

Programarea algoritmilor

Marinescu-Ghemeci Ruxandra

ruxandra.marinescu@fmi.unibuc.ro

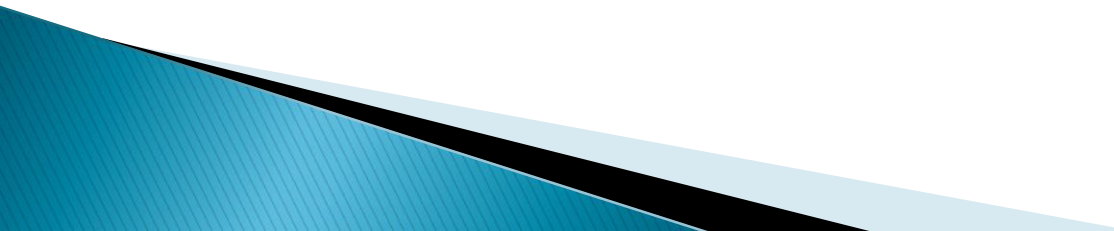
ruxandra.marinescu@unibuc.ro

Programa



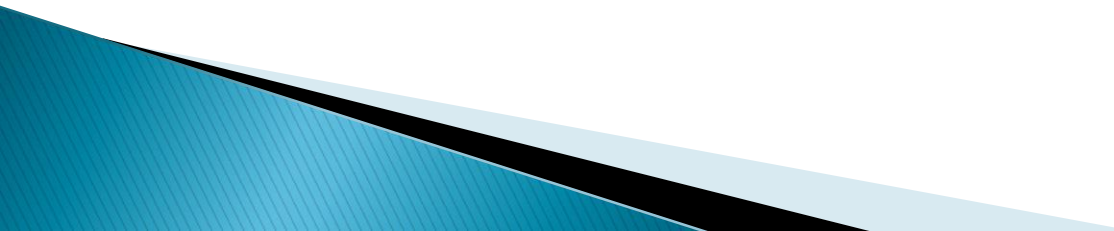
Programa

► Introducere în limbajul Python

- Elemente de bază
 - Colecții
 - Șiruri de caractere
 - Funcții
 - Fișiere
 - Excepții
- 

Programa

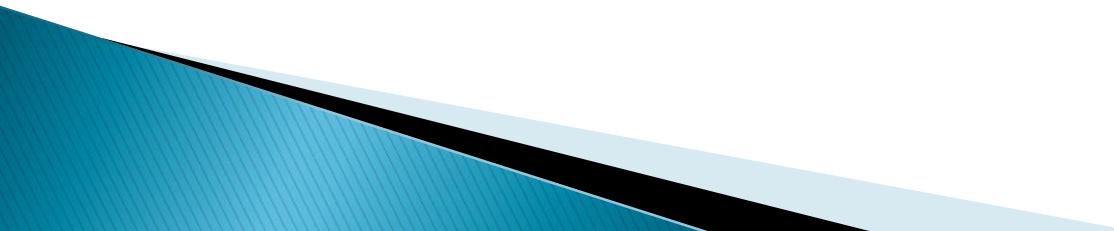
► Tehnici de programare:

- Algoritmi. Complexitate
 - Greedy
 - Divide et Impera
 - Programare dinamica
 - Backtracking
- 

Obiective generale

- ▶ Însușirea principalelor tehnici de elaborare a algoritmilor și a tipurilor de probleme la care se pretează acestea
- ▶ Însușirea elementelor de bază ale limbajului Python, utilizarea corectă a structurilor de date și algoritmilor puși la dispoziție de acest limbaj pentru implementarea algoritmilor elaborați

Obiective specifice

- ▶ cunoașterea principalelor tehnici de programare
 - ▶ abilități de utilizare a **structurilor de date** și tehnicilor potrivite în rezolvarea unei probleme
 - ▶ dezvoltarea unei gândiri algoritmice
 - ▶ abilități de justificare a **corectitudinii** algoritmilor propuși și de determinare a **complexității** acestora
 - ▶ abilități de implementare a algoritmilor în limbajul Python, de testare
- 

Obiective. Motivații

▶ Python

- elemente de bază
- lucrul cu structuri de date

Obiective. Motivații

► Python – avantaje

- sintaxa simplă, sugestivă
- dinamic
- de actualitate
- numeroase facilități (incluse automat): dezvoltare software, web, GUI, module pentru IA, ML (Google – motoare de căutare)
- portabil
- open-source: www.python.org
- garbage collection

Objective. Motivații

► Tehnici de programare

- algoritmi eficienți

"Perhaps the most important principle for the good algorithm designer is to refuse to be content" –

Aho, Hopcroft, and Ullman, The Design and Analysis of Computer Algorithms

Obiective. Motivații

► Tehnici de programare

- algoritmi eficienți

Exemple de probleme

- Aflarea minimului și maximului dintr-un vector
- Cele mai apropiate două puncte dintr-o mulțime de puncte din plan dată
- Numărul de inversiuni dintr-un vector
- Înmulțirea a două numere / matrice

Obiective. Motivații

► Tehnici de programare

- algoritmi corecți

Exemple de probleme

- Dată o mulțime de intervale, să se determine o submulțime de cardinal maxim de intervale care nu se suprapun
- Dată o mulțime de intervale, fiecare interval având asociată o pondere, să se determine o submulțime de intervale care nu se suprapun având ponderea totală maximă

Obiective. Motivații

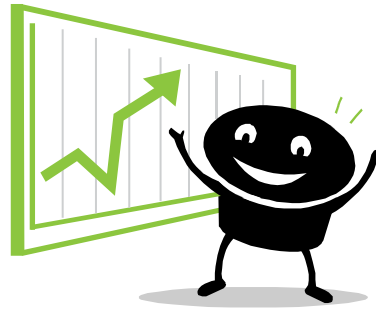
► Tehnici de programare

- algoritmi eficienți (chiar dacă există soluții evidente polinomiale – se poate mai bine?)
- corectitudinea algoritmilor – demonstrații
- probleme dificile \rightarrow NP-completitudine
- pentru ce tipuri de probleme se aplica metodele
- Complexitate – structuri de date

Objective. Motivații

- Numeroase aplicații
 - Bioinformatică, procesare texte, imagini
 - Geometrie computațională
 - Căutare web, similitudini, aliniere
 - Probleme de planificare
 - Proiectare, jocuri, strategii
 - Baze de date – arbori de căutare optimi
- Probleme interviuri

Evaluare



Structura


▶ Curs

- 2 ore pe săptămâna
- finalizat cu examen

▶ Laborator

- 2 ore la două săptămâni
- limbaj Python
- finalizat cu test de laborator (parte din examenul final)

▶ Seminar

- 2 ore la două săptămâni
 - discuții probleme curs/laborator
 - nu este notat separat, subiecte legate de seminar se vor regăsi la examen
- 

Evaluare

- ▶ Examen final (care va include și test de laborator)

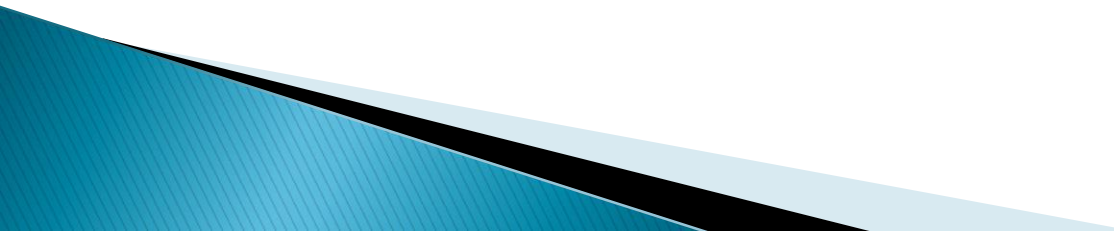
Condiție necesară:

Nota test laborator ≥ 5 puncte

BIBLIOGRAFIE

- ❖ Jon Kleinberg, Éva Tardos, **Algorithm Design**, Addison–Wesley 2005
<https://www.cs.princeton.edu/~wayne/kleinberg-tardos/>
- ❖ T.H. Cormen, C.E. Leiserson, R.R. Rivest – **Introducere in algoritmi**, Mit Press, trad. Computer Libris Agora
- ❖ S. Dasgupta, C.H. Papadimitriou, U.V. Vazirani, **Algorithms**, McGraw–Hill, 2008

BIBLIOGRAFIE

- ❖ Horia Georgescu. **Tehnici de programare.** Editura Universității din București 2005
 - ❖ Leon Livovschi, Horia Georgescu. **Sinteza și analiza algoritmilor.** 1986
 - ❖ Dana Lica, Mircea Pașoi, **Fundamentele programării,** L&S Infomat
- 

BIBLIOGRAFIE

- ❖ **coursera.org**

Algorithms, Part II – Princeton University

Algorithms: Design and Analysis – Stanford University

- ❖ **MIT** <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/>

- ❖ **infoarena.ro**

BIBLIOGRAFIE – Python

- ❖ <https://docs.python.org/3/>
- ❖ Magnus Lie Hetland– **Beginning Python From Novice to Professional** – Apress (2017)
- ❖ Naomi Ceder – **The Quick Python Book** –Manning Publications, 3rd ed (2018)

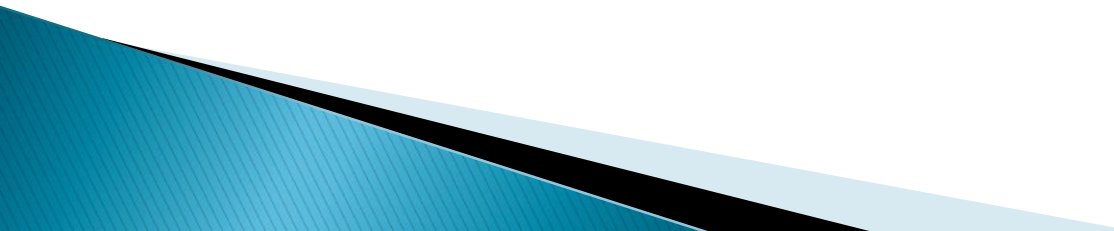
Despre algoritmi



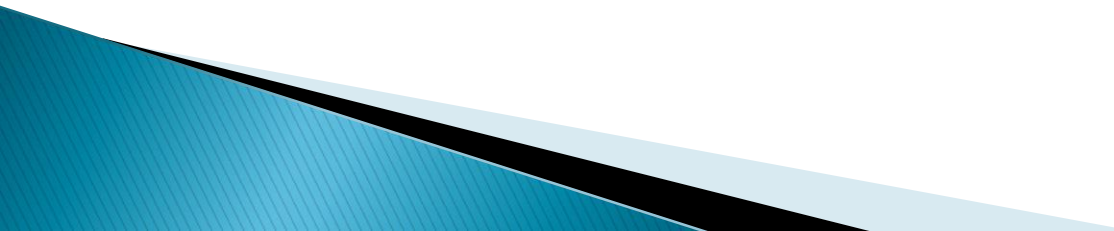
De ce despre algoritmi?

- ▶ numeroase aplicații
- ▶ în practică este importantă eficiența algoritmilor
- ▶ ar fi util să știm dacă algoritmii pe care îi propunem sunt corecți
 - 😊 corectitudine \neq nu a găsit cineva încă un contraexemplu

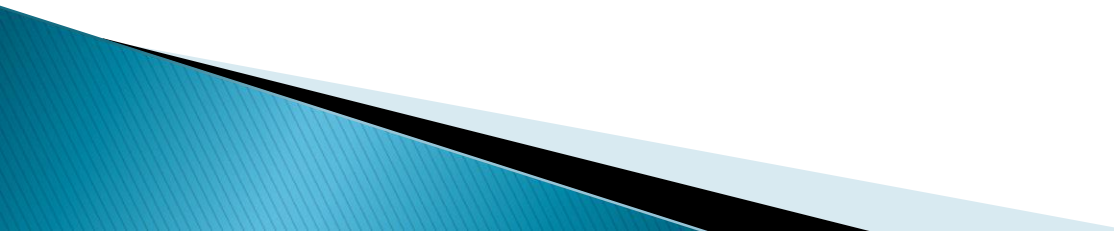
Aspecte generale care apar la rezolvarea unei probleme

- ▶ ***Teoretic***, pașii elaborării un algoritm sunt următorii:
 1. demonstrarea faptului că este **posibilă** elaborarea unui algoritm pentru determinarea unei soluții
 - 2.
 - 3.
 - 4.
 - 5.
- 

Aspecte generale care apar la rezolvarea unei probleme

- ▶ ***Teoretic***, pașii elaborării un algoritm sunt următorii:
 1. demonstrarea faptului că este **posibilă** elaborarea unui algoritm pentru determinarea unei soluții
 2. elaborarea algoritmului
 3. demonstrarea **corectitudinii** algoritmului
 - 4.
 - 5.
- 

Aspecte generale care apar la rezolvarea unei probleme

- ▶ ***Teoretic*, pașii elaborării un algoritm sunt următorii:**
 1. demonstrarea faptului că este **posibilă** elaborarea unui algoritm pentru determinarea unei soluții
 2. **elaborarea** algoritmului
 3. demonstrarea **corectitudinii** algoritmului
 4. determinarea **timpului de executare** a algoritmului
 5. demonstrarea **optimalității** algoritmului
- 

