

Despre algoritmi

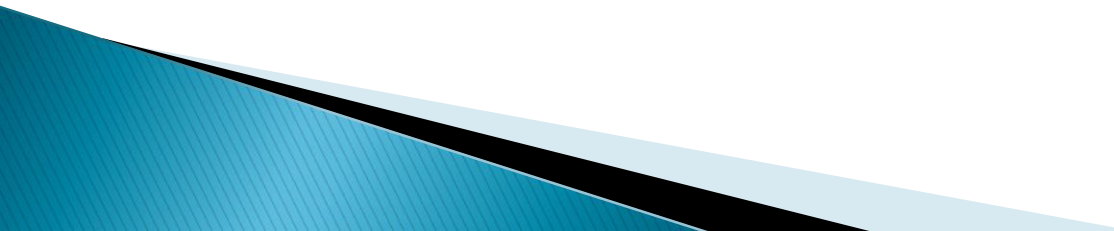


De ce despre algoritmi?

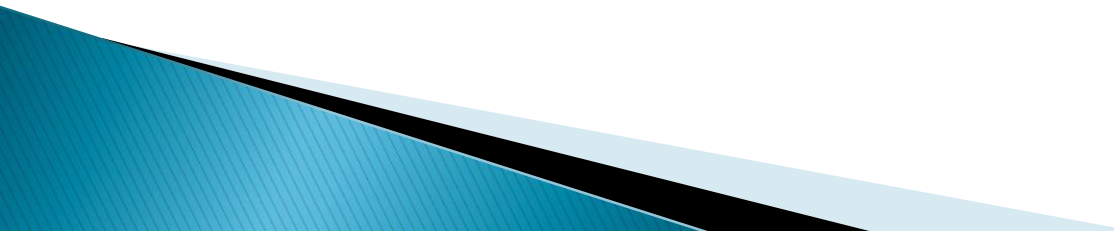
- ▶ numeroase aplicații
- ▶ în practică este importantă eficiența algoritmilor
- ▶ ar fi util să știm dacă algoritmii pe care îi propunem sunt corecți
 - 😊 corectitudine \neq nu a găsit cineva încă un contraexemplu

Aspecte generale care apar la rezolvarea unei probleme

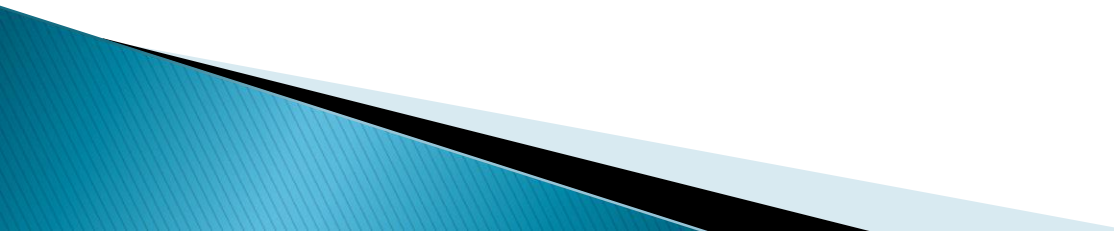
► ***Teoretic***, pașii elaborării un algoritm sunt următorii:

