test-e.ro

CAIET ELEV

Lecția 20

Sintaxa def, return, parametri în Python. Împachetarea algoritmilor numerici cunoscuți în subprograme

Un subprogram e o cutie cu două deschideri: pe una intră datele, pe cealaltă iese rezultatul.

DISCIPLINA Informatică **CLASA** a IX-a · mate-info **LIMBAJ** Python 3

NUMELE ELEVULUI .....

CUM FOLOSEȘTI CAIETUL

Rezolvă exercițiile pe hârtie: urmărești execuția pas cu pas, completezi, asociezi și scrii singur soluțiile, apoi le verifici pe platformă. Ce ai voie să folosești: secvențial, if/elif/else, while, for cu range, break, continue, %, //, ord/chr, listă literală, indexare, len(), for x in listă, metodele listei, def și return, iar nou de la această lecție atribuirea simultană (a, b = b, a % b). Fără min/max/sum și fără import math (de la L21), fără slicing, fără open/with, fără class. Cifrele se parcurg doar cu while și % 10 // 10 — nicio conversie la șir. Paritatea se scrie n % 2 != 0. Fără diacritice în codul Python.

CUPRINS

1 Ce ne trebuie din lecția trecută .....
2 Intrarea în subprogram .....
3 Ieșirea din subprogram .....
4 Împachetăm ce știm deja .....
5 Instrumente chemate de zeci de ori .....
6 Un instrument care primește o listă .....
7 Cod ghidat · șase probleme .....

SECȚIUNEA 1 · REACTUALIZARE

1 Ce ne trebuie din lecția *trecută*

Lecția trecută ai construit primele subprograme. Astăzi le duci mai departe, așa că merită să-ți încălzești mâinile: reamintește-ți cum arată un antet, ce face `return` și cum se plimbă un indice printr-o listă.

Un subprogram are un antet care începe cu `def`, un nume, o listă de nume între paranteze și două puncte. Sub el, indentat, stă corpul — instrucțiunile care se execută la fiecare apel. Când corpul ajunge la `return`, valoarea calculată pleacă înapoi în locul de unde a fost chemat subprogramul, iar execuția lui se oprește pe loc.

Aceeași idee în Python (cu sintaxă reală):

```
1  def dublu(x):
2      return x + x
3  print(dublu(5))
5  print(dublu(dublu(3)))
```

Al doilea apel merită citit de două ori: `dublu(3)` se calculează întâi și dă 6, apoi 6 intră în `dublu` și iese 12.

Din lecțiile despre liste ai nevoie astăzi de trei lucruri: `len(v)` spune câte elemente sunt, `v[i]` ia elementul de pe poziția `i` (pozițiile încep de la 0), iar o listă se parcurge cu un indice care crește într-un `while`.

> **DE REȚINUT**
>
> Numărarea cifrelor unui număr rămâne treaba lui `while` cu `% 10` și `// 10`. Nu convertim numărul la șir de caractere — vrem să se vadă aritmetica, nu s-o ascundem.

A

EXERCIȚIUL A · GRILĂ

Ce ai reținut din lecția trecută

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce face instrucțiunea `return` într-un subprogram?

   `Șterge variabilele locale, dar continuă execuția instrucțiunilor următoare`

   `Trimite valoarea înapoi în locul apelului și oprește imediat subprogramul`

- Afișează valoarea pe ecran și lasă subprogramul să ruleze mai departe
- Salvează valoarea într-o variabilă globală, ca să poată fi folosită mai târziu

2. Pentru `v = [8, 3, 5]`, ce afișează `print(len(v), v[0])`?

- `3 5`
- `3 8`
- `8 3`
- `2 8`, pentru că numărătoarea pozițiilor începe de la zero

3. Câte împărțiri face bucla `while n > 0: n = n // 10` pentru `n = 904`?

- 2, pentru că zeroul din mijloc nu se numără
- 3
- 4, pentru că se numără și pasul în care `n` devine `0`
- 9, adică valoarea primei cifre

4. Pentru `def f(x): return x * 2`, ce afișează `print(f(3) + f(4))`?

- Eroare — un subprogram nu poate fi apelat de două ori în aceeași expresie
- 24, pentru că se înmulțesc între ele rezultatele celor două apeluri
- 14
- 7, pentru că `x * 2` se aplică o singură dată sumei

B

EXERCIȚIUL B · DRY-RUN

Suma elementelor unei liste

DRY-RUN

Urmărește pas cu pas cum se adună elementele listei. Coloana i îți este dată; tu completezi valoarea v[i] luată la pasul acela și valoarea lui s după adunare. Primul rând îți este dat ca exemplu, iar ultimul rând, cel pe care bucla se oprește, îți este dat complet și nu se completează: el arată doar de ce s-a oprit bucla. Toate valorile pe care le completezi sunt numerice.

```
1  v = [5, 9, 2]
2  i = 0
3  s = 0
4  while i < len(v):
5      s = s + v[i]
6      i = i + 1
7  print(s)
```

i (dat)	v[i]	s după adunare
0	5	5
1		
2		
3 — bucla se oprește	—	16

HINT *la rândurile pe rând: s nou este s vechi plus elementul de pe poziția i, iar i crește cu 1 după fiecare adunare.*

SPRIJIN · SCRIE CODUL

Aria dreptunghiului

SCRIERE COD

Se citesc două numere naturale a și b, fiecare de pe câte un rând. Scrie un subprogram aria(a, b) care întoarce produsul lor și afișează rezultatul apelului. Ex.: pentru a = 3 și b = 4, subprogramul întoarce 12, deci se afișează 12.

