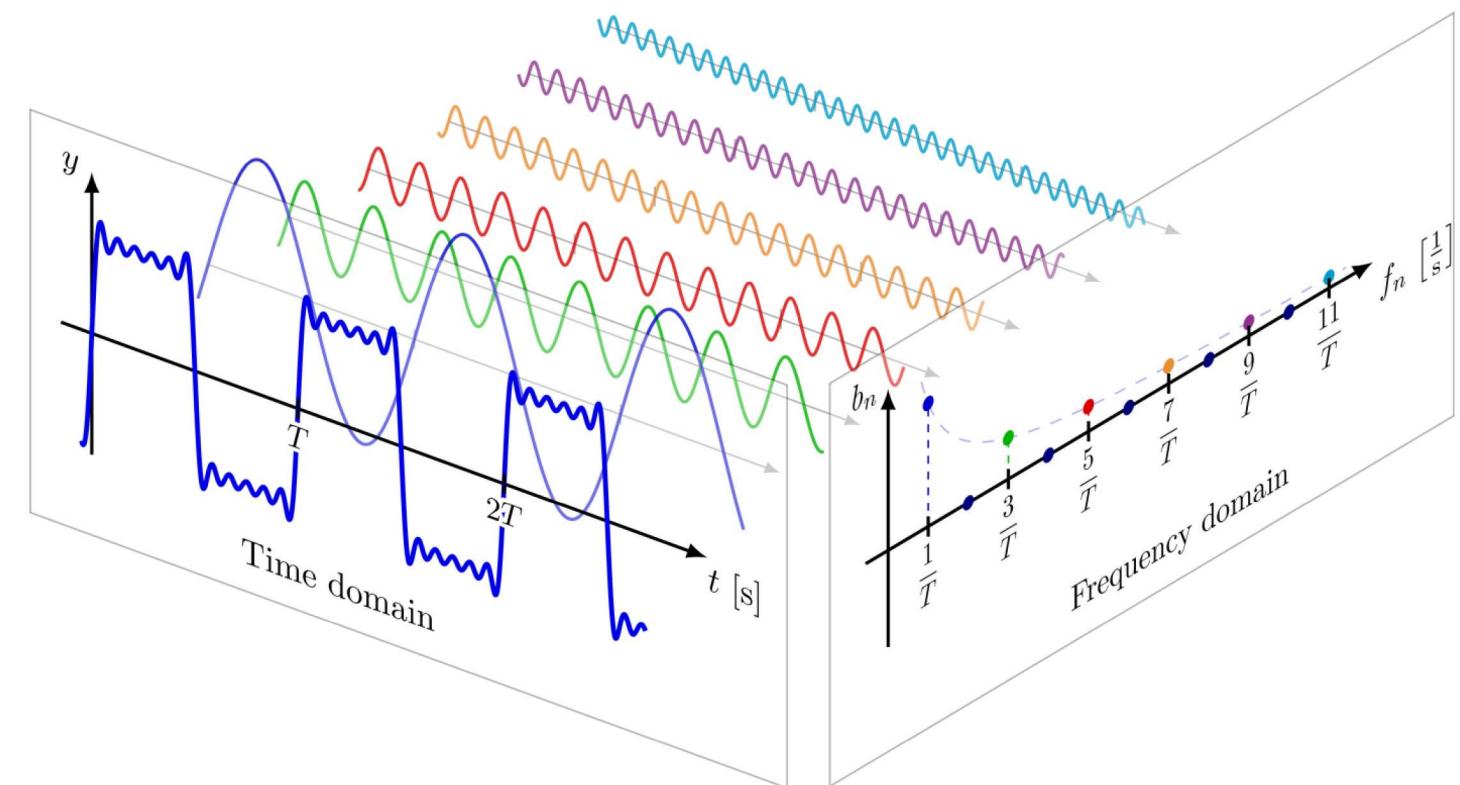


# CURS 1 PD

## Introducere în limbajul Python



**Introducere**

**Bazele Python**

**Structuri de control**

1. Introducere

2. Bazele Python

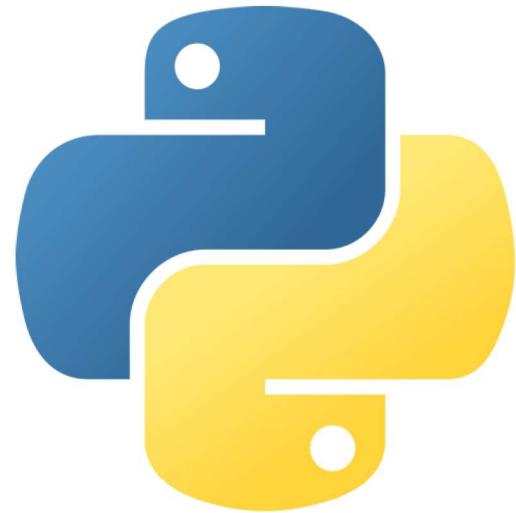
3. Structuri de control

# Introducere

Structuri de control

**Python** este un limbaj de programare de nivel înalt, de utilitate general, putând fi folosit în aproape orice domeniu:

- dezvoltare de aplicații mobile / PC / web
- dezvoltare de jocuri video
- dezvoltare de aplicații industriale sau de laborator:
  - inteligență artificială
  - procesare de date
  - controlul sistemelor
  - calcul ingineresc
  - etc.



### Avantaje

1. Este considerat limbaj de nivel înalt – instrucțiunile acestuia sunt apropriate de limbajul uman
2. Codul nu este compilat, ci interpretat
3. Există un număr foarte mare de librării ce sunt disponibile în mod gratuit
4. Oferă posibilitatea mai multor tipuri de programare: structurată, funcțională, orientată pe obiect

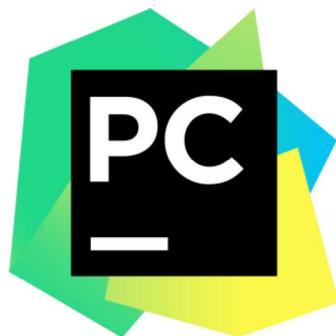
Descărcarea kit-ului de instalare se poate face din secțiunea „**Downloads**” a site-ului oficial Python: <https://www.python.org/>. Se recomandă instalarea versiunii **Python 3.9** sau a unei versiuni mai recente pentru buna funcționare a exemplelor de aplicații ce vor fi prezentate.