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

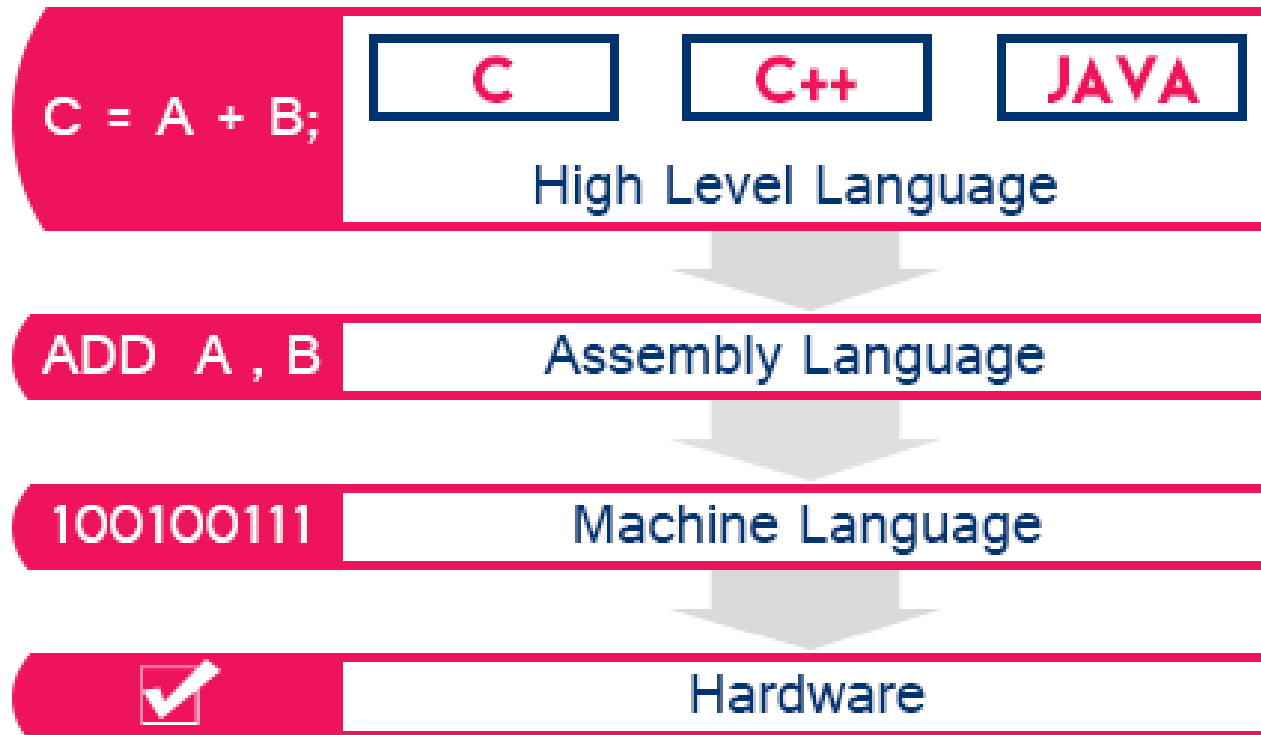
Despre limbaje de programare

Despre limbaje de programare

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

Limbas low/high level

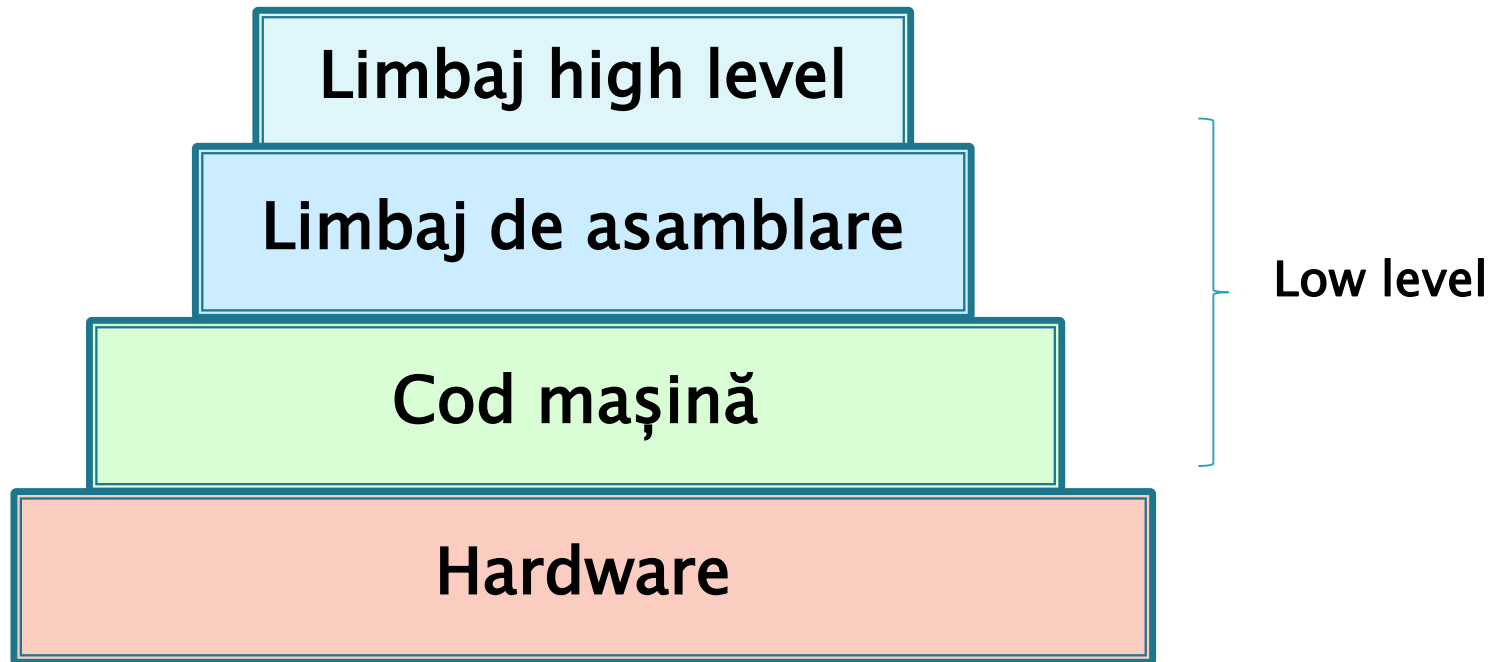
Limbaj low/high level



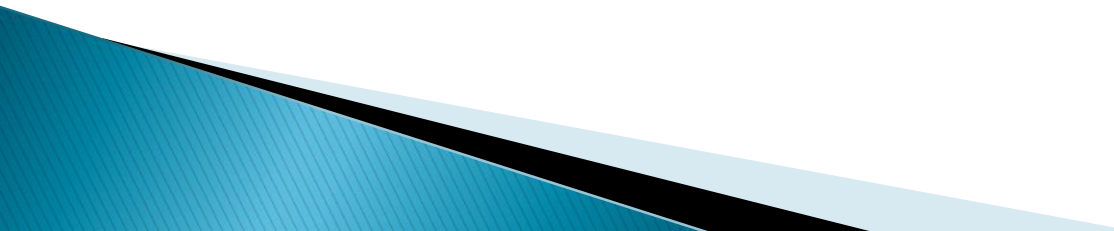
https://bournetocode.com/projects/GCSE_Computing_Fundamentals/pages/3-2-9-class_prog_langs.html

Limbaaj 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

rapide,
utilizare eficientă a memoriei,
comunică direct cu hardware

nu necesită compilator/interpretor

cod dependent de mașină, greu de urmărit