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

Câte elemente și primul element

SCRIERE COD

Se citește un număr natural n, apoi n numere naturale, fiecare de pe câte un rând. Memorează-le într-o listă și afișează pe un singur rând, despărțite printr-un spațiu, câte elemente are lista și valoarea primului element. Ex.: pentru n = 3 și valorile 5, 9, 2, lista este [5, 9, 2], deci se afișează 3 5.

HINT *n este cel puțin 1, deci lista nu rămâne niciodată goală.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

Ultima cifră, ca instrument

SCRIERE COD

Se citesc două numere naturale a și b, fiecare de pe câte un rând. Scrie un subprogram ultima(n) care întoarce ultima cifră a numărului primit. Afișează pe același rând, despărțite printr-un spațiu, ultima cifră a lui a și ultima cifră a lui b. Subprogramul se scrie o singură dată și se apelează de două ori. Ex.: pentru a = 1234 și b = 7 se afișează 4 7.

HINT *Ultima cifră a unui număr este restul împărțirii lui la 10. Nu ai nevoie de nicio buclă — corpul subprogramului are un singur rând.*

HINT **Atenție.** *Subprogramul trebuie să* **întoarcă** *valoarea, nu s-o afișeze.* `print` *stă în programul principal, pe rezultatul apelului.*

SOLUȚIA TA · PYTHON

2 SECȚIUNEA 2 · PARAMETRII DE INTRARE

Intrarea în *subprogram*

Numele din antet și valorile din apel nu sunt același lucru — au chiar nume diferite în programă. Iar unii parametri pot veni cu o valoare pregătită dinainte, ca să nu-i mai scrii de fiecare dată.

> **CONȚINUT DE EXTINDERE**
>
> Parametrii impliciți și apelul cu nume, din a doua jumătate a acestei secțiuni, nu sunt cerute de programa clasei a IX-a. Îi învățăm pentru că apar peste tot în codul real și pentru că fac exercițiile mult mai scurte. Prima jumătate — parametrii formali și actuali — este, în schimb, conținut obligatoriu.

Când scrii `def pret_final(pret, tva):`, numele `pret` și `tva` sunt **parametri formali**. Ele sunt goale: există doar ca etichete, ca să știe corpul subprogramului cum să numească datele care vor veni. Când scrii apelul `pret_final(200, 11)`, valorile 200 și 11 sunt **parametri actuali**. Potrivirea dintre ele se face după poziție: primul actual intră în primul formal, al doilea în al doilea, și tot așa.

DE REȚINUT

Parametrii formali stau în **definiție** și sunt nume. Parametrii actuali stau în **apel** și sunt valori. Un subprogram are o singură definiție, dar poate avea zeci de apeluri — deci un set de parametri formali și zeci de seturi de parametri actuali.

Uneori un parametru are aproape mereu aceeași valoare. Cota standard de TVA în România este 21%, așa că ar fi obositor s-o scrii la fiecare apel. (Pentru câteva categorii — alimente de bază, medicamente, cărți, cazare — se aplică o cotă redusă de 11%.) Îi dăm atunci cotei standard o valoare pregătită dinainte, direct în antet:

```
1  def pret_final(pret, tva=21):
2      return pret + pret * tva // 100
3  print(pret_final(200))
5  print(pret_final(200, 11))
6  print(pret_final(200, tva=0))
```

Primul apel afișează 242, al doilea 222, al treilea 200. `tva=21` din antet este un **parametru implicit**: se folosește doar dacă apelul nu spune altceva. Iar `tva=0` din al treilea apel este un **parametru cu nume**: în loc să numeri pozițiile, spui direct cui îi dai valoarea.

CIFRA SE SCHIMBĂ, MECANISMUL NU

Cotele de TVA se modifică din când în când prin lege — cea folosită aici este cea în vigoare la data redactării lecției. Ce înveți acum nu este cifra, ci mecanismul: o valoare scrisă în antet ține locul celei mai frecvente situații, iar apelul o poate înlocui oricând.

ATENȚIE

Parametrii impliciți se scriu **la sfârșitul** listei din antet. `def f(a, b=2, c)` este o **eroare de sintaxă**: programul nici măcar nu pornește, fiindcă Python nu poate citi antetul — după ce a apărut un parametru cu valoare pregătită, nu mai știe unde se termină numărătoarea pozițiilor. Iar un apel căruia îi lipsește un parametru fără valoare implicită — de exemplu `f(b=3)` — este o **eroare de executare**: programul pornește normal și se oprește abia în momentul apelului greșit, cu mesajul `missing 1 required positional argument`.

Mesajele de eroare ale Pythonului sunt în engleză și folosesc cuvântul *argument*: el numește exact ceea ce noi numim **parametru actual**. Când vezi `argument` într-o eroare, citește „valoare din apel”.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Aplicația de livrare a mâncării calculează prețul cu un parametru implicit pentru comision și unul pentru taxa de transport — tu vezi doar prețul final, dar cei doi parametri sunt acolo. Când printezi un document, programul cheamă un subprogram de tipărire cu zeci de parametri

impliciți (o copie, alb-negru, față-verso oprit); tu schimbi cu numele doar ce te interesează. La fel, un formular online pornește cu unele câmpuri deja completate — județul, anul curent — și te lasă să schimbi doar ce nu se potrivește.

A

EXERCIȚIUL A · GRILĂ

Cine primește ce

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. În definiția `def pret_final(pret, tva=21):` și în apelul `pret_final(200)`, cum se numesc `pret` și `tva`?