## Utilizare

Există mai multe metode de a utiliza Python:

1. Comenzile pot fi introduse în mod interactiv. Pentru această metodă se va deschide o fereastră de „**Command Prompt**” sau „**PowerShell**” în care se va introduce comanda **python**. După apariția prompt-ului **>>>**, se pot introduce instrucțiuni **Python** ce se vor executa în momentul apăsării tastei **Enter**.
2. Instrucțiunile **Python** pot fi scrise într-un fișier script cu extensia **.py** folosind orice editor de text și apoi rulate din „**Command Prompt**” sau „**PowerShell**” introducând comanda **python** urmată de numele fișierului script, inclusiv extensia.
3. Utilizarea unui mediu de dezvoltare (IDE – Integrated Development Environment)

Mediul de dezvoltare (IDE – Integrated Development Environment) facilitează dezvoltarea aplicațiilor prin îmbinarea, într-o singură interfață, a unei serii de instrumente pentru dezvoltare precum: managementul aplicației, editor de cod sursă, managementul librăriilor, depanare, etc. În cadrul acestui curs, se va folosi versiunea community a mediului de dezvoltare **PyCharm**. Descărcarea kit-ului de instalare al acestui IDE se poate face din secțiunea „**Downloads**” a site-ului oficial PyCharm: <https://www.jetbrains.com/pycharm/>.



