1. Limbaje de programare

1.1. De la algoritm la program

Un **algoritm** este o succesiune finită și clară de pași prin care se rezolvă o problemă. Pentru ca algoritmul să poată fi executat de calculator, acesta trebuie transpus într-un **program**.

> **Definiție.** Un limbaj de programare este un ansamblu de simboluri, cuvinte și reguli folosite pentru scrierea instrucțiunilor pe care calculatorul le poate executa.

Un limbaj de programare are:

- un **vocabular** — simbolurile și cuvintele care pot fi folosite;
- o **sintaxă** — regulile de scriere a instrucțiunilor;
- o **semantică** — semnificația instrucțiunilor.

1.2. Vocabularul

Într-un program întâlnim:

Felul cuvintelor	Exemple
cuvinte-cheie, rezervate de limbaj	`if elif else while for def return`
identificatori — numele alese de programator	`n suma media numarElevi`
operatori	`+ - * / % ==`
constante literale, scrise direct în program	`25 3.14 "text" True`

> Cuvintele-cheie nu pot fi folosite ca nume de variabile. De exemplu, `for` nu poate fi numele unei variabile.

1.3. Sintaxa

Sintaxa stabilește cum trebuie scrise instrucțiunile. În Python sunt necesare semnul `:` la sfârșitul condiției și indentarea instrucțiunilor din bloc:

```
if n % 2 == 0:
    print("par")
```

O eroare de sintaxă împiedică, de regulă, începerea programului. Cele mai frecvente sunt lipsa caracterului `:` și indentarea greșită.

1.4. Erori de logică și erori de execuție

Un program poate fi corect din punct de vedere sintactic și totuși să producă un rezultat greșit:

```
Ai scris:  suma = a - b
Ai dorit:  suma = a + b
```

Aceasta este o **eroare de logică**. Programul poate rula, dar nu rezolvă corect problema. Erorile de logică se descoperă prin testare.

O **eroare de execuție** apare după ce programul a pornit, atunci când întâlnește o operație imposibilă:

```
print("Programul a început.")
x = 10 / 0
```

Mesajul este afișat, apoi programul se oprește, deoarece împărțirea la zero nu este permisă.

1.5. Cum ajunge programul să fie executat

Calculatorul nu execută direct instrucțiunile scrise de programator. Codul Python este transformat, de regulă, într-o formă intermediară numită *bytecode*, executată de interpretorul Python. Programul pornește fără un pas separat de compilare, iar erorile sunt semnalate în momentul în care execuția ajunge la instrucțiunea greșită.

2. Analiza unei probleme

Înainte de a scrie programul trebuie să stabilim ce date cunoaștem și ce rezultat trebuie obținut.

Exemplu. Se citesc lungimea și lățimea unui dreptunghi. Să se calculeze aria și perimetrul.

- **Date de intrare:** lungimea L și lățimea l;
- **Date de ieșire:** aria A și perimetrul P;
- **Relații de calcul:** $A = L \cdot l$, $P = 2 \cdot (L + l)$.

Pseudocodul algoritmului este:

```
citește L, l
A ← L × l
P ← 2 × (L + l)
scrie A, P
```

Acesta este un **algoritm liniar**: instrucțiunile se execută o singură dată, în ordinea în care au fost scrise.

Pașii rezolvării unei probleme

1. identificăm datele de intrare;
2. stabilim datele de ieșire;
3. alegem variabilele necesare;
4. scriem relațiile de calcul sau condițiile;
5. construim algoritmul;
6. implementăm algoritmul;
7. testăm programul.

3. Variabile și constante

3.1. Ce este o variabilă?

> **Definiție.** O variabilă este un nume asociat unei valori care poate fi folosită și modificată în timpul executării programului.

În problema dreptunghiului putem folosi variabilele `lungime`, `latime`, `arie` și `perimetru`.

```
lungime = 8
latime = 5
arie = lungime * latime
```

După executarea instrucțiunilor, variabila `arie` are valoarea 40.

3.2. Identificatori

Numele unei variabile se numește **identificator**. Un identificator poate conține litere, cifre și caracterul `_`, dar:

- nu poate începe cu o cifră;
- nu poate conține spații;
- nu poate conține semne precum -, +, ? sau %;
- nu poate fi un cuvânt-cheie al limbajului;
- literele mari și mici sunt considerate diferite.