Aspecte generale care apar la rezolvarea unei probleme

- ▶ După elaborare – **Implementare**

=> limbaje de programare

Aspecte generale care apar la rezolvarea unei probleme

- ▶ După elaborare – **Implementare**

=> limbaje de programare

- ▶ Limbajul Python

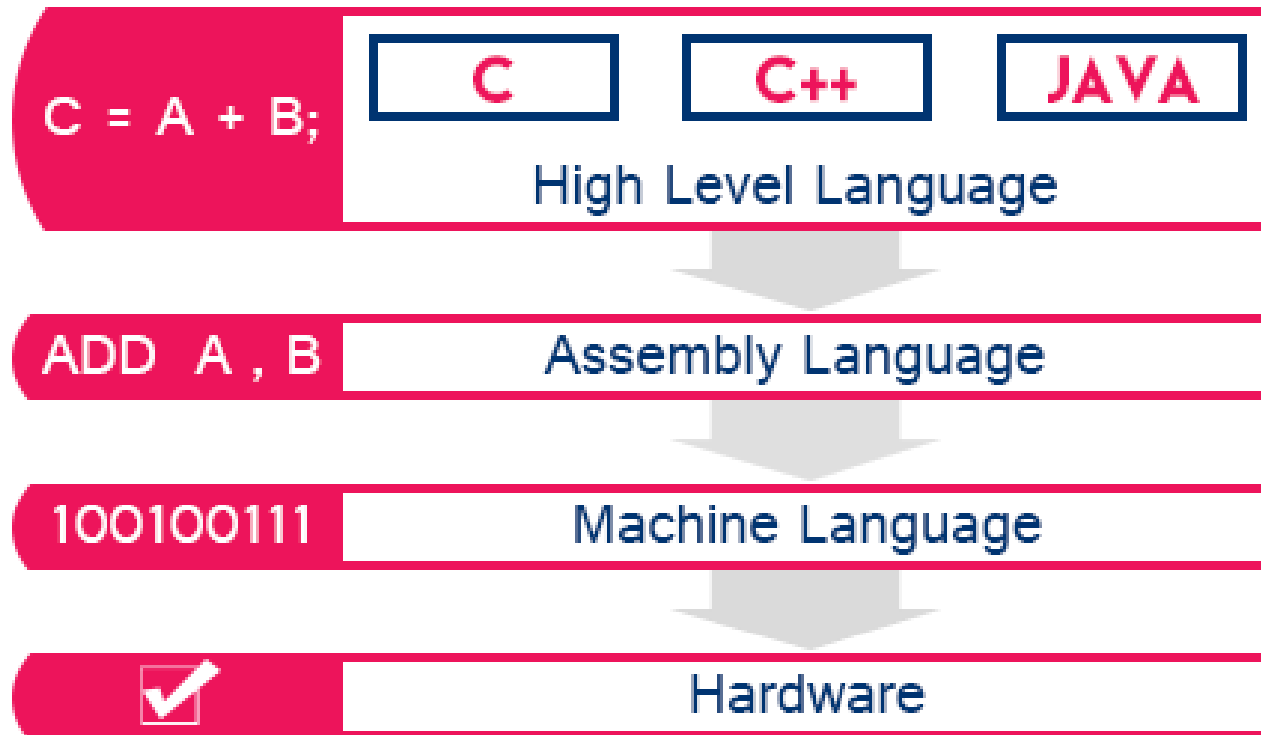
Despre limbaje de programare

Despre limbaje de programare

- ▶ Limbaj low level / high level
- ▶ Limbaj compilat / limbaj interpretat
- ▶ Paradigme de programare

Limbas low/high level

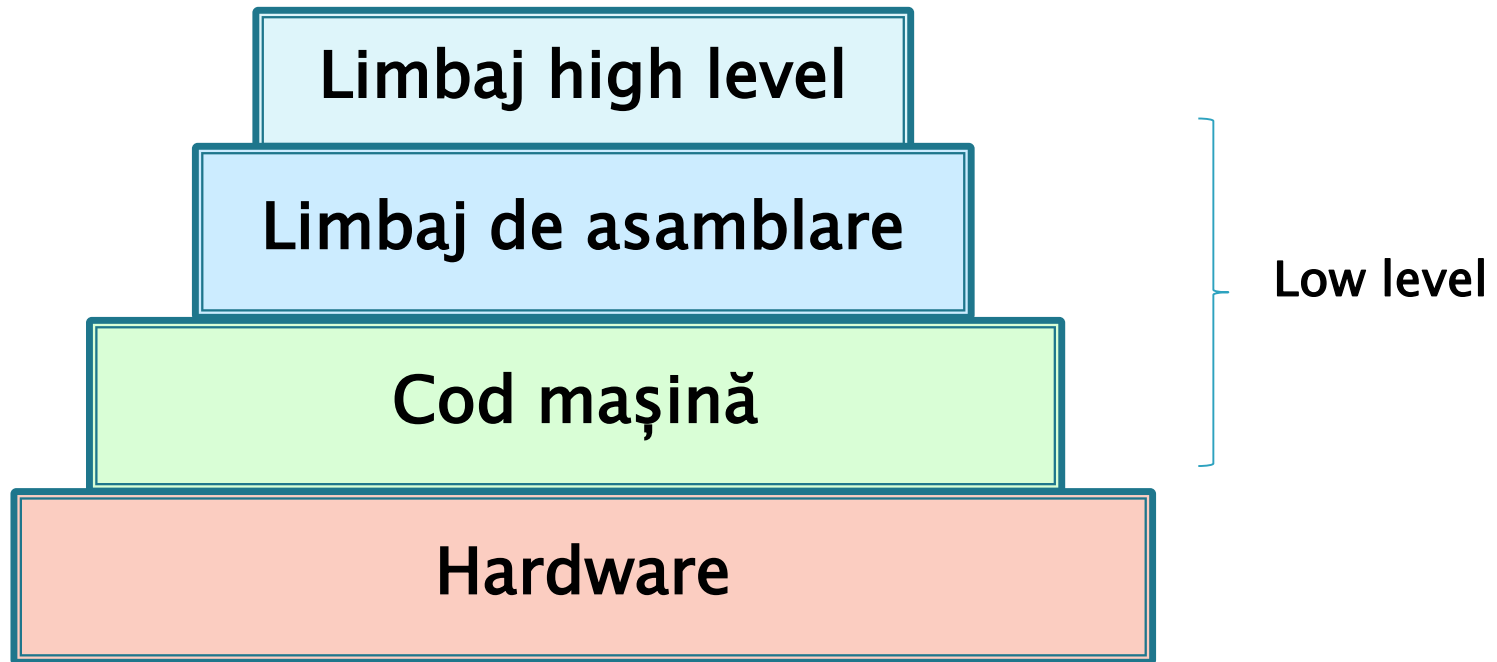
Limbaj low/high level



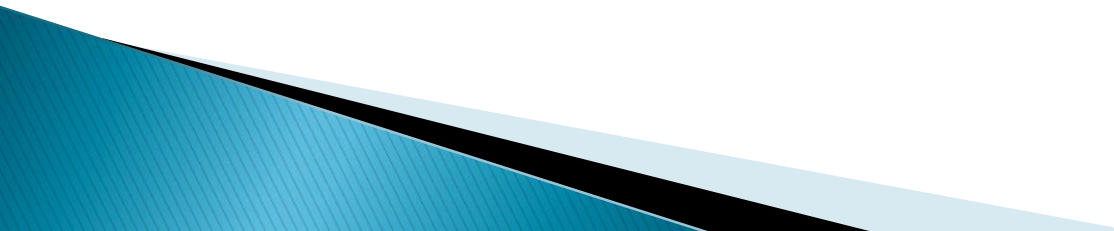
https://bournetocode.com/projects/GCSE_Computing_Fundamentals/pages/3-2-9-class_prog_langs.html

Limбай low/high level

- ▶ Clasificare în funcție de apropierea unui limbaj de limbajul mașină



Limbaaj low/high level

- ▶ **Limbaaj mașină** – cod binar, comunică direct cu hardware
 - ▶ **Limbaaj de asamblare** – o îmbunătățire: abrevieri de cuvinte în engleză pentru a specifica instrucțiuni (nu cod binar) => este necesar un “traducător” în cod mașină (Assambler)
 - ▶ **Limbaaj high level** – apropiat limbajului natural (uman)
- 

LIMBAJ LOW LEVEL	LIMBAJ HIGH LEVEL
rapide, utilizare eficientă a memoriei, comunică direct cu hardware	mai ușor de utilizat, de detectat erori, nivel mai mare de abstractizare
nu necesită compilator/interpretor	mai lente, trebuie traduse în cod mașină
cod dependent de mașină, greu de urmărit	independente de mașină, pot fi portabile
programatorul are nevoie de cunoștințe legate de arhitectura calculatorului pe care dezvoltă programul	nu comunică direct cu hardware => folosesc mai puțin eficient resursele
utilizate pentru dezvoltare SO, aplicații specifice	arie largă de aplicații

Limbaj compilat/ interpretat

Limbaș compilat / interpretat

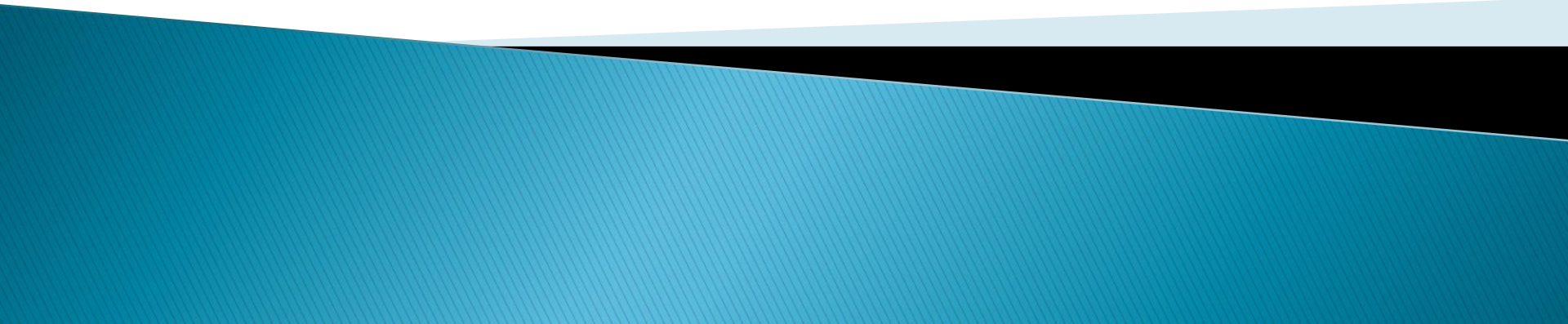
- ▶ **Compiler:** traduce codul sursă high-level în cod mașină (low-level, binar) => fișier care poate fi rulat pe sistemul de operare **pe care a fost creat** (tradus)
- ▶ **Interpreter:** traduce și execută instrucțiune cu instrucțiune