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and the current project name "Aplicatii Python - main.py". The left sidebar displays the "Project" structure, showing a folder named "Aplicatii Python" containing a "venv" folder and a file named "main.py". The main code editor window shows the following Python code:

```
def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}') # Press Ctrl+F8 to toggle the breakpoint.

# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

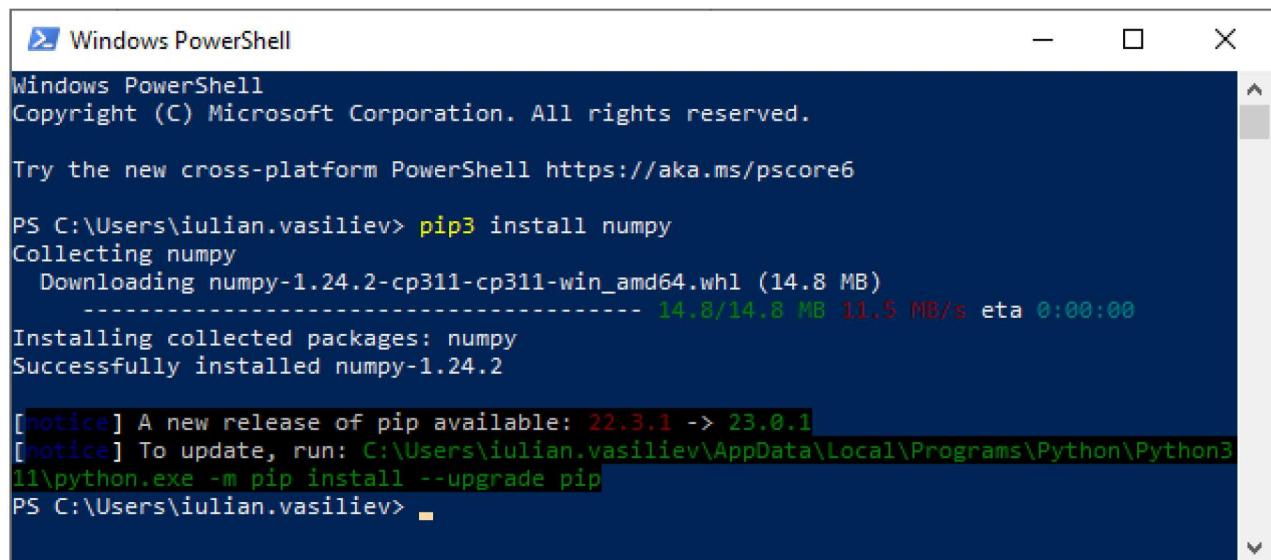
# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

The "Run" tool window at the bottom shows the command: "C:\Users\Iulian Vasiliev\Desktop\Aplicatii Python\venv\Scripts\python.exe" "C:\Users\Iulian Vasiliev\Desktop\Aplicatii Python\main.py". The output pane shows "Hi, PyCharm" and "Process finished with exit code 0".

Extinderea capacitațiilor de bază ale limbajului **Python** se face cu ajutorul librăriilor. Librăriile sunt colecții de module specializate pe anumite funcționalități (ex: **numpy** – lucru cu matrice multi-dimensionale, **Matplotlib** – afișarea grafică a datelor, etc.). Aceste librării sunt disponibile în mod gratuit și se instalează diferit în funcție de modul de rulare al aplicațiilor **Python**.

### Instalare

În cazul în care Python se utilizează în mod interactiv sau scripturile sunt rulate din linie comandă, instalarea unei librării se face din „**Command Prompt**” sau „**PowerShell**” folosind comanda `pip3 install` urmată de numele librăriei ce se dorește instalată.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

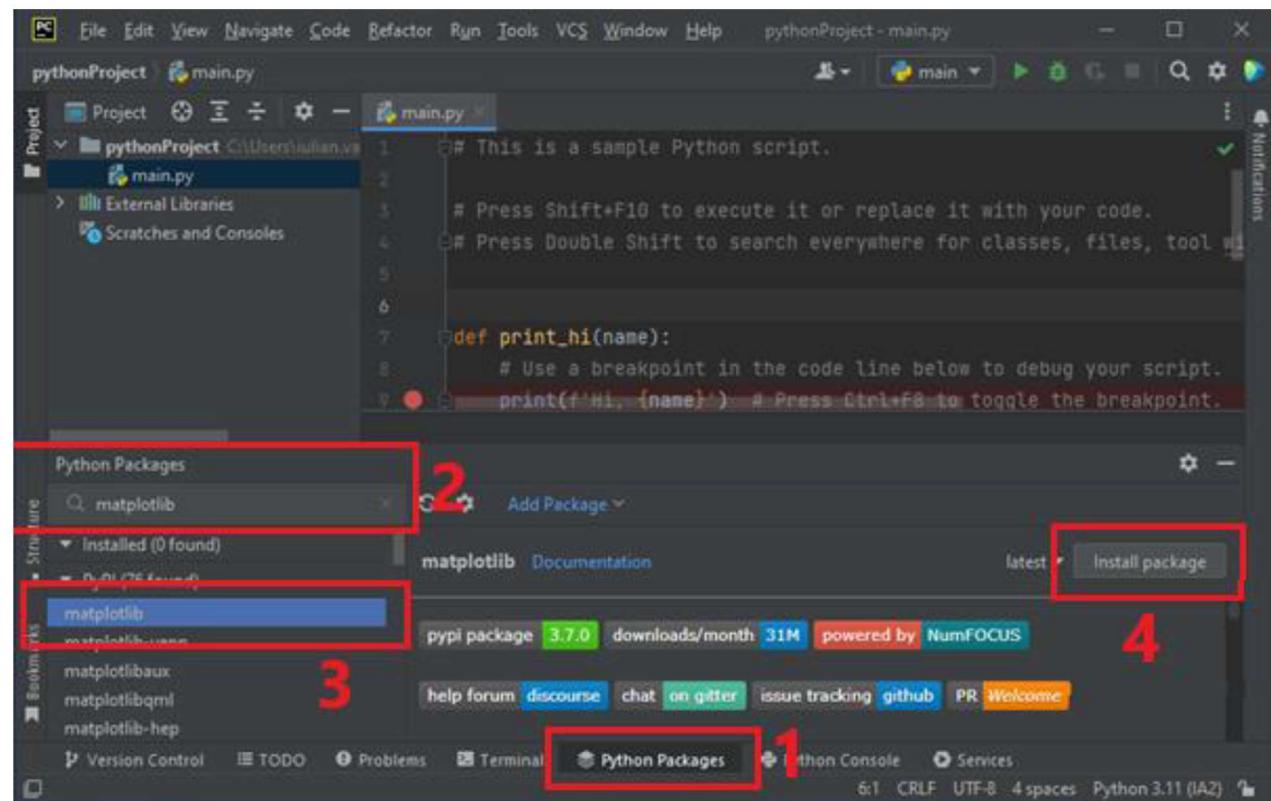
PS C:\Users\iulian.vasiliev> pip3 install numpy
Collecting numpy
  Downloading numpy-1.24.2-cp311-cp311-win_amd64.whl (14.8 MB)
    14.8/14.8 MB 11.5 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.24.2

[notice] A new release of pip available: 22.3.1 -> 23.0.1
[notice] To update, run: C:\Users\iulian.vasiliev\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
PS C:\Users\iulian.vasiliev>
```

