

# Programare 1

Introducere în programare



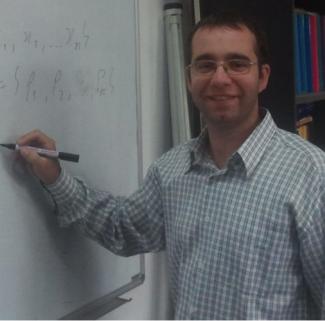
**Teodora  
Selea**



**Elena  
Flondor**



**Alexandru  
Ionașcu**



**Cosmin  
Bonchiș**



**Adrian  
Spătaru**



**Florin  
Roșu**



**Petru  
Mulcuța**



**Sorin  
Valcan**



**Darius  
Oanea**

# Despre ce discutăm astăzi ?

## Informații despre curs

- Organizare și evaluare

## Elemente de bază despre Python

- Variabile și tipuri

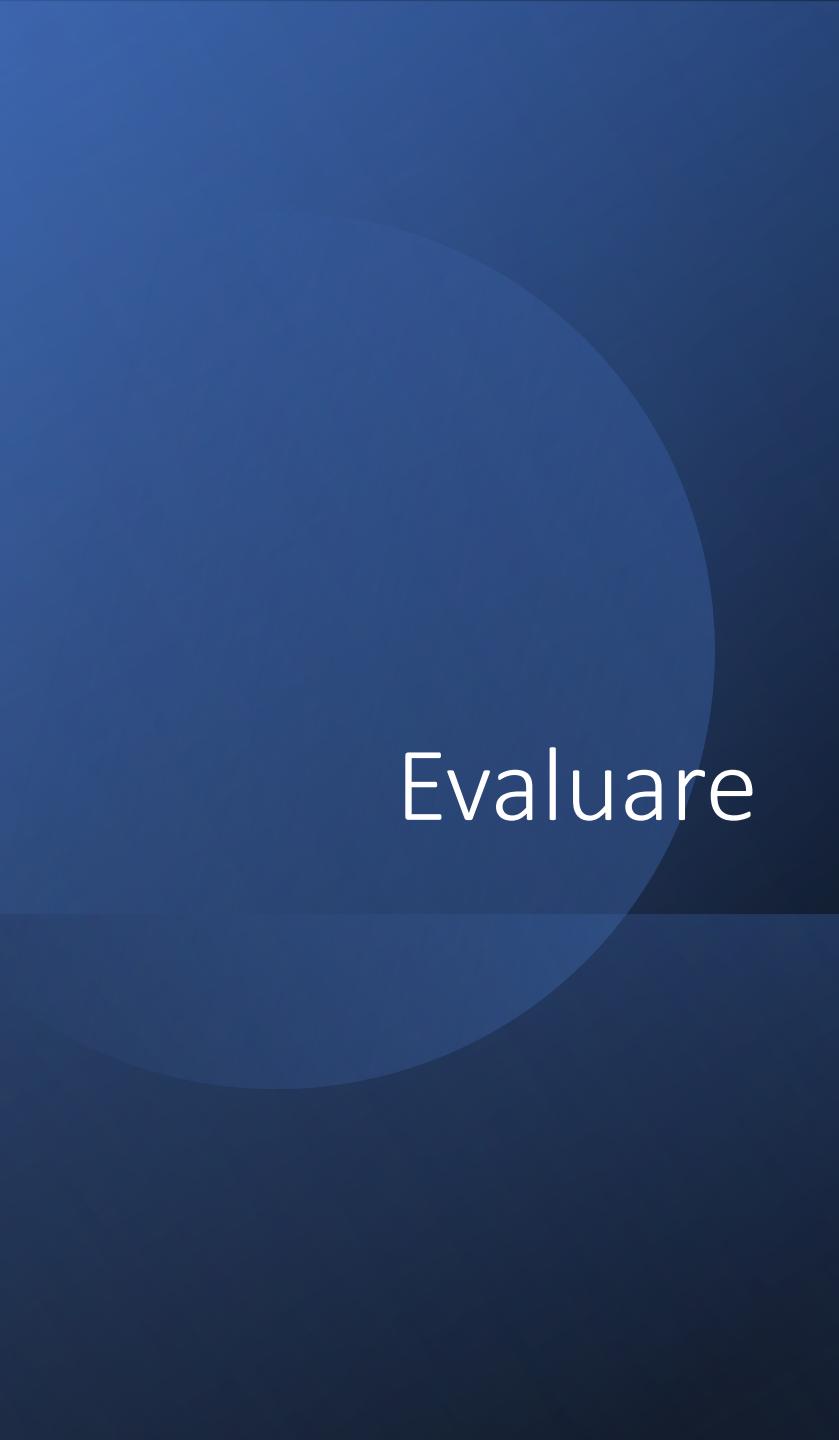
## Operații matematice

# Despre curs

Este un curs introducțiv !!!

Pentru începători:

- Nu puteți învăța programare în mod pasiv
- Să nu vă fie frică să interacționați cu Python. Eșecul reprezintă o oportunitate pentru a învăța
- Descărcați materialele noastre și încercați să le urmăriți în cadrul laboratoarelor și cursurilor
- Exersați