LIMBAJ COMPILAT	LIMBAJ INTERPRETAT
compilatorul “traduce” tot programul, dar după execuția este mai rapidă (poate fi executat de mai multe ori dacă nu se modifică sursa)	este mai lent
erorile sunt semnalate la finalul compilării (în procesul de compilare nu sunt executate instrucțiunile)	procesul de interpretare (cu executare instrucțiuni) se oprește la prima eroare
se distribuie executabilul (rezultatul compilării), nu sursa	nu este generat executabil, se distribuie sursa
nu portabil (executabilul se poate rula doar pe aceeași platformă)	este portabil este suficient să avem interpretorul (el e dependent de platformă)
trebuie recompilat după modificări	nesiguranța suportului
C, C++	JavaScript, PHP, Java???

Limbaș compilat / interpretat

- ▶ Nu se poate face mereu o încadrare strictă

Paradigme de programare



Paradigme de programare

- ▶ clasificarea limbajelor în funcție de stilul de programare și facilitățile oferite (controlul fluxului, modularitate, clase etc)

Paradigme de programare

- ▶ Programare imperativă
 - Programare procedurală
 - Programare orientată pe obiecte
 - Programare paralelă
- ▶ Programare declarativă
 - Programare logică
 - Programare funcțională
 - Programare la nivelul bazelor de date

Paradigme de programare

▶ Programare imperativă

- cea mai veche
- programatorul dă instrucțiuni mașinii (care trebuie executate în ordine) despre pașii care trebuie să îi execute
- cod mașină, Fortran, C, C++ etc

Paradigme de programare

▶ Programare procedurală

- program modularizat, bazat pe apeluri de proceduri
- C, Pascal

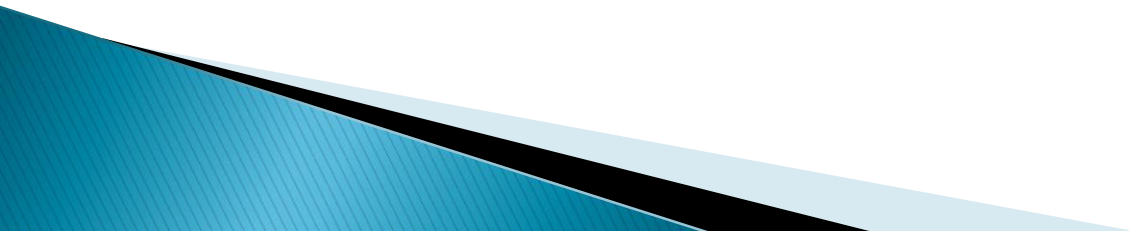
Paradigme de programare

▶ Programare procedurală

- program modularizat, bazat pe apeluri de proceduri
- C, Pascal

▶ Programare orientată pe obiecte

- bazată pe conceptul de obiecte (care interacționează)
- C++, Java, C#



Paradigme de programare

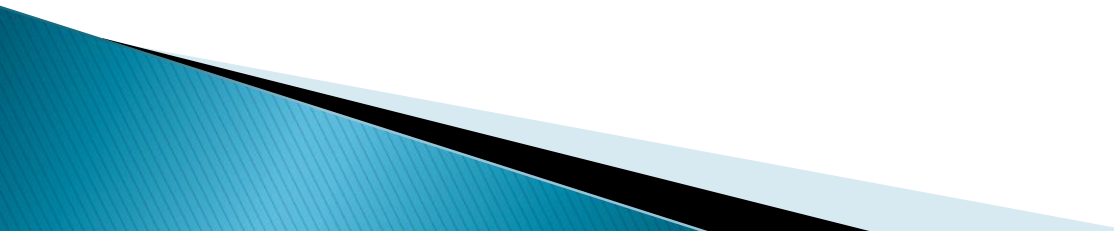
▶ Programare procedurală

- program modularizat, bazat pe apeluri de proceduri
- C, Pascal

▶ Programare orientată pe obiecte

- bazată pe conceptul de obiecte (care interacționează)
- C++, Java, C#

▶ Programare paralelă

- programul poate fi împărțit în seturi de instrucțiuni ce pot fi executate în paralel pe mai multe mașini
 - Ex.: Go, Java, Scala
- 

Paradigme de programare

▶ Programare declarativă

- programatorul declară proprietăți ale rezultatului dorit, nu cum se obține rezultatul

Paradigme de programare

▶ Programare logică

- Rezultatul este răspunsul la o interogare a unui sistem de date și reguli (bază de cunoștințe); execuția înseamnă activarea unui proces deductiv (bazat pe logică).
- Ex.: Prolog

Paradigme de programare

▶ Programare logică

- Rezultatul este răspunsul la o interogare a unui sistem de date și reguli (bază de cunoștințe); execuția înseamnă activarea unui proces deductiv (bazat pe logică).
- Ex.: Prolog

▶ Programare funcțională

- rezultatul este definit ca valoare a aplicării succesive ale unor funcții (matematice)
- Ex.: Haskell, Lisp, Scala

Paradigme de programare


▶ Programare logică

- Rezultatul este răspunsul la o interogare a unui sistem de date și reguli (bază de cunoștințe); execuția înseamnă activarea unui proces deductiv (bazat pe logică).
- Ex.: Prolog

▶ Programare funcțională

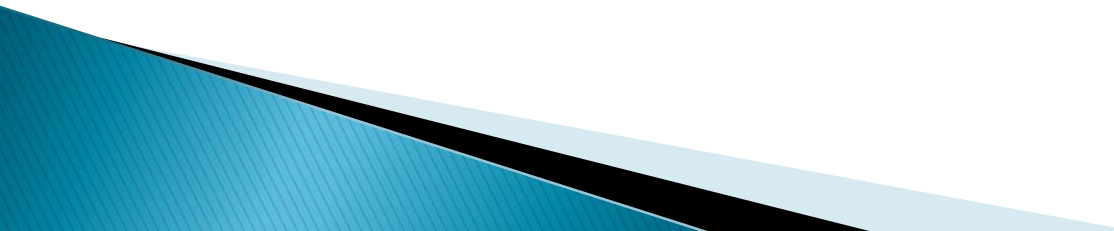
- rezultatul este definit ca valoare a aplicării succesive ale unor funcții (matematice)
- Ex.: Haskell, Lisp, Scala

▶ Programare la nivelul bazelor de date

- programul rezolvă cerințele unei gestiuni corecte și consistente a bazelor de date.
 - Ex.: FoxPro, SQL
- 

Limbajul Python

Limbajul Python

- high – level
 - Interpretat...
 - Paradigme de programare (hibrid):
 - procedural: subprograme și module
 - orientat obiect: clase
 - funcțional: funcții ca argumente ale altor funcții
(filter, map, lambda)
- 

Limbajul Python

- tip dinamic: **variabilele nu au tip de date static (declarat)**

Variabilelor li se pot asocia valori de tipuri diferite (valorile au tip) pe parcursul execuției programului
=> li se schimbă tipul

