test-e.ro

CAIET ELEV

Lecția 3

Structuri repetitive (while · repetă)

A treia structură fundamentală: repetiția. Bucla cu test inițial (while), bucla cu test final (repetă … până când), instrucțiunile break și continue, și echivalența dintre cele două forme.

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

NUMELE ELEVULUI .....

CUM FOLOSEȘTI CAIETUL

Acest caiet adună exercițiile scrise din Lecția 3. Rezolvă-le pe hârtie — depanezi pseudocod, prezici ce afișează o buclă, desenezi scheme logice, ordonezi instrucțiuni și scrii programe. Folosește doar ce s-a predat: **while** și **repetă** (fără **for** / **range()**, fără funcții). Spațiile punctate și liniate sunt pentru scrisul tău; verificarea automată rămâne pe platformă.

CUPRINS

1 Recapitulare: secvențial și decizie .....
2 Structura while („cât timp") .....
3 Structura repetă … până când .....
4 while în practică · contor .....
5 break și continue .....
6 Echivalența while ↔ repetă .....
7 Verifică-ți codul .....

SECȚIUNEA 1 · RECAPITULARE

1 Ce știm deja: *secvențial* și *decizie*

Înainte de a învăța repetiția, ne asigurăm că primele două structuri sunt solide: **secvențială** (instrucțiuni una după alta) și **de decizie** (`if … elif … else`). Reține operatorii pe cifre: `n % 10` dă ultima cifră, iar `n // 10` elimină ultima cifră.

A EXERCIȚIUL A · RECAPITULARE

Ce-ți mai aduci aminte?

GRILĂ COMPLETARE

Încercuiește `True` / `False` sau completează pe linia punctată.

1. `27 % 10` are valoarea _____
2. `27 // 10` are valoarea _____
3. Într-un `if … elif … else` se execută cel mult o singură ramură.

True False

4. Un număr este par dacă `n % 2` este egal cu _____

B EXERCIȚIUL B · DRY-RUN

Mini-calculator cu o decizie

DRY-RUN

Algoritmul citește două numere `x`, `y` și un operator `op` (`+`, `-` sau `*`). Execută-l în minte pentru fiecare rând și scrie ce afișează.

```
citește x, y, op
dacă op = "+" atunci
    scrie x + y
altfel dacă op = "-" atunci
    scrie x - y
altfel
    scrie x * y
```

x	y	op	Ce afișează?
6	4	+	
9	2	-	
5	3	*	

C

SCRIE SINGUR · DOAR CU DECIZII

Câte cifre are numărul?

SCRIERE COD

Se citește un număr natural `n` de **cel mult 5 cifre**. Afișează câte cifre are, folosind **doar** `if … elif … else` (fără buclă — repetiția vine în secțiunea următoare).

HINT *compară `n` cu pragurile `10`, `100`, `1000`, `10000`: mai mic decât 10 → o cifră, mai mic decât 100 → două cifre ș.a.m.d.*

SOLUȚIA TA · PYTHON

2 SECȚIUNEA 2 · STRUCTURA WHILE

Structura *while* („cât timp")

Bucla `while` repetă un bloc de instrucțiuni **cât timp** o condiție rămâne adevărată. Condiția se verifică **înainte** de fiecare execuție a corpului (test inițial) — de aceea corpul poate rula și de **zero** ori.

CELE PATRU COMPONENTE ALE UNEI BUCLE

1. **Inițializarea** — pregătești variabila de control înainte de buclă (`i = 1`).
2. **Condiția** — expresia logică testată înainte de fiecare repetare (`i <= n`).
3. **Corpul** — instrucțiunile care se repetă.
4. **Pasul (modificare)** — modifică variabila de control ca, în cele din urmă, condiția să devină falsă (`i = i + 1`). Fără el, bucla rulează la nesfârșit.

