Salvează fișierul cu extensia .py.
4. Rulează modulul cu Run Module sau tasta F5.
5. Citește mesajele și rezultatele din Shell; corectează și rulează din nou.

Primul program

```
print('Salut, lume!')
```

- Python diferențiază literele mari de cele mici: print este diferit de Print.
- De regulă, fiecare instrucțiune se scrie pe rând separat; punctul și virgula este permis în anumite situații, dar nu este necesar.
- Indentarea delimitează blocurile și face parte din sintaxă.

Vocabularul limbajului Python

Element	Rol	Exemple
Caractere	Formează codul sursă.	litere, cifre, spații și semne speciale
Identificatori	Denumesc variabile, funcții, clase sau module.	nota, suma_note, total2
Cuvinte cheie	Au sens rezervat și nu pot fi folosite ca identificatori.	if, else, for, while, True, False, and, or, not
Delimitatori și separatori	Grupează și separă componente.	( ) [ ] { } , : ; și indentarea
Comentarii	Explică intenția codului; nu sunt executate.	# calculează media

Reguli pentru identificatori

- Pot conține litere, cifre și caracterul de subliniere.
- Nu pot începe cu o cifră.
- Nu pot fi cuvinte cheie.
- Sunt sensibili la majuscule: varsta și Varsta sunt nume diferite.
- Pentru variabile se recomandă nume clare, scrise uzual cu litere mici și underscore.

Exemple

Nume	Valid	Explicație
nota1	Da	Începe cu literă și poate conține cifre.
_contor	Da	Poate începe cu underscore.
media_note	Da	Nume clar, format din cuvinte separate prin underscore.
1nota	Nu	Începe cu cifră.
nota finală	Nu	Conține spațiu.
For	Nu	Este cuvânt cheie.

Variabile și atribuiri

În Python, o variabilă apare atunci când unui identificator i se atribuie o valoare. Nu se scrie separat tipul variabilei; obiectul referit are un tip, iar numele poate fi asociat ulterior cu o valoare de alt tip. Pentru programele introductive este util să păstrăm totuși aceeași semnificație și același tip pentru o variabilă.

```
varsta = 15
inaltime = 1.72
nume = 'Maria'
este_elev = True
```

Citirea datelor

Funcția input afișează opțional un mesaj și returnează întotdeauna un șir de caractere. Pentru calcule numerice, valoarea trebuie convertită explicit.

```
nume = input('Cum te numești? ')
varsta = int(input('Ce vârstă ai? '))
inaltime = float(input('Înălțimea în metri: '))
```

Afișarea datelor

Funcția print poate afișa texte, valori și expresii. F-string-urile integrează expresii între acolade și permit formatarea clară a rezultatului.

```
print(f'Bună, {nume}! Ai {varsta} ani.')
print(f'Înălțimea este {inaltime:.2f} m.')
```

Situație	Cod	Rezultat pentru intrarea 12
Fără conversie	x = input(); print(x + x)	1212
Conversie la întreg	x = int(input()); print(x + x)	24
Conversie la real	x = float(input()); print(x / 5)	2.4

Tipuri de date fundamentale

Tip	Conținut	Exemplu	Observație
Int	Numere întregi	-7, 0, 2026	Are precizie arbitrară, limitată practic de memorie.
Float	Numere reale în virgulă mobilă	3.14, -0.5	Unele valori zecimale nu se reprezintă exact.
complex	Numere complexe	2 + 3j	Partea imaginară folosește litera j.
Bool	Valori logice	True, False	Este folosit în condiții și comparații.
Str	Șiruri de caractere	'Ana', "9"	Se scriu între apostrofuri sau ghilimele.

Determinarea şi conversia tipului

```
x = 12
print(type(x))      # <class 'int'>
y = float(x)
text = str(x)
z = int('25')
```

Conversia reușește numai dacă valoarea sursă are o formă compatibilă. De exemplu, int("25") este valid, dar int("2.5") produce o eroare; pentru acest text este necesară mai întâi conversia la float, dacă intenția este justificată.

Atenție la numerele reale

```
print(0.1 + 0.2)      # poate afișa 0.30000000000000004
print(round(0.1 + 0.2, 2))
```

Această diferență apare din reprezentarea binară aproximativă a numerelor în virgulă mobilă, nu dintr-o greșeală a operației matematice.