Identificator	Corect?	Explicație
`varsta`	Da	conține numai litere
`nota1`	Da	cifra nu este la început
`numar_elevi`	Da	poate conține _
`1nota`	Nu	începe cu o cifră
`numar elevi`	Nu	conține spațiu
`nota-finala`	Nu	conține semnul -
`for`	Nu	este cuvânt-cheie în Python

Numele trebuie să arate ce reprezintă valoarea:

```
medie = 9.50       # nume sugestiv
x = 9.50           # corect, dar mai puțin clar
```

3.3. Atribuirea

În Python nu se declară tipul variabilei: numele apare în program atunci când îi atribuim o valoare, iar tipul se stabilește din valoarea primită.

```
varsta = 15
medie = 9.75
nume = "Ana"

# numele apare când primește o valoare
```

Operatorul `=` atribuie variabilei din stânga valoarea expresiei din dreapta:

```
x = 5
x = x + 1     # acum x este 6
```

După a doua instrucțiune, valoarea lui `x` este 6. Mai întâi se calculează `x + 1`, apoi rezultatul este memorat în `x`.

3.4. Atribuire și comparație

Operator	Rol	Exemplu
`=`	atribuie o valoare	`x = 5`
`==`	compară două valori	`x == 5`

```
x = 5
print(x == 5)     # True
```

3.5. Constante

O **constantă** este o valoare care nu se modifică în timpul executării programului. Valorile scrise direct în program sunt constante literale: `25`, `3.14`, `"Ana"`, `True`.

Python nu are un cuvânt prin care să se interzică modificarea unei variabile. Constantele se scriu, prin convenție, cu litere mari. Convenția este respectată de programatori, dar nu este verificată de limbaj:

```
PI = 3.14159   # prin convenție, cu litere mari
```

---

4. Tipuri de date

Tipul unei date arată ce valori pot fi memorate și ce operații pot fi efectuate asupra lor.

Tipul informației	În Python	Exemple
Număr întreg	`int`	−7, 0, 125
Număr real	`float`	3.14, −0.5
Valoare logică	`bool`	True, False
Șir de caractere	`str`	"A", "Ana", "Python"

4.1. Numere întregi

```
elevi = 28
temperatura = -5
```

În Python, numerele întregi nu au o limită superioară fixată de limbaj; dimensiunea lor este mărginită numai de memoria disponibilă.

4.2. Numere reale

În programe, separatorul zecimal este punctul: `9.75`.

```
medie = 9.75
```

4.3. Valori logice

Tipul `bool` are două valori. Ele apar, de obicei, în urma comparațiilor:

```
prezent = True

varsta = 15
print(varsta >= 14)     # True
```

> `True` și `False` se scriu cu literă mare. Scrise cu literă mică, Python le ia drept nume de variabile și programul se oprește cu eroare.

4.4. Șiruri de caractere

În Python nu există un tip separat pentru un singur caracter: și o literă, și un text întreg sunt de tip `str`. Ele diferă numai prin lungime.

```
initiala = "A"
nume = "Ana"

print(len(initiala))    # 1
print(len(nume))        # 3
```

4.5. Aflarea tipului

```
x = 15
print(type(x))          # <class 'int'>
print(type(9.75))       # <class 'float'>
print(type("Ana"))      # <class 'str'>
```

4.6. Conversii

```
n = int("25")
x = float("3.5")
s = str(120)

print(int(3.9))       # 3
print(int(-3.9))      # -3
```

> **Conversia nu înseamnă rotunjire.** Trecerea de la real la întreg elimină partea fracționară, apropiind rezultatul de zero: `int(3.9)` dă 3, iar `int(-3.9)` dă −3.

5. Citirea și afișarea datelor

5.1. Citirea

Funcția `input()` întoarce întotdeauna un șir de caractere:

```
nume = input("Numele: ")
varsta = int(input("Vârsta: "))
inaltime = float(input("Înălțimea: "))

# două numere pe aceeași linie
a, b = map(int, input().split())
```

> Dacă scriem numai `x = input()` și utilizatorul introduce 25, variabila `x` conține textul `"25"`, nu numărul. Folosit într-o operație aritmetică, un șir de caractere produce eroare.

```
x = input()      # daca se tastează 25,
                 # x conține textul "25"

x = int(input()) # asa obținem numărul 25
```

5.2. Afișarea

```
a = 7
b = 5
suma = a + b

print(suma)
print("Suma este", suma)
print(f"Suma numerelor {a} și {b} este {suma}.")
```

Ultima formă este un *f-string*. Expresiile dintre acolade sunt înlocuite cu valorile lor.

6. Operatori și expresii

6.1. Operatori aritmetici

Operația	Operator
Adunare	`+`
Scădere	`-`
Înmulțire	`*`
Împărțire reală	`/`
Împărțire întreagă	`//`
Restul împărțirii	`%`
Ridicare la putere	`**`

```
a = 17
b = 5

print(a / b)     # 3.4
print(a // b)    # 3
print(a % b)     # 2
```

> **Cele două împărțiri dau rezultate de tipuri diferite.** `/` dă întotdeauna un număr real, chiar și când împărțirea este exactă: `10 / 2` dă `5.0`, nu `5`. Pentru câtul întreg se folosește `//`.