A

EXERCIȚIUL A · REPARARE

Pseudocodul care „uită" să repete

DEPANARE

Pseudocodul de mai jos ar trebui să afișeze numerele de la `1` la `n`. Din cauza unei erori, se comportă greșit (buclă infinită). Găsește linia lipsă / greșită și rescrie corect algoritmul.

```
citește n
i ← 1
cât timp i ≤ n execută
    scrie i
```

CE LIPSEȘTE / CUM SE CORECTEAZĂ

B

EXERCIȚIUL B · DRY-RUN

Execută bucla în minte

DRY-RUN

Bucla calculează suma `1 + 2 + … + n`. Pentru fiecare `n`, scrie ce afișează la final. Atenție la cazul-limită `n = 0`!

```
i = 1
s = 0
while i <= n:
    s = s + i
```

```
    i = i + 1
print(s)
```

n	Ce afișează?
0	
1	
4	
5	

C

EXERCIȚIUL C · CONCEPEREA SCHEMEI

Construiește schema buclei while

SCHEMĂ LOGICĂ

Traduce pseudocodul într-o schemă logică. Rombul `i ≤ n ?` are **două ieșiri**: pe **DA** se execută corpul, apoi fluxul revine la romb (bucla); pe **NU** se merge la STOP. Folosește: **oval** pentru START/STOP, **paralelogram** pentru citire/scriere, **dreptunghi** pentru calcul, **romb** pentru condiție.

```
citește n
i ← 1
cât timp i ≤ n execută
    scrie i
    i ← i + 1
```

DESENEAZĂ SCHEMA AICI · NU UITA RAMURILE DA / NU ȘI ÎNTOARCEREA LA ROMB

D

EXERCIȚIUL D · COMPLETEAZĂ COD

Completează pasul care lipsește

COMPLETARE

Programul ar trebui să afișeze numerele **pare** de la `2` la `n` (`2, 4, 6, …`). Lipsește **pasul (modificare)**. Completează-l ca bucla să avanseze din 2 în 2.

```
i = 2
while i <= n:
    print(i)

    ....................
```

LINIA CARE LIPSEȘTE

SECȚIUNEA 3 · STRUCTURA REPETĂ

3 Structura *repetă … până când*

La `while` condiția se verifică **înainte** de corp. La **repetă … până când** ordinea e inversă: **execută întâi** corpul, apoi verifică o **condiție de oprire** — de aceea corpul rulează **cel puțin o dată** (test final). Python nu are `repetă`, așa că o simulăm cu `while`.

PSEUDOCOD · REPETĂ

```
repetă
    citește x
până când 1 ≤ x ≤ 100
```

PYTHON · WHILE CU CITIRE ÎN PLUS

```
x = int(input())
while not (1 <= x <= 100):
    x = int(input())
```

HINT *condiția de oprire („repetă") este exact* **negarea** *condiției de continuare („while"): te oprești când* `1 ≤ x ≤ 100` ⟺ *continui* `while not (1 <= x <= 100)`.

A

EXERCIȚIUL A · DRY-RUN

De câte ori rulează corpul?

DRY-RUN

Algoritmul citește numere **până când** citește un număr valid (între 1 și 100), apoi se oprește. Pentru fiecare șir de numere introduse (în ordine), scrie de **câte ori** se execută corpul (de câte ori se citește). Atenție când primul număr e deja valid!

```
repetă
    citește x
până când 1 ≤ x ≤ 100
```

Numere citite (în ordine)	Câte execuții?
50	
-3, 200, 7	
0, 0, 100	

B

EXERCIȚIUL B · ORDONARE

Ordonează pseudocodul

ORDONARE

Instrucțiunile de mai jos formează, amestecate, un algoritm cu **repetă … până când** care citește un număr valid (între 1 și 100). Numerotează-le de la **1** la **4** în ordinea corectă. O instrucțiune este o **capcană** (nu face parte din algoritm) — taie-o.

- `până când 1 ≤ x ≤ 100`
- `citește x`
- `repetă`
- `scrie x`
- `cât timp x < 0 execută` ← *capcană?*

C

EXERCIȚIUL C · COMPLETEAZĂ COD

Completează condiția de oprire

COMPLETARE

Programul (varianta cu `break`) citește un număr **până când** utilizatorul introduce un număr **pozitiv** (strict mai mare ca 0). Completează condiția de oprire din `if`.

```
while True:
    x = int(input())
    if            :
        break
```

CONDIȚIA CARE LIPSEȘTE (DUPĂ IF)

SECȚIUNEA 4 · WHILE ÎN PRACTICĂ

4 *while* în practică: aplicații cu contor

Punem bucla la treabă pe probleme uzuale, toate cu un **contor** sau o variabilă care se schimbă la fiecare pas. Studiază exemplul comentat, apoi rezolvă.