Șiruri de caractere și liste

Șiruri de caractere

Tipul str reprezintă o succesiune ordonată de caractere. Un șir poate fi delimitat cu apostrofuri sau ghilimele. Trei delimitatoare consecutive permit texte pe mai multe rânduri.

```
mesaj = 'Salut'
propozitie = "Învăț Python."
text_lung = '''Prima linie
A doua linie'''
```

Liste

Lista este o colecție ordonată și modificabilă. Elementele sunt separate prin virgulă și delimitate de paranteze pătrate. O listă poate conține valori de tipuri diferite, deși în multe probleme școlare este mai clar să folosim elemente cu aceeași semnificație.

```
note = [8, 9, 10]
print(note[0])      # 8
print(note[-1])     # 10
print(len(note))    # 3
note.append(7)
note.insert(1, 6)
note.extend([9, 8])
```

Operație	Efect
lista[index]	Accesează elementul; primul index este 0, iar -1 indică ultimul element.
len(lista)	Returnează numărul elementelor.
append(x)	Adaugă un element la final.
insert(i, x)	Inserează x pe poziția i și deplasează elementele următoare.
extend(colectie)	Adaugă la final toate elementele colecției primite.

Expresii și operatori aritmetici

O expresie combină operanzi și operatori și produce o valoare. Evaluarea respectă precedența operatorilor, asociativitatea și parantezele explicite.

Operator	Semnificație	Exemplu	Rezultat
+ - *	Adunare, scădere, înmulțire	12 - 3 * 2	6
/	Împărțire reală	7 / 2	3.5
//	Împărțire cu rotunjire în jos	7 // 2; -7 // 2	3; -4
%	Rest compatibil cu împărțirea //	17 % 5	2
**	Ridicare la putere	2 ** 3 ** 2	512
+x -x	Operatori unari	-(-5)	5
~x	Complement pe biți pentru întregi	~5	-6

Pentru orice întregi a și b nenul, Python respectă relația a = (a // b) × b + (a % b). Operatorul // rotunjește câtul în jos, aspect important pentru valorile negative. Pentru întregi, ~x este egal cu -(x + 1).

Prelucrarea cifrelor unui număr natural

Scop	Formulă	Exemplu pentru n = 4732
Ultima cifră	n % 10	2
Eliminarea ultimei cifre	n // 10	473
Cifra de pe poziția k de la dreapta, numerotată de la 0	(n // 10 ** k) % 10	pentru k = 2 rezultă 7

Operatori relaționali și logici

Operatori relaționali

Operatorii <, <=, >, >=, == și != compară valori și produc rezultatul True sau False. Operatorul == testează egalitatea, în timp ce = realizează atribuirea.

```
varsta = 15
print(varsta >= 14)     # True
print(varsta == 18)     # False
print(1 < varsta < 18)  # True
```

Python permite comparații înlănțuite. Expresia 1 < varsta < 18 este echivalentă logic cu 1 < varsta and varsta < 18.

Operatori logici

p	q	p and q	p or q
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

p	not p
False	True
True	False

Operatorul not inversează valoarea logică. Expresia p and q este adevărată numai când ambele valori sunt adevărate, iar p or q este adevărată când cel puțin una este adevărată.

Evaluare scurtcircuitată

Python poate opri evaluarea unei expresii logice imediat ce rezultatul este cunoscut. De exemplu, n != 0 and 10 / n > 2 nu calculează împărțirea atunci când n este 0.

Atribuiri și precedența operatorilor

Atribuirea

Operatorul = evaluează expresia din dreapta și asociază rezultatul numelui din stânga. Atribuirile compuse actualizează valoarea folosind un operator.

Formă extinsă	Formă compusă
x = x + 3	x += 3
x = x – 2	x -= 2
x = x * 5	x *= 5
x = x / 4	x /= 4
x = x // 10	x //= 10
x = x % 10	x %= 10
x = x ** 2	x **= 2

Ordine practică de evaluare

Prioritate de sus în jos	Operatori sau construcții	Observație
1	( )	Parantezele controlează explicit ordinea.
2	**	Ridicarea la putere este asociativă la dreapta.
3	+x, -x, ~x	Operatori unari.
4	*, /, //, %	Multiplicativi.
5	+, -	Aditivi.
6	<, <=, >, >=, ==, !=	Comparații; pot fi înlănțuite.
7	Not	Negație logică.
8	And	Conjuncție logică.
9	Or	Disjuncție logică.