La numere negative, `//` rotunjește în jos, nu spre zero:

```
print(-17 // 5)    # -4
print(-17 % 5)     # 3
```

6.2. Ordinea operațiilor aritmetice

1. parantezele;
2. ridicarea la putere;
3. înmulțirea, împărțirea și restul;
4. adunarea și scăderea.

Astfel, `2 + 3 * 4` dă 14, iar `(2 + 3) * 4` dă 20. Folosim paranteze atunci când fac expresia mai ușor de înțeles.

6.3. Operatori relaționali

Operatorii relaționali compară două valori și produc `True` sau `False`.

Semnificație	Operator
Egal cu	`==`
Diferit de	`!=`
Mai mic decât	`<`
Mai mare decât	`>`
Mai mic sau egal	`<=`
Mai mare sau egal	`>=`

6.4. Operatori logici

Semnificație	Operator
ȘI logic	`and`
SAU logic	`or`
Negare	`not`

Pentru a verifica dacă x aparține intervalului [10, 99]:

```
10 <= x <= 99

# sau, echivalent:
x >= 10 and x <= 99
```

Prima formă se scrie ca în matematică și este acceptată de Python.

---

7. Structura liniară

Într-un algoritm liniar, instrucțiunile sunt executate o singură dată, în ordinea în care au fost scrise.

> **Exemplu.** Se citește un număr natural de exact trei cifre. Să se afișeze suma cifrelor sale.

Pentru n = 472: cifra unităților este 2, a zecilor 7, a sutelor 4, iar suma 13.

Pseudocod:

```
citește n
unități ← n % 10
zeci ← (n div 10) % 10
sute ← n div 100
suma ← sute + zeci + unități
scrie suma
```

În Python:

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

unitati = n % 10
zeci = n // 10 % 10
sute = n // 100

suma = sute + zeci + unitati
print("Suma cifrelor este", suma)
```

Testare

n	Rezultat așteptat	Motivul alegerii
`472`	13	caz obișnuit
`100`	1	conține cifre egale cu zero
`999`	27	toate cifrele au valoarea maximă

---

8. Structura alternativă

Structura alternativă permite alegerea unei ramuri de execuție în funcție de o condiție.

8.1. Forma generală

```
if conditie:
    instructiuni
else:
    alte_instructiuni
```

Indentarea face parte din sintaxă și arată care instrucțiuni aparțin fiecărei ramuri.
Modificarea ei schimbă înțelesul programului.

8.2. Exemplu: paritatea unui număr

Un număr este par dacă restul împărțirii sale la 2 este 0.

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

if n % 2 == 0:
    print("Numărul este par.")
else:
    print("Numărul este impar.")
```

8.3. Alternative multiple

Problemă. Se citește un punctaj cuprins între 0 și 100: cel puțin 80 — *Foarte bine*; cel puțin 50, dar mai puțin de 80 — *Bine*; mai puțin de 50 — *Mai trebuie exersat*.

```
punctaj = int(input("Punctaj = "))

if punctaj >= 80:
    print("Foarte bine")
elif punctaj >= 50:
    print("Bine")
else:
    print("Mai trebuie exersat")
```

> **Ordinea condițiilor este importantă.** Programul verifică mai întâi pragul cel mai mare. A doua condiție se scrie doar `punctaj >= 50`, fiindcă programul ajunge acolo numai dacă prima a fost falsă — deci se știe deja că punctajul este sub 80.

8.4. Problemă aplicativă

Problemă. Un magazin acordă o reducere de 10% dacă valoarea cumpărăturilor este de cel puțin 200 de lei. Să se afișeze suma finală.

```
valoare = float(input("Valoarea cumpărăturilor: "))

if valoare >= 200:
    plata = valoare * 0.90
else:
    plata = valoare

print(f"Suma de plată este {plata:.2f} lei.")
```

Testare

Valoare	Rezultat așteptat
`150`	150.00
`200`	180.00
`350`	315.00

> Valoarea 200 este un **caz-limită**. Enunțul spune „cel puțin 200”, deci condiția trebuie să conțină `>=`, nu `>`.

---

9. Structuri repetitive

O structură repetitivă execută un grup de instrucțiuni de mai multe ori.

9.1. Repetiție cu număr cunoscut de pași

Se afișează numerele de la 1 la 5.

```
for i in range(1, 6):
    print(i)
```

> `range(1, 6)` generează valorile de la 1 la **5**: ultima valoare nu este inclusă. Pentru a ajunge la n se scrie `range(1, n + 1)`.

9.2. Repetiție condiționată

Se citesc numere până la introducerea valorii 0 și se calculează suma lor.

```
suma = 0
n = int(input())

while n != 0:
    suma = suma + n
    n = int(input())

print(suma)
```

Valoarea 0 oprește citirea și nu este adunată. O astfel de valoare se numește uneori **santinelă**.

