

# Programare 1

Cicluri, structuri  
Cursul 2, 3, 4

Despre ce am  
discutat în cursul  
precedent?

### Informații generale despre curs

- Despre cerințe și evaluare

### Elemente de bază despre Python

- Variabile și tipuri de date

### Operații matematice elementare

# Despre ce o să discutăm astăzi?

Procesul de dezvoltare software

Instrucțiuni repetitive

- For, while, break, continue

Tipuri de date predefinite

- List, tupluri, dictionare, set-uri

# Să ne gândim la următorul scenariu...

- Sunteți un informatician
- Vă mutați în SUA
- Dimineața ascultați știrile la radio
- Surpriză: auziți la radio că afară sunt 90 de grade Fahrenheit
- PROBLEMĂ! – ca să știți cum să vă îmbrăcați trebuie să înțelegeți ce înseamnă!
- ~~Soluția: – căutați pe Google!~~
- Soluția: scrieți un program care face asta!

Scriveți un  
program....

- Ce informații trebuie să îi furnizăm calculatorului ?
  - O valoare reprezentând valoarea în grade Fahrenheit
- Care este formula pentru conversie ?
  - Găsim formula de calcul:  $C = (F - 32) * 5 / 9$
- Ce trebuie să ne spună calculatorul?
  - O valoare reprezentând valoarea în grade Celsius

Vă aduceți aminte  
discuția privind  
cunoașterea ?

# ... programul ...

```
tempFahrenheit = int ( input ("Temperatura în grade Fahrenheit:"))  
tempCelsius = (tempFahrenheit - 32) * 5/9  
print("Temperatura în grade Celsius este: ", tempCelsius)
```

## TESTAȚI PROGRAMUL!

Testați programul pentru valori cunoscute și verificați dacă returnează ce trebuie!

Temperatura în grade Fahrenheit: 0

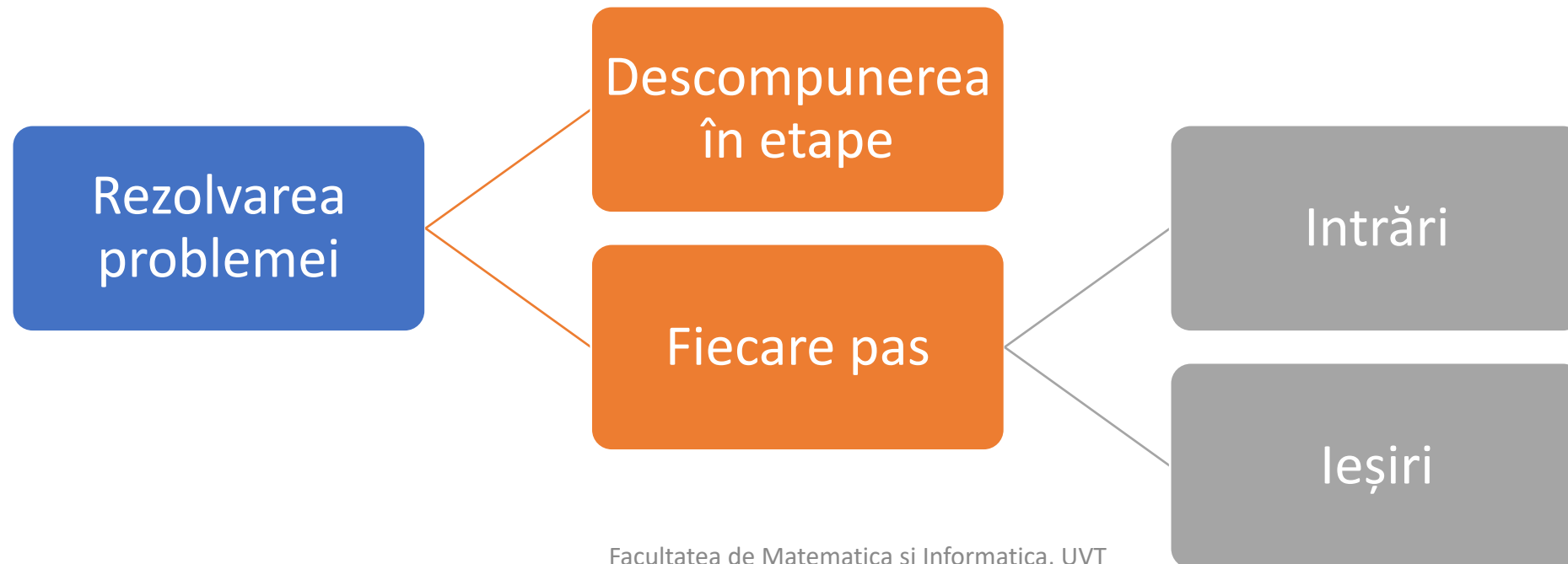
Temperatura în grade Celsius este: -17.77777777777778