Funcții numerice utile

Funcție	Rezultat	Exemplu
abs(x)	Valoarea absolută sau modulul, după tip.	abs(-7) → 7
round(x, n)	Rotunjește la n zecimale; n este opțional.	round(3.14159, 2) → 3.14
math.floor(x)	Cel mai mare întreg mai mic sau egal cu x.	math.floor(-2.3) → -3
math.ceil(x)	Cel mai mic întreg mai mare sau egal cu x.	math.ceil(-2.3) → -2
math.sqrt(x)	Rădăcina pătrată reală pentru x nenegativ.	math.sqrt(25) → 5.0

Funcțiile floor, ceil și sqrt se apelează după importarea modulului math.

```
import math

x = float(input('x = '))
if x >= 0:
    print(f'Rădăcina este {math.sqrt(x):.2f}')
else:
    print('Nu există rădăcină pătrată reală.')
```

Despre round

Funcția round urmează regulile Python și efectele reprezentării în virgulă mobilă; la egalitate exactă folosește rotunjirea către valoarea pară. Nu trebuie confundată cu simpla formatare pentru afișare, precum f"{x:.2f}".

Testare

x	Ramură	Rezultat
25	x >= 0	5.00
2	x >= 0	1.41
0	x >= 0	0.00
-4	x < 0	mesaj explicativ

Exemplu rezolvat: analiza unei note

Cerință: Se citesc numele unui elev și o notă reală din intervalul [1, 10]. Se afișează un mesaj formatat care conține numele, nota cu două zecimale și situația PROMOVAT sau NEPROMOVAT.

Analiza problemei

Categorie	Conținut
Intrare	nume de tip str; nota de tip float
Ieșire	mesaj formatat și situația elevului
Restricție	1 <= nota <= 10
Condiție de promovare	nota >= 5

Implementarea în Python

```
nume = input('Numele elevului: ')
nota = float(input('Nota: '))

if nota < 1 or nota > 10:
   print('Notă invalidă')
else:
   situatie = 'PROMOVAT' if nota >= 5 else 'NEPROMOVAT'
   print(f'{nume}: nota {nota:.2f} - {situatie}')
```

Tabel de teste

Nume	Notă	Tip	Rezultat așteptat
Ana	8.5	general	Ana: nota 8.50 - PROMOVAT
Mihai	5	limită	Mihai: nota 5.00 - PROMOVAT
Ioana	4.99	limită	Ioana: nota 4.99 - NEPROMOVAT
Radu	11	invalid	Notă invalidă

Activități și exerciții pentru elevi

Activitate ghidată

Sarcină: Rulează expresiile 7 / 2, 7 // 2, -7 // 2 și 7 % 2. Notează tipul și valoarea fiecărui rezultat, apoi explică diferențele.

Criterii de reușită: rezultatele sunt prezise înainte de rulare; tipurile sunt verificate cu type; explicația folosește sensul operatorilor.

Exerciții

1. Clasifică identificatorii ca valizi sau invalizi: nume_elev, 2note, notaFinala, for, _suma.
2. Corectează programul care afișează 23 în loc de 5 pentru intrările 2 și 3.
3. Stabilește valorile expresiilor 2 + 3 * 4, (2 + 3) * 4 și 2 ** 3 ** 2.
4. Pentru n = 5837, determină ultima cifră, numărul fără ultima cifră și cifra de pe poziția 2 de la dreapta.
5. Scrie o expresie care verifică dacă x aparține intervalului [10, 99].
6. Creează o listă cu trei orașe, adaugă unul la final și afișează primul și ultimul element.
7. Scrie un program care citește raza unui cerc și afișează aria cu două zecimale. Folosește math.pi.

Diferențiere

- Sprijin: elevul primește un tabel cu tipurile și conversiile posibile.
- Nivel de bază: exerciții cu o singură operație și intrări valide.
- Aprofundare: cazuri cu valori negative, precizie float și evaluare scurtcircuitată.

Evaluare

1. Ce tip returnează input și de ce este necesară conversia?
2. Care este diferența dintre / și // pentru valorile pozitive?
3. Ce rezultat produce expresia 3 < 5 and not False?