9.3. Prelucrarea cifrelor unui număr

Pentru a prelucra toate cifrele unui număr natural folosim repetat:

- `n % 10` — obține ultima cifră;
- `n // 10` — elimină ultima cifră.

Exemplu: suma cifrelor unui număr natural cu oricâte cifre.

```
n = int(input("n = "))
suma = 0

while n > 0:
    cifra = n % 10
    suma = suma + cifra
    n = n // 10

print(suma)
```

10. Depanarea programelor

Depanarea înseamnă identificarea și corectarea greșelilor dintr-un program. Programul următor ar trebui să verifice dacă un punctaj este cel puțin 50:

```
punctaj = input("Punctaj = ")

if punctaj > 50
print("Promovat")
else:
    print("Mai trebuie exersat")
```

Greșelile

1. valoarea citită nu este convertită în număr;
2. după condiție lipsește :;
3. primul print() nu este indentat;
4. valoarea-limită 50 trebuie acceptată, deci folosim >=.

Varianta corectă

```
punctaj = int(input("Punctaj = "))

if punctaj >= 50:
    print("Promovat")
else:
    print("Mai trebuie exersat")
```

11. Activitate practică

Problemele pot fi rezolvate mai întâi în pseudocod, apoi în Python. Programul se poate rula direct în pagină, în caseta de la finalul secțiunii.

1. **Valoarea absolută**
   Se citește un număr întreg. Să se afișeze valoarea sa absolută, fără a folosi o funcție predefinită.
   `Intrare: -17 → Ieșire: 17`

```
n = int(input())

if n < 0:
    n = -n

print(n)
```

2. **Maximul a două numere**
   Se citesc două numere întregi distincte. Să se afișeze valoarea mai mare.
   `Intrare: 14 9 → Ieșire: 14`

```
a, b = map(int, input().split())

if a > b:
    print(a)
else:
    print(b)
```

3. **Număr de două cifre**
   Se citește un număr natural. Să se afișeze DA dacă are exact două cifre și NU în caz contrar.
   `Intrare: 47 → Ieșire: DA`

```
n = int(input())

if 10 <= n <= 99:
    print("DA")
else:
    print("NU")
```

4. **Cea mai mare cifră**

   Se citește un număr natural de exact trei cifre. Să se afișeze cea mai mare cifră.

   `Intrare: 472 → Ieșire: 7`

```
n = int(input())

maxim = n % 10
n = n // 10

while n > 0:
    cifra = n % 10
    if cifra > maxim:
        maxim = cifra
    n = n // 10

print(maxim)
```

5. **Anotimpul**

   Se citește numărul unei luni. Să se afișeze iarna pentru 12, 1, 2; primavara pentru 3, 4, 5; vara pentru 6, 7, 8; toamna pentru 9, 10, 11; valoare invalida pentru orice alt număr.

   `Intrare: 4 → Ieșire: primavara`

```
luna = int(input())

if luna == 12 or luna == 1 or luna == 2:
    print("iarna")
elif 3 <= luna <= 5:
    print("primavara")
elif 6 <= luna <= 8:
    print("vara")
elif 9 <= luna <= 11:
    print("toamna")
else:
    print("valoare invalida")
```

6. **Suma numerelor de la 1 la n**

   Se citește un număr natural n. Să se calculeze suma 1 + 2 + … + n folosind o structură repetitivă.

   `Intrare: 5 → Ieșire: 15`

```
n = int(input())

suma = 0
for i in range(1, n + 1):
    suma = suma + i

print(suma)
```

7. **Numărul de cifre**

   Se citește un număr natural. Să se determine câte cifre are.

   `Intrare: 2048 → Ieșire: 4`

```
n = int(input())

if n == 0:
    cifre = 1
else:
    cifre = 0
    while n > 0:
        cifre = cifre + 1
        n = n // 10

print(cifre)
```

12. Ce trebuie să reținem

- Un algoritm este o succesiune finită și clară de pași.
- Programul este transpunerea algoritmului într-un limbaj de programare.
- Înaintea programării stabilim datele de intrare, datele de ieșire și relațiile dintre ele.
- Variabila este un nume asociat unei valori care se poate modifica.

- În Python, tipul este stabilit din valoarea atribuită, nu declarat de programator.
- Indentarea arată ce instrucțiuni fac parte dintr-un bloc.
- = realizează atribuirea, iar == realizează comparația.
- input() întoarce un șir; pentru numere este necesară conversia.
- / realizează împărțirea reală, iar // împărțirea întreagă.
- Structura liniară execută instrucțiunile în ordine.
- Structura alternativă alege o ramură în funcție de o condiție.
- Structura repetitivă execută un bloc de mai multe ori.
- Programele trebuie testate cu valori obișnuite, valori-limită și cazuri speciale.