## Instalare

În cazul în care se utilizează un mediu de dezvoltare pentru crearea aplicațiilor **Python**, instalarea unei librării poate dифeri de la IDE la IDE. În cazul **PyCharm**, instalarea unei librării implică o serie de pași:

1. se accesează sub-fereastra „**Python Packages**” din partea de jos a ferestrei principale.
2. se scrie numele librăriei în câmpul de căutare al sub-ferestrei.
3. se selectează librăria ce se dorește instalată.
4. se apasă butonul „**Install package**”



**1. Introducere**

**2. Bazele Python**

**3. Structuri de control**

# Introducere

Structuri de control

1. Introducere

2. Bazele Python

3. Structuri de control

Introducere

# Bazele Python

Structuri de control

### Funcția print

Una dintre cele mai simple instrucțiuni de bază ale limbajului **Python** este funcția `print` ce poate fi folosită la afișarea mesajelor în consolă.

### Comentarii

În limbajul **Python** există două tipuri de comentarii în cod

- comentarii pe o singură linie, acestea fiind precedate de simbolul `#`
- comentarii pe mai multe linii, acestea fiind cuprinse între seturi de ghilimelele `"""`

### Exemplu

`main.py`

```
print("Procesarea Datelor")
# Exemplu de comentariu pe o singură linie
"""

Exemplu de comentariu
pe mai multe linii
"""
```

`rezultat`

Procesarea Datelor

**Important !!!**

1. **NU** este necesar să punem simbolul ; (punct și virgulă) la sfârșitul liniilor de cod
2. identarea este foarte important:
  - o linie de cod este identată **DOAR DACĂ** ea aparține interiorului unei structuri de control (condițională, buclă, funcție, etc.)
  - **Python NU** utilizează accolade pentru a defini blocurile interioare structurilor de control, ci identarea
  - Identarea se poate face atât cu spații, cât și folosind caracterul Tab
  - O identare greșită va duce la returnarea unei erori