Repere pentru profesor

Sugestii metodice

- Solicitați predicția rezultatului înainte de rulare și verificarea ulterioară în consolă.
- Separați clar atribuirea = de comparația ==.
- Evidențiați faptul că input returnează str chiar dacă utilizatorul tastează cifre.
- Folosiți valori negative pentru a clarifica faptul că // rotunjește în jos.
- Corectați ideea că listele conțin numai șiruri; ele pot conține obiecte de tipuri diferite.
- Prezentați True și False cu majusculă, conform sintaxei Python.
- Pentru expresii complexe, preferați paranteze explicite în locul memorării mecanice a unui tabel foarte extins.

Soluții orientative

Exercițiu	Răspuns orientativ
1	Valizi: nume_elev, notaFinala, _suma. Invalizi: 2note și for.
2	a = int(input()); b = int(input()); print(a + b).
3	14; 20; 512.
4	Ultima cifră 7; fără ultima cifră 583; cifra de pe poziția 2 este 8.
5	10 <= x <= 99.
6	orase = ['Iași', 'Cluj', 'Brașov']; orase.append('Sibiu'); print(orase[0], orase[-1]).
7	import math; r = float(input()); aria = math.pi * r ** 2; print(f"{aria:.2f}").

Lecția 4 – Structuri de control în Python. Structura secvențială, alternativă și repetitivă

Fișa de identificare a lecției

Element	Conținut
Clasa	a IX-a
Durata recomandată	Trei ore; conținutul poate fi împărțit în decizie și repetare
Tipul lecției	Dobândire, aplicare și consolidare
Limbaj și mediu	Python 3; IDLE sau alt mediu compatibil
Prerechizite	Variabile, tipuri de date, expresii, operatori, input și print
Resurse	Calculator, interpretor Python, videoproiector și fișă de lucru
Conținuturi	Secvență, if, elif, else, while, for, range, break și continue

Competențe urmărite

- Alegerea structurii de control adecvate unei cerințe.
- Construirea și testarea programelor Python cu decizii și repetări.
- Urmărirea valorilor variabilelor pe ramuri și iterații.
- Identificarea condițiilor de oprire și prevenirea repetărilor infinite.

Obiective de învățare

La finalul lecției, elevul va fi capabil:

- să distingă structurile secvențială, alternativă și repetitivă;
- să utilizeze corect indentarea blocurilor;
- să scrie decizii simple, complete și multiple;
- să folosească while pentru repetări controlate de condiție;
- să folosească for și range pentru parcurgeri;
- să explice efectul instrucțiunilor break și continue;
- să testeze cazuri generale și limită.

Scenariul didactic

Etapă	Timp	Activitatea profesorului	Activitatea elevilor	Dovada învățării
Conectare	10 min	Prezintă trei fragmente de cod cu flux diferit.	Identifică secvența, decizia și repetarea.	Clasificare argumentată
Explorare	30 min	Modelează execuția pas cu pas.	Urmărește variabilele și traseele.	Tabele de urmărire
Explicare	35 min	Clarifică sintaxa și condițiile de oprire.	Compară formele și formulează reguli.	Răspunsuri explicate
Aplicare	65 min	Propune probleme progresive.	Implementează și testează soluțiile.	Programe și teste
Verificare	10 min	Aplică evaluarea.	Răspunde și se autoevaluează.	Răspunsuri scurte

Întrebarea esențială

Întrebare: Cum alegem structura care descrie corect ordinea, decizia sau repetarea cerută de problemă?

Situația de pornire

Un program trebuie să citească temperaturi până la introducerea valorii 999 și să numere valorile negative. Elevii decid ce instrucțiuni se execută o singură dată, ce decizie se ia pentru fiecare temperatură și ce condiție oprește citirea.

Structurile fundamentale

Python este un limbaj de nivel înalt. Implementarea uzuală execută programul prin intermediul unui interpretor: codul sursă este analizat și transformat într-o formă internă, apoi instrucțiunile sunt executate. Pentru început este suficient să reținem că programul nu este livrat elevului ca un executabil obținut printr-un pas manual de compilare, ca în fluxul introductiv folosit adesea pentru C++.