   - Parametri actuali, pentru că apar scriși negru pe alb în definiția subprogramului
   - Parametri formali; la apel, `pret` primește 200, iar `tva` rămâne cu valoarea implicită 21
   - Parametri formali, dar `tva` rămâne fără valoare până când apelul îi dă una explicit
   - Variabile globale, pentru că sunt scrise pe primul rând al programului

2. Pentru `def g(a, b=2): return a + b`, care apel se oprește cu eroare?

   - `g(b=3)`
   - `g(5, 3)`
   - `g(5, b=3)`
   - `g(5)`

3. Ce afișează `print(pret_final(100, tva=0))`, pentru subprogramul din teorie?

   - 100
   - Eroare, pentru că un parametru implicit nu poate fi schimbat la apel
   - 0, pentru că întregul rezultat se înmulțește cu cota de TVA primită
   - 121, pentru că valoarea implicită 21 are întâietate față de cea din apel

B

EXERCIȚIUL B · ASOCIERE

Ce valoare ia tva la fiecare apel

ASOCIERE

Pentru subprogramul `def pret_final(pret, tva=21)` de mai sus, scrie în căsuța fiecărui apel litera rezultatului potrivit. Gândește-te de fiecare dată dacă `tva` primește o valoare din apel sau rămâne cu cea implicită. Fiecare literă se folosește o singură dată.

- ☐ `pret_final(200)`
- ☐ `pret_final(200, 11)`
- ☐ `pret_final(200, tva=0)`
- ☐ `pret_final(tva=11, pret=100)`

A ambii parametri sunt dați prin nume, deci ordinea nu contează → 111

B tva rămâne 21, valoarea implicită din antet → 242

C tva primește 11 după poziție, fiind al doilea parametru actual → 222

D tva primește 0 prin nume, deci TVA-ul nu se mai adaugă → 200

C

EXERCIȚIUL C · SCRIE CODUL

Prețul cu TVA, în două feluri

SCRIERE COD

Se citesc două numere naturale: prețul p al unui produs, în lei, și cota t de TVA, în procente. Scrie un subprogram pret_final(pret, tva=21) care întoarce valoarea pret + pret * tva // 100. Afișează pe primul rând rezultatul apelului fără să dai cota (deci cu valoarea implicită), iar pe al doilea rând rezultatul apelului în care dai cota citită prin nume. Ex.: pentru p = 100 și t = 11: primul apel dă 100 + 100 * 21 // 100 = 121, iar al doilea dă 100 + 100 * 11 // 100 = 111. Se afișează 121 pe primul rând și 111 pe al doilea.

HINT *Împărțirea // taie zecimalele, deci pentru p = 37 și t = 21 rezultatul este 37 + 7 = 44, nu 44,77.*

HINT **Atenție.** *Aici contează cum scrii, nu doar ce iese. Ai putea obține aceleași două rânduri scriind pret_final(p, 21) pe primul rând și pret_final(p, t) pe al doilea. Rezultatul ar fi identic, dar n-ai fi folosit nici valoarea implicită, nici apelul prin nume — adică exact cele două lucruri pe care le exersăm. Verificarea automată se uită și la felul în care ai scris apelurile.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 3 · VALOAREA ÎNTOARSĂ

3 Ieșirea din *subprogram*

`return` nu e o afișare. E o trimitere de valoare înapoi, în locul exact de unde a fost chemat subprogramul — și, în același timp, o oprire. Iar dacă rezultatul are două jumătăți, poți trimite ambele deodată.

Când execuția ajunge la `return`, se întâmplă două lucruri simultan: valoarea calculată pleacă înapoi la apel, iar subprogramul se oprește. Tot ce e scris după acel `return` nu se mai execută niciodată. De aceea un subprogram poate avea mai multe instrucțiuni `return` — se execută prima la care ajunge.

Un subprogram fără niciun `return` nu e o eroare, dar întoarce `None`, adică „nimic”. Dacă îl afișezi, vezi chiar cuvântul `None` pe ecran.

DOUĂ NUME PENTRU ACELAȘI LUCRU

În programa școlară și în manual, o astfel de „cutie” se numește **subprogram**. În Python și în documentația lui, ea se numește **funcție** — de la cuvântul-cheie `def`, prescurtare de la *define*. Sunt exact același lucru: `def suma_cifrelor(n)` definește un subprogram, adică o funcție. Vei întâlni ambele cuvinte, uneori în aceeași pagină; când citești „funcție predefinită”, înțelege „subprogram care vine gata scris în Python” — despre acestea este lecția următoare.

ATENȚIE

`print` înăuntrul subprogramului și `return` sunt lucruri diferite. `print` arată valoarea pe ecran și atât — programul principal nu o poate folosi mai departe. `return` o trimite înapoi, iar acolo o poți aduna, compara sau da mai departe altui subprogram. Un algoritm care se termină cu `print` în loc de `return` este un instrument care vorbește singur, dar nu-ți dă nimic în mână.

Uneori rezultatul are, în mod natural, două jumătăți: împărțirea cu rest dă un cât **și** un rest. Python le trimite pe amândouă deodată:

```
1  def imparte(a, b):
2      return a // b, a % b
3  c, r = imparte(17, 5)
4  print(c, r)
```

Programul afișează `3 2`. Cele două valori pleacă împachetate într-o pereche (Python o numește *tuplu*), iar scrierea `c, r = ...` le desface la loc, în ordinea în care au fost trimise.

DE REȚINUT

Aceeași scriere ne dă acum și cea mai scurtă formă a algoritmului lui Euclid: `a, b = b, a % b`. Partea dreaptă se calculează **întâi, în întregime**, cu valorile vechi, și abia apoi se fac ambele atribuiri. De aceea nu mai avem nevoie de variabila temporară `r` din lecția despre CMMDC. (La secțiunea următoare vom pune cele două forme una lângă alta: cea cu variabilă temporară rămâne singura care se poate scrie în pseudocod.)

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Un GPS întoarce dintr-un singur calcul două valori: latitudinea și longitudinea. O aplicație de sport îți dă, după alergare, distanța și timpul — nu are niciun sens să le calculeze separat, fiindcă vin din aceeași parcurgere a datelor. Iar la bancomat, o singură operație produce suma extrasă și noul sold.