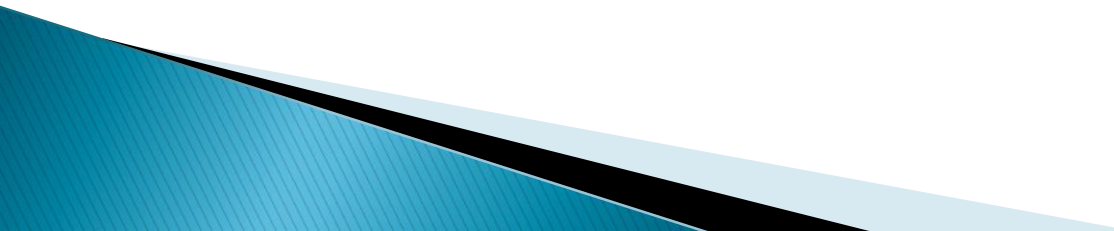
- orice valoare este un obiect, variabilele sunt referințe spre obiecte
- Garbage collector

Avantaje

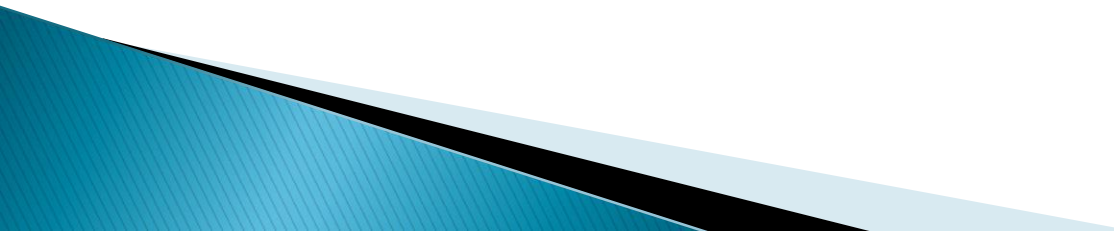
- Am discutat și la obiective și motivații



Dezavantaje

- mai lent – high level, interpretat
 - nu are atât de multe biblioteci ca alte limbaje, nu are suport pentru mobile
 - nu verifică tipurile de date ale variabilelor la compilare
 - nu folosește bine facilități precum procesoare multi-core
- 

Scurt istoric

- conceput de Guido van Rossum (în C) 1980
 - implementarea a început în decembrie 1989
 - lansări majore:
 - Python 1.0: ian. 1994
 - Python 2.0: oct. 2000
 - **Python 3.0**: dec. 2008
 - Python 3.8: 2019
- 

Scurt istoric

- ▶ Python 3.x **nu** este 100% compatibil cu Python 2.x

```
print "Pyhon 2"
```

```
print("Pyhon 3")
```

- ▶ Vom folosi Python 3.x

Scurt istoric

- este denumit după trupa de comedie / serialul BBC al anilor '70 Monty Python
- Tim Peters – set de principiile limbajului, îl putem afla cu instrucțiunea

```
import this
```



Cum rulăm o instrucțiune Python?

Instalare, rulare

- <https://www.python.org/downloads/>
- mai multe variante pot coexista
- variabila de mediu PATH – bifați la instalare opțiunea **Add python to PATH**

Instalare, rulare

- linie de comanda: comanda `python`

```
>>> import this
```

```
>>> print("Python 3")
```

- Medii de dezvoltare IDE : PyCharm, IDLE etc

Elemente de bază ale limbajului

- ▶ Nu sunt necesari delimitatori de blocuri de tip `{}` sau `begin/end` etc, este obligatorie indentarea blocurilor de cod (și **suficientă** pentru delimitarea acestora)
- ▶ Nu este nevoie sa punem `;` la finalul unei linii (dacă nu mai urmează alte linii de cod pe aceeași linie)

```
i = 1
while i<10:
    print(i)
    i = i + 1

print("gata afisarea")
```

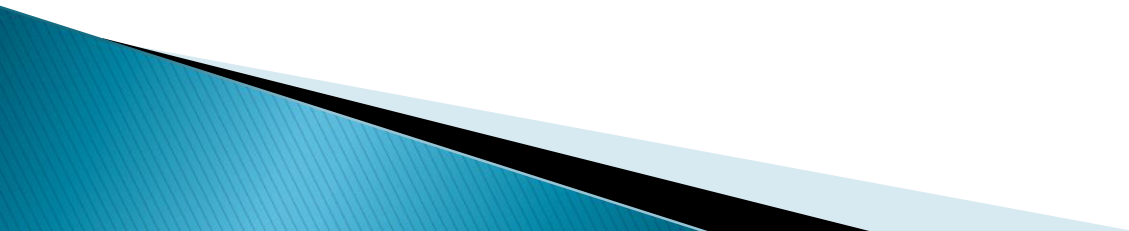
Elemente de bază ale limbajului

1. Afișarea unei variabile + tipul acesteia (al valorii asignate)



Elemente de bază ale limbajului

2. Citirea de la tastatură + funcții de conversie

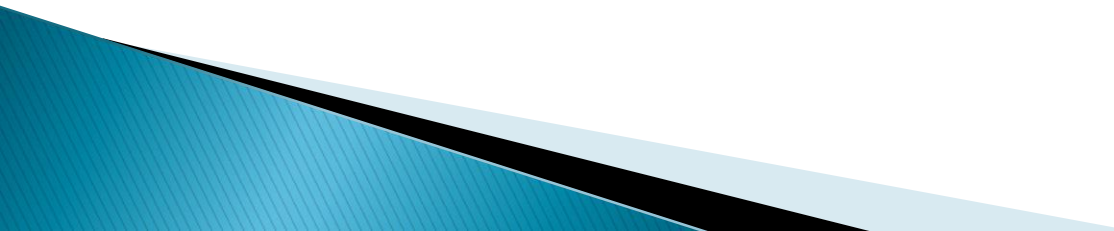


Elemente de bază ale limbajului

3.

```
i = 1  #i="ab", i=-1
print(i)
if i>0:
    print("ok")
else:
    print(i+" este negativ")
```

Variabile

- În C/C++ o variabilă are: tip, adresa, valoare
 - În Python variabilele sunt **referințe spre obiecte** (orice valoare este un obiect)
 - Un obiect **ob** are asociat:
 - un număr de identificare: **id(ob)**
 - un tip de date: **type(ob)**
 - o valoare – poate fi convertită la șir de caractere **str(ob)**
- 