Evaluare

---

**30% Examen**

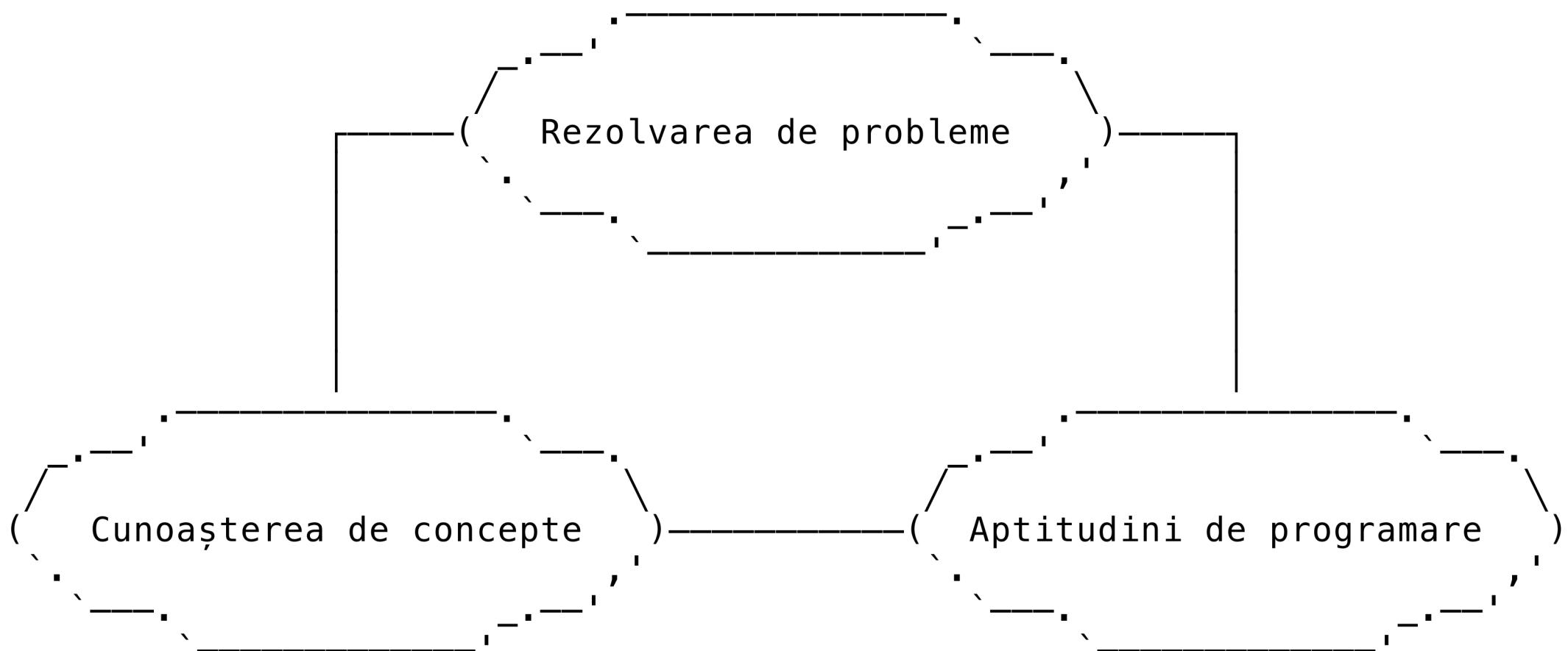
---

**70% Laborator (4 Teste)**

---

**Extra 1 punct activitate**

# Despre curs



# Tematică curs

---

Ce este un calculator și cum funcționează

---

Reprezentarea conceptelor folosind structuri de date

---

Iterație și recursivitate

---

Abstractizarea procedurilor și tipurilor de date

---

Organizarea sistemelor folosind module, clase și obiecte

---

Diferite tipuri de algoritmi: căutare și sortare

---

Complexitatea algoritmilor

# Ce face un calculator ?

## Fundamental:

- Efectuează miliarde de **calcule** pe secundă
- Reține rezultatele acestor calcule: sute de giga octeți de memorie

## Tipuri de calcule ?

- Calcule **predefinite** în limbaj
- Calcule **definite de noi** în calitate de programatori

Calculatoarele știu doar ce le spunem, nimic altceva

# Tipuri de cunoaștere

## Cunoaștere declarativă

- Aserțiuni privind niște fapte
- De exemplu:  $\sqrt{x}$  este definit ca acel  $y$  pentru care  $y^2 = x$  și  $y > 0$