programatorul are nevoie de cunoștințe legate de arhitectura calculatorului pe care dezvoltă programul

utilizate pentru dezvoltare SO, aplicații specifice

LIMBAJ HIGH LEVEL

mai ușor de utilizat,
de detectat erori,
nivel mai mare de abstractizare

mai lente, trebuie traduse în cod mașină

independente de mașină,
pot fi portabile

nu comunică direct cu hardware
=> folosesc mai puțin eficient resursele

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

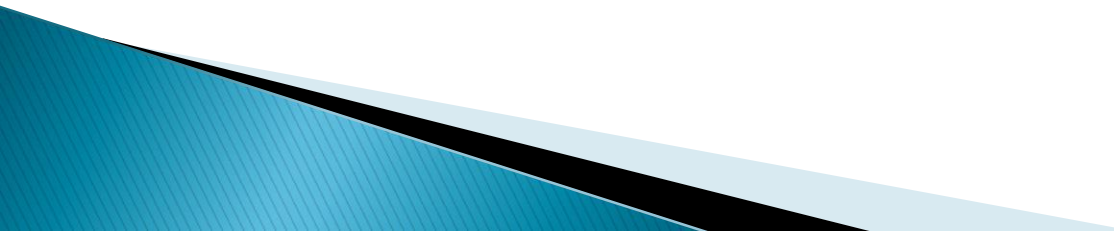
▶ Programare procedurală

- 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

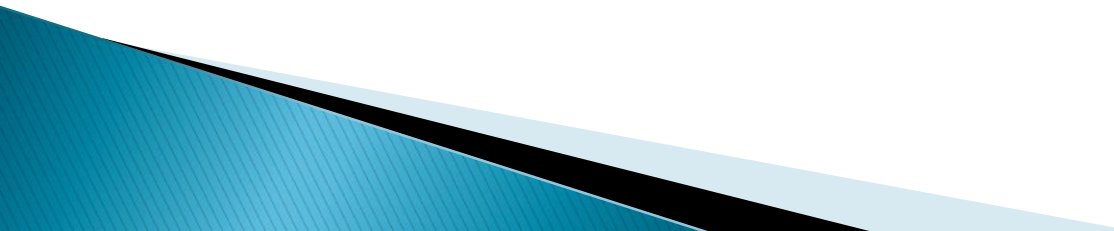
- rezultatul este definit ca valoare a aplicării succesive ale unor funcții (matematice)
- Ex.: Haskell, Lisp, Scala