EXEMPLU COMENTAT · NUMĂRAREA CIFRELOR

```
n = int(input())
nr = 0
while n > 0:          # cât timp mai am cifre
    nr = nr + 1       # am găsit încă o cifră
    n = n // 10       # elimin ultima cifră
print(nr)
```

Trucul `% 10` (ultima cifră) și `// 10` (elimină ultima cifră) revine des în problemele cu numere. Atenție: algoritmul e corect doar pentru `n > 0`.

A

EXERCIȚIUL A · COMPLETEAZĂ COD

Suma cifrelor unui număr

COMPLETARE

Programul calculează **suma cifrelor** lui `n`. Adună deja ultima cifră, dar lipsește linia care **elimină** ultima cifră — fără ea, bucla nu se mai termină. Completeaz-o!

```
n = int(input())
s = 0
while n > 0:
    s = s + n % 10

    ..................
print(s)
```

LINIA CARE LIPSEȘTE

B

EXERCIȚIUL B · DRY-RUN

Câte cifre are numărul?

DRY-RUN

Folosind bucla din exemplul comentat, execută în minte pentru fiecare valoare (toate `> 0`) și scrie ce afișează la final.

n	Ce afișează?
7	
58	
2473	
90000	

C

EXERCIȚIUL C · CONCEPEREA SCHEMEI

Schema numărării cifrelor

SCHEMĂ LOGICĂ

Desenează schema logică a algoritmului care numără cifrele lui `n` (pentru `n > 0`). Rombul `n > 0 ?` are două ieșiri: pe **DA** numeri și elimini o cifră, apoi revii la romb; pe **NU** afișezi rezultatul și STOP.

```
citește n
nr ← 0
cât timp n > 0 execută
    nr ← nr + 1
    n ← n div 10
scrie nr
```

DESENEAZĂ SCHEMA · RAMURILE DA / NU

D

EXERCIȚIUL D · ORDONARE

Ordonează algoritmul

ORDONARE

Instrucțiunile formează, amestecate, un algoritm care calculează **suma numerelor de la 1 la n** cu `while`. Numerotează-le de la **1** la **5** în ordinea corectă. O instrucțiune este o capcană — taie-o.

```
i = i + 1
s = 0
while i <= n:
i = 1
s = s + i
n = n // 10   ← capcană?
print(s)
```

5

SECȚIUNEA 5 · BREAK ȘI CONTINUE

Traficul în buclă: *break* și *continue*

Două instrucțiuni care schimbă cursul unei bucle: `break` **oprește imediat** bucla (trece la instrucțiunea de după ea), iar `continue` **sare peste restul corpului** pentru repetarea curentă și trece la următoarea verificare a condiției.

BREAK — OPRIRE (AFIȘEAZĂ 1 2 3 4)

CONTINUE — SĂRI PESTE (1 2 3 4 6 … 10)

```python
i = 0
while i < 10:
    i = i + 1
    if i == 5:
        break
    print(i)
```

```python
i = 0
while i < 10:
    i = i + 1
    if i == 5:
        continue
    print(i)
```

ATENȚIE *dacă pasul (`i = i + 1`) e scris* **după** *un `continue`, el va fi sărit — și bucla poate deveni infinită. De aceea contorul crește* **înainte** *de eventualul `continue`.*

A

EXERCIȚIUL A · PREZICE OUTPUT-UL

Ce afișează bucla cu continue?