- În exemplu avem o definiție, o axiomă. Dar în general nu ne ajută să găsim radicalul, maxim să testăm o valoare
- Ne spune **CE** dar **NU CUM**

## Cunoaștere imperativă

- Rețetă. Un set de reguli.
- Ne spune cum putem să deducem ceva
- Exemplu, pentru radical

# Tipuri de cunoaștere

## Metoda lui Heron din Alexandria

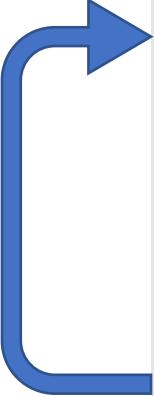
Alegem **aleator** un  $G$

Dacă  $G*G$  este **suficient de aproape** de  $x$ :  
atunci ne oprim și  $G$  este răspunsul

Altfel

ghicim din nou și  $G$  devine  $(G+x/G) / 2$

Repetăm

- 
- Spre deosebire de cunoașterea declarativă aici știm exact cum trebuie să procedăm
  - Este clar!

# Tipuri de cunoștere

## Metoda lui Heron din Alexandria

Pentru  $x=9$

<b>g</b>	<b><math>g^*g</math></b>	<b><math>x/g</math></b>	<b><math>(g+x/g)/2</math></b>
2	4	4.5	3,25
3,25	10,5625	2,7692	3,0096
3,0096	9,0576	2,9904	3

**Rapid, nu ?**

# Rețeta

---

O succesiune de **pași**

---

Un mecanism de **gestiune a controlului**,  
care ne spune când este executat fiecare pas