▶ Programare la nivelul bazelor de date

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

Limbaajul Python

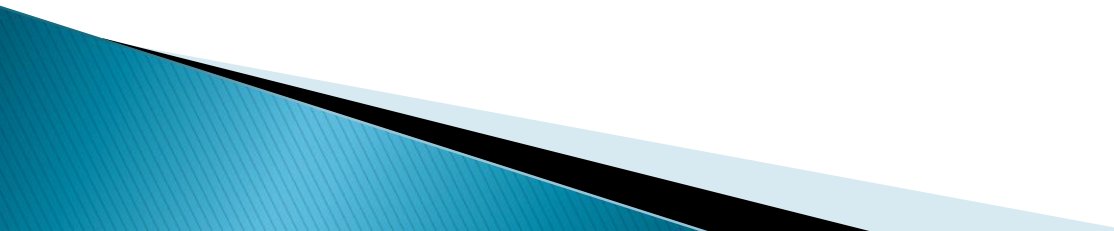
Limbajul Python

- 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

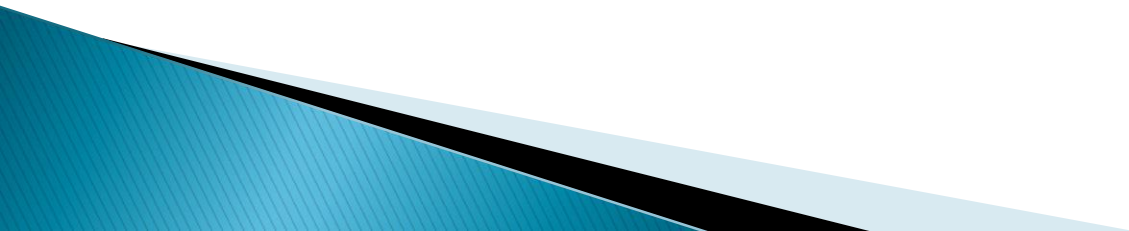
- **tip dinamic**: variabilele nu au tip de date static (declarat)

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

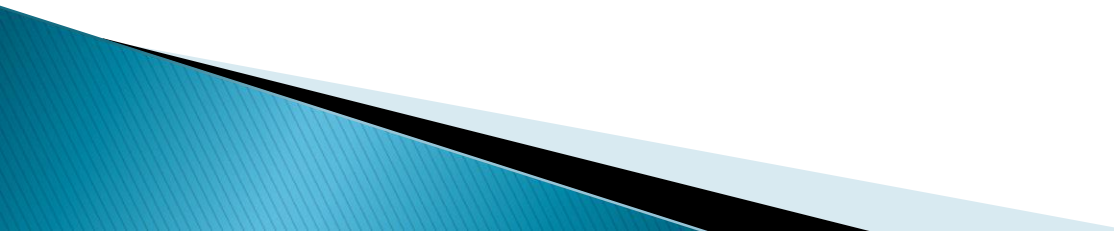
- orice valoare este un obiect, variabilele sunt referințe spre obiecte
 - Garbage collector
- 

Avantaje

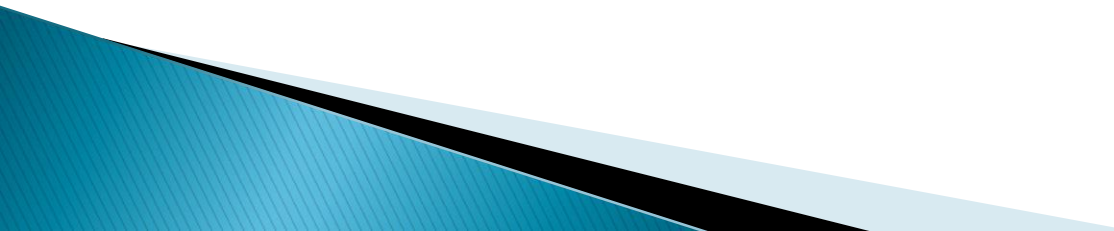
- Am discutat și la obiective și motivații



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, Spyder etc