DRY-RUN

Bucla parcurge numerele de la 1 la 10, dar folosește `continue` pentru a sări peste multiplii de 3. Prezice exact ce afișează (numerele separate prin spațiu).

```python
i = 0
while i < 10:
    i = i + 1
    if i % 3 == 0:
        continue
    print(i, end=" ")
```

CE AFIȘEAZĂ PROGRAMUL

B

EXERCIȚIUL B · COMPLETEAZĂ COD

Oprire la prima potrivire

COMPLETARE

Programul caută **primul divizor** al lui `n` mai mare ca 1. De îndată ce îl găsește, trebuie să se oprească. Completează **instrucțiunea care oprește forțat bucla** chiar în acel punct. Nu modifica variabila de control ca să iasă bucla „de la sine" — se cere instrucțiunea dedicată.

```
n = int(input())
gasit = n
d = 2
while d < n:
    if n % d == 0:
        gasit = d

        ..................
    d = d + 1
print(gasit)
```

INSTRUCȚIUNEA CARE LIPSEȘTE

SECȚIUNEA 6 · WHILE ↔ REPETĂ

6 Aceeași logică, *două forme*

Aceeași problemă poate fi rezolvată cu oricare dintre cele două structuri — cu o singură rezervă: dacă bucla poate să nu ruleze deloc (0 repetări), doar `while` este corectă. Cheia conversiei: condiția de continuare (while) este **negarea** condiției de oprire (repetă).

WHILE (TEST INIȚIAL)

```
i ← 1
cât timp i ≤ n execută
    scrie i
    i ← i + 1
```

REPETĂ (TEST FINAL)

```
i ← 1
repetă
    scrie i
    i ← i + 1
până când i > n
```

DIFERENȚĂ SUBTILĂ

Dacă `n = 0`, varianta `while` nu afișează nimic (condiția e falsă din start), pe când `repetă` afișează totuși `1` (corpul rulează o dată înainte de verificare). Sunt echivalente doar când corpul trebuie oricum să ruleze cel puțin o dată.

A

EXERCIȚIUL A · CONVERSIE

Transformă while în repetă

GRILĂ

Bucla `while` de mai jos citește un număr `x` cât timp el este negativ. Bifează varianta `repetă … până când` **echivalentă**.

```
citește x
cât timp x < 0 execută
    citește x
```

a) `repetă citește x până când x < 0`

b) `repetă citește x până când x ≥ 0`

c) `repetă citește x până când x > 0`

HINT *„repetă" rulează corpul măcar o dată — la fel ca aici, unde `citește x` apare oricum înainte de test. Condiția de oprire e negarea lui `x < 0`.*

B

EXERCIȚIUL B · ALEGE STRUCTURA

Care structură se potrivește?

ASOCIERE

Pentru fiecare scenariu, scrie `while` sau `repetă`. Întreabă-te: corpul trebuie să ruleze **măcar o dată** (→ repetă) sau **se poate să nu ruleze deloc** (→ while)?

1. Ceri o parolă și o verifici; repeți până e corectă. _____
2. Procesezi mesajele necitite; dacă nu există niciunul, nu faci nimic. _____
3. Joci o rundă, apoi întrebi „mai joci?". _____

7 Verifică-ți *codul*

SECȚIUNEA 7 · VERIFICĂ-ȚI CODUL

Scrie soluția pe hârtie cu `while` (sau simularea lui `repetă`), apoi testeaz-o pe platformă. Gândește-te la cazurile-limită.

1

PROBLEMA 1

Suma cifrelor pare

SCRIERE COD

Citește un număr natural `n` (`n > 0`) și calculează **suma cifrelor pare** ale lui (0, 2, 4, 6, 8).
Exemplu: pentru `n = 2473` → `2 + 4 = 6`.

SOLUȚIA TA · PYTHON

2

PROBLEMA 2

Citire validată (repetă)

SCRIERE COD

Cere un număr și **repetă cererea până când** numărul este între 1 și 100 (inclusiv), apoi afișează-l. Simulează `repetă` cu tiparul „citire o dată, apoi `while` cât timp NU e valid".

SOLUȚIA TA · PYTHON

3

PROBLEMA 3 · PROVOCARE

Cel mai mare divizor mai mic decât n

SCRIERE COD

Citește un număr natural `n` (`n > 1`) și afișează **cel mai mare divizor al lui mai mic decât n**. Exemplu: `n = 30` → `15`; pentru un număr prim (`n = 7`) → `1`.

HINT *primul divizor `d ≥ 2` al lui `n` dă răspunsul `n // d`. Atenție la cazul „n este prim".*

SOLUȚIA TA · PYTHON