INFORMATICĂ · PYTHON · CLASA A IX-A · SUBPROGRAME · LECȚIA 8

Subprograme (1): definire, parametri, apel

Durată: 100 min Nivel: mate-info · Python Înainte: lecțiile py9-02, py9-06

CE VEI ȘTI DUPĂ LECȚIE

- Explici rolul unui subprogram și identifici antetul, corpul, parametrii și valoarea returnată.
- Definești funcții cu def, le apelezi cu argumente potrivite și folosești return, inclusiv pentru mai multe valori.
- Urmărești mecanismul de executare al apelurilor, inclusiv când o funcție apelează altă funcție.
- Descompui o problemă simplă în subprograme și testezi fiecare subprogram separat.

1. De ce avem nevoie de subprograme

În lecția py9-06, la operațiile cu fracții, bucla lui Euclid apărea **identic** de mai multe ori. Dacă găseai o greșeală în ea, trebuia s-o corectezi peste tot. Un **subprogram** (în Python: **funcție**) este o secvență de instrucțiuni cu un **nume**, scrisă o singură dată și **apelată** de câte ori ai nevoie.

Câștigi **reutilizare** (scrii `cmmdc` o dată, îl folosești în zece programe), **claritate** (`if este_prim(x):` se citește ca o propoziție), **testare separată** (verifici fiecare funcție pe cazurile ei) și **lucru în echipă** (fiecare scrie un modul, apoi modulele se asamblează). Ai folosit deja subprograme **predefinite**: `print` , `input` , `int` , `abs` . Acum le scrii pe ale tale.

2. Definirea: antet și corp

```
def arie(lungime, latime):       # ANTETUL: def, numele, parametrii, două puncte
    s = lungime * latime         # CORPUL: instrucțiunile indentate
    return s                     # rezultatul trimis înapoi

print(arie(20, 15))              # apel → 300
print(arie(12.5, 8))             # alt apel → 100.0
total = arie(20, 15) + arie(10, 10)
print(total)                     # 400
```

	C++	Python
antet	`int arie(int lungime, int latime)`	`def arie(lungime, latime):`
corpul	între `{ }`	blocul indentat
tipuri	declarate pentru parametri și rezultat	nu se declară
fără rezultat	`void f()`	o funcție fără `return` (întoarce `None`)

Definiția doar „învață" programul ce înseamnă `arie`; **nu execută nimic**. Corpul rulează abia la apel. De aceea, funcția trebuie definită **înainte** de primul apel care se execută.

3. Parametri și argumente

- **Parametrii** sunt numele din antet (`lungime`, `latime`). Sunt ca niște variabile ale funcției, care primesc valori la fiecare apel.
- **Argumentele** sunt valorile date la apel (`20`, `15`, sau `L`, sau `2 * x + 1`). Ele se potrivesc cu parametrii **în ordine**: primul argument merge în primul parametru și așa mai departe.

```
def pret_final(pret, reducere):        # reducere în procente
    return pret - pret * reducere / 100

print(pret_final(200, 15))             # 170.0
print(pret_final(15, 200))             # -15.0: ordinea contează!
print(pret_final(reducere=15, pret=200))   # 170.0: argumente cu nume
```

Numărul argumentelor trebuie să fie egal cu numărul parametrilor: `pret_final(200)` dă `TypeError: pret_final() missing 1 required positional argument: 'reducere'`. O funcție poate avea și **zero** parametri: `def meniu():` se apelează cu `meniu()`. Parantezele sunt obligatorii.

4. return: rezultatul și ieșirea din funcție

`return valoare` face două lucruri: **trimite** valoarea în locul apelului și **oprește** imediat funcția. Poți avea mai multe `return`-uri, pe ramuri diferite:

```
def este_prim(n):
    if n < 2:
        return False             # ieșire rapidă pentru 0 și 1
    d = 2
    while d * d <= n:
        if n % d == 0:
            return False         # am găsit un divizor: nu mai căutăm
        d += 1
    return True                  # ajungem aici doar dacă nu am găsit niciunul

print(este_prim(1), este_prim(2), este_prim(91), este_prim(97))   # False True False True
```

Mai multe valori deodată. Scrii valorile separate prin virgulă și le primești în tot atâtea variabile:

```
def ore_minute(m):
    return m // 60, m % 60

h, mn = ore_minute(135)
print(h, mn)                     # 2 15
```

ATENȚIE — GREȘELI FRECVENTE

`print` **nu este** `return`. O funcție care doar *afișează* rezultatul nu îl dă înapoi: `r = f(5)` primește `None`, iar `f(5) + 1` dă eroare. Regula: funcțiile care calculează **returnează**; afișarea o face programul principal.

5. Mecanismul de executare

La fiecare apel, Python:

1. calculează argumentele;
2. creează un **cadru de apel** (un „spațiu de lucru” nou) în care parametrii primesc valorile argumentelor;
3. execută corpul funcției;
4. la `return` (sau la sfârșitul corpului) șterge cadrul și se **întoarce exact în locul apelului**, cu valoarea returnată.