**INCORECT****main.py**

```
def test():
    x = 7
    y = 2 # Identare greșită

    if x < y:
        print("x este mai mic decât y")
    else:
print("x nu este mai mic decât y") # Identare greșită

test()
```

**rezultat**

```
File "C:\User\Desktop\test.py", line 3
    y = 2 # Identare greșită
IndentationError: unexpected indent
```

**CORECT****main.py**

```
def test():
    x = 7
    y = 2 # Identare greșită

    if x < y:
        print("x este mai mic decât y")
    else:
        print("x nu este mai mic decât y") # Identare greșită

test()
```

**rezultat**

```
x nu este mai mic decât y
```

**Python** oferă mai multe tipuri de date, dintre care cele mai simple sunt cele numerice și sirurile de caractere. Datorită orientării spre obiecte a acestui limbaj, orice variabilă **Python** poate fi văzută ca un obiect. În **Python** nu trebuie declarat tipul de variabilă înainte. Astfel, pentru a defini și atribui o valoare unei variabile se va scrie numele variabilei, urmat de simbolul = și de valoarea pe care o atribuim.

**Important !!!**

- Fiecare variabilă trebuie să aibă nume unic
- Nu sunt permise spații în numele variabilelor
- Numele variabilelor trebuie să înceapă cu literă sau cu „\_”
- Cifrele sunt permise în numele variabilelor (dar nu ca prim caracter)
- Variabilele sunt CASE-SENSITIVE;
- Variabilele își pot schimba valorile și tipul de date în timpul execuției programului (suprascriere).

Limbajul **Python** oferă suport pentru trei tipuri de date numerice:

### 1. numere întregi

Exemplu	
main.py	rezultat
<pre># numere intregi nr_intreg = 42 print(nr_intreg)</pre>	42

### 2. numere în virgulă mobilă

Exemplu	
main.py	rezultat
<pre># numere in virgula mobila nr_real1 = 7.0 print(nr_real1) nr_real2 = float(7) print(nr_real2)</pre>	7.0 7.0

### 3. numere complexe

Exemplu	
main.py	rezultat
<pre># numere complexe nr_complex = 7 + 0j print(nr_complex)</pre>	(7+0j)

Un alt tip de date de bază este sirul de caractere, numit și string. Acesta se poate defini folosind atât ghilimele simple, cât și ghilimele duble.

Exemplu	
main.py	rezultat
<pre>sir_de_caractere = "Procesarea Datelor" print(sir_de_caractere)</pre>	Procesarea Datelor

Un alt tip de date disponibil în **Python** este lista, reprezentând un vector unidimensional ce poate conține orice număr de elemente de orice tip. Valoarea acesteia se poate inițializa enumerând valorile elementelor acesteia, separate de virgulă între un set de paranteze drepte. Ulterior, un element al listei se poate accesa scriind numele variabilei urmat de indexul elementului între paranteze pătrate. În **Python** numărarea index-urilor începe de la 0.

Exemplu	
main.py	rezultat
<pre>lista = [3, 2, 1] print(lista) print(lista[1])</pre>	[3, 2, 1] 2

Operatori matematici de bază:

- |                       |                        |
|-----------------------|------------------------|
| 1. adunarea           | – operatorul <b>+</b>  |
| 2. scădere            | – operatorul <b>-</b>  |
| 3. înmulțire          | – operatorul <b>*</b>  |
| 4. împărțire          | – operatorul <b>/</b>  |
| 5. ridicare la putere | – operatorul <b>**</b> |
| 6. modulo (rest)      | – operatorul <b>%</b>  |
| 7. împărțire (cât)    | – operatorul <b>//</b> |

Adunarea și înmulțirea se pot utiliza și cu siruri de caractere

Inserarea unei valori numerice într-un sir de caractere se poate face cu ***Literal String Interpolation***

Ordinea operațiilor este aceeași din matematică, existând și posibilitatea grupării cu paranteze rotunde.

