INFORMATICĂ · PYTHON · CLASA A IX-A · SUBPROGRAME · LECȚIA 9

Subprograme (2): variabile locale și globale, funcții predefinite

Durată: 100 min Nivel: mate-info · Python Înainte: lecțiile py9-08

CE VEI ȘTI DUPĂ LECȚIE

- Deosebești variabilele locale de cele globale și explici de ce funcțiile bune comunică prin parametri și return, nu prin global.
- Prezici, intuitiv, ce se întâmplă cu un argument întreg și cu o listă transmise unei funcții.
- Folosești corect funcțiile predefinite: abs, round, int, float, str, math.sqrt, math.floor, math.ceil, len, min, max, sum, inclusiv în cazurile limită.
- Descompui o aplicație mică în subprograme și o testezi modul cu modul.

1. Variabile locale

O variabilă creată **în interiorul** unei funcții (inclusiv parametrii) este **locală**: există doar în cadrul de apel al funcției și dispare la `return` (vezi py9-08, secțiunea 5).

```
def arie_cerc(r):
    pi = 3.14159          # pi, a și r sunt locale
    a = pi * r * r
    return a

print(arie_cerc(2))       # 12.56636
print(a)                  # NameError: name 'a' is not defined
```

Două funcții pot avea fiecare câte o variabilă `s` sau `i`, fără să se încurce. Asta face funcțiile **independente**: poți scrie și testa `suma_cifrelor` fără să știi ce nume folosește restul programului.

2. Variabile globale și de ce le evităm

O variabilă creată **în afara** oricărei funcții este **globală**. O funcție o poate **citi**:

```
REDUCERE = 20             # procent; constantă globală, scrisă cu majuscule
def pret_redus(p):
    return p - p * REDUCERE / 100

print(pret_redus(150))    # 120.0
```

Dar dacă funcția o **atribuie**, Python o consideră locală, deci încercarea de a o modifica eșuează:

```
contor = 0
def creste():
    contor = contor + 1   # UnboundLocalError: cannot access local variable 'contor'…
```

Cu `global contor` pe primul rând al funcției, merge. Totuși, **evită** `global`:

cu `global`	cu parametru și `return`
`def creste():` ↵ `global contor` ↵ `contor += 1`	`def creste(c):` ↵ `return c + 1`
apel: `creste()`; din apel nu vezi ce se schimbă	apel: `contor = creste(contor)`; se vede tot
rezultatul depinde de starea întregului program	același argument dă mereu același rezultat
greu de testat separat	se testează cu un `assert`

SFAT

Regula practică: o funcție primește tot ce are nevoie prin **parametri** și dă tot ce a calculat prin `return`. Globalele le folosești, cel mult, pentru **constante** citite (ca `REDUCERE`).

3. Ce primește funcția: un int sau o listă

DE ȘTIUT

Mini-introducere în liste (pe larg, în lecțiile py9-14 și py9-15). O **listă** ține mai multe valori sub un singur nume: `note = [7, 9]`. Elementele se numără de la 0 (`note[0]` este `7`, `note[1]` este `9`), `len(note)` este numărul lor, `note.append(0)` adaugă o valoare la sfârșit, iar `for x in note:` parcurge elementele pe rând. Atât ne trebuie în această lecție.

Parametrul primește **aceeași valoare** pe care o dai la apel, nu o copie. Ce se întâmplă mai departe depinde de tipul valorii:

```
def dubleaza(x):
    x = x * 2                    # x primește o valoare NOUĂ
    print("în funcție:", x)

n = 5
dubleaza(n)                      # în funcție: 10
print("după apel:", n)           # după apel: 5

def adauga_zero(lista):
    lista.append(0)              # modifică lista PRIMITĂ
    print("în funcție:", lista)

note = [7, 9]
adauga_zero(note)                # în funcție: [7, 9, 0]
print("după apel:", note)        # după apel: [7, 9, 0]
```

Intuitiv: un număr nu poate fi „modificat pe loc". `x = x * 2` doar face ca numele local `x` să indice altă valoare, iar `n` rămâne 5. O listă **poate** fi modificată pe loc (`append`), iar `note` și `lista` sunt aceeași listă, văzută sub două nume. Dacă însă în funcție scrii `lista = [1, 2, 3]`, numele local indică o listă nouă, iar `note` nu se schimbă. Listele le studiem pe larg în lecțiile py9-14 și py9-15; deocamdată reține regula: **o funcție care modifică o listă primită o modifică și pentru cel care a apelat-o**.

4. Funcții predefinite pentru calcule și conversii

Unele sunt mereu disponibile (`abs`, `round`, `int`, ...). Altele sunt în modulul `math`, care se încarcă cu `import math`.

apel	rezultat	de reținut
`abs(-7)`, `abs(3.5)`	`7`, `3.5`	valoarea absolută
`round(3.14159, 2)`	`3.14`	rotunjire la 2 zecimale
`round(9.46)`, `round(9.5)`, `round(8.5)`	`9`, `10`, `8`	la „,,5" exact, alege numărul **par**
`int(9.99)`, `int(-2.7)`	`9`, `-2`	**taie** zecimalele (spre 0)
`math.floor(-2.7)`, `math.ceil(2.1)`	`-3`, `3`	partea întreagă (în jos), respectiv în sus
`math.sqrt(16)`, `math.sqrt(2)`	`4.0`, `1.4142135623730951`	radicalul este **real**
`math.sqrt(-1)`	`ValueError: math domain error`	validează datele
`int("12") + 1`, `float("3.5") * 2`	`13`, `7.0`	conversii din text
`int("3.5")`	`ValueError`	`int` nu acceptă un text cu virgulă zecimală
`str(12) + "3"`	`'123'`	conversie în text

ATENȚIE — GREȘELI FRECVENTE

`int()` nu rotunjește. Pentru media 9,5, `int(9.5)` dă `9`, iar `round(9.5)` dă `10`. Dar `round(8.5)` dă `8`, pentru că Python rotunjește „,,5" exact spre numărul par. Dacă regula școlii este „de la ,50 în sus", folosește `math.floor(m + 0.5)` (pentru note pozitive).