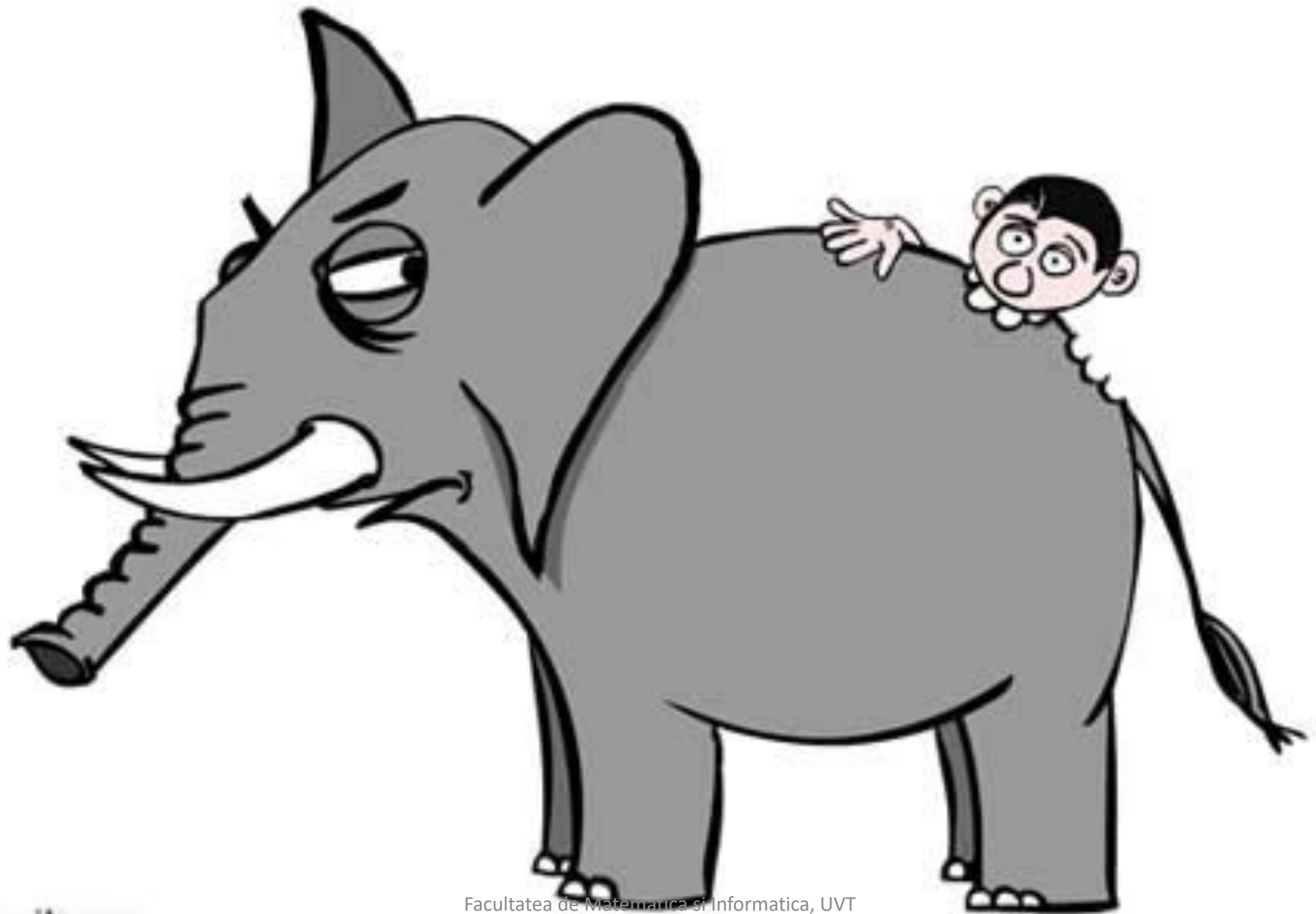
Temperatura în grade Fahrenheit: 100

Temperatura în grade Celsius este: 37.77777777777778

# Procesul de dezvoltare software

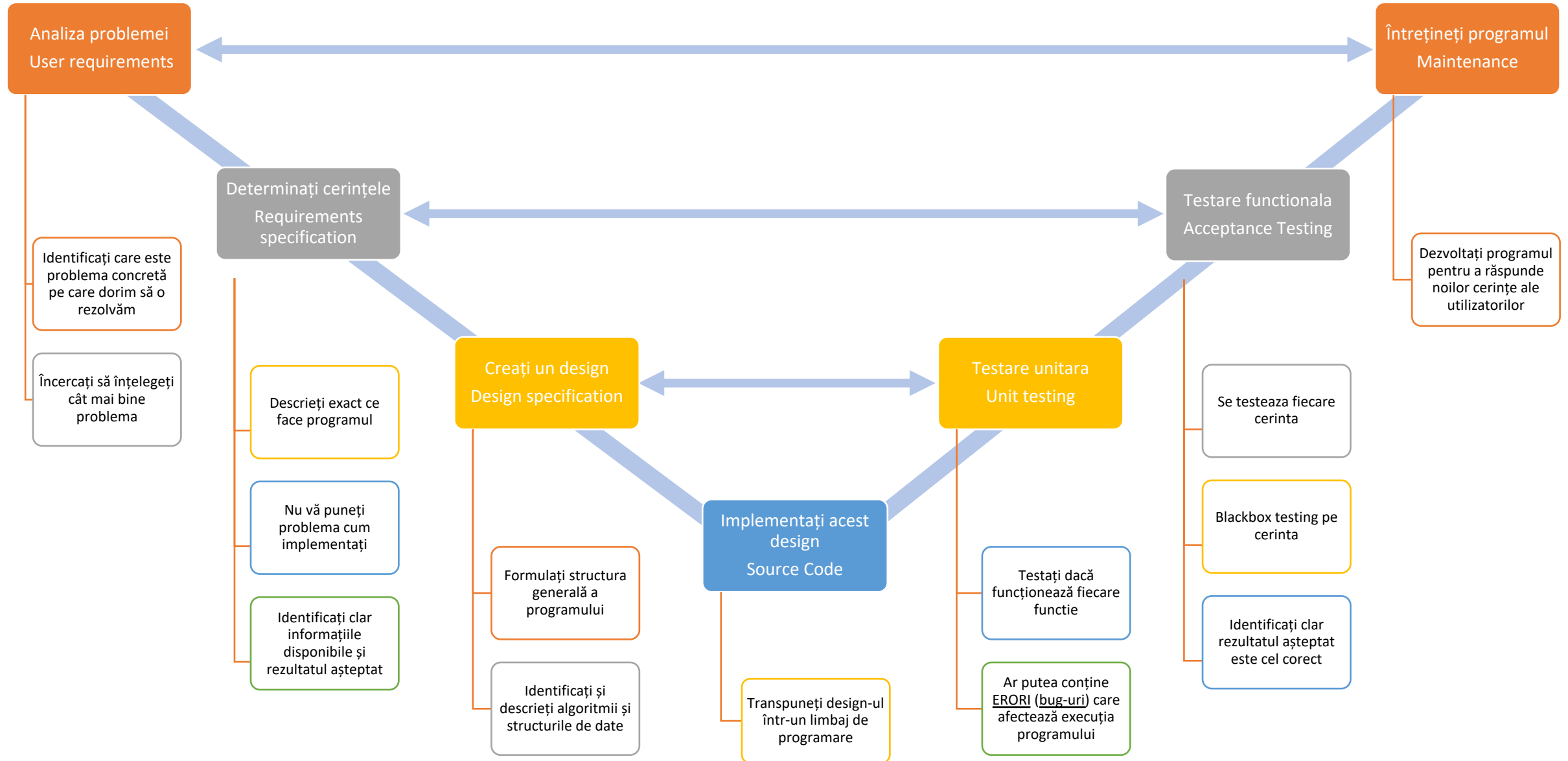
Vă aduceți aminte: un calculator nu acceptă ambiguitate, trebuie să îi spunem până în ultimul detaliu ce să facă

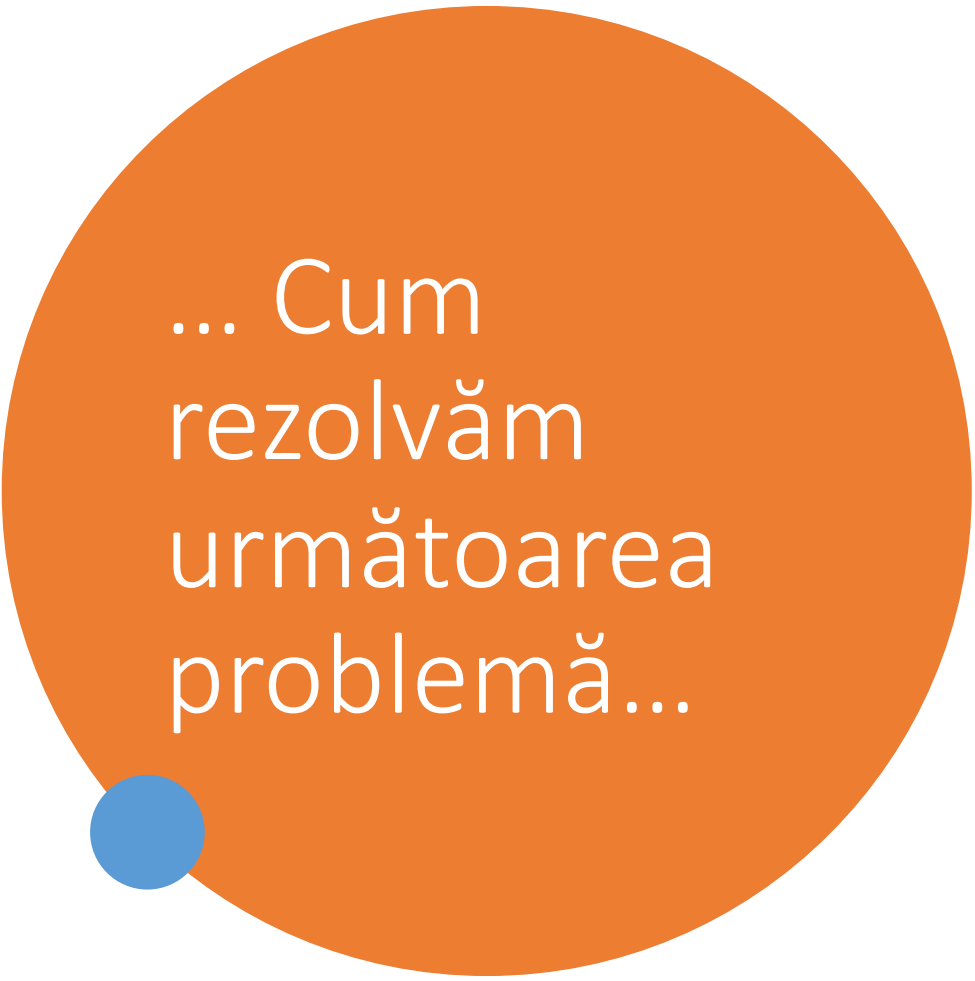






# Ce trebuie sa facă un programator?





... Cum  
rezolvăm  
următoarea  
problemă...

- Calculați suma:

$$S_n = \sum_{i=0}^n i = 1 + 2 + \dots + n$$

- Dacă  $n=2$ ?
- Dacă  $n=3$ ?
- Dacă  $n = 100$ ?

... Cum rezolvăm  
următoarea  
problemă...

- Calculați suma

$$S_n = \sum_{i=0}^n i = 1 + 2 + \dots + n$$

- Dacă rescriem formula astfel:

$$S_n = S_{n-1} + n$$

- Dar pentru  $S_i$ ?

$$S_i = S_{i-1} + i$$

- Algoritm ?

# ... Cum rezolvăm următoarea problemă...

- Calculați suma

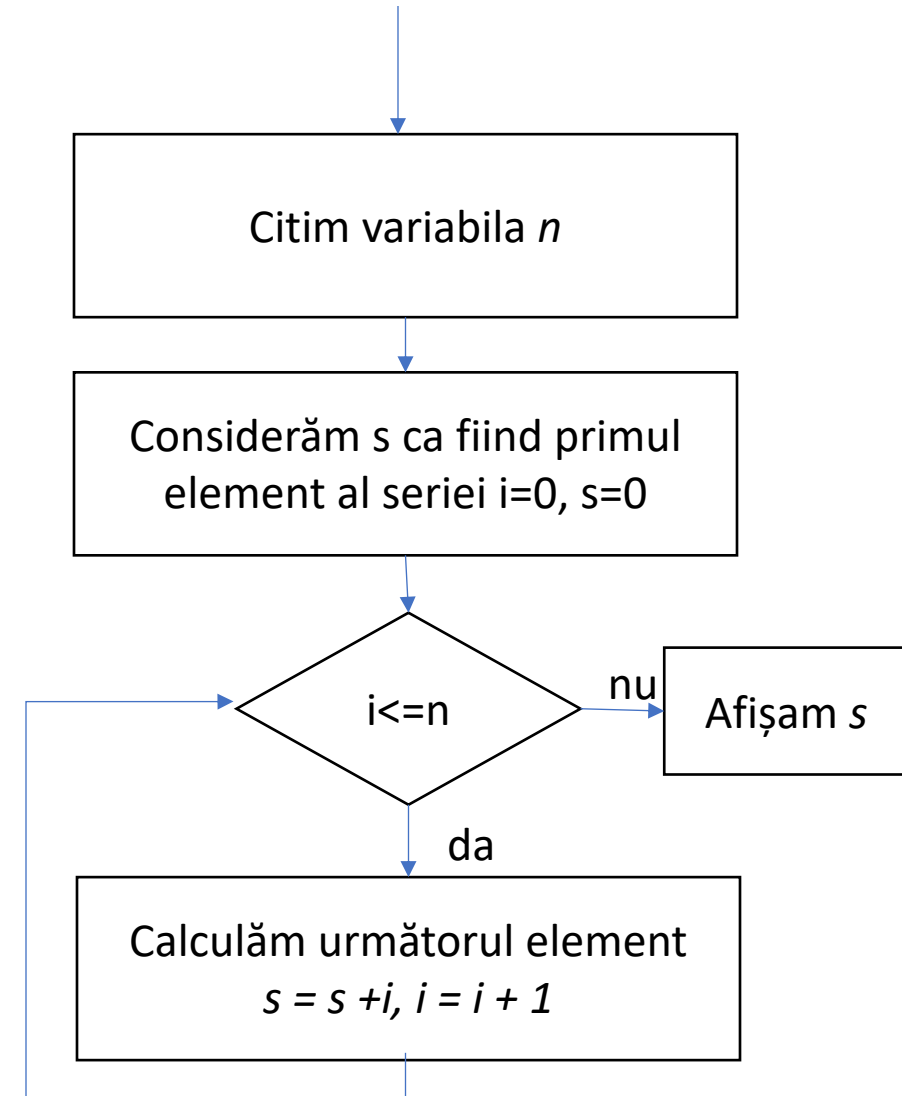
$$S_n = \sum_{i=0}^n i = 0 + 1 + 2 + \dots + n$$