Variable

`m = 10`

C

m: 10

Python

m → 10
are adresă

Variable

C

Python

```
m = 10
```

m: 10

m → 10
are adresă

```
m = m+1
```

Variable

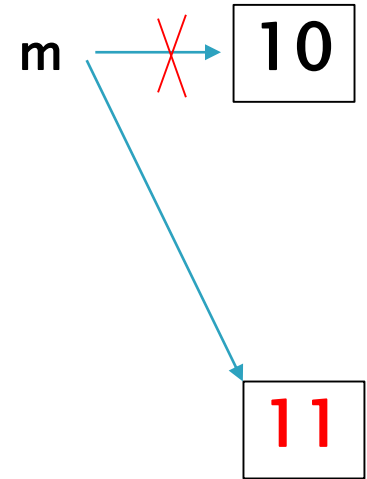
```
m = 10
```

```
m = m+1
```

C

m: 11

Python



Variable

```
m = 10
```

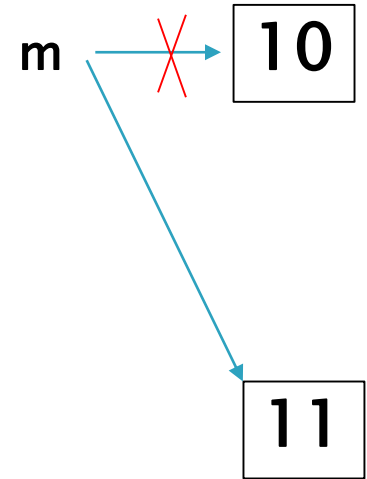
```
m = m+1
```

```
n = m
```

C

m: 11

Python




Variable

C

Python

```
m = 10
```

m: 11

m  10

```
m = m+1
```

n: 11

 11

```
n = m
```

n 

Variable

```
m = 10
```

```
m = m+1
```

```
n = m
```

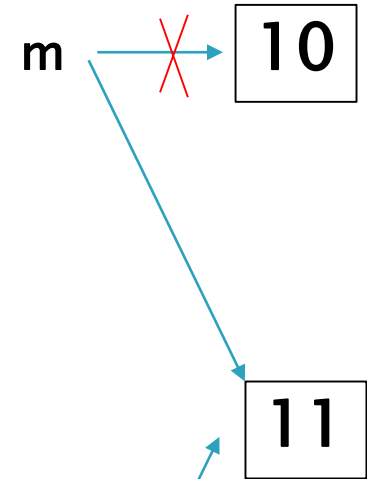
```
n = n+1 ???
```

C

m: 11

n: 11

Python



n

Variabile

- Tipul unei variabile se stabilește prin inițializare și se poate schimba prin atribuiri de valori de alt tip
- Numele unei variabile – identificatori
- Recomandare nume:

`litere_mici_separate_prin_underscore`



Variabile

- optimizare: numerele întregi din intervalul $[-5, 256]$ sunt prealocate (în cache) – **toate obiectele care au o astfel de valoare sunt identice (au același id)**
- variabile cu aceeași valoare **pot avea** același id (dacă este o valoare prealocată, atunci sigur da)

Variabile

▶ Exemple

```
x = 1
```

```
y = 0
```

```
y = y + 1
```

```
z = x
```

```
print(x,y,z,x*x)
```

```
print(id(x),id(y),id(z),id(x*x))
```

Variabile

▶ Exemple

```
x = 1000
```

```
y = 999
```

```
y = y+1
```

```
z = x
```

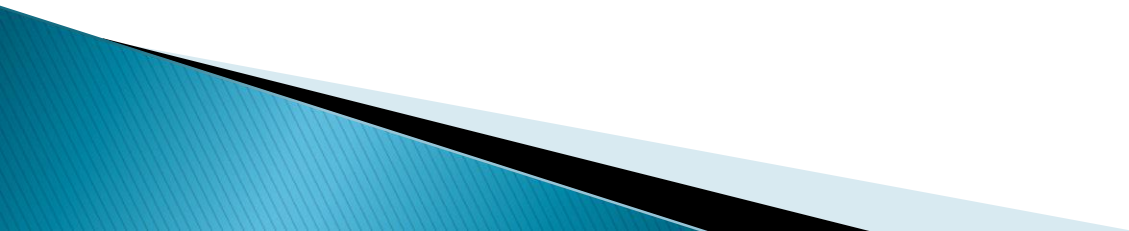
```
print(x,y,z,10*x//10)
```

```
print(id(x),id(y),id(z),id(10*x//10))
```

Variabile

- `del x` – șterge o variabilă din memorie
- Garbage collector – șterge obiecte către care nu mai sunt referințe

Tipuri de date



Tipuri de date

- **int**
 - numere întregi cu oricât de multe cifre (limita dată doar de performanța sistemului pe care se rulează)

Tipuri de date

- **int**

- numere întregi cu oricât de multe cifre (limita dată doar de performanța sistemului pe care se rulează)
- memorate ca vectori de “cifre” din reprezentarea în baza 2^{30} (cu cifre de la 0 la $2^{30}-1 = 1073741823$)

Exemplu: Reprezentarea pentru 234254646549834273498:

ob_size	3		
ob_digit	462328538	197050268	203

deoarece $234254646549834273498 = 462328538 \times (2^{30})^0 + 197050268 \times (2^{30})^1 + 203 \times (2^{30})^2$

Tipuri de date

- **int**

- constante în baza 10 (implicit), dar și în bazele 2 (prefix 0b,0B), 8 (prefix 0o, 0O), 16 (prefix 0x,0X):

```
print(0b101, 0o10, 0xAB)
```

Tipuri de date

- **int**

- constante în baza 10 (implicit), dar și în bazele 2 (prefix 0b,0B), 8 (prefix 0o, 0O), 16 (prefix 0x,0X):


```
print(0b101, 0o10, 0xAB)
```

- putem folosi `int(sir)` pentru creare/conversie (există și varianta `int(sir, base=baza)`)

```
int(9.5)    #round(9.5)
```

```
int("101", base=2)
```

```
int("101", 2)
```



Tipuri de date

- **float**

- IEEE-754 double precision
- Constante: 3.5, 1e-2 (notație științifică)
- float([x]):

```
float("inf"); float("infinity"); float("nan")
```

Tipuri de date

- `float`
 - operațiile aritmetice cu tipul de date *float* nu au precizie absolută:

NU: `0.1*0.1 == 0.01`

DA: `abs(0.1*0.1-0.01) < 1e-9`

Tipuri de date

- **complex**
 - de forma $a + bj$ (!!! nu i, merge și J)

Tipuri de date

- `complex`

```
z = complex(-1, 4)

print("Numarul complex:", z)

print("Partea reala:", z.real)

print("Partea imaginara:", z.imag)

print("Conjugatul:", z.conjugate())

print("Modul:", abs(z))
```

Tipuri de date

- **bool**
 - True, False
 - putem folosi `bool()` pentru conversie
 - În context boolean – **conversia oricărei valori la bool**

Context boolean – condiție if, while; operand pentru operatori logici – conversii

Tipuri de date

- **bool**
 - Se consideră **False**:
 - **None, False**
 - **0, 0.0, 0j, Decimal(0), Fraction(0,1)**
 - **Colecții și secvențe vide** (+obiecte în care `__bool__()` returnează False sau `__len__()` returnează 0)

```
print(bool(0), bool(-5))
```

```
print(bool(""), bool(" "))
```

```
print(bool(None), bool([]))
```

Tipuri de date

- NoneType

- **None**

- Nu exista char

`ord("a")`

`chr(97)`

Tipuri de date

Secvențe:

Mutable (le putem modifica) și imutable

- liste `list`: `a = [3, 1, 4, 7]` – mutable
- tupleuri `tuple`: `a = (3, 1, 4, 7)`
- șiruri de caractere `str`: `a = "3147sir"`

Tipuri de date

Mulțimi:

- set: $a = \{1, 4, 5\}$
- frozenset: $fa = \text{frozenset}(a)$ – nu se poate modifica

Dicționare

Operatori

- aritate (număr de operanzi)
- prioritate (precedența)
- asociativitatea: $x \text{ op } y \text{ op } z$

Operatori

- Operatori aritmetici

+	adunare
-	scădere
*	înmulțire
/	Împărțire exactă, rezultat float (nu ca în C/C++ sau Python 2)
//	împărțire cu rotunjire la cel mai apropiat întreg mai mic sau egal decât rezultatul împărțirii exacte dacă un operator este float rezultatul este de tip float
%	restul împărțirii
**	ridicare la putere

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

`3 + 2.0; 2j*3j`

`1/1; 1/2`

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

`3 + 2.0; 2j*3j`

`1/1; 1/2`

`4//2; 5//2.5`

`2.5//1.5`

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

`3 + 2.0; 2j*3j`

`1/1; 1/2`

`4//2; 5//2.5`

`2.5//1.5`

`11//3; 11//3; -11//3; -11//-3`

Operatori

- $x \% y = x - ((x // y) * y)$

`11%3; -11%3; 11%-3; -11 %-3`

`10.5%2; 3%1.5`

`10**7**2`

!!! Prioritățile și asociativitatea operatorilor

Operatori

- Operatori relaționali

$x == y$	x este egal cu y
$x != y$	x nu este egal cu y
$x > y$	x mai mare decât y
$x < y$	x mai mic decât y
$x >= y$	x mai mare sau egal y
$x <= y$	x mai mic sau egal y