---

O modalitate de a ne spune **când sa ne oprim**

---

**Când avem aceste trei caracteristici  
îndeplinite vorbim despre un algoritm**

# Primele calculatoare: Calculatoarele numerice cu program fix

---

Calculatoarele de birou: fac doar aritmetică



[Calculatorul Atanasoff–Berry](#) (1942): rezolvarea de ecuații liniare

---



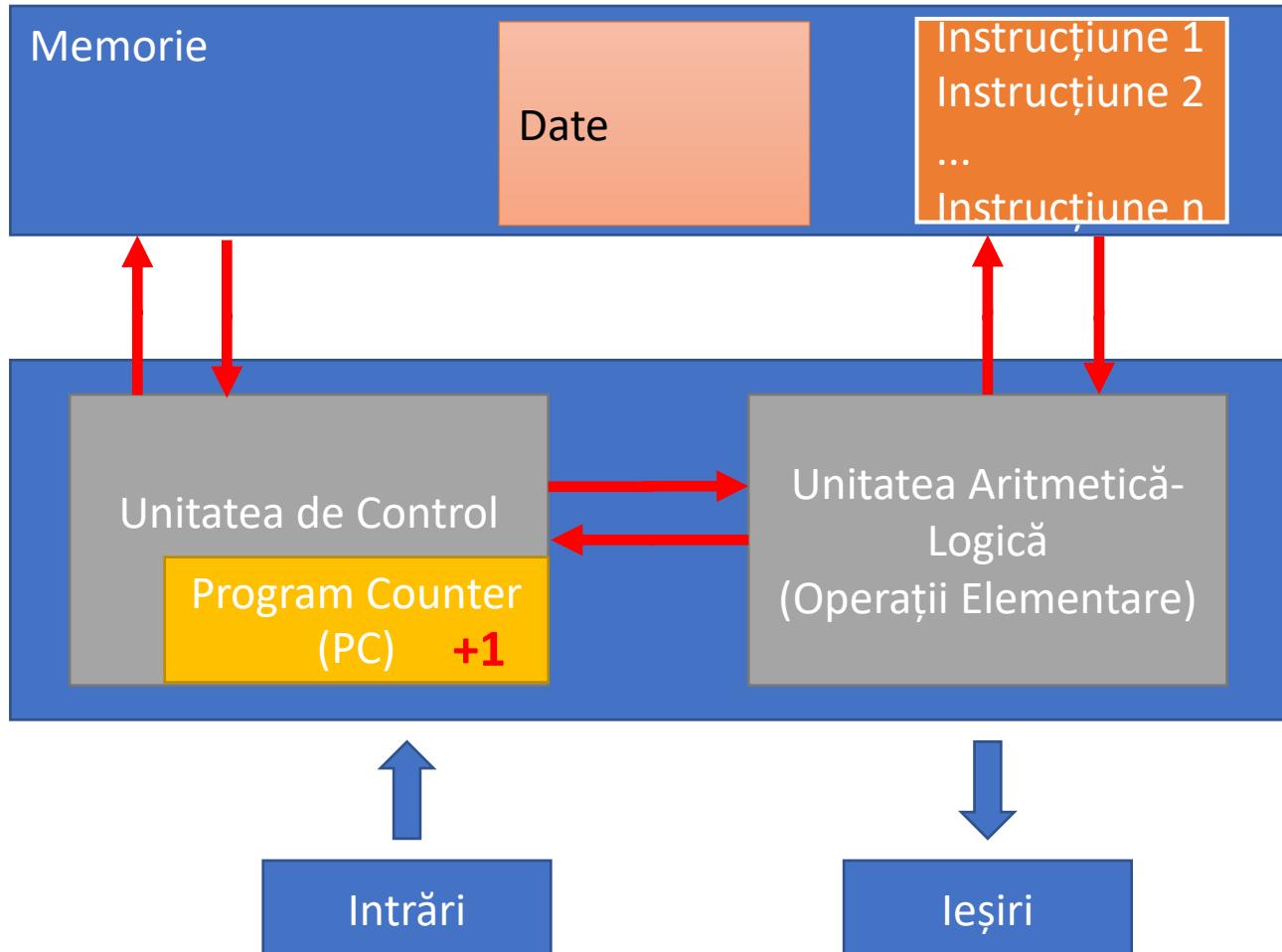
„Bomba” Turing: un dispozitiv electro-mecanic menit a descifra mesajele codate cu dispozitivul german Enigma

# Primele calculatoare: Calculatoarele numerice cu program fix

- Oare nu putem crea un calculator cu program fix care primește ca intrare diagrama de circuit a unui alt calculator și care se reconfigurează în aşa fel încât să se comporte ca acea diagramă ? 😊
  - Adică s-ar putea comporta precum un calculator de birou sau ca un calculator Atanasoff-Berry ?

EXISTĂ! Este în inima oricărui calculator: un *interpreter*.

# Primele calculatoare: Calculatoarele numerice cu program stocat



## Arhitectură elementară

- Unele instrucțiuni ar putea modifica PC în aşa fel încât să nu fie incrementat ci să meargă la o instrucțiune specifică
  - Control al fluxului

# Calculatoare numerice cu program stocat

O succesiune de instrucțiuni sunt stocate  
în memoria unui calculator

- Instrucțiuni dintr-un set predefinit de instrucțiuni primitive
  - Aritmetică și logică
  - Teste simple
  - Copiere de date

Un interpreter (un program special)  
execută aceste instrucțiuni în ordine

- Se folosește de teste pentru controlul execuției secvenței de instrucțiuni
- Se oprește când termină

# Ce este un program ?

Un program nu este nimic altceva decât o rețetă!