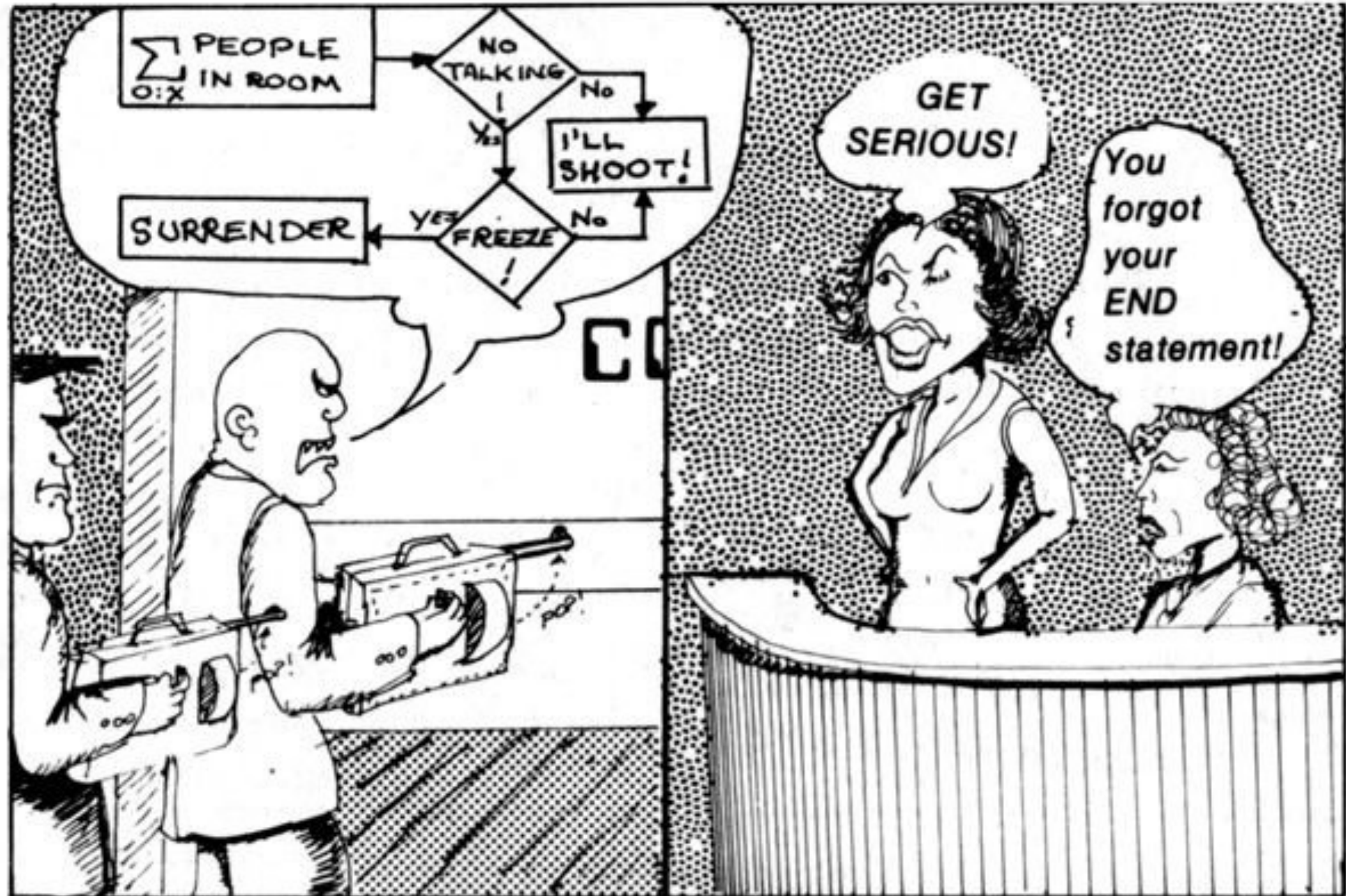
- Dar pentru  $S_i$ ?

$$S_i = S_{i-1} + i$$

## Algoritm ?

- 1) Citim o variabilă  $n$
- 2) Considerăm  $s$  ca fiind primul element al seriei ( $s=0$ ) și considerăm  $i=0$
- 3) Dacă  $i \leq n$  atunci
- 4) Calculăm următorul element al seriei  $s = s + i$
- 5) Incrementăm  $i$  ( $i=i+1$ )
- 6) Revenim la pasul 3





# Instrucțiuni repetitive

În majoritatea programelor instrucțiunile trebuie repetate de mai multe ori în funcție de o condiție (flow control / control al fluxului)

Instrucțiunile repetitive (ciclurile) în Python sunt **for** și **while**

Ciclurile sunt o structură de control care ne permite să repetăm un grup de pași dintr-un program

- Corpul ciclului reprezintă instrucțiunile care trebuie repetate

# Instrucțiunile repetitive - while

- Sintaxa

**while** <condiție> :

Instrucțiuni

[**else:**

Instrucțiuni]

- O expresie evaluată ca valoare logică (True, False)

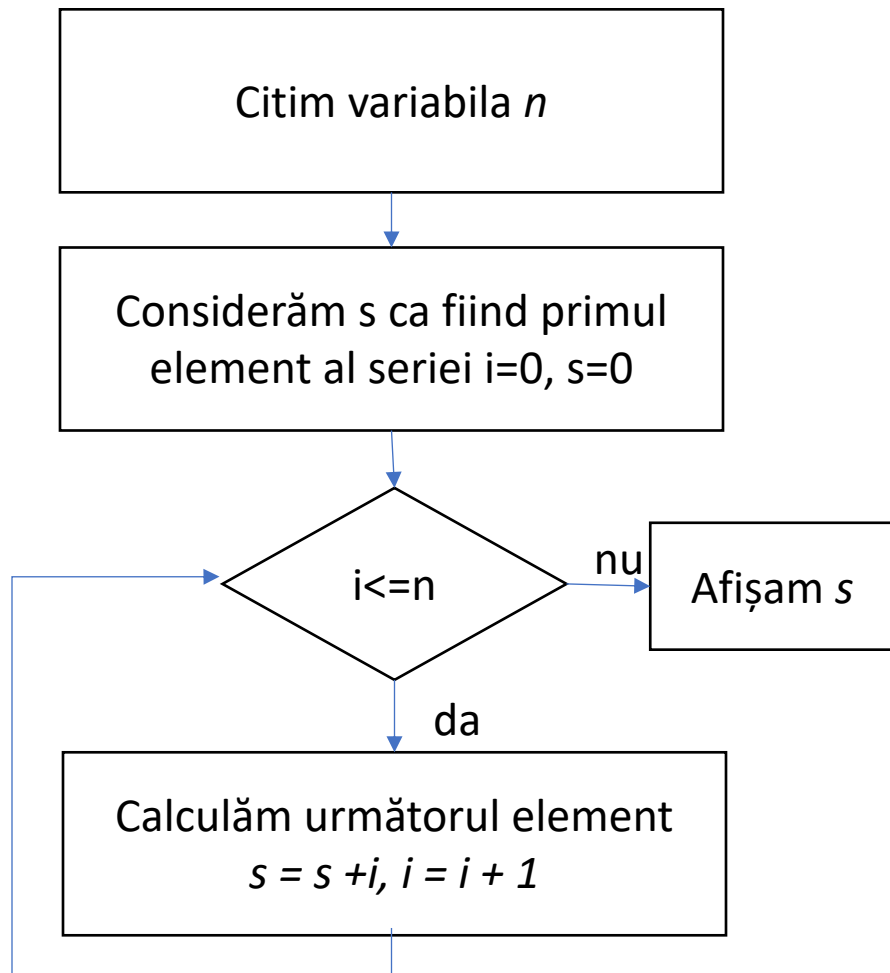
- Corpul ciclului, este executat atât timp cât <condiție> este adevărată (True)
- Poate fi formată din una sau mai multe instrucțiuni
- Toate instrucțiunile de după **while** trebuie să fie indentate la dreapta cu cel puțin un spațiu

Un ciclu este considerat ciclu infinit dacă <condiție> este tot timpul adevărat (True).

