Seminar de programare în python

Programarea ca meșteșug

Constantin Radu Mirescu, 4 noiembrie 2020

PROGRAMAREA
(ca meserie)

Obiectul seminarului este câștigarea unei experiențe de programator pentru participanți.

Fiecare participant va scrie cod și teste.

Limbajul folosit va fi python care are un trend ascendent de popularitate.

NOTĂ: Seminarul se bazează pe experiența de programator a propunătorului care începe efectiv pe 1 noiembrie 2001.

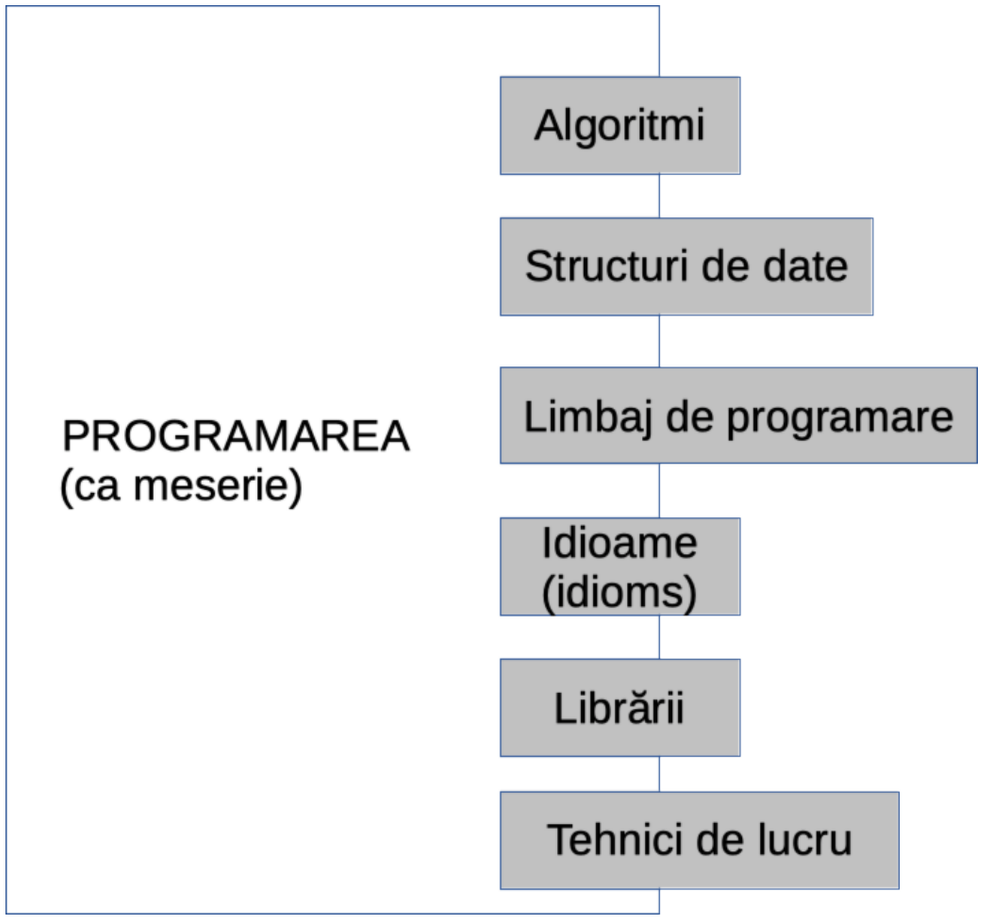
Fiind autodidact și fără a urma o programă analitică dintr-o facultate de profil este (foarte) posibil ca anumite conexiuni făcute aici să nu reflecte părerea unanim răspândită în rândul celor care urmează programa standard. Acest lucru nu înseamnă automat că ceea ce se spune aici este greșit.

Materialul reflectă nivelul de înțelegere actual al propunătorului și este conceput dintr-o perspectivă pragmatică.

Programarea ca meșteșug

Părți componente

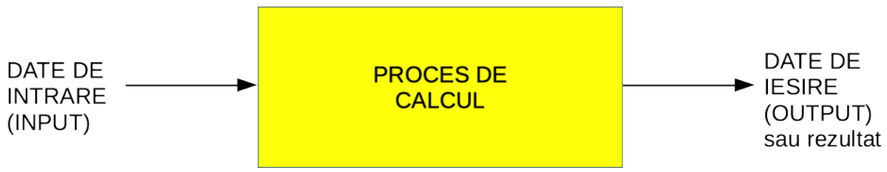
- Algoritmi
- Structuri de date
- Limbaj de programare
- Idioame (rețete corecte de utilizare a limbajului)
- Librării (biblioteci)
- Tehnici de lucru



Def1. A programa = a “pilota” un proces de calcul.