A

EXERCIȚIUL A · GRILĂ

Ce iese din cutie

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce întoarce un subprogram care nu conține nicio instrucțiune `return`?

   - Ultima valoare calculată în corpul subprogramului, oricare ar fi ea
   - Valoarea 0, pentru că este valoarea neutră a adunării
   - Eroare de execuție, pentru că orice subprogram trebuie să întoarcă ceva
   - `None`, adică „nimic”

2. Pentru `def pereche(): return 3, 8` și apelul `x, y = pereche()`, ce valori au `x` și `y`?

   - `x` este 8, iar `y` este 3, pentru că valorile se desfac în ordine inversă
   - `x` este 3, iar `y` este 8
   - Amândouă primesc perechea întreagă, adică `3, 8`
   - Eroare, pentru că `return` poate trimite înapoi o singură valoare

3. Ce se întâmplă cu instrucțiunile scrise după un `return` care s-a executat?

   - Se execută doar dacă în subprogram mai există o instrucțiune `return`
   - Se execută normal, dar rezultatul lor nu mai poate fi afișat pe ecran
   - Se execută la următorul apel al aceluiași subprogram, înaintea celorlalte
   - Nu se mai execută niciodată, pentru că subprogramul s-a oprit deja

B

EXERCIȚIUL B · DRY-RUN

Trei apeluri ale aceluiași subprogram

DRY-RUN

Subprogramul imparte întoarce două valori deodată. Coloana cu apelul îți este dată; tu completezi câtul c și restul r care ies din fiecare apel. Tabelul are câte un rând per apel, nu per iterație — subprogramul acesta nu conține nicio buclă. Primul rând îți este dat ca exemplu, iar valorile de completat sunt numerice.

```
1  def imparte(a, b):
2      return a // b, a % b
3  c, r = imparte(17, 5)
5  print(c, r)
6  c, r = imparte(9, 3)
7  print(c, r)
8  c, r = imparte(4, 7)
9  print(c, r)
```

Apelul (dat)	c	r
imparte(17, 5)	3	2
imparte(9, 3)		
imparte(4, 7)		

HINT *Câtul este a // b, adică de câte ori intră b în a, iar restul este a % b, adică ce rămâne după aceste împărțiri.*

C

EXERCIȚIUL C · SCRIE CODUL

Câtul și restul, dintr-un singur apel

SCRIERE COD

Se citesc două numere naturale a și b, fiecare de pe câte un rând, cu b diferit de zero. Scrie un subprogram imparte(a, b) care întoarce două valori: câtul împărțirii întregi și restul. Desfă cele două valori în două variabile și afișează-le pe același rând, despărțite printr-un spațiu. Ex.: pentru a = 17 și b = 5, subprogramul întoarce perechea 3 și 2, deci se afișează 3 2.

HINT **Atenție.** *Nu scrie două subprograme și nici două return unul sub altul — al doilea n-ar fi executat niciodată. Un singur return trimite ambele valori.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 4 · TRUSA DE INSTRUMENTE (1)

4 Împachetăm ce știm *deja*

De la lecția a noua încoace ai scris de zeci de ori aceeași buclă pe cifrele unui număr. Astăzi n-o mai scrii: o închizi într-un subprogram, o dată, și o chemi pe nume ori de câte ori ai nevoie.

Împachetarea nu schimbă nimic din interiorul algoritmului. Bucla rămâne exact cea de anul trecut, literă cu literă. Se schimbă doar rama: un antet deasupra, care dă algoritmului un nume și primește datele, și un `return` acolo unde înainte era un `print`.

SUMA CIFRELOR

Bucla scoate ultima cifră cu `% 10`, o adună, apoi taie numărul cu `// 10`. Împachetată, devine un instrument pe care îl poți chema oriunde:

OGLINDITUL

Aceeași buclă, dar cifrele scoase se lipesc într-un număr nou, care crește de zece ori la fiecare pas: `o = o * 10 + n % 10`. Pentru 2026 iese 6202, iar pentru 100 iese 1 — zerourile de la coadă devin zerouri în față și dispar, ceea ce e corect matematic.

ATENȚIE

Când scrii `n = n // 10` înăuntrul subprogramului, **legi numele `n` de o valoare nouă, doar în interior**. Variabila cu același nume din programul principal nu se clintește. De aceea poți scrie liniștit `print(n, oglindit(n))`: apelul nu strică valoarea lui `n` de afară. La lecția trecută trebuia să salvezi numărul într-o a doua variabilă înainte de buclă — acum subprogramul face asta pentru tine.

REGULA EXACTĂ, PENTRU MAI TÂRZIU

Reatribuirea unui parametru nu schimbă variabila din exterior. Formulată așa, regula e adevărată pentru orice fel de dată — numere, dar și liste. Există însă o nuanță pe care o vom studia la lecția despre metodele listei: dacă în loc să reatribui parametrul îl **modifici pe loc** — de exemplu cu `v.append(...)` sau `v.sort()` — atunci lista din afară chiar se schimbă, fiindcă e una și aceeași listă, nu o copie a ei. Toate subprogramele din lecția de azi doar citesc datele primite și întorc rezultate; niciunul nu modifică nimic pe loc.

Un instrument poate fi chemat și de un alt instrument, chiar în buclă. Rădăcina digitală a unui număr se obține adunând cifrele, apoi adunând cifrele rezultatului, până rămâne o singură cifră: pentru 12345 → 15 → 6.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Cifra de control de pe cardul bancar și de pe codul de bare al oricărui produs din magazin se calculează dintr-o sumă a cifrelor — nu chiar cea simplă de aici, ci una în care cifrele de pe anumite poziții se numără de două ori. Ideea rămâne aceeași: o singură cifră, calculată din toate celelalte, care trădează orice greșeală de tastare. Casa de marcat o recalculează la fiecare bip. Numerele-oglindă apar la verificarea palindroamelor, folosite în testele de memorie și în puzzle-urile de logică. Iar rădăcina digitală e trucul prin care contabilii verificau, înainte de calculatoare, dacă o adunare lungă e plauzibilă.