Structură	Comportament	Semnal în enunț
Secvențială	Instrucțiunile se execută în ordinea scrierii, o singură dată.	calculează, apoi afișează
Alternativă	Se execută o ramură aleasă printr-o condiție.	dacă, în funcție de, altfel
Repetitivă	Un bloc se execută de mai multe ori.	pentru fiecare, cât timp, până când

Structura secvențială

Cerință: Se citește latura unui cub și se calculează aria totală și volumul.
Relații: $A = 6 \times l^2$ și $V = l^3$, pentru $l > 0$.

```
latura = float(input('Latura cubului: '))
aria = 6 * latura ** 2
volumul = latura ** 3
print(f'Aria totală: {aria:.2f}')
print(f'Volumul: {volumul:.2f}')
```

Testare

Latura	Aria	Volumul	Observație
2	24	8	caz general
1	6	1	caz simplu
0	0	0	valoare-limită; trebuie decis dacă este permisă
-3	54	-27	invalidă pentru lungime; programul necesită validare

Structura secvențială nu verifică singură validitatea datelor. Dacă latura trebuie să fie pozitivă, introducem o structură alternativă.

Structura alternativă

Instrucțiunea if evaluează o expresie. Dacă rezultatul este True, execută blocul indentat. Blocul else este opțional și descrie situația în care expresia este False.

```
if condiție:
    instrucțiuni_pentru_adevărat
else:
    instrucțiuni_pentru_fals
```

Exemplu: compararea vârstelor

Cerință: Se citesc vârstele a doi copii despre care știm că sunt diferite. Se afișează care copil este mai mare.

```
varsta_ana = int(input('Vârsta Anei: '))
varsta_mihai = int(input('Vârsta lui Mihai: '))

if varsta_ana > varsta_mihai:
   print('Ana este mai mare.')
else:
   print('Mihai este mai mare.')
```

Indentarea

- Instrucțiunile aceluiași bloc trebuie să aibă aceeași indentare.
- Se recomandă patru spații pentru fiecare nivel.
- Amestecarea necontrolată a tabulatorilor și spațiilor poate produce erori sau structură greu de observat.
- Indentarea arată exact ce instrucțiuni depind de condiție.

Decizii cu mai multe ramuri

Forma if, elif, else verifică pe rând condițiile. Se execută numai blocul primei condiții adevărate; după aceea, structura se încheie. Blocul else tratează toate situațiile rămase și poate lipsi.

```
if condiție_1:
   ramura_1
elif condiție_2:
   ramura_2
else:
   ramura_3
```

Exemplu calificativ

Cerință: Se citește o notă între 1 și 10 și se afișează un calificativ orientativ.

```
nota = float(input('Nota: '))
if nota < 1 or nota > 10:
   print('Notă invalidă')
elif nota < 5:
   print('Insuficient')
elif nota < 7:
   print('Suficient')
elif nota < 9:
   print('Bine')
else:
   print('Foarte bine')
```

Ordinea condițiilor

Condițiile sunt evaluate de sus în jos. După ce nota a trecut de verificările anterioare, fiecare ramură poate folosi informația acumulată. De exemplu, în ramura nota < 7 știm deja că nota este cel puțin 5.

Notă	Prima condiție adevărată	Rezultat
0	nota < 1	Notă invalidă
4.50	nota < 5	Insuficient
6	nota < 7	Suficient
8.25	nota < 9	Bine
10	Else	Foarte bine

Repetarea cu while

Instrucțiunea while testează condiția înaintea fiecărei iterații. Cât timp condiția este True, blocul indentat se execută. Corpul poate să nu se execute deloc dacă expresia este falsă de la început.

```
while condiție:
    instrucțiuni
    actualizarea_dată_care_poate_schimba_condiția
```

Exemplu numere pare și impare

Cerință: Se citesc numere întregi până la valoarea 0. Se numără valorile pare și impare; valoarea 0 nu se ia în calcul.

```
pare = 0
impare = 0
numar = int(input('Număr: '))
while numar != 0:
    if numar % 2 == 0:
        pare += 1
    else:
        impare += 1
    numar = int(input('Număr: '))
print('Pare:', pare)
print('Impare:', impare)
```

Rolurile variabilelor

Variabilă	Rol	Inițializare
numar	valoarea curentă și controlul opririi	prima citire, înainte de while
pare	contor pentru valori pare	0
impare	contor pentru valori impare	0

Urmărirea și oprirea repetării

Intrare curentă	Acțiune	pare	impare
8	pare += 1	1	0
-3	impare += 1	1	1
4	pare += 1	2	1
0	oprire; nu se numără	2	1

Evitarea repetărilor infinite

- Condiția trebuie să poată deveni False.
- Cel puțin o variabilă folosită în condiție trebuie actualizată în corp sau printr-un eveniment controlat.
- Pentru citirea cu santinelă, următoarea valoare se citește în interiorul repetării.
- Testați și situația în care prima valoare este chiar santinela.