- Clauză opțională executată atunci când ciclul se termină. Este specifică Python și este executată în momentul terminării execuției ciclului (în mod natural)

# Instrucțiunile repetitive - while

**Calculați  $S_i = S_{i-1} + i$**



**Transpus în limbajul Python**

```
n = int(input("n="))
s = 0
i = 0
while i <= n:
    s = s + i
    i = i + 1
else:
    print("S=", s)
```



# Instrucțiunile repetitive – while –alte exemple

## Înmulțirea „A la russe”

Înmulțiți  
două  
numere x și  
y folosind  
următoarea  
metodă:

- Scrieți x și y pe aceeași linie
- Împărțiți x la 2 și scrieți câtul sub x
- Înmulțiți y cu 2 și scrieți rezultatul sub y
- Continuați cât timp x este diferit de 1
- Rezultatul înmulțirii este suma valorilor de pe coloana y care corespund numerelor impare de pe coloana x

Example

X = 13	Y = 25
13	25
6	50
3	100
1	200

Rezultat:  $x*y = 25 + 100 + 200 = 325$

Analiza problemei

# Instrucțiunile repetitive - while

X = 13	Y = 25
13	25
6	50
3	100
1	200

Rezultat:  $x*y = 25 + 100 + 200 = 325$

- Înmulțirea „A la russe”
  - Înmulțiți două numere x și y folosind următoarea metodă:
    - Scrieți x și y pe aceeași linie
    - Împărțiți x la 2 și scrieți câtul sub x
    - Înmulțiți y cu 2 și scrieți rezultatul sub y
    - Continuați cât timp x este diferit de 1
    - Rezultatul înmulțirii este suma valorilor de pe coloana y care corespund numerelor impare de pe coloana x

Să reformulăm:

*Pas1*: `result = 0`

*Pas2*: Dacă x este impar atunci `result = result + y`

*Pas3*: x devine `x/2`

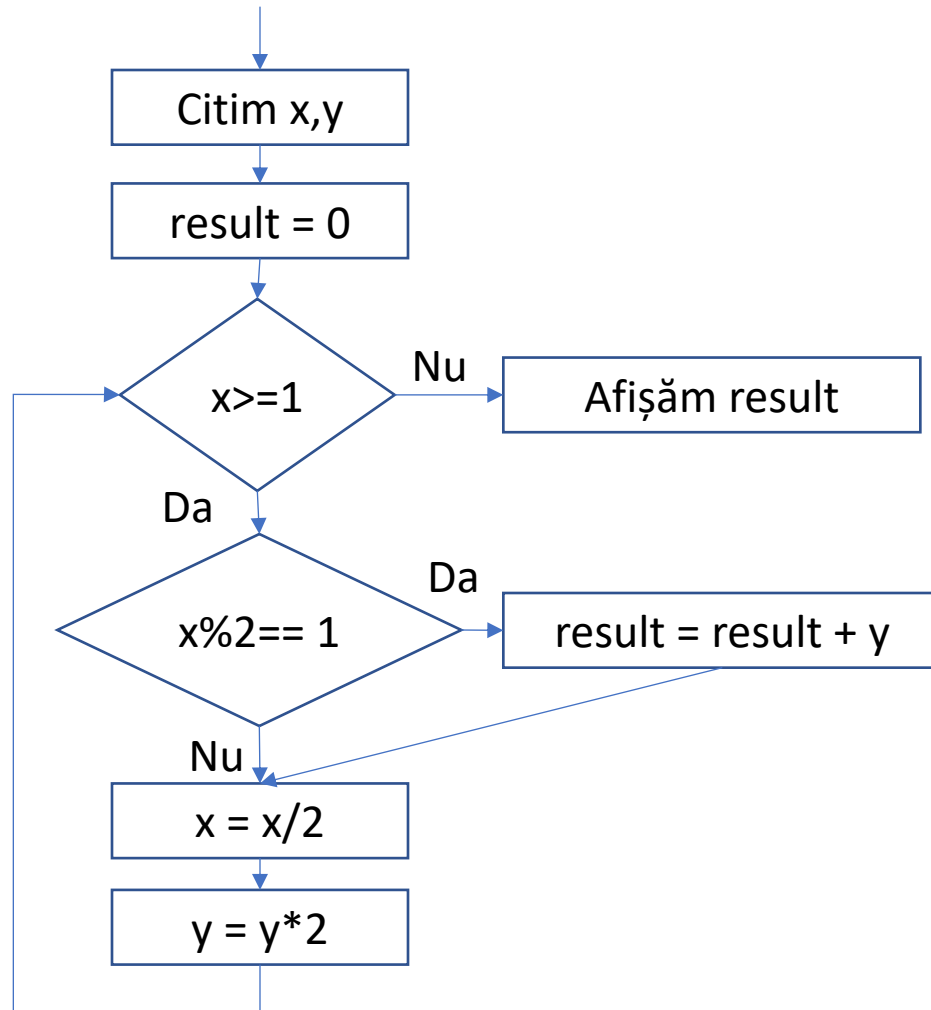
*Pas4*: y devine `y*2`

*Pas5*: dacă x nu este egal cu 1 mergi la *Pas2*;  
altfel `result = result + y`

*Pas6*: afișăm result

Determinați cerințele

# Instrucțiunile repetitive - while



X = 13	Y = 25
13	25
6	50
3	100
1	200

Rezultat:  $x*y = 25 + 100 + 200 = 325$

Să reformulăm:

*Pas1:*  $result = 0$

*Pas2:* Dacă x este impar atunci  $result = result + y$

*Pas3:* x devine  $x/2$

*Pas4:* y devine  $y*2$

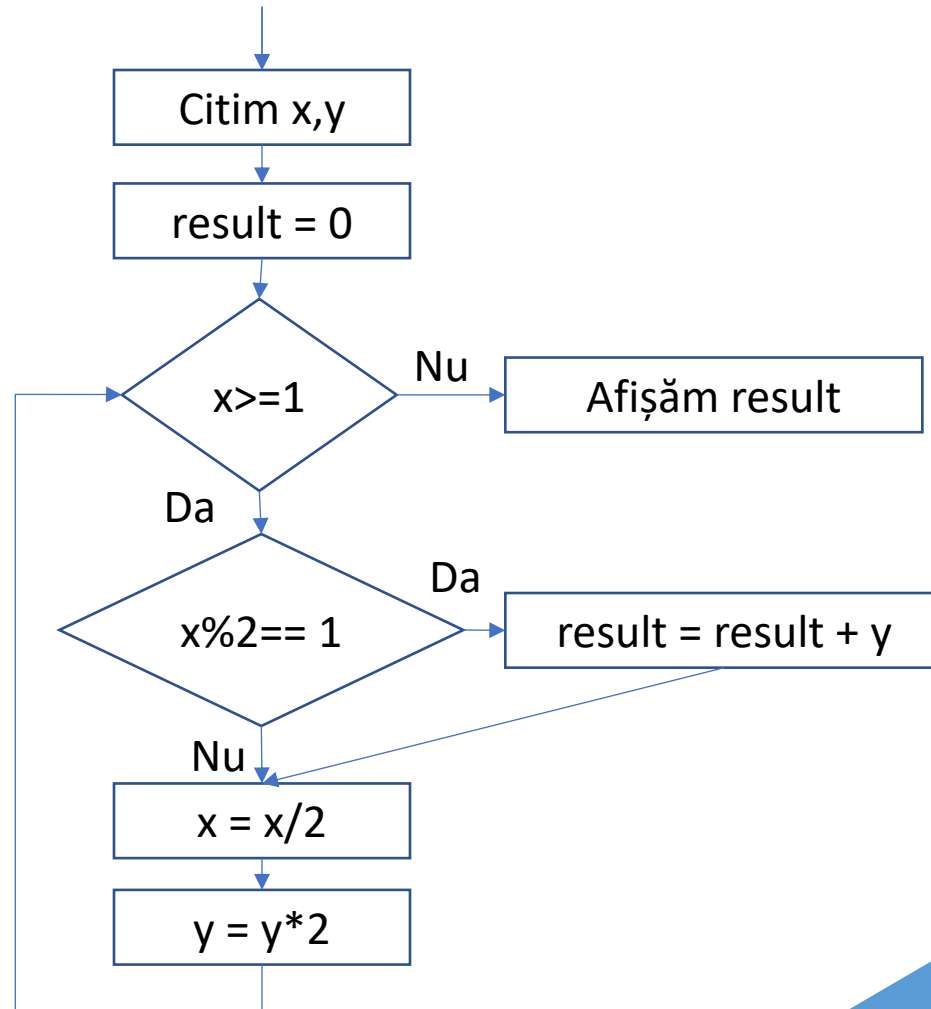
*Pas5:* dacă x nu este egal cu 1 mergi la *Pas2*;  
altfel  $result = result + y$