A

EXERCIȚIUL A · COMPLETARE

Rețeta împachetării

COMPLETARE

Completează golurile folosind cuvintele din rezervă (două cuvinte rămân nefolosite).

`antetul` `formal` `global` `return` `indentarea` `actual`

Când împachetăm un algoritm deja învățat, bucla rămâne neatinsă: deasupra ei adăugăm _____, care începe cu def și primește datele, iar afișarea finală o înlocuim cu _____, ca rezultatul să plece înapoi la apel. Valoarea trimisă în momentul apelului este parametrul _____, iar numele scris în definiție este parametrul _____.

B

EXERCIȚIUL B · SCRIE CODUL

Oglinditul unui număr, ca instrument

SCRIERE COD

Se citește un număr natural n. Scrie un subprogram oglindit(n) care întoarce numărul obținut prin citirea cifrelor lui n de la dreapta la stânga. Afișează pe același rând numărul citit, un spațiu, apoi valoarea întoarsă de subprogram. Ex.: pentru n = 2026, cifrele citite invers dau 6202, deci se afișează 2026 6202.

HINT **Atenție.** *Pentru n = 100 rezultatul este 1, nu 001: zerourile ajunse în față nu se scriu într-un număr. Pentru n = 0 rezultatul este 0, fiindcă bucla nu rulează niciodată.*

SOLUȚIA TA · PYTHON

★

PERFORMANȚĂ · SCRIE CODUL

Rădăcina digitală

SCRIERE COD

Se citește un număr natural n. Scrie două subprograme: suma_cifrelor(n), care întoarce suma cifrelor, și radacina(n), care aplică repetat suma cifrelor până când rămâne un număr de o singură cifră și îl întoarce. Al doilea subprogram trebuie să-l apeleze pe primul. Afișează pe același rând numărul citit, un spațiu, apoi rădăcina lui digitală. Ex.: pentru n = 12345: suma cifrelor este 15, care are încă două cifre, deci se reia — suma cifrelor lui 15 este 6, care are o singură cifră. Se afișează 12345 6.

HINT *Condiția de oprire se scrie while n > 9: un număr este de o singură cifră exact atunci când nu depășește 9. Pentru n = 0 nu se face niciun pas, iar rezultatul e chiar 0.*

HINT **Curiozitate.** *Pentru orice număr natural* **nenul** *, rădăcina digitală este restul împărțirii lui la 9 — cu excepția multiplilor lui 9, care dau 9. Numărul 0 face excepție: rădăcina lui digitală este chiar 0, așa cum ai văzut mai sus. Poți verifica singur pe câteva exemple, dar cerința se rezolvă cu bucla, nu cu formula.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · TRUSA DE INSTRUMENTE (2)

5 Instrumente chemate de zeci de *ori*

Un instrument bun se recunoaște după cât de des îl folosești fără să te gândești la el. Verificarea primalității și CMMDC-ul sunt exact așa: le scrii o dată și le chemi apoi în buclă, de sute de ori.

ESTE PRIM

Algoritmul e cel de la lecția despre divizori: încerci divizorii de la 2 în sus, dar te oprești când `d * d` depășește `n`. Împachetat, câștigă ceva ce înainte nu putea avea — mai multe ieșiri:

De îndată ce a găsit un divizor, răspunsul e cunoscut și subprogramul se oprește — nu mai are rost să caute. Instrumentul întoarce 1 pentru „da” și 0 pentru „nu”, deci se folosește într-un `if` scris `if este_prim(x) == 1:`.

Condiția `d * d <= n` nu e o șmecherie: dacă `n` are un divizor mai mare decât rădăcina lui, atunci are obligatoriu și unul mai mic, care vine la pereche cu el — și pe acela l-am fi găsit deja. Ce s-ar întâmpla dacă am încerca, în schimb, toți divizorii posibili până sub `n`? Pentru n = 97, bucla cu `d < n` face 95 de pași, iar cea cu `d * d <= n` face doar 8 — de aproape douăsprezece ori mai puțini. Diferența crește odată cu numărul: la un număr de nouă cifre, prima variantă face aproape un miliard de pași, a doua câteva zeci de mii.

ATENȚIE LA UN CARACTER

Condiția de contrast trebuie scrisă `d < n`, nu `d <= n`. Dacă `d` ajunge să fie egal cu `n`, atunci `n % d` este 0 și subprogramul ar întoarce 0 — adică **niciun** număr n-ar mai fi declarat prim, nici măcar 2. Un algoritm poate fi făcut lent fără să fie stricat, dar aici linia dintre „lent” și „greșit” e de un singur caracter. Merită încercat în editor: schimbă `d * d <= n` în `d <= n` și vezi ce răspunde subprogramul pentru 7.

DE CE 0 ȘI 1, ȘI NU TRUE ȘI FALSE?

Am ales să întoarcem numere pentru că un număr se poate aduna și număra — la exercițiul de mai jos vei aduna direct rezultatele ca să numeri primele dintr-un interval. Python are și valorile `True` și `False` pentru răspunsuri de tip da/nu, iar în programele scrise de profesioniști le vei vedea des în locul lui 1 și 0. Sunt două convenții pentru același lucru; important e să fii consecvent în cadrul aceluiași program.