Operatori

- Operatori relaționali

- Se pot înlănțui: $1 < x < 10$
- operatorul `is` – testeaza daca obiectele au *acelasi id*

```
x = 1000
```

```
y = 999
```

```
y = y+1
```

```
print(x == y)
```

```
print(x is y)
```

```
x = 1
```

```
y = 0
```

```
y = y+1
```

```
print(x == y)
```

```
print(x is y)
```

!! variabile din cache

Operatori

- Operatori de atribuire

=

+=, -=, *=, /=, **=, //=, %=,

&=, |=, ^=, >>=, <<= (v. operatori pe biți)

Operatori

- Operatori logici

`not, and, or`

- se evaluează prin scurtcircuitare
- în context Boolean orice valoare se poate evalua ca True/False;
=> operatorii logici nu se aplica doar pe valori de tip `bool` (ci pentru orice valori)

Operatori

- Operatori logici

$$x \text{ and } y = \begin{cases} x, & \text{dacă } x \text{ se evaluează ca } True \\ y, & \text{altfel} \end{cases}$$

$$x \text{ or } y = \begin{cases} x, & \text{dacă } x \text{ se evaluează ca } True \\ y, & \text{altfel} \end{cases}$$

$$\text{not } x = \begin{cases} False, & \text{dacă } x \text{ se evaluează ca } True \\ True, & \text{altfel} \end{cases}$$

Operatori

```
x = 0
```

```
y = 4
```

```
if x:
```

```
    print(x)
```

```
print(x and y)
```

```
print(x or y)
```

```
print(not x, not y)
```

```
print((x<y) and y)
```

```
print((x<y) or y)
```



Operatori

Observație: `not` are prioritate mai mică decât operatorii de alte tipuri:

`not a==b` \Leftrightarrow `not (a == b)`

`a == not b` eroare de sintaxă (trebuie `a == not(b)`)

Operatori

- Operatori pe biți

–rapizi, asupra reprezentării interne

$\sim x$	complement față de 1
$x \& y$	și pe biți
$x y$	sau pe biți
$x \wedge y$	sau exclusiv pe biți
$x \gg k$	deplasare la dreapta cu k biți
$x \ll k$	deplasare la stânga cu k biți

\wedge	0	1
0	0	1
1	1	0

Operatori

Observații:

$x = x \gg k \iff x = x // (2^{k})$**

$x = x ** k \iff x = x * (2^{k})$**

Exemple – seminar + laborator

1. Se citește un număr întreg x . Să se testeze dacă x este par
2. Să se interschimbe valorile a două variabile folosind \wedge

Operatori

```
x = 272
```

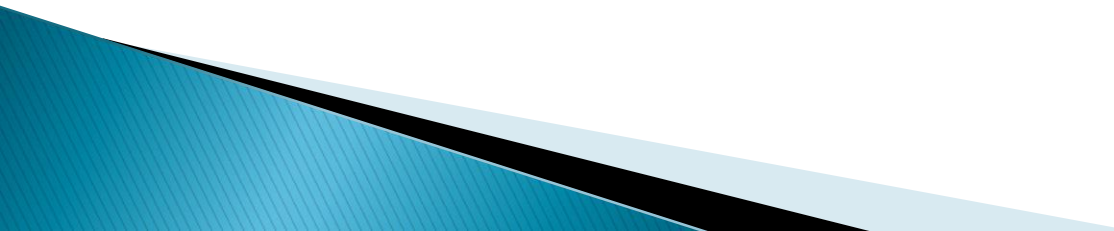
```
print(bin(x))
```

```
print(bin(x&0b10001), x&0b10001)
```

```
print(bin(17|0b10001), 17|0b10001)
```

```
print(bin(~x), ~x)
```

```
print(bin(x>>1), x>>1)
```



Operatori

- Operatorul condițional (ternar)

expresie_1 **if** *conditie* **else** *expresie_2*

Operatori

- Operatorul condițional (ternar)

expresie_1 **if** *conditie* **else** *expresie_2*

x = 5; y = 20

z = x-y if x>y else y-x

print(z)



Operatori

- Operatorul condițional (ternar)

expresie_1 **if** *conditie* **else** *expresie_2*

```
x = 5; y = 20
```

```
z = x-y if x>y else y-x
```

```
print(z)
```

- evaluat tot prin scurtcircuitare
- orice expresie

```
x = 5; y= 20
```

```
print(x) if x>y else print(y)
```

Operatori

- **Operatori de identitate:** `is`, `is not`
- **Operatori de apartenență :** `in`, `not in` (la o colecție)

Operatori

- **Precedența operatorilor:**

<https://docs.python.org/3/reference/expressions.html>

Comentarii

- Prefixat de # => comentariu pe o linie
- Pentru mai multe linii – # pe fiecare linie sau delimitatori de șiruri de caractere
 - Încadrat de ' ' ' => pe mai multe linii
 - Încadrat de " " " => docstring – comentariu pe mai multe linii, folosit în mod special pentru documentare

Instrucțiuni

- Instrucțiunea de atribuire

x = 1

x = y = 1

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

```
x, y = 1, 2 #atribuire de tupluri
```

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

```
x, y = 1, 2
```

```
x, y = y, x #!!! Interschimbare (tupluri)
```

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

```
x, y = 1, 2
```

```
x, y = y, x ### Interschimbare (tupluri)
```

```
x, y = min(x,y), max(x,y)
```

```
print("intervalul [" + str(x) + ", " + str(y) + "])"
```

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

```
x, y = 1, 2
```

```
x, y = y, x #!!! Interschimbare (tupluri)
```

```
v = [11, 12, 13, 14]
```

```
i = 2
```

```
i, v[i] = v[i], i !!!???????
```

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

```
x, y = 1, 2
```

```
x, y = y, x #!!! Interschimbare (tupluri)
```

```
v = [11, 12, 13, 14]
```

```
i = 2
```

```
i, v[i] = v[i], i #Nu, mai bine v[i], i = i, v[i]
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

```
x=int(input())
```

```
if x<0:
```

```
    print('valoare incorecta')
```

```
x=abs(int(input()))
```

```
if x%2==0:
```

```
    print('numar par')
```

```
else:
```

```
    print('numar impar')
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

```
k = int(input())  
print('ultima cifra a lui 3**',k, 'este',end=" ")
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

```
k = int(input())
print('ultima cifra a lui 3**',k, 'este',end=" ")
r = k%4
if r==0:
    print(1)
elif r==1:
    print(3)
elif r==2:
    print(9)
else:
    print(7)
```


Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

```
k = int(input())
print('ultima cifra a lui 3**',k, 'este',end=" ")
r = k%4
if r==0:
    print(1)
elif r==1:
    print(3)
elif r==2:
    print(9)
else:
    print(7)
```

- `else` poate lipsi
- Nu există `switch`

Instrucțiuni

- Instrucțiunea repetitiva cu test inițial while

`#suma cifrelor unui numar`

Instrucțiuni

- Instrucțiunea repetitiva cu test inițial while

```
#suma cifrelor unui numar
```

```
m = n = int(input())
```

```
s = 0
```

```
while n>0:
```

Instrucțiuni

- Instrucțiunea repetitiva cu test inițial while