*Pas6:* afișăm result

Creați un design

# Instrucțiunile repetitive - while

X = 13	Y = 25
13	25
6	50
3	100
1	200



Să reformulăm:

*Pas1:* `result = 0`

*Pas2:* Dacă x este impar atunci `result = result + y`

*Pas3:* x devine `x/2`

*Pas4:* y devine `y*2`

*Pas5:* dacă x nu este egal cu 1 mergi la *Pas2*;  
altfel `result = result + y`

*Pas6:* afișăm result

Implementați acest design

```
x = int(input("x="))
y = int(input("y="))
result = 0
while x >= 1:
    if x % 2 == 1:
        result = result + y
    x = x // 2
    y = y * 2
print('x*y=', result)
```

# Inmultire

```
x = int(input("x="))
y = int(input("y="))
result = 0
while x>=1:
    if x%2 == 1:
        result = result + y
    x = x // 2
    y = y * 2
print('x*y=', result)
```

```
x = int(input("x="))
print(x)
y = int(input("y="))
print(y)
result = 0
while x>=1:
    print(x, y, result)
    if x%2 == 1:
        result = result + y
    x = x // 2
    y = y * 2
print('x*y=', result)
```

testInmultire.py

X = 13	Y = 25
13	25
6	50
3	100
1	200

```
>>> %Run testInmultire.py
x=13
13
y=25
25
13 25 0
6 50 25
3 100 25
1 200 125
x*y= 325
```

Unit testing

# Inmultire

```
x = int(input("x="))
y = int(input("y="))
result = 0
while x>=1:
    if x%2 == 1:
        result = result + y
    x = x // 2
    y = y * 2
print('x*y=', result)
```

## Test 3 - numere negative

```
>>> %Run testInmultire.py
x=4
y=-2
x*y= -8
```

OK

## Test 1 – numere mici

```
>>> %Run testInmultire.py
x=2
y=3
x*y= 6
```

OK

## Test 2 – numere mari

```
>>> %Run testInmultire.py
x=300
y=4000
x*y= 1200000
```

OK

## Test 4 – tot negative

```
>>> %Run testInmultire.py
x=-1
y=5
x*y= 0
```

Not OK

Testare functionala

# S-a gasit un BUG unde este problema?

## Înmulțirea „A la russe”

Înmulțiți  
două  
numere  $x$  și  
 $y$  folosind  
următoarea  
metodă:

- Scrieți  $x$  și  $y$  pe aceeași linie
- Împărțiți  $x$  la 2 și scrieți câtul sub  $x$
- Înmulțiți  $y$  cu 2 și scrieți rezultatul sub  $y$
- Continuați cât timp  $x$  este diferit de 1
- Rezultatul înmulțirii este suma valorilor de pe coloana  $y$  care corespund numerelor impare de pe coloana  $x$

Example

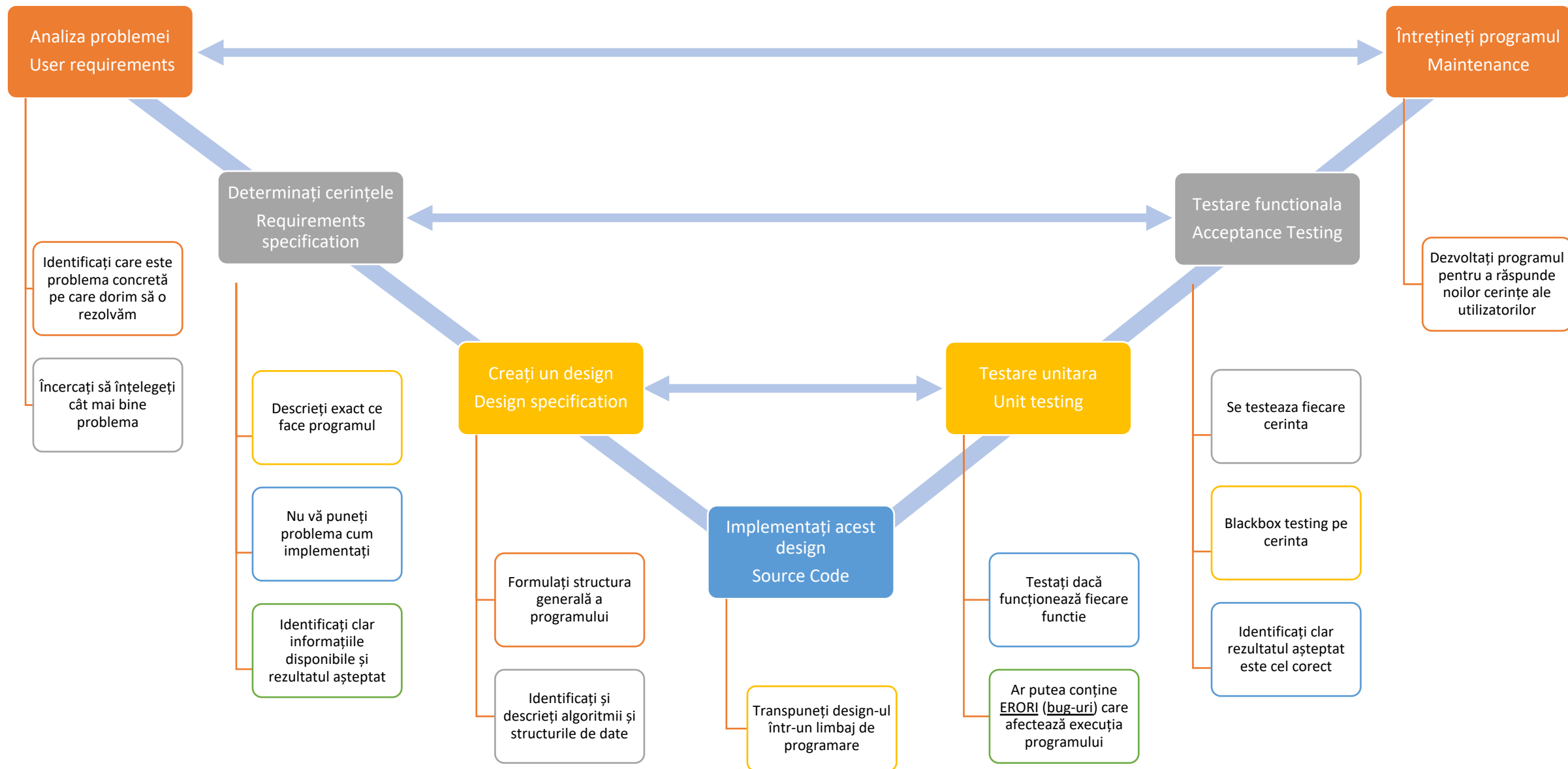
$X = 13$	$Y = 25$
<b>13</b>	<b>25</b>
6	50
<b>3</b>	<b>100</b>
<b>1</b>	<b>200</b>

Rezultat:  $x*y = 25 + 100 + 200 = 325$



Mentenanța

# Ce trebuie sa facă un programator?





# Instrucțiunile repetitive - while

- Folosit pentru a citi ceva de la utilizator

```
m="Răspundeți corect? (3+4-2)"))"
r=int(input(m))
while r != 5:
    r=int(input(m))
```

- Folosit pentru a număra ceva

```
i=0 # Inițializare
while i < 5:
    print(i)
    i += 1 # modificare
```

# for

—

# instrucțiune repetitivă

- Instrucțiunea **for** se comportă altfel în Python decât în limbaje precum C, C++, Java sau Pascal
  - Iterează peste obiecte iterabile (liste)
  - Nu se folosește de expresii pentru controlul iterației

=>

Dar mai întâi să discutăm pe scurt despre liste

# Structuri de date

Liste

Seturi

Tupluri

Dicționare

# Liste

## Ce este o listă ?

- O colecție de obiecte
- Reprezintă o secvență ordonată de obiecte

## Exemple

- [1, 2, -3, 5, 7]
- ['abc', 'efg', "hij"]
- []
- lst = [3, 5, 8]

# Generarea unei liste de numere

## Funcția range:

- Sintaxă
  - `range([start,] stop [, pas])`
- Generează o listă de valori din intervalul  $[start, stop)$  cu un anumit *pas*

## Exemple

- `range(5) → [0, 1, 2, 3, 4]`
- `range(2,5) → [2, 3, 4]`
- `range(0,5,2) → [0, 2, 4]`
- `range(10, 0, -2) → [10, 8, 6, 4, 2]`

# Revenind la instrucțiuni repetitive - for

Instrucțiunea for iterează peste o secvență de valori

## Sintaxa

- **for** <variabilă> **in** <secvență>:
  - instrucțiuni
- **[else:**
  - instrucțiuni]

## Exemple

- Afișarea conținutului unei liste folosind instrucțiunea for
  - `lst = [1, 3, 5, 7]`
  - `for el in lst:`
    - `print (el)`

# Revenind la instrucțiuni repetitive - for

Rescrieți folosind for

`i=0` # inițializare

`while i < 5:`

`print(i)`

`i += 1` # modificare

Folosind FOR

`for i in range(5):`

`print (i)`

în Python:

Nu tot ce puteți scrie folosind **while** poate fi scris cu **for**.