Când o funcție apelează altă funcție, cadrele se pun unul peste altul, ca o **stivă**: ultimul apelat este primul care se termină. Am pus `print`-uri ca să vedem ordinea:

```
def cmmdc(a, b):
    print("  intru în cmmdc cu", a, b)
    while b != 0:
        a, b = b, a % b
    print("  ies din cmmdc cu", a)
    return a

def cmmmc(a, b):
    print(" intru în cmmmc cu", a, b)
    r = a // cmmdc(a, b) * b
    print(" ies din cmmmc cu", r)
    return r

print("încep")
print(cmmmc(60, 90))
print("gata")
```

```
încep
 intru în cmmmc cu 60 90
  intru în cmmdc cu 60 90
  ies din cmmdc cu 30
 ies din cmmmc cu 180
180
gata
```

În interiorul lui `cmmdc`, stiva este: `cmmdc(a=60, b=90)` deasupra lui `cmmmc(a=60, b=90)`, deasupra programului principal. După `return 30`, cadrul lui `cmmdc` dispare, iar `cmmmc` continuă: `r = 60 // 30 * 90 = 180`. Observă că `cmmdc` își modifică propriii `a` și `b` (ajung `30` și `0`), dar `a` și `b` din `cmmmc` rămân `60` și `90`. Fiecare cadru are variabilele lui. Despre asta vorbim pe larg în lecția py9-09.

6. Proiectare modulară: o problemă, trei subprograme

Problema (din programă): câte numere dintr-un șir citit au **suma cifrelor un număr prim**?

Analiză. Cerința se sparge în bucăți independente: (1) suma cifrelor unui număr; (2) testul de primalitate; (3) parcurgerea șirului și numărarea. Fiecare devine un modul.

```python
def suma_cifrelor(n):
    s = 0
    while n > 0:
        s += n % 10
        n //= 10
    return s

def este_prim(n):
    ...                          # exact ca în secțiunea 4

# programul principal
c = 0
for x in map(int, input().split()):
    if este_prim(suma_cifrelor(x)):
        c += 1
print(c)
```

Pentru intrarea `12 29 47 100 55` afișează `3` (sumele sunt 3, 11, 11, 1, 10).

Testare pe module. Înainte să rulezi tot programul, testezi fiecare funcție separat:

apel	așteptat	de ce acest test
`suma_cifrelor(9081)`	`18`	caz obișnuit, cu un 0 în mijloc
`suma_cifrelor(0)`	`0`	bucla nu se execută
`este_prim(1)`	`False`	1 nu este prim
`este_prim(2)`	`True`	cel mai mic prim
`este_prim(91)`	`False`	7 · 13: „pare” prim

Testele se pot automatiza cu `assert suma_cifrelor(9081) == 18`: dacă trece, nu se întâmplă nimic; dacă nu, programul se oprește cu `AssertionError` și știi exact ce modul să repari.

> **LA BAC**
>
> La Bac, subprogramele se cer de obicei cu antet precizat („Scrieți definiția completă a subprogramului `f`, cu parametrii...”). Respectă exact numele, ordinea parametrilor și ce se returnează. Nu citi și nu afișa în funcție decât dacă ți se cere.

7. Verifică-te

1. Care este antetul, care este corpul și care sunt parametrii în `def medie(a, b, c): return (a + b + c) / 3`?
2. Ce afișează `def dublu(x): return 2 * x` urmat de `print(dublu(dublu(3)) + dublu(1))`?
3. Ce afișează programul următor și de ce?

```python
def f(n):
    if n % 2 == 0:
        return "par"
    print("impar")

print(f(4), f(7))
```

4. Scrie o funcție `min_max(a, b)` care returnează mai întâi minimul, apoi maximul.
5. În ce ordine se afișează mesajele?

```python
def a():
    print("A")
def b():
    print("B1")
    a()
    print("B2")
b()
a()
```

Răspunsuri

1. Antetul: `def medie(a, b, c):`; corpul: `return (a + b + c) / 3`; parametrii: `a`, `b`, `c`.
2. `14`: `dublu(dublu(3)) = dublu(6) = 12`, plus `dublu(1) = 2`.
3. Întâi `impar` (afișat în timpul apelului `f(7)`), apoi `par None`. `f(7)` nu ajunge la niciun `return`, deci întoarce `None`. Argumentele lui `print` se calculează înainte de afișare.
4. `def min_max(a, b):` · `if a < b: return a, b` · `return b, a`.
5. `B1`, `A`, `B2`, `A`.

Probleme de lucru în clasă (pbinfo)

Le rezolvăm împreună în ședință, în ordinea de mai jos.

- [ ] **#368** **nrpal** — ușoară
  https://www.pbinfo.ro/probleme/368/nrpal
- [ ] **#64** **numarperfect** — ușoară
  https://www.pbinfo.ro/probleme/64/numarperfect

Temă pentru acasă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] **#1574** **prietene** — ușoară
  https://www.pbinfo.ro/probleme/1574/prietene
- [ ] **#375** **urmatorul-prim** — ușoară
  https://www.pbinfo.ro/probleme/375/urmatorul-prim