```
#suma cifrelor unui numar
```

```
m = n = int(input())
```

```
s = 0
```

```
while n>0:
```

```
    s += n%10
```

```
    n //= 10 #!!nu /
```

```
print("suma cifrelor lui", m, "este",s)
```

Instrucțiuni

- **Instrucțiunea repetitiva cu test inițial while**
 - while poate avea else
 - Nu există do... while

Instrucțiuni

- Instrucțiunea repetitiva cu număr fix de iterații (for)

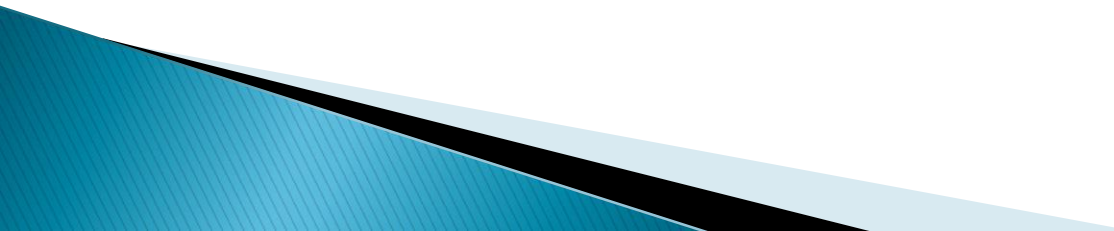
Doar “for each”, de forma

for *variabila* in *colectie_iterabila*

de exemplu:

```
for litera in sir:
```

```
for elem in lista:
```



Instrucțiuni

```
s = "abcde"
```

```
for litera in s:
```

```
    print(ord(litera))
```

```
for i in [0,1,2,3,4]:
```

```
    print(s[i])    #nu chiar pythonic
```

Instrucțiuni

- Instrucțiunea repetitiva cu număr fix de iterații (for)

for i in [0,1,2,3,4]



for i in [0,..., n] **????!**

Instrucțiuni

- Instrucțiunea repetitiva cu număr fix de iterații (for)

```
for i in [0,1,2,3,4]
```

```
for i in [0,..., n] ????
```



Funcția `range ()`

Instrucțiuni

- Instrucțiunea repetitiva cu număr fix de iterații (for)

```
for i in [0,1,2,3,4]
```

```
for i in [0,..., n] ????
```



Funcția `range()`

```
for i in range(0,n+1):
```

Instrucțiuni

- Funcția `range()` – clasa `range`, o secvență (iterabilă)

`range(b)` => de la 0 la $b-1$

`range(a,b)` => de la a la $b-1$

`range(a,b,pas)` => $a, a+p, a+2p...$

↑
`pas` poate fi negativ

Instrucțiuni

- Funcția `range()` – clasa `range`, o secvență (iterabilă)
 - memorie puțină, **un element este generat doar cand este nevoie de el**, nu se memorează toate de la început (secvența este generată **element cu element**)



Instrucțiuni

`range(10) =>`

`range(1,10) =>`

`range(1,10,2) =>`

`range(10,1,-2) =>`

`range(1,10,-2) =>`

Instrucțiuni

`range(10) => 0 1 2 3 4 5 6 7 8 9`

`range(1,10) => 1 2 3 4 5 6 7 8 9`

`range(1,10,2) => 1 3 5 7 9`

`range(10,1,-2) => 10 8 6 4 2`

`range(1,10,-2) => vid`

Instrucțiuni

```
print(*range(1,10,2))
```

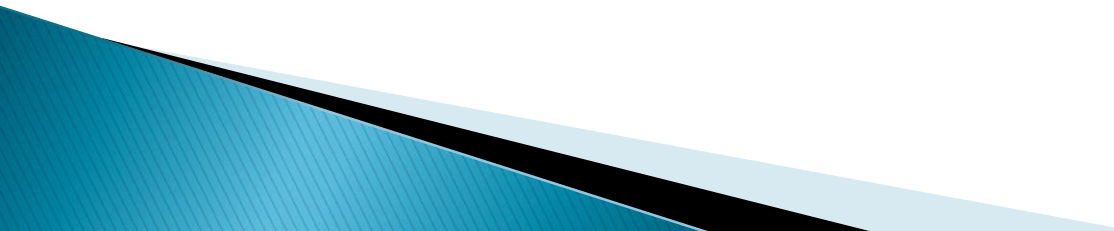
```
print(*range(10,1,-2))
```

```
print(*range(1,10,-2))
```

```
s = "abcde"
```

```
for i in range(len(s)):
```

```
    print(s[i])
```



Instrucțiuni

- **break, continue + clauza else pentru instrucțiuni repetitive**
 - **break, continue** – aceeași semnificație ca în C

Instrucțiuni

- **break, continue + clauza else pentru instrucțiuni repetitive**
 - **break, continue** – aceeași semnificație ca în C

```
while True:
```

```
    comanda = input('>> ')
```

```
    if comanda == 'exit()':
```

```
        break
```

Instrucțiuni

#primul divizor propriu



Instrucțiuni

```
#primul divizor propriu
```

```
x = int(input())
```

```
dx = None
```

```
for d in
```



Instrucțiuni

```
#primul divizor propriu
```

```
x = int(input())
```

```
dx = None
```

```
for d in range(2,x//2+1):
```

```
    if x%d == 0:
```

```
        dx = d
```

```
        break
```



Instrucțiuni

```
#primul divizor propriu
```

```
x = int(input())
```

```
dx = None
```

```
for d in range(2,x//2+1):
```

```
    if x%d == 0:
```

```
        dx = d
```

```
        break
```

```
if dx: #if dx is not None:
```

```
    print("primul divizor propriu:",dx)
```

```
else:
```

```
    print("numar prim")
```

Instrucțiuni

```
#numarul de divizori propria - cu continue
```

```
x = int(input())
```

```
k = 0
```

```
for d in range(2,x):
```

```
    if x%d != 0:
```

```
        continue
```

```
    k+=1
```

```
print("numarul de divizori proprii:",k)
```

Instrucțiuni

```
#numarul de divizori propria - cu continue
```

```
x = int(input())
```

```
k = 0
```

```
for d in range(2,x):
```

```
    if x%d != 0:
```

```
        continue
```

```
    k+=1
```

```
print("numarul de divizori proprii:",k)
```



Instrucțiuni

Clauza `else` a unei structure repetitive: nu se executa daca s-a iesit din ciclu cu `break`

```
x = int(input())  
for d in range(2,x//2+1):  
    if x%d == 0:  
        print("primul divizor propriu:",d)  
        break  
else: #al for-ului, nu al if-ului  
    print("numar prim")
```


Instrucțiuni

```
x = int(input())  
for d in range(2,x//2+1):  
    if x%d == 0:  
        print("primul divizor propriu:",d)  
        break  
else: #al for-ului, nu al if-ului  
    print("numar prim")
```

Instrucțiuni

Exercițiu: Date a și b , să se determine cel mai mic număr prim din intervalul $[a, b]$

Instrucțiuni

#cel mai mic număr prim din intervalul [a,b]

```
a = int(input())
```

```
b = int(input())
```

```
for x in range(a,b+1):
```

Instrucțiuni

#cel mai mic număr prim din intervalul [a,b]

```
a = int(input())
```

```
b = int(input())
```

```
for x in range(a,b+1):
```

```
    for d in range(2,x//2+1):
```

```
        if x%d == 0:
```

```
            break
```



Instrucțiuni

#cel mai mic număr prim din intervalul [a,b]

```
a = int(input())  
b = int(input())  
for x in range(a,b+1):  
    for d in range(2,x//2+1):  
        if x%d == 0:  
            break  
    else:  
        print(x)  
        break
```

Instrucțiuni

- **pass**

```
x=int(input())
```

```
if x<0:
```

```
    pass #urmeaza sa fie implementat
```

Funcții predefinite

- **Modulul builtins**

<https://docs.python.org/3/library/functions.html#built-in-funcs>

Funcții predefinite

- **Conversie**

– constructori `int()`, `float()`, `str()`

```
print(bin(23), hex(23)) #str
```


Funcții predefinite

- **Matematică**

```
print(abs(-5))
```

```
print(min(5,2))
```

```
x = 3.0
```

```
print(x.is_integer())
```

```
print(round(x + 0.7))
```

```
import math
```

```
print(math.sqrt(4))
```