CMMDC (EUCLID)

Împărțim repetat și păstrăm restul, până când restul devine zero; ultimul rest nenul e răspunsul. Algoritmul e cel de la lecția despre CMMDC, neschimbat. Înainte de Python, iată-l scris în pseudocod, în forma cerută de programă: **Pseudocodul nu are `return`.** În convențiile din programă, rezultatul se anunță o singură dată, în antet, după cuvântul *returnează*; nu există o instrucțiune separată de retur în corp, așa cum este `return`-ul din Python. Când subprogramul se termină, valoarea anunțată în antet este cea care pleacă înapoi la apel. Este una dintre puținele locuri unde pseudocodul și Python nu se suprapun rând cu rând — merită ținut minte pentru lucrările în care ți se cere pseudocod. În Python, același algoritm îl poți scrie acum în două feluri: În stânga, forma pe care o ai deja în caiet — și singura care se poate transcrie ca atare în pseudocod, unde fiecare atribuire are un singur nume în stânga săgeții. În dreapta, aceeași buclă scrisă cu atribuirea simultană de la secțiunea trecută: patru rânduri în loc de șase. Fac exact același lucru și dau același rezultat; în editor le poți folosi pe amândouă. Când ți se cere însă **pseudocodul**, scrii varianta din stânga.

Pentru 48 și 18: 48 % 18 = 12, apoi 18 % 12 = 6, apoi 12 % 6 = 0 — deci CMMDC este 6.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

> Numerele prime mari sunt cheia criptării cu care se protejează plățile online. Lacătul din bara de adrese înseamnă că datele tale circulă criptate cu chei construite din două numere prime uriașe, de câteva sute de cifre fiecare. Ele se generează o singură dată, la crearea cheii, cu teste de primalitate mult mai iscusite decât al nostru: la asemenea numere, căutarea divizorilor unul câte unul n-ar termina niciodată, nici dacă ar fi pornit odată cu universul. CMMDC-ul apare când simplifici o fracție, dar și când o aplicație de fotografii calculează raportul unei imagini de 1920 × 1080 și îți spune că e 16:9. Iar în muzică, două ritmuri suprapuse se sincronizează exact la cel mai mic multiplu comun al lungimilor lor.

A

EXERCIȚIUL A · GRILĂ

Cum funcționează instrumentele

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. De ce bucla din `este_prim` merge cu `d * d <= n` și nu cu `d <= n`?

   - `Pentru că divizorii unui număr sunt întotdeauna mai mici decât rădăcina lui pătrată`
   - `Pentru că altfel numărul n ar fi găsit ca divizor al lui însuși și rezultatul ar fi greșit`
   - `Pentru că înmulțirea se calculează mai repede decât comparația simplă între două numere`
   - `Pentru că un divizor mai mare decât rădăcina lui n are pereche unul mai mic, găsit deja — deci pașii se reduc mult`

2. Ce întoarce apelul `este_prim(1)`, pentru subprogramul din teorie?

   - `0`
   - `1, pentru că 1 se împarte exact doar la el însuși`
   - `None, pentru că bucla while nu se execută niciun pas`
   - `Eroare, pentru că d * d <= n este falsă de la bun început`

3. Cât este `cmmdc(48, 18)`?

   - `2`
   - `6`
   - `3, pentru că este cel mai mare divizor prim comun al celor două numere`
   - `18, pentru că 18 este cel mai mic dintre cele două numere date`

B

EXERCIȚIUL B · DRY-RUN

Euclid pas cu pas

DRY-RUN

Urmărește cum se schimbă a și b la fiecare rotire a buclei lui Euclid — aceeași buclă care stă în corpul lui cmmdc, scrisă aici fără antet, ca să vezi doar mecanismul. Numărul pasului îți este dat; tu completezi valorile lui a și b la începutul pasului și restul a % b care se calculează acolo. Primul rând

îți este dat ca exemplu, iar ultimul rând, cel pe care bucla se oprește, îți este dat complet și nu se completează: acolo b este zero, deci restul nici nu s-ar putea calcula. Valorile pe care le completezi sunt numerice.

```
a = 48
b = 18
while b != 0:
    a, b = b, a % b
print(a)
```

Pasul (dat)	a	b	a % b
1	48	18	12
2			
3			
4 — bucla se oprește	6	0	—

HINT *La fiecare pas, noul a este vechiul b, iar noul b este restul a % b calculat cu valorile vechi.*

C

EXERCIȚIUL C · SCRIE CODUL

Câte numere prime sunt într-un interval

SCRIERE COD

Se citesc două numere naturale a și b, fiecare de pe câte un rând, cu a mai mic sau egal cu b. Scrie un subprogram este_prim(n) care întoarce 1 dacă n este prim și 0 în caz contrar, apoi folosește-l ca să afișezi câte numere prime sunt în intervalul de la a la b, capetele incluse. Ex.: pentru a = 1 și b = 10, numerele prime sunt 2, 3, 5 și 7, deci se afișează 4.

HINT **Atenție.** *Numerele 0 și 1 nu sunt prime, iar subprogramul trebuie să le respingă explicit — pentru ele bucla while nu face niciun pas, deci fără verificarea n < 2 ar fi declarate prime.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 6 · UN SUBPROGRAM PENTRU O LISTĂ

6 Un instrument care primește o *listă*

Până acum instrumentele noastre primeau numere. Dar un parametru poate fi la fel de bine o listă întreagă — și atunci un singur apel prelucrează toate valorile deodată.

Un parametru nu e obligat să fie un număr. `def minim_maxim(v):` primește o listă întreagă, o parcurge cu un indice și întoarce două valori: cea mai mică și cea mai mare dintre elemente.

```
def minim_maxim(v):
    mi = v[0]
    ma = v[0]
    i = 1
    while i < len(v):
        if v[i] < mi:
            mi = v[i]
        if v[i] > ma:
            ma = v[i]
        i = i + 1
    return mi, ma
mi, ma = minim_maxim([5, 9, 2])
print(mi, ma)
```

Programul afișează `2 9`.