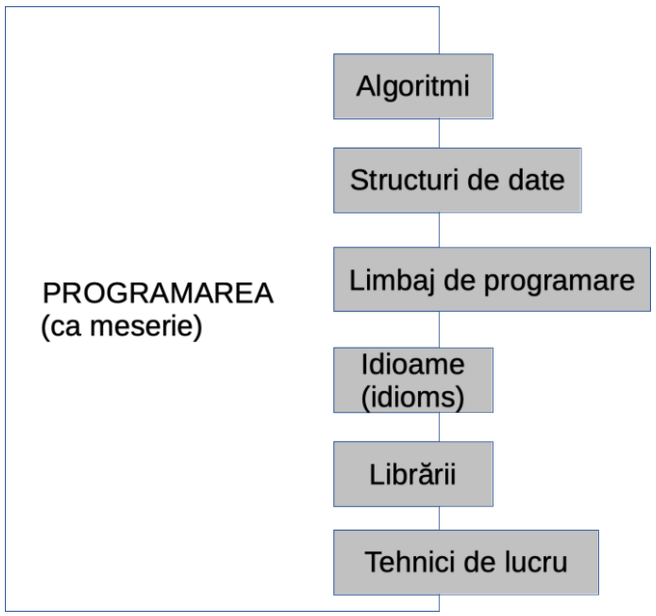
Def2. A programa = a “da comenzi” calculatorului

The Big Picture (Programarea din avion)



A programa înseamnă a prelua date de undeva, a le procesa și a obține în final rezultatul dorit.

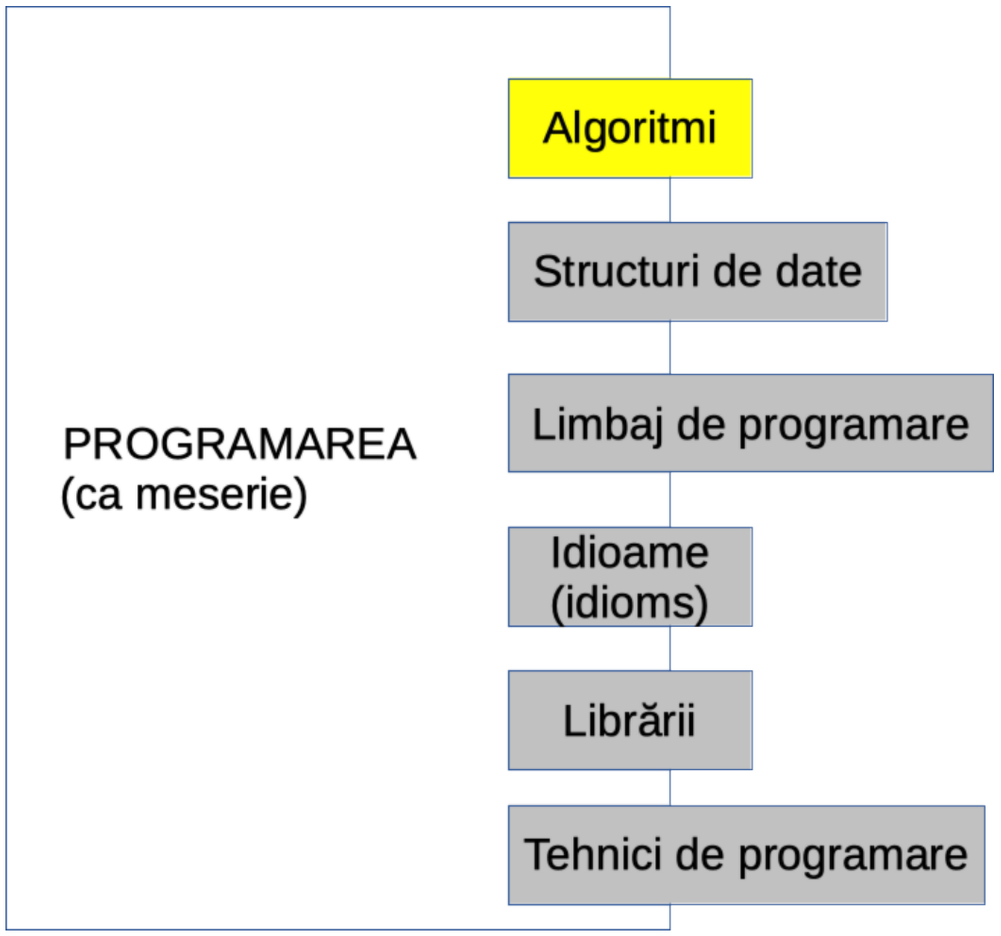
Datele sunt simple (numere, șiruri de caractere, valori de tip boolean) sau compuse - structurate în **structuri de date**. În seminar vom avea de a face cu **list** (lista) și **dict** (dictionare) în principal.