- Este format dintr-un **set fix** de instrucțiuni primitive
  - Cu acest set de instrucțiuni se poate programa orice
- Care sunt aceste primitive ?
- În 1936 Alan Turing a arătat că 6 primitive simple sunt suficiente pentru a programa orice program care poate fi descris într-un proces mecanic.

O implicație interesantă este că orice program care poate fi scris într-un limbaj de programare poate fi scris în orice alt limbaj de programare.

- Conceptul se cheamă „Compatibilitate Turing”.

# Ce este un program ?

- Pentru a descrie rețetele avem nevoie de un limbaj. Un **limbaj de programare!** În acest curs o să folosim limbajul de programare Python



Acest curs **NU** este despre Python

# Rețete (din nou)

---

Un limbaj de programare ne oferă un set de **operații** primitive

---

**Expresiile** reprezintă combinații complexe (dar valide) a acestor primitive oferite de limbajul de programare

---

Expresiile și calculele au **valori** și semnificație într-un limbaj de programare

# Aspectele unui limbaj

- Construcții primitive
    - În limba Română: cuvinte
    - Într-un limbaj de programare: numere, siruri de caractere, operatori simpli



# Limbaje de programare

De nivel înalt sau  
jos ?

C

Python

JavaScript

Go

De uz general sau  
specializate ?

Matlab

Python

JavaScript

Interpretate sau  
compilate ?

C

Java

Python

JavaScript

# Limbaje de programare

- **Sintaxă**
  - Care expresii sunt legale in acest limbaj de programare ?
  - „Băiatul o pisică casă” 🤔
  - Erorile de sintaxă sunt foarte frecvente și ușor de detectat!
- **Semantică**
  - **Statică:** ne spune care programe au sens. Ce expresii au sens ?
    - „Cursul acesta este gustos” .
    - Sintactic este corect, nu ?
    - Acet tip de erori pot să ducă la comportament imprevizibil...
  - **Completă:** ce semnifică programul ? Ce se va întâmpla când rulam programul ?
    - ✓ **STIL:** Aici depinde mult de voi!
    - ✓ Rezultate: programul crapă, rulează o veșnicie, se oprește, dă un rezultat diferit de ce ne așteptăm

# Obiecte (în Python)

Programele manipulează obiecte

Obiectele au un tip care ne definește ce poate un program să facă cu aceste obiecte

- Lulu este un câine și poate să facă „hau hau”
- Pisi este o pisică și poate să facă „miau miau”

Obiectele pot să fie:

- Valori scalare (ex. valoarea 42)
- Non-scalare (au o anumită structură internă care poate fi accesată)

# Valori Scalare

- Reprezentare pentru date fundamentale
  - Numere:
    - 3 – int
    - 3.14 – float
  - Logice:
    - True/False – bool
  - NoneType
    - O singură valoare: None
  - Siruri de caractere:
    - ‘Timișoara’ – str
- În mod ușual discutăm despre perechea (tip, valoare)
- Pentru fiecare tip avem un set de operatori:
  - Pentru numere: + - \* / %
  - Pentru siruri: +, \*

Putem folosi type() ca să aflăm tipul unui obiect:

```
>> type(5)
Int
>> type(3.14)
float
```

# Conversie între tipuri

---

Putem să convertim obiecte de un tip în obiecte de alt tip

---

`float(3)` → 3.0

---

`int(3.9)` → 3

# Operatori pe `int` și `float`

- $i+j \rightarrow$  suma  Dacă amândoi operanții sunt int rezultatul este int
  - $i-j \rightarrow$  diferența  Dacă unul dintre operanți este float rezultatul este float
  - $i*j \rightarrow$  produsul 
  - $i/j \rightarrow$  împărțirea  Rezultatul este float
- 
- $i \% j \rightarrow$  restul împărțirii lui  $i$  la  $j$
  - $i ** j \rightarrow i$  la puterea  $j$