Observă că `minim_maxim` doar **citește** lista: n-adaugă nimic, nu șterge, nu sortează. Un subprogram care primește o listă și o modifică pe loc este un instrument de alt fel — pe acela îl vom întâlni la lecția despre metodele listei.

ATENȚIE

Cele două variabile pornesc de la **primul element al listei**, nu de la zero. Dacă ai porni minimul de la 0, orice listă cu numere pozitive ar avea minimul 0 — o valoare care nici măcar nu se află în listă. Din același motiv indicele pornește de la 1: primul element a fost deja luat în calcul.

DE REȚINUT

Python are gata făcute subprogramele `min` și `max`, dar le vom folosi abia de la lecția următoare. Faptul că le putem scrie singuri, în zece rânduri, nu e o pierdere de timp: aceeași idee stă la baza lor — deși varianta din Python e scrisă într-un limbaj mai apropiat de mașină și funcționează pe orice fel de colecție. Cine le-a scris o dată nu le mai confundă niciodată.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Aplicația de vreme îți arată minima și maxima zilei dintr-o singură parcurgere a temperaturilor măsurate din oră în oră. Catalogul electronic calculează, dintr-o singură trecere prin note, cea mai mică și cea mai mare notă a clasei. Iar un magazin online care îți arată „prețuri între 40 și 260 lei” a chemat exact un astfel de subprogram peste lista produselor filtrate.

A

EXERCIȚIUL A · ORDONARE

Cum se împachetează un algoritm cunoscut

ORDONARE

Numerotează pașii de la 1 la 5, în ordinea în care procedezi când transformi un algoritm deja scris într-un subprogram reutilizabil. Pașii îți apar amestecați.

- [ ] Copiem în corp bucla deja scrisă, indentată cu un nivel în plus
- [ ] Înlocuim afișarea finală cu return, ca rezultatul să plece înapoi la apel
- [ ] În programul principal citim datele și apelăm subprogramul, o dată sau de mai multe ori
- [ ] Alegem un nume care spune, în unul-două cuvinte, ce face algoritmul
- [ ] Scriem antetul cu def și punem între paranteze datele de care algoritmul are nevoie

B

EXERCIȚIUL B · SCRIE CODUL

Minimul și maximul, dintr-un singur apel

SCRIERE COD

Se citește un număr natural n, apoi n numere naturale, fiecare de pe câte un rând. Scrie un subprogram minim_maxim(v) care primește lista și întoarce două valori — cel mai mic și cel mai mare element. Afișează-le pe același rând, despărțite printr-un spațiu, în ordinea minim, maxim. Ex.: pentru n = 3 și valorile 5, 9, 2, subprogramul întoarce perechea 2 și 9, deci se afișează 2 9.

HINT **Atenție.** *Nu folosi min și max — le studiem la lecția următoare. Pornește ambele variabile de la primul element al listei, nu de la 0.*

HINT *n este cel puțin 1, deci lista nu e niciodată goală. Dacă toate valorile sunt egale, minimul și maximul sunt egale și se afișează de două ori aceeași valoare.*

SOLUȚIA TA · PYTHON

★

PERFORMANȚĂ · SCRIE CODUL

Ce se întâmplă cu o listă goală

SCRIERE COD

Se citește un număr natural n, care de data aceasta poate fi și 0, apoi n numere naturale, fiecare de pe câte un rând. Scrie un subprogram minim_maxim(v) care întoarce minimul și maximul listei, dar care nu se oprește cu eroare atunci când lista e goală: în acest caz întoarce valorile -1 și -1, iar

programul principal afișează LISTA GOALA. Ex.: pentru n = 0 se afișează LISTA GOALA. Pentru n = 3 și valorile 5, 9, 2 se afișează 2 9.

HINT *Instrumentele bune nu se strică pe intrări neobișnuite. Un subprogram care primește o listă goală și încearcă totuși să citească v[0] se oprește cu eroare de executare — iar într-o aplicație reală asta înseamnă un program care „cade” în fața utilizatorului. Un instrument bine făcut verifică întâi dacă are pe ce lucra.*

HINT **Curiozitate.** *Convenția „-1 și -1” funcționează aici doar fiindcă valorile citite sunt numere naturale, deci -1 nu poate fi niciodată un rezultat adevărat. Dacă lista ar putea conține și numere negative, convenția s-ar strica — și tocmai de aceea limbajele moderne au mecanisme speciale pentru „nu există răspuns”.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 7 · COD GHIDAT

7 Scrie cod care *trece testele*

Șase probleme. Scrie soluția pe hârtie, urmărește-o mental pe exemplul dat și abia apoi tastează-o pe platformă, unde e verificată automat pe 5 cazuri de test.

1

PROBLEMA 7.1 · COD GHIDAT

Trusa la lucru: două numere

SCRIERE COD

Se citesc două numere naturale a și b, fiecare de pe câte un rând. Scrie un subprogram suma_cifrelor(n) care întoarce suma cifrelor numărului primit și afișează, pe același rând și despărțite printr-un spațiu, suma cifrelor lui a și suma cifrelor lui b. Subprogramul se scrie o singură dată și se apelează de două ori. Ex.: pentru a = 1234 și b = 7: suma cifrelor lui 1234 este 10, iar a lui 7 este 7, deci se afișează 10 7.