Revenind la  
instrucțiuni  
repetitive - for

Câteodată instrucțiunile  
repetitive trebuie întrerupte

Folosind instrucțiuni de  
întrerupere

- **Break**
  - Întrerupe un ciclu
- **Continue**
  - Sare peste instrucțiunile următoare



# Instrucțiunea break

- O instrucțiune de control a ciclului folosită pentru întrerupere
- Imediat ce este executată instrucțiunea:
  - Ciclul nu mai este executat
  - Controlul revine din ciclu la prima instrucțiune de după ciclu (atenție: nu mai este executată clauza else)

- Exemplu
  - Simulăm aruncarea a două zaruri. Ne oprim când suma lor este 7

```
from random import random
while True:
    dice1 = 1 + int(random()*6)
    dice2 = 1 + int(random()*6)
    print ("dice1=", dice1,
"dice2=", dice2)
    if dice1+dice2 == 7:
        break
```

# Instrucțiunea continue

- O instrucțiune de control a ciclului folosită pentru a sări peste următoarele instrucțiuni din ciclu
  - Condiția ciclului este verificată pentru a determina dacă mai este necesară executarea ciclului

- Exemplu

- Calcularea sumei numerelor pare dintr-o listă

```
l = [23, 45, 66, 77, 98]
s = 0
for el in l:
    if el % 2 == 1:
        continue
    s += el
print("S=", s)
```

# Cicluri imbricate

- În același fel în care putem imbrica instrucțiunea if putem face aceasta și cu while/for

- Cum desenăm următoarea figură:

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

- Soluția

```
n = int(input("n="))
for i in range(n):
    for j in range(n):
        print('*', end=' ')
    print()
```

# Structuri de date din nou...

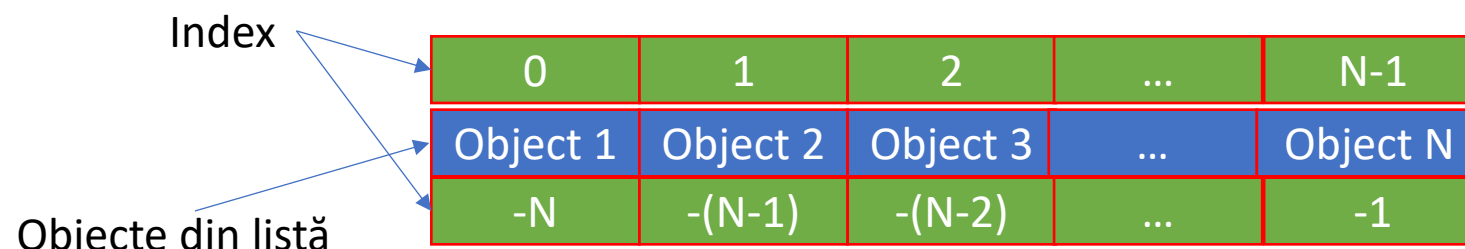
- Limbajul Python suportă următoarele tipuri de date nativ
  - Liste
  - Seturi
  - Tupluri
  - Dicționare

# Liste

- Ce este o listă?
  - O colecție de obiecte
  - Reprezintă o secvență ordonată de date
  - Listele sunt obiecte mutabile

- Exemple:

- [1, 2, -3, 5, 7]
- L1 = ['abc', 'efg', 'hij']
- []
- lst = [3, 5, 8]



Intern listele sunt stocate sub formă de tablouri

# Mai mult despre liste

Listele sunt specificate folosind []

Elementele listei pot fi

- Omogene (toate de același tip)
- Ne-omogene pot conține obiecte de diverse tipuri

Elementele listei pot fi referite folosind indexul lor

- Primul index este 0
- Ultimul index este lungimea listei -1

# Operații pe liste

```
lst = ["aa", 3, "bb", [1, 2]]
```

- Determinarea numărului de elemente dintr-o listă:
  - `len(lst) → 4`
- Accesarea unui element dintr-o listă
  - `lst[3] → [1, 2]`
- Modificarea unui element din listă
  - `lst[3] = "asd" → ["aa", 3, "bb", "asd"]`
- Adăugarea unui element în listă
  - `lst.append("zzz") → ["aa", 3, "bb", [1, 2], "zzz"]`
  - `lst.insert(2, "cc") → ["aa", 3, "cc", "bb", [1, 2]]`
- Ștergerea unui element din listă
  - `lst.pop() → ["aa", 3, "bb"]`
  - `lst.remove(3) → ["aa", "bb", [1, 2]]`
  - `del(lst[2]) → ["aa", 3, [1, 2]]`

# Operații pe liste

`lst = ["aa", 3, "bb", [1, 2]]`

Determinarea numărului de elemente dintr-o listă:

- `len(lst) → 4`

Accesarea unui element dintr-o listă

- `lst[3] → [1, 2]`

Modificarea unui element din listă

- `lst[3] = "asd" → ["aa", 3, "bb", "asd"]`

Adăugarea unui element în listă

- `lst.append("zzz") → ["aa", 3, "bb", [1, 2], "zzz"]`
- `lst.insert(2, "cc") → ["aa", 3, "cc", "bb", [1, 2]]`

Ștergerea unui element din listă

- `lst.pop() → ["aa", 3, "bb"]`
- `lst.remove(3) → ["aa", "bb", [1, 2]]`
- `del(lst[2]) → ["aa", 3, [1, 2]]`



## Operații pe listă

### Felierea (sliceing)

- Extragerea unei sub-liste dintr-o listă

### Exemple

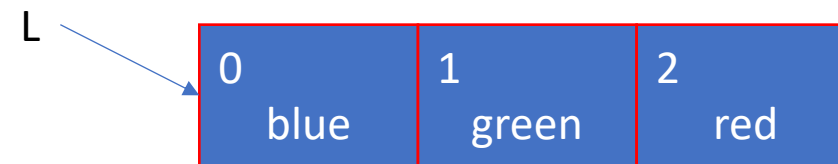
- $L = [8, 9, 10, 11, 12, 13, 14, 15]$
- $L[3:5] \rightarrow [11, 12]$
- $L[:3] \rightarrow [8, 9, 10]$
- $L[5:] \rightarrow [13, 14, 15]$
- $L[0:6:2] \rightarrow [8, 10, 12]$

# Operații pe listă

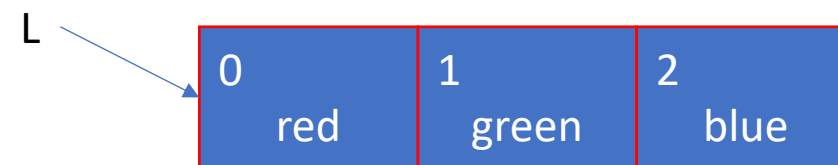
- Sortare
  - `sort()`
  - `sorted()`
- Exemple
  - `L = ["red", "green", "blue"]`



- 
- `L.sort() -> ["blue", "green", "red"]`  
`print(L)`



- 
- `print(sorted(L))`  
`print(L)`



Funcția `sorted()` returnează o nouă listă

# Tupluri

**Imutabil** – Nu poate fi schimbată structura

- Ce sunt tuplurile?
  - O secvență **imutabilă**, ordonată de obiecte.
- Sunt reprezentate folosind paranteze
- Exemple:
  - `T = ()` # tuplu gol
  - `T = ( "Programare 1", "S1", 6)`
  - `T[1]` -> accesarea valorii "S1"
  - `len(T)` -> este evaluat la 3
  - `( "Programare 1", "S1", 6) + (3, 4)` -> `( "Programare 1", "S1", 6, 3, 4)`
  - `T[1:3]` -> este evaluat la `('S1', 6)`
  - `T[1:2]` -> este evaluat la `('S1', )`

Virgula este folosită pentru a spune python că avem de a face cu un tuplu

# Tuplurile sunt utile pentru:

- Interschimbarea valorilor

$X=Y$

$Y=X$

NOT OK

$aux=X$

$X=Y$

$Y=aux$

OK

$(X, Y) = (Y, X)$

OK

- Returnarea a mai multor valori dintr-o funcție
  - O funcție poate returna o singură valoare
  - Tuplurile ne permit să returnăm mai multe

# Tuplu - Imutabil

## Imutabil

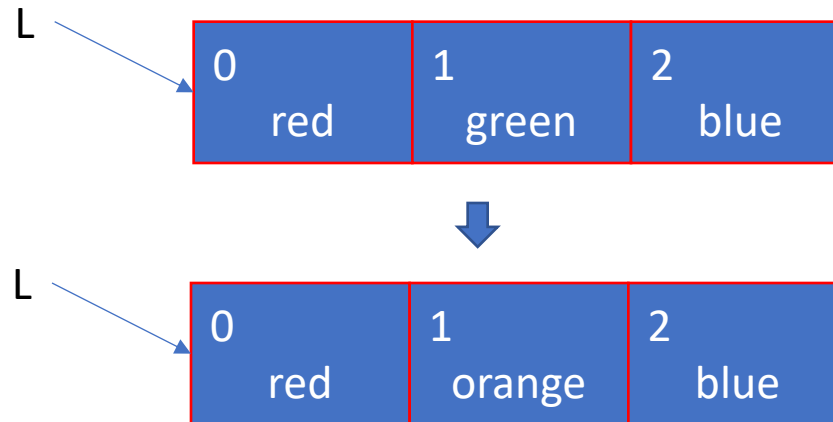
- Nu putem schimba valorile

## Exemplu:

- $T = ( \text{"Programare I"}, \text{"S1"}, 6 )$
- $T[1] = \text{"S2"} \rightarrow \text{ERROR}$