# Operatori

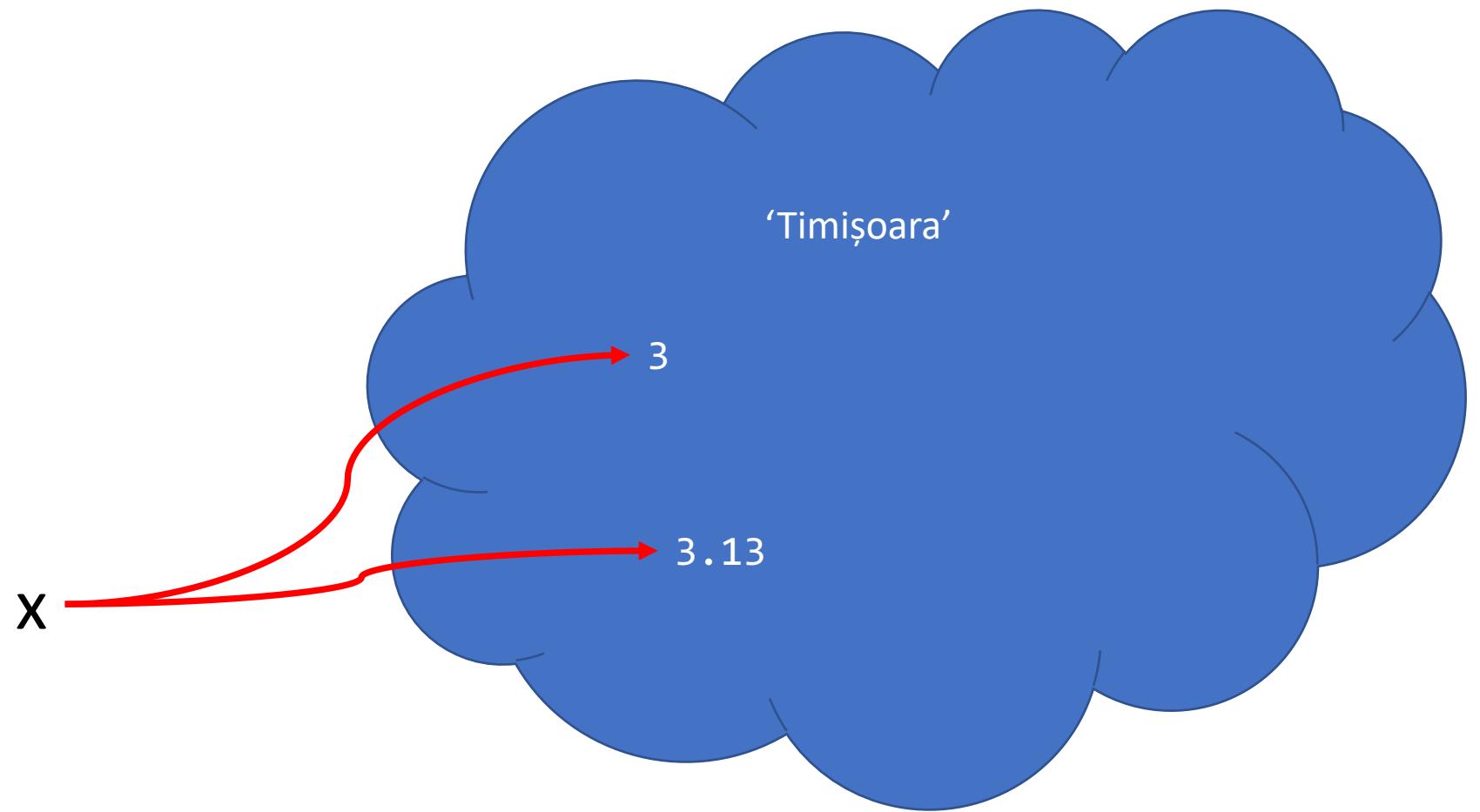
Operator	Descriere
<code>**</code>	Exponențiere (ridicare la putere)
<code>~ + -</code>	Complement, plus și minus unar
<code>* / % //</code>	Înmulțire, împărțire, module și „floor division”
<code>+ -</code>	Adunare și scădere
<code>&gt;&gt; &lt;&lt;</code>	deplasare de biți (la dreapta, la stanga)
<code>&amp;</code>	ȘI pe biți
<code>^  </code>	<b>SAU EXCLUSIV</b> pe biți, <b>SAU</b> pe biți
<code>&lt;= &lt; &gt; &gt;=</code>	operatori de comparare
<code>&lt;&gt; == !=</code>	operatori de egalitate
<code>= %= /= //=-+= *= **=</code>	operatori de atribuire
<code>is is not</code>	operatori de identitate
<code>in not in</code>	operatori de apartenență
<code>not or and</code>	operatori logici