Aplicație (din programă): distanța dintre două puncte.

```
import math
def distanta(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

print(distanta(0, 0, 3, 4), distanta(1, 1, 4, -3))   # 5.0 5.0
```

5. Funcții predefinite pentru colecții

apel, pentru `note = [7, 10, 9, 8, 10]`	rezultat
`len(note)`	`5`
`min(note)`, `max(note)`	`7`, `10`
`sum(note)`	`44`

apel, pentru `note = [7, 10, 9, 8, 10]`	rezultat
`sum(note) / len(note)`	`8.8` (media)
`max(3, 8, 5)`	`8` : merge și cu valori date direct

Cazuri limită de știut: `sum([])` este `0` și `len([])` este `0`, deci media unei liste goale dă `ZeroDivisionError`. `max([])` dă `ValueError: max() iterable argument is empty`. Iar `min([3, "a"])` dă `TypeError`: nu se pot compara numere cu texte. Textele între ele se compară alfabetic, dar majusculele vin înaintea minusculelor: `min(["mere", "Banane", "caise"])` este `'Banane'`.

> **DE ȘTIUT**
>
> `sum(v)` face exact ce făcea bucla ta `s = 0; for x in v: s += x`. Este mai scurt, mai clar și mai greu de greșit. Algoritmul trebuie totuși să-l știi: la Bac, în pseudocod, nu există `sum`.

6. Mini-proiect modular: raportul meteo

Cerința: se citesc temperaturile dintr-o săptămână. Afișează numărul de zile, media, minimul, maximul, amplitudinea și numărul de zile peste medie.

Descompunere. Citire → calcule (fiecare cu funcția lui) → afișare. Programul principal doar le leagă.

```
def citeste_valori():
    return list(map(int, input().split()))   # lista valorilor de pe un rând
    # split() dă lista textelor "12", "15", ...; map(int, ...) le convertește pe rând;
    # list(...) adună rezultatele într-o listă (lista vidă dacă rândul e gol)

def media(v):
    return sum(v) / len(v)

def numara_peste(v, prag):
    c = 0
    for x in v:
        if x > prag:
            c += 1
    return c

def raport(v):
    m = media(v)
    print(f"zile: {len(v)}")
    print(f"media: {m:.2f}")
    print(f"minim: {min(v)}, maxim: {max(v)}, amplitudine: {max(v) - min(v)}")
    print(f"zile peste medie: {numara_peste(v, m)}")

# programul principal
t = citeste_valori()
if len(t) == 0:
    print("nu există date")      # altfel media ar împărți la 0
else:
    raport(t)
```

intrare	ieșire
`12 15 9 14 18 11 13`	`zile: 7`, `media: 13.14`, `minim: 9, maxim: 18, amplitudine: 9`, `zile peste medie: 3`
`20`	`zile: 1`, `media: 20.00`, `minim: 20, maxim: 20, amplitudine: 0`, `zile peste medie: 0`
(rând gol)	`nu există date`

Observă: nicio funcție nu folosește variabile globale. Fiecare poate fi testată separat (`assert numara_peste([1, 5, 9], 4) == 2`), iar o echipă de trei elevi și-ar putea împărți funcțiile.

7. Verifică-te

1. Ce afișează `print(round(7.5), round(6.5), int(7.9), math.floor(-0.5))`?
2. Ce afișează programul?

```python
x = 10
def f(x):
    x = x + 5
    return x
print(f(x), x)
```

3. După `def g(v): v.append(len(v))`, `a = [4, 4]`, `g(a)`, `g(a)`, ce conține `a`?
4. De ce dă eroare `media = sum(v) / len(v)` pentru `v = []`? Cum o eviți?
5. Scrie, cu o singură expresie, ipotenuza unui triunghi dreptunghic cu catetele `a` și `b`.
6. Ce afișează `print(max("10", "9"), max(10, 9))`? De ce diferă?

Răspunsuri

1. `8 6 7 -1`: la „.5" exact se alege numărul par (8, respectiv 6); `int` taie; `floor` coboară.
2. `15 10`: `x` din funcție este parametrul local; `x` global rămâne 10.
3. `[4, 4, 2, 3]`: lista este modificată de ambele apeluri.
4. `len([])` este 0: `ZeroDivisionError`. Verifici întâi `if len(v) > 0`.
5. `math.sqrt(a ** 2 + b ** 2)` (după `import math`).
6. `9 10`: textele se compară caracter cu caracter, iar `"9" > "1"`; numerele se compară după valoare.

Probleme de lucru în clasă (pbinfo)

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

- [ ] **#1255** **lipsa** — ușoară
  https://www.pbinfo.ro/probleme/1255/lipsa
- [ ] **#435** **factoriprimi** — ușoară
  https://www.pbinfo.ro/probleme/435/factoriprimi

Temă pentru acasă (pbinfo)

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

- [ ] #365 **cifre-comune** — ușoară
  https://www.pbinfo.ro/probleme/365/cifre-comune
- [ ] #445 **pseudoperfect** — ușoară
  https://www.pbinfo.ro/probleme/445/pseudoperfect