Exemplu numărul de cifre

Cerință: Se citește un număr întreg și se determină câte cifre are. Semnul minus nu este cifră, iar numărul 0 are o cifră.

```
n = int(input('n = '))
n = abs(n)
if n == 0:
   cifre = 1
else:
   cifre = 0
   while n > 0:
      cifre += 1
      n //= 10
print(cifre)
```

Aplicarea funcției abs permite aceeași prelucrare pentru numere pozitive și negative. Tratarea separată a lui 0 evită rezultatul greșit 0 cifre.

Repetarea cu for

Instrucțiunea for parcurge pe rând elementele unui obiect iterabil, precum o listă, un șir de caractere sau o secvență produsă de range. La fiecare iterație, variabila de control primește următorul element.

```
for element in colecție:
   instrucțiuni
```

Exemplu: clasificarea valorilor

Cerință: Pentru o listă de numere, se determină câte valori sunt pozitive, negative și nule.

```
valori = [4, -2, 0, 7, -1, 0]
pozitive = negative = nule = 0
for x in valori:
   if x > 0:
      pozitive += 1
   elif x < 0:
      negative += 1
   else:
      nule += 1
print(pozitive, negative, nule)
```

Urmărire

x	Ramură	pozitive	negative	nule
4	x > 0	1	0	0
-2	x < 0	1	1	0
0	else	1	1	1
7	x > 0	2	1	1
-1	x < 0	2	2	1
0	else	2	2	2

Secvențe produse de range

Funcția range produce o secvență de întregi. Limita finală nu este inclusă. Pasul trebuie să fie nenul și poate fi negativ.

Formă	Secvență
range(5)	0, 1, 2, 3, 4
range(2, 6)	2, 3, 4, 5
range(2, 10, 3)	2, 5, 8
range(5, 0, -1)	5, 4, 3, 2, 1

Exemplu suma numerelor de la 1 la n

```
n = int(input('n = '))
suma = 0
for i in range(1, n + 1):
    suma += i
print(suma)
```

Pentru n pozitiv, range(1, n + 1) include valorile de la 1 la n. Programul folosește O(n) pași; aceeași sumă se poate calcula și în O(1) timp prin formula n × (n + 1) / 2, dacă n este natural.

Parcurgerea indicilor

```
culori = ['roșu', 'verde', 'albastru']
for i in range(len(culori)):
    print(i, culori[i])
```

Când este necesară numai valoarea, forma for culoare in culori este mai clară. Indicii sunt utili când poziția elementului face parte din cerință.

Minimul și numărul aparițiilor

Cerință: Se citesc n numere întregi, cu n cel puțin 1. Se determină valoarea minimă și numărul ei de apariții.

Ideea algoritmului

1. Citește prima valoare și folosește-o pentru inițializarea minimului.
2. Inițializează numărul aparițiilor cu 1.
3. Pentru fiecare valoare rămasă, compară cu minimul curent.

4. Dacă valoarea este mai mică, actualizează minimul și resetează contorul la 1.
5. Dacă valoarea este egală cu minimul, mărește contorul.

Implementarea

```
n = int(input('n = '))
prima = int(input('Valoarea 1: '))
minim = prima
aparitii = 1
for i in range(2, n + 1):
    x = int(input(f'Valoarea {i}: '))
    if x < minim:
        minim = x
        aparitii = 1
    elif x == minim:
        aparitii += 1
print('Minim:', minim)
print('Apariții:', aparitii)
```

De ce nu inițializăm cu 0

Dacă toate valorile sunt pozitive, 0 nu aparține șirului și ar rămâne în mod greșit minim. Inițializarea cu prima valoare folosește o valoare reală din date și funcționează pentru orice întreg.

Date	Minim	Apariții
5; 3, 1, 4, 1, 2	1	2
4; -2, -5, -5, 0	-5	2
1; 7	7	1

Instrucțiunea break

Instrucțiunea break încheie imediat cea mai apropiată repetare care o conține. Execuția continuă cu prima instrucțiune de după repetare. Se folosește când o condiție internă face inutilă continuarea.