Exemplu	
main.py	rezultat
<pre>rezultat = 5 + 2 * 4 ** (2 - 1) - (3 + 2j) / 1 print(rezultat)</pre>	(10-2j)

Exemplu	
main.py	rezultat
<pre>print("Procesarea" + " " + "Datelor") print("Laborator" * 3)</pre>	Procesarea Dator LaboratorLaboratorLaborator

Exemplu	
main.py	rezultat
<pre>nume = "Procesarea Datelor" nr_curs = 1 print(f"Acesta este cursul numarul {nr_curs} de {nume}")</pre>	Acesta este cursul numarul 1 de Procesarea Datelor

Mare parte din funcțiile matematice de bază nu sunt disponibile direct din limbajul de bază, fiind necesară utilizarea librăriei **math** (preinstalată). Pentru a importa o librărie în codul aplicației se va folosi instrucțiunea `import` urmată de numele librăriei. Funcțiile și constantele definite de acea librărie vor putea fi accesate scriind numele librăriei, urmat de simbolul punct și de numele constantei / funcției dorite.

#### Constante Matematice

`math.pi` Constanta matematică  $\pi$

`math.e` Constanta matematică e (numărul lui Euler)

#### Funcții pentru reprezentarea numerelor

`math.ceil(x)` Cel mai mic număr întreg mai mare sau egal cu x

`math.floor(x)` Cel mai mare număr întreg mai mic sau egal cu x

`round(x)` Cel mai apropiat întreg de numărul x – funcția este în limbajul de bază (nu trebuie importată librăria math)

`abs(x)` Modulul numărului x – funcția este în limbajul de bază

#### Funcții exponentiale și logaritmice

`math.exp(x)` e la puterea x

`math.log(x, b)` Logaritm în bază b din x (dacă b lipsește va calcula logaritm natural)

`math.log2(x)` Logaritm în baza 2 din x

`math.log10(x)` Logaritm în baza 10 din x

`math.sqrt(x)` Rădăcina pătrată a lui x

#### Funcții trigonometrice

`math.sin(x)` Sinus de x (x în radiani)

`math.cos(x)` Cosinus de x (x în radiani)

`math.tan(x)` Tangent de x (x în radiani)

`math.asin(x)` Arc sinus de x în radiani (rezultatul între  $-\pi/2$  și  $\pi/2$ )

`math.acos(x)` Arc cosinus de x în radiani (rezultatul între 0 și  $\pi$ )

`math.atan(x)` Arc tangent de x în radiani (rezultatul între  $-\pi/2$  și  $\pi/2$ )

`math.atan2(y, x)` Arc tangent de y / x în radiani (rezultatul între  $-\pi$  și  $\pi$ )

#### Funcții de conversie unghiulară

`math.degrees(x)` Convertește în grade unghiul x specificat în radiani

`math.radians(x)` Convertește în radiani unghiul x specificat în grade

**Constante Matematice**

**math.pi** Constanta matematică  $\pi$

**math.e** Constanta matematică e (numărul lui Euler)

**Funcții pentru reprezentarea numerelor**

**math.ceil(x)** Cel mai mic număr întreg mai mare sau egal cu x

**math.floor(x)** Cel mai mare număr întreg mai mic sau egal cu x

**round(x)** Cel mai apropiat întreg de numărul x – funcția este în limbajul de bază (nu trebuie importată librăria math)