HINT *Pentru 0, bucla nu face niciun pas, deci suma cifrelor iese 0 — ceea ce e corect.*

SOLUȚIA TA · PYTHON

2

PROBLEMA 7.2 · COD GHIDAT

Prim și în oglindă

SCRIERE COD

Se citește un număr natural n. Scrie două subprograme, este_prim(n) și oglindit(n), și afișează pe același rând numărul citit, un spațiu, apoi DA dacă atât n, cât și oglinditul lui sunt numere prime, sau NU în caz contrar. Ex.: pentru n = 13: 13 este prim, iar oglinditul lui, 31, este tot prim, deci se afișează 13 DA. Pentru n = 19: 19 este prim, dar oglinditul lui, 91, este 7 × 13, deci se afișează 19 NU.

HINT **Atenție.** *Numărul 1 nu este prim, deci pentru n = 1 răspunsul este NU. Cele două subprograme nu trebuie să se strice unul pe altul: fiecare lucrează pe propriul parametru, iar ce se schimbă înăuntru rămâne înăuntru.*

HINT **Curiozitate.** *Numerele prime al căror oglindit este un prim diferit se numesc emirp — cuvântul „prime” citit invers. 13, 17, 31 și 37 sunt exemple: fiecare are ca oglindit un alt număr prim. Numerele prime care coincid cu propriul oglindit (2, 3, 5, 7, 11, 101) se numesc prime palindromice; trec și ele testul din problemă, dar nu sunt emirp.*

SOLUȚIA TA · PYTHON

3 PROBLEMA 7.3 · COD GHIDAT

Amplitudinea unui șir de valori

SCRIERE COD

Se citește un număr natural n, apoi n numere naturale, fiecare de pe câte un rând. Scrie un subprogram minim_maxim(v) care întoarce cele două valori extreme ale listei și afișează diferența dintre cea mai mare și cea mai mică dintre ele. Ex.: pentru n = 3 și valorile 5, 9, 2: maximul este 9, minimul este 2, deci se afișează 7.

HINT *Dacă toate valorile sunt egale, diferența este 0. n este cel puțin 1.*

HINT **Atenție.** *Subprogramul întoarce două valori dintr-un singur return; nu-l scrie de două ori, o dată pentru minim și o dată pentru maxim.*

SOLUȚIA TA · PYTHON

4

PROBLEMA 7.4 · REPARĂ CODUL

Subprogramul care uită să răspundă

SCRIERE COD

Programul de mai jos ar trebui să citească un număr natural n și să afișeze câte cifre are. Bucla este corectă, dar programul afișează None pentru orice intrare — inclusiv None pentru 1234, în loc de 4. Repară-l și, în plus, fă-l să afișeze 1 pentru n = 0, fiindcă și zero se scrie cu o cifră. Ex.: pentru n = 1234 trebuie să se afișeze 4, iar pentru n = 0 trebuie să se afișeze 1.

Cod stricat:

```
def numar_cifre(n):
    c = 0
    while n > 0:
        c = c + 1
        n = n // 10
n = int(input())
print(numar_cifre(n))
```

HINT **Atenție.** *Codul de mai sus nu are nicio eroare de sintaxă și nu se oprește cu vreun mesaj: rulează până la capăt și afișează ceva. Este o eroare de logică — cel mai greu de găsit dintre cele trei feluri, fiindcă nimeni nu te avertizează. Teoria de la secțiunea a treia spune că un subprogram fără return întoarce None; asta e exact ce*

se întâmplă aici. Al doilea defect e mai discret: pentru n = 0 bucla nu face niciun pas, deci contorul rămâne 0; tratează cazul separat, înainte de buclă.

PROGRAMUL CORECTAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Două instrumente într-un singur rând

SCRIERE COD

Se citește un număr natural n, mai mare sau egal cu 1. Scrie două subprograme, suma_cifrelor(n) și cmmdc(a, b), și afișează pe același rând numărul citit, un spațiu, apoi cel mai mare divizor comun dintre n și suma cifrelor sale. Rezultatul trebuie obținut dintr-un singur apel al lui cmmdc, care primește direct rezultatul celuilalt subprogram. Ex.: pentru n = 1234: suma cifrelor este 10, iar cel mai mare divizor comun dintre 1234 și 10 este 2, deci se afișează 1234 2.

HINT *Apelul se scrie într-un singur rând: cmmdc(n, suma_cifrelor(n)). Se calculează întâi apelul din interior, apoi cel din exterior — exact ca la dublu(dublu(3)) din prima secțiune.*

HINT **Curiozitate.** *Numerele pentru care rezultatul este chiar suma cifrelor se numesc numere Harshad: 18 este unul dintre ele, fiindcă se împarte exact la 9.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.6 · PROVOCARE

Numere Harshad dintr-un interval

SCRIERE COD

Un număr natural nenul se numește Harshad dacă se împarte exact la suma cifrelor sale — de exemplu 18, fiindcă 1 + 8 = 9, iar 18 % 9 este 0. Se citesc două numere naturale a și b, cu a mai mic sau egal cu b. Scrie două subprograme: suma_cifrelor(n), care întoarce suma cifrelor, și este_harshad(n), care întoarce 1 dacă n este Harshad și 0 în caz contrar. Al doilea trebuie să-l apeleze pe primul. Afișează câte numere Harshad se află în intervalul de la a la b, capetele incluse. Ex.: pentru a = 20 și b = 30, numerele Harshad sunt 20, 21, 24, 27 și 30, deci se afișează 5.

HINT **Atenție.** *Numărul 0* **nu** *este Harshad, iar tratarea lui nu este opțională: suma cifrelor lui 0 este 0, iar o împărțire la 0 oprește programul cu eroare de executare. Verifică n* **înainte** *de a face împărțirea — exact ca la lista goală din secțiunea a șasea.*

HINT **Curiozitate.** *Numele vine din sanscrită: harșa („bucurie”) + da („dătător”). Primele nouă numere naturale nenule sunt toate Harshad, fiindcă fiecare este egal cu suma cifrelor sale; abia mai departe apar primele excepții.*

SOLUȚIA TA · PYTHON