# Listă - Mutabilă

- Listele sunt mutabile
  - Putem modifica membrii listei
- Exemple
  - $L = [\text{"red"}, \text{"green"}, \text{"blue"}]$
  - $L[1] = \text{"orange"}$



# Listă - Mutabilă

---

Listele sunt mutabile

---

Se comportă altfel decât structurile imutabile

---

Este un obiect în memorie

---

Numele de variabilă "arată" direct spre obiecte

---

Orice variabilă care arată spre un obiect mutabil este afectată de modificări

---

Este important atunci când lucrăm cu liste să avem în vedere efectele laterale ale modificărilor

**MODIFICARE, ALIASING, CLONARE**

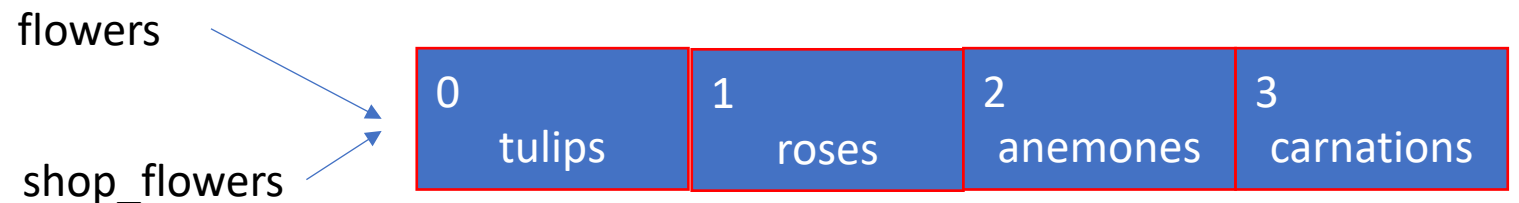
# Alias-uri

```
a=1  
b=a  
b=2  
print(a)  
print(b)
```



```
flowers = ["tulips", "roses", "anemones"]  
shop_flowers = flowers  
shop_flowers.append("carnations")  
print(flowers)  
print(shop_flowers)
```

A blue arrow points from the code to the right. To the right of the arrow, two identical list representations are shown, one above the other: ['tulips', 'roses', 'anemones', 'carnations'].



Alias-urile sunt nume care arată spre același obiect  
Orice schimbare în valoare este reflectată în toate variabilele

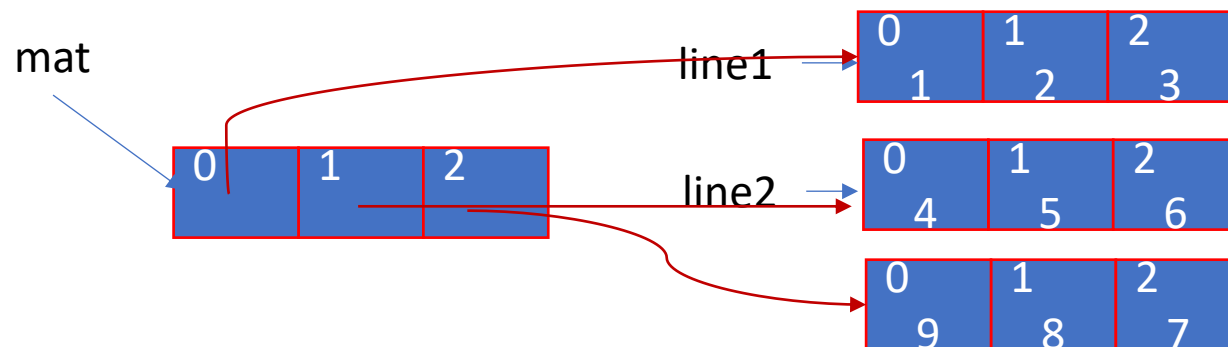


# Listă de liste...

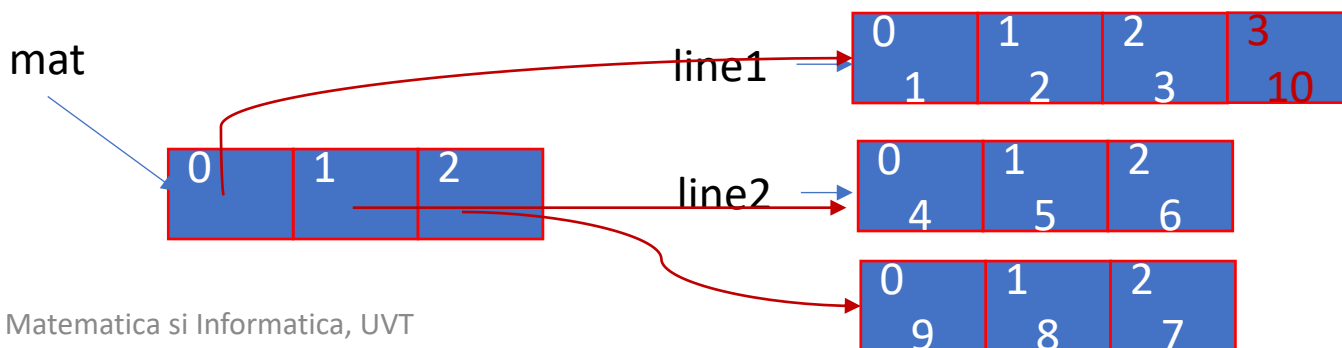
- Listele fiind simple obiecte putem avea liste în liste
- Modificările pot avea efecte laterale

```
line1 = [1, 2, 3]
line2 = [4, 5, 6]
mat = [line1, line2, [9, 8, 7]]
print(mat)
```

```
line1.append(10)
print(mat)
```



```
[[1, 2, 3], [4, 5, 6], [9, 8, 7]]
[[1, 2, 3, 10], [4, 5, 6], [9, 8, 7]]
```



# Clonare

```
line1 = [1, 2, 3]
```

```
line2 = [4, 5, 6]
```

```
mat = [line1[:], line2, [9, 8, 7]]
```



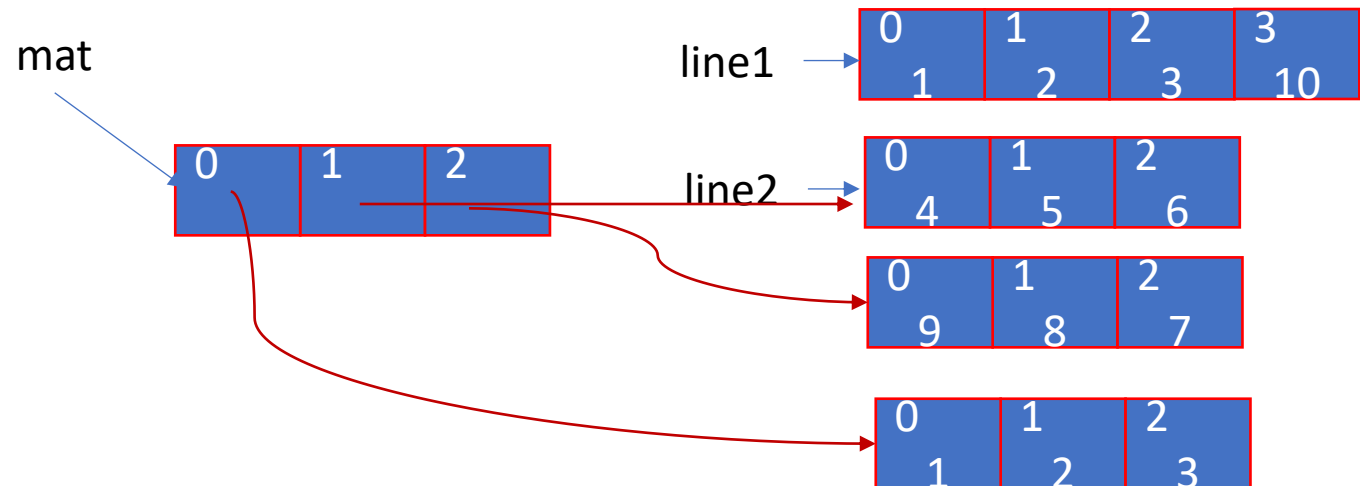
```
[[1, 2, 3], [4, 5, 6], [9, 8, 7]]
```

```
[[1, 2, 3], [4, 5, 6], [9, 8, 7]]
```

```
print(mat)
```


```
line1.append(10)
```

```
print(mat)
```



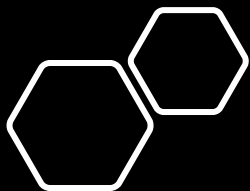
## Clonare

Creează o nouă listă conținând  
toate elementele listei folosind  
[:]



Exemplu:

```
new_list = L1[:]
```



# Set

Un set este o colecție neordonată de obiecte

Fiecare element este unic și imutabil

Totuși setul este mutabil, putem adauga și sterge elemente (intern este implementat ca dicționar)

Poate fi folosit pentru a implementa operații precum uniune, intersecție, diferență simetrică

# Set

## Creare

- `S = set()` # set vid
- `S = {1, 2, 3}`
- `S={}` #NOT OK este inițializarea unui alt tip de obiect – dicționarul
- `print(type(S))`

## Adăugarea de elemente

- `S.add(2)`
- `S.add(2)`

# Operații pe seturi

- Ștergere
  - `S.remove(2)` # șterge elementul cu valoarea 2
- Uniune  $A \cup B$ 
  - `A.union(B)`
- Intersecție  $A \cap B$ 
  - `A.intersection(B)`
- Diferență  $A - B$ 
  - `A.difference(B)`
- Apartenență  $element \in B$ 
  - `element in A`

# Dicționare

Cum am putea salva informații despre studenți ?