**abs(x)** Modulul numărului x – funcția este în limbajul de bază

**Funcții exponentiale și logaritmice**

**math.exp(x)** e la puterea x

**math.log(x, b)** Logaritm în bază b din x (dacă b lipsește va calcula logaritm natural)

**math.log2(x)** Logaritm în baza 2 din x

**math.log10(x)** Logaritm în baza 10 din x

**math.sqrt(x)** Rădăcina pătrată a lui x

**Funcții trigonometrice**

**math.sin(x)** Sinus de x (x în radiani)

**math.cos(x)** Cosinus de x (x în radiani)

**math.tan(x)** Tangent de x (x în radiani)

**math.asin(x)** Arc sinus de x în radiani (rezultatul între  $-\pi/2$  și  $\pi/2$ )

**math.acos(x)** Arc cosinus de x în radiani (rezultatul între 0 și  $\pi$ )

**math.atan(x)** Arc tangent de x în radiani (rezultatul între  $-\pi/2$  și  $\pi/2$ )

**math.atan2(y, x)** Arc tangent de y / x în radiani (rezultatul între  $-\pi$  și  $\pi$ )

**Funcții de conversie unghiulară**

**math.degrees(x)** Convertește în grade unghiul x specificat în radiani

**math.radians(x)** Convertește în radiani unghiul x specificat în grade

**Exemplu**

**main.py**

```
import math
print(math.sin(2 * math.pi / 3))
```

**rezultat**

0.8660254037844387

**Python** utilizează logica booleană pentru a evalua condiții.

Evaluarea unei expresii logice (condiții) poate returna una din cele două valori booleene: **True** sau **False**

### Operatori logici:

Operator	Descriere	Exemplu pentru <code>x = 2</code>	Rezultat
<code>==</code>	egal cu	<code>print(x == 2)</code>	<code>True</code>
<code>!=</code>	diferit de	<code>print(x != 2)</code>	<code>False</code>
<code>&lt;</code>	mai mic ca	<code>print(x &lt; 2)</code>	<code>False</code>
<code>&lt;=</code>	mai mic sau egal cu	<code>print(x &lt;= 2)</code>	<code>True</code>
<code>&gt;</code>	mai mare ca	<code>print(x &gt; 2)</code>	<code>False</code>
<code>&gt;=</code>	mai mare sau egal cu	<code>print(x &gt;= 2)</code>	<code>True</code>

**Operatorul `in`** verifică dacă o valoare se regăsește într-un obiect iterabil.

Exemplu	
main.py	rezultat
<pre>lista_forme = ["triunghi", "patrat", "cerc"] print("triunghi" in lista_forme) print("hexagon" in lista_forme)</pre>	<pre>True False</pre>

### Operatori booleeni:

Operator	Descriere	Exemplu	Rezultat
<code>not</code>	negarea expresiei precedate de operator	<code>not True</code>	<code>False</code>
<code>or</code>	operația de „sau logic” între două expresii	<code>True or False</code>	<code>True</code>
<code>and</code>	operația de „și logic” între două expresii	<code>True and False</code>	<code>False</code>

1. Introducere

2. Bazele Python

3. Structuri de control

Introducere

# Bazele Python

Structuri de control

1. Introducere

2. Bazele Python

3. Structuri de control

Introducere

# Structuri de control

- Structura condițională permite executarea unor secvențe de cod doar dacă anumite condiții sunt îndeplinite.
- Utilizează 3 cuvinte cheie: `if`, `elif` și `else`
- Identarea este foarte importantă (nu se utilizează accolade pentru delimitarea blocurilor de cod)
- După definirea fiecărei ramuri, înainte de blocul aferent acelei ramuri, se pune simbolul “două puncte” (`:`)
- Ramurile `elif` și `else` pot lipsi din structurile condiționale
- Pot exista mai multe ramuri `elif`

### Cum funcționează?

- Blocul de cod interior ramurii `if` se execută doar dacă expresia logică `expr_logica1` dată acestei ramuri are valoarea **True**.
- Blocul de cod interior ramurii `elif` se execută doar dacă expresia logică `expr_logica2` dată acestei ramuri are valoarea **True** și dacă expresiile logice date ramurilor precedente au avut valoarea **False**.
- Blocul de cod aferent ramurii `else` se execută dacă nici una din expresiile logice aferent ramurilor precedente nu au avut valoarea **True**.

### Sintaxa generală

```
if expr_logica1:  
    # bloc de cod - ramura if  
    ...  
elif expr_logica2: # else if  
    # bloc de cod - ramura elif  
    ...  
# alte ramuri elif  
else:  
    # bloc de cod - ramura else  
    ...
```

- Structura condițională permite executarea unor secvențe de cod doar dacă anumite condiții sunt îndeplinite.
- Utilizează 3 cuvinte cheie: `if`, `elif` și `else`
- Identarea este foarte importantă (nu se utilizează accolade pentru delimitarea blocurilor de cod)
- După definirea fiecărei ramuri, înainte de blocul aferent acelei ramuri, se pune simbolul “două puncte” (`:`)
- Ramurile `elif` și `else` pot lipsi din structurile condiționale
- Pot exista mai multe ramuri `elif`