Cerință: Se citesc numere și se calculează suma lor până la valoarea 0.

```
suma = 0
while True:
    x = int(input('Număr: '))
    if x == 0:
        break
    suma += x
print(suma)
```

Comparație cu o condiție în while

Variantă	Avantaj	Atenție
Citire înainte și la finalul corpului	Condiția while exprimă direct oprirea.	Instrucțiunea de citire apare de două ori.
while True și break	Citirea apare într-un singur loc.	Condiția de oprire trebuie identificată clar în corp.

Ambele variante pot fi corecte. Alegerea trebuie să favorizeze claritatea și să facă imposibilă omiterea actualizării care duce la oprire.

Instrucțiunea continue

Instrucțiunea continue oprește numai iterația curentă și trece la următoarea. Repetarea nu se încheie. Folosirea sa trebuie să simplifice logica, nu să ascundă actualizări esențiale.

Cerință: Se citește n și se afișează numerele impare de la 1 la n.

```
n = int(input('n = '))
for i in range(1, n + 1):
    if i % 2 == 0:
        continue
    print(i)
```

Variantă fără continue

```
for i in range(1, n + 1, 2):
    print(i)
```

A doua variantă exprimă direct faptul că pasul este 2 și este mai scurtă pentru această problemă. Prima variantă este utilă pentru a înțelege efectul lui continue și se generalizează când filtrul este mai complex.

Atenție în while

Dacă continue apare înaintea actualizării variabilei din condiție, repetarea poate deveni infinită. Actualizarea trebuie realizată înainte de continue sau structura trebuie reorganizată.

Activități și exerciții pentru elevi

Activitate ghidată

Sarcină: Pentru datele de intrare 6, -1, 8 și 0, completează tabelul de urmărire al programului care numără valorile pare și impare.

Criterii de reușită: valoarea 0 nu este numărată; fiecare valoare urmează o singură ramură; citirea următoarei valori precedă retestarea condiției.

Exerciții

1. Scrie un program secvențial care citește baza și înălțimea unui triunghi și afișează aria.
2. Modifică programul vârstelor astfel încât să trateze și egalitatea.
3. Scrie un program care clasifică un număr ca pozitiv, negativ sau zero.
4. Determină câte cifre pare are un număr întreg; tratează separat numărul 0.
5. Pentru o listă de note, numără valorile cel puțin egale cu 5.
6. Afișează multiplii lui 3 din intervalul [1, n] folosind range.
7. Determină maximul și numărul aparițiilor sale într-un șir de n valori.
8. Explică diferența dintre break și continue prin câte un exemplu.
9. Provocare: verifică dacă un număr natural este prim și oprește căutarea primului divizor cu break.

Diferențiere

- Sprijin: se oferă scheletul programului și un tabel de urmărire parțial completat.
- Nivel de bază: o singură structură de control și date valide.
- Aprofundare: structuri imbricate, validare și comparație între două soluții.

Evaluare

1. Când este preferabil while în locul lui for?
2. Ce valori produce range(2, 9, 3)?
3. Care este diferența dintre break și continue?

Repere pentru profesor

Sugestii metodice

- Folosiți tabele de urmărire înaintea rulării pentru a face vizibil fluxul programului.
- Insistați asupra ordinii condițiilor din lanțurile if, elif, else.
- Testați repetările cu situația în care corpul nu se execută deloc.
- Tratați explicit 0 și numerele negative la prelucrarea cifrelor.
- Inițializați minimul sau maximul cu prima valoare, nu cu o presupunere despre domeniu.
- Prezentați break și continue numai împreună cu justificarea clarității soluției.

Soluții orientative

Exercițiu	Răspuns orientativ
1	b = float(input()); h = float(input()); print(b * h / 2).
2	Se adaugă ramura elif varsta_ana < varsta_mihai, iar else afișează aceeași vârstă.
3	if x > 0: pozitiv; elif x < 0: negativ; else: zero.
4	Se aplică abs; pentru 0 rezultatul este o cifră; altfel se elimină cifrele cu // 10 și se testează paritatea.
5	Pentru fiecare notă, dacă nota >= 5 se mărește contorul.
6	for x in range(3, n + 1, 3): print(x).
7	Se adaptează algoritmul minimului, înlocuind comparația < cu >.
8	break încheie repetarea; continue sare doar peste restul iterației curente.
9	Se testează divizorii de la 2 până la rădăcina numărului; la primul divizor se folosește break.