- Names = ['Ionescu Ion', 'Popescu Pavel', 'Marinecu Maria']
- Current\_year\_mean = [9.4, 8, 6.78]
- Year = [1, 2, 1]

O **listă separată** pentru fiecare element ?

Fiecare listă trebuie să aibă **aceeași lungime**

Informațiile trebuie stocate la același index. Index-ul este folosit pentru a ne referi la persoane

# Cum actualizăm datele?

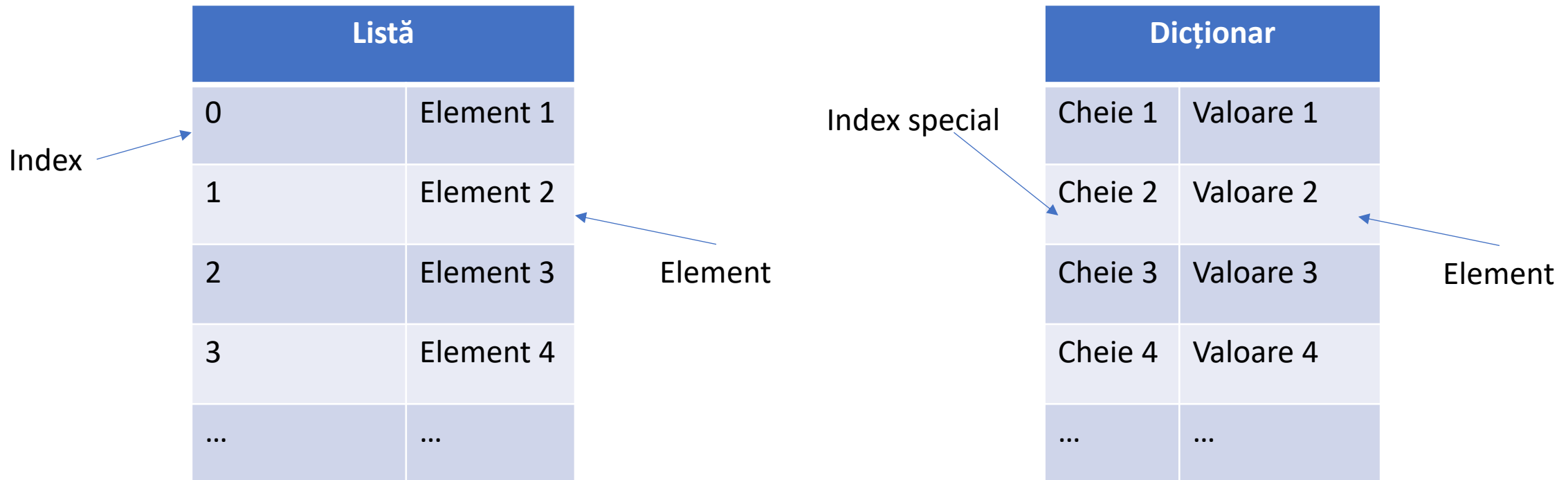
```
name = input("Nume student: ")  
i = names.index(name)  
Current_year_mean[i] = 8.7  
Year[i] = 2
```

- **dezorganizat** dacă trebuie să ținem evidența pentru multe date
- Trebuie să avem grijă de mai **multe liste**
- Trebuie să indexăm tot timpul folosind numere
- Trebuie să ținem minte să schimbăm mai multe liste



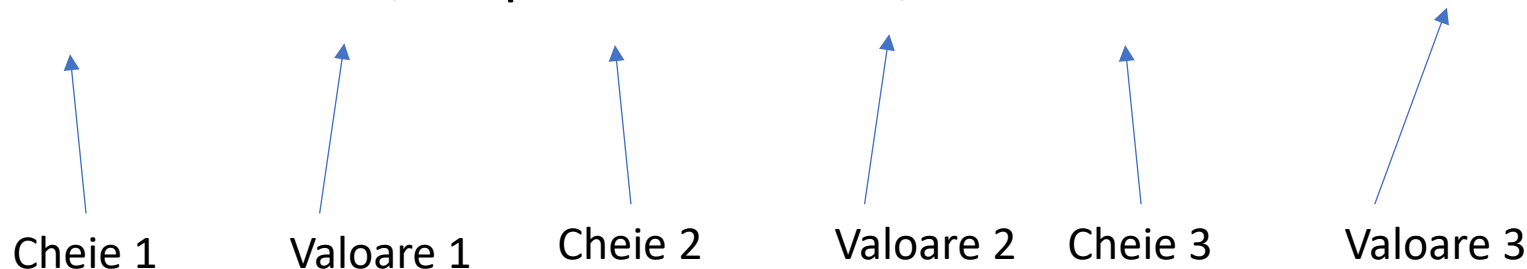
# Mai bine și mai isteț -- dicționarele

- O singură structură de date
- Indexarea este bazată pe o cheie și nu pe un index



# Dicționare

- Stochează perechi de date
  - (cheie, valoare)
- Creare
  - `dict1={} # dicționar vid`
  - `dict_grades= {'Ionescu Ion' : 9.4, 'Popescu Pavel' : 8, 'Marinecu Maria' : 6.78}`



# Dicționare

## Accesarea elementelor

- Similar cu o listă
- Folosind cu o cheie

## Exemple

- `dict_grades= {'Ionescu Ion' : 9.4, 'Popescu Pavel' : 8, 'Marinecu Maria' : 6.78}`
- `dict_grades['Ionescu Ion']` este evaluat la 9.4
- `dict_grades['Ionescu Vasile']` evaluat la: *error key does not exist*



# Operații pe dicționare

```
dict_grades= {'Ionescu Ion' : 9.4, 'Popescu Pavel' : 8,  
'Marinecu Maria' : 6.78}
```

## Adăugare

- `dict_grades['Enescu Ene'] = 8.7`

## Verificat dacă o cheie este în dicționar

- `'Ionescu Ion' in dict_grades`

## Ștergerea unei intrări

- `del dict_grades['Ionescu Ion']`

# Dicționare - iterare

```
dict_grades= {'Ionescu Ion' : 9.4, 'Popescu Pavel' : 8, 'Marinecu  
Maria' : 6.78}
```

## Obținerea cheilor

- `dict_grades.keys()`
- `for key in dict_grades.keys():`
  - `print(key)`

## Obținerea valorilor

- `dict_grades.values()`
- `for value in dict_grades.value():`
  - `print(value)`

## Obținerea perechilor (cheie, valoare)

- `dict_grades.items()`
- `for key, value in dict_grades.items():`
  - `print(key, ":", value)`

# Dicționare: chei și valori

## Chei

- Trebuie să fie **unice**
- Trebuie să fie de tip **imutabil** (int, float, string, tuple, bool)
- Mai exact trebuie să fie un tip care este "hashable"
- Tipurile imutabile sunt hashable
  - Mare grijă cu float!
  - **Dicționarele nu sunt ordonate!**  
`d = {4:{1:0}, (1,3):"twelve", 'const':[3.14,2.7,8.44]}`

## Valori

- Orice tip (**mutabil și imutabil**)
- Pot fi **duplicate**
- Valorile pot fi orice obiecte (chiar și alte dicționare)

# Liste versus Dicționare

## Liste

- Secvențe **ordonate** de obiecte
- Referențierea se face după un număr, index
- Indicii au o **ordine**
- Un index este un **integer**

## Dicționare

- **asociază** “chei” la “valori”
- Caută un element pe baza altui element
- Nu garantează **ordinea**
- Cheile pot să fie de orice tip **imutabil**





# Bibliografie

- <https://youtu.be/0jljZRnHwOI?t=1020>
- <https://www.youtube.com/watch?v=RvRKT-jXvko>
- [John Zelle](#), **Python Programming: An Introduction to Computer Science (chapter 2)**

## Sa analizam cateva probleme:

- Scrieți un program care să facă **adunarea a două matrice** pătrate.
- Scrieți un program care să calculeze **media notelor** unui elev/student.
- Scrieți un program care citește un număr și creează o structura care conține **toate cifrele numărului**.
- Scrieți un program care creează o **statistică cu numărul de vocale** care apar într-o propoziție.
- Scrieți un program care creează o statistică cu **valorile pixelilor** dintr-o imagine.