# Variabile

- Creează o asociere/legătură între un nume și o valoare
  - Operația de atribuire: `x=3`, Creează o legătură/asociere între numele x și valoarea 3
- Care este tipul unei variabile ?
  - În cazul Python răspunsul este că tipul este moștenit de la valoare. Mai exact variabila nu are tip ci valoarea referită de ea
  - Tipul variabilelor este dinamic, se schimbă în funcție de valoarea referită
  - Sfat: nu schimbați în mod arbitrar „tipul” variabilelor
- Unde putem folosi variabilele: oriunde putem folosi o valoare!
- Numele variabilelor: important să aibă sens!
  - Nu puteți folosi cuvinte cheie rezervate

# Variabile

$x = 3$



# Abstractizarea expresiilor

---

De ce să dăm nume expresiilor ?

---

Pentru a reutiliza nume în locul valorilor !

---

Putem schimba codul mai ușor, mai târziu!

---

$\pi = 3.14159$

---

$raza = 2.2$

---

$aria = \pi * (raza ^ 2)$

# Şiruri de caractere

Litere, numere, caractere speciale, spaţii

Sunt introduse folosind simbolul " sau '

- `hi = 'salut'`

concatenare de şiruri:

- `nume = 'ion'`
- `salut = hi + nume`

Alte operaţii pe şiruri:

- `test = hi + " " + nume * 2`

# Intrare/ieșire: print

Este folosit pentru a afișa ceva la ieșirea standard

Funcția folosită este print:

```
x = 1
print(x)
x_str = str(x)
print ("un numar este", x, ".", "x=", x)
print ("un numar este" + x_str + ". " + "x= " + x_str)
```

# Intrare/Ieșire: `input("")`

- afișează ce primește ca argument (ce avem între ghilimele)
- citește ceva de la utilizator până când acesta tastează tasta enter
- returnează o valoare pe care o legăm de o variabilă

```
text = input("Introduceți ceva: ")
```

```
print(5*text) 🤔
```

- `input` ne dă un sir de caractere pe care trebuie să îl convertim la tipul dorit de noi

```
num = int(input("Introduceți un număr"))
```

```
print(5*num)
```

# Operatori de comparație pe **int**, **float**, **str**

---

i și j sunt nume de variabilă

---

comparațiile de mai jos sunt evaluate ca un boolean

---

i > j

---

i >= j

---

i < j

---

i <= j

---

i == j → operator de **egalitate**, True dacă i este egal cu j

---

i != j → operator de **inegalitate**, True dacă i este diferit de j

---

i is j → operator de **identitate**, True dacă i este același cu j

# Operatori logici pe bool

- **a și b** sunt variabile cu valori booleene

not a → True dacă a este False

False dacă a este True

a and b → True dacă a și b sunt True

a or b → True dacă unul dintre a și b este True

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

# Controlul fluxului

```
if <conditie>:  
    <expresie>  
    <expresie>  
    ...
```

```
if <conditie>:  
    <expresie>  
    <expresie>  
    ...  
else:  
    <expresie>  
    <expresie>  
    ...
```

```
if <conditie>:  
    <expresie>  
    <expresie>  
    ...  
elif <conditie>:  
    <expresie>  
    <expresie>  
    ...  
else:  
    <expresie>  
    <expresie>  
    ...
```

- <conditie> are valoarea True sau False
- evaluam expresiile dintr-un bloc dacă condiția este evaluată ca True

# Indentare

- este importantă în Python
- este folosită pentru a identifica blocuri de cod

```
x = float(input("x="))
y = float(input("y="))
if x == y:
    print("x și y sunt egale")
    if y != 0:
        print ("asadar, x/y este", x/y)
elif x<y:
    print ("x este mai mic")
else:
    print("y este mai mic")
print ("gata")
```