Procesarea poate fi făcută de maniera rețetelor de bucătărie folosind instrucțiuni de prelucrare (**algoritmi**, **programare procedurală**) sau poate fi reprezentată mental ca un apel în cascadă de funcții (programare **funcțională**). **Cele două stiluri pot fi combinate**, cel funcțional e considerat mai curat.

Limbajele de programare conțin instrucțiuni atât pentru declararea datelor cât și a operațiilor pe acestea și au anumite moduri de a fi scrise în mod optimal (rețete de lucru - **idioms**).

Odată cu apariția multor librării o bună parte din programare s-a mutat din sfera programării **imperative** (în care programatorul comandă ce să se întâmple, la programarea **declarativă** - în care pur și simplu programatorul descrie rezultatul dorit, iar o librărie ”are grijă” să îndeplinească (când și cum știe ea).



ALGORITMI

Permit descopunerea problemei propuse în pași.

Un algoritm = **o secvență de pași.**

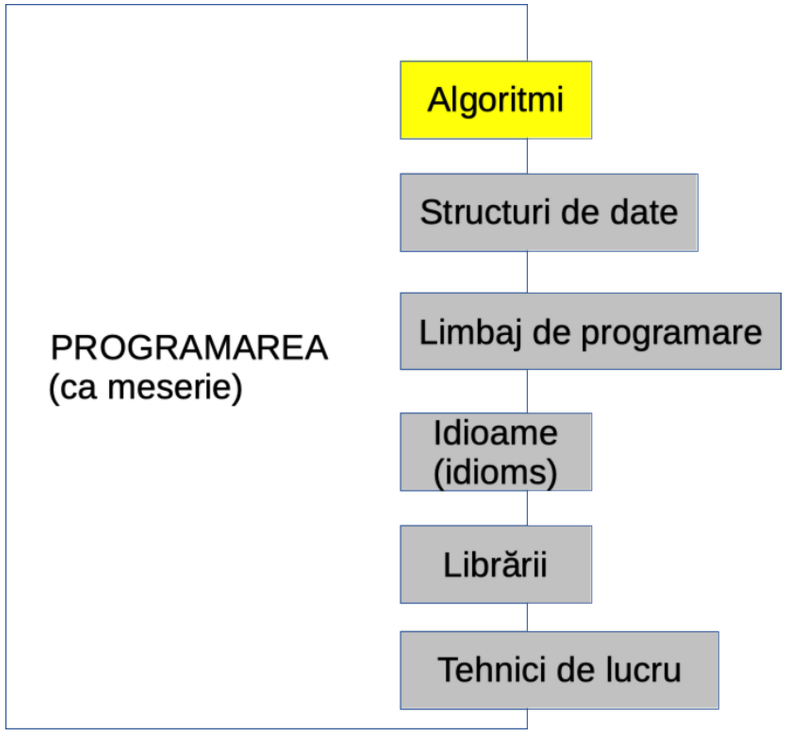
Oamenii folosesc în mod obișnuit algoritmi pentru rezolvarea problemelor de zi cu zi.

O clasă de algoritmi o reprezintă rețetele de bucate (de gătit).

Exemplu: Rețeta pentru salată boeuf

1. Se curăță ingredientele
2. Se fierb ingredientele
3. Se toacă ingredientele
4. Se prepară maioneza
5. Se amestecă maioneza cu ingredientele

Pregătirea salatei = o secvență de pași.



Concluzie. Algoritmii se compun din blocuri de instrucțiuni executate în secvență.
Unele instrucțiuni de execută de mai multe ori, repetitiv (bloc **while**) altele se execută numai dacă anumite condiții sunt îndeplinite (**if**).