### Sintaxa generală

```
if expr_logica1:
    # bloc de cod - ramura if
    ...
elif expr_logica2: # else if
    # bloc de cod - ramura elif
    ...
# alte ramuri elif
else:
    # bloc de cod - ramura else
    ...
```

### Exemplu

#### main.py

```
x = 33
if x % 2 == 0:
    print("x este divizibil cu 2!")
elif x % 5 == 0:
    print("x este divizibil cu 5!")
else:
    print("x nu este divizibil nici cu 2, nici cu 5!")
```

#### rezultat

`x nu este divizibil nici cu 2, nici cu 5!`

- Structura repetitive **while** permite repetarea unui bloc de cod atât timp cât o condiție este adevărată.
- Identarea este foarte important (nu se utilizează acolade pentru delimitarea blocului de cod)
- După definirea condiției, înainte de blocul de cod ce se dorește repetat, se pune simbolul “două puncte” (:)

**Sintaxa generală**

```
while expr_logica:  
    # bloc de cod  
    ...
```

**Exemplu****main.py**

```
a = 0
while a < 4:
    print(f"Iteratia {a}")
    a = a + 1
```

**rezultat**

Iteratia 0  
Iteratia 1  
Iteratia 2  
Iteratia 3

- Structura repetitive **for** permite repetarea unei secvențe de cod pentru fiecare element al unui obiect iterabil.
- Identarea este foarte important (nu se utilizează accolade pentru delimitarea blocului de cod)
- După instrucțiunea **for**, înainte de blocul de cod ce se dorește repetat, se pune simbolul “două puncte” (:)

**Sintaxa generală**

```
for variabila in obiect_iterabil:
    # bloc de cod
    ...

```

**Exemplu**

main.py	rezultat
<pre>lista_forme = ["triunghi", "patrat", "cerc"] for forma in lista_forme:     print(forma)</pre>	triunghi patrat cerc

- Funcția **range(start, stop, step)**, poate fi folosită pentru a crea un obiect iterabil ce conține o secvență de numere

Parametru	Descriere	Observație
<b>start</b>	valoarea de start a secvenței	poate lipsi, caz în care se consideră valoarea implicită 0
<b>stop</b>	valoarea de stop a secvenței	nu poate lipsi; de asemenea, valoarea de stop a secvenței nu este inclusă în secvență
<b>step</b>	pasul de incrementare	poate lipsi, valoarea implicită 1

**Exemplu**

main.py	rezultat
<pre>for x in range(1, 7, 2):     print(x)</pre>	1 3 5

- În buclele `while` și `for` se pot utiliza două comenzi pentru a opri execuția blocului de cod

Comanda	Descriere
<code>break</code>	oprește execuția blocului de cod și ieșe din buclă
<code>continue</code>	oprește execuția blocului de cod și continuă cu următoarea iterare a buclei

Exemplu	
main.py	rezultat
<pre># Exemplul 1: bucla se va opri cand cnt ajunge la 4 cnt = 0 while True:     print(cnt)     cnt = cnt + 1     if cnt &gt;= 4:         break  # Exemplul 2: Va afisa doar numerele pare for cnt in range(10):     if cnt % 2 != 0:         continue     print(cnt)</pre>	<pre>1 3 5 7 9  0 2 4 6 8</pre>

- Buclele `while` și `for` permit utilizarea unei ramuri `else`, caz în care blocul de cod aferent ramurii `else` se va executa doar dacă bucla **NU** este oprită de o instrucțiune `break`

#### Exemplul 1

main.py	rezultat
<pre>for i in range(5):     print(i)     if i &gt;= 5:         break else:     print("Ramura Else")</pre>	0 1 2 3 4 Ramura Else

#### Exemplul 2

main.py	rezultat
<pre>for i in range(5):     print(i)     if i &gt;= 3:         break else:     print("Ramura Else")</pre>	0 1 2 3

1. Introducere

2. Bazele Python

3. Structuri de control

Introducere

Bazele Python

# Structuri de control

**1. Introducere**

**2. Bazele Python**

**3. Structuri de control**

**Introducere**

**Bazele Python**

**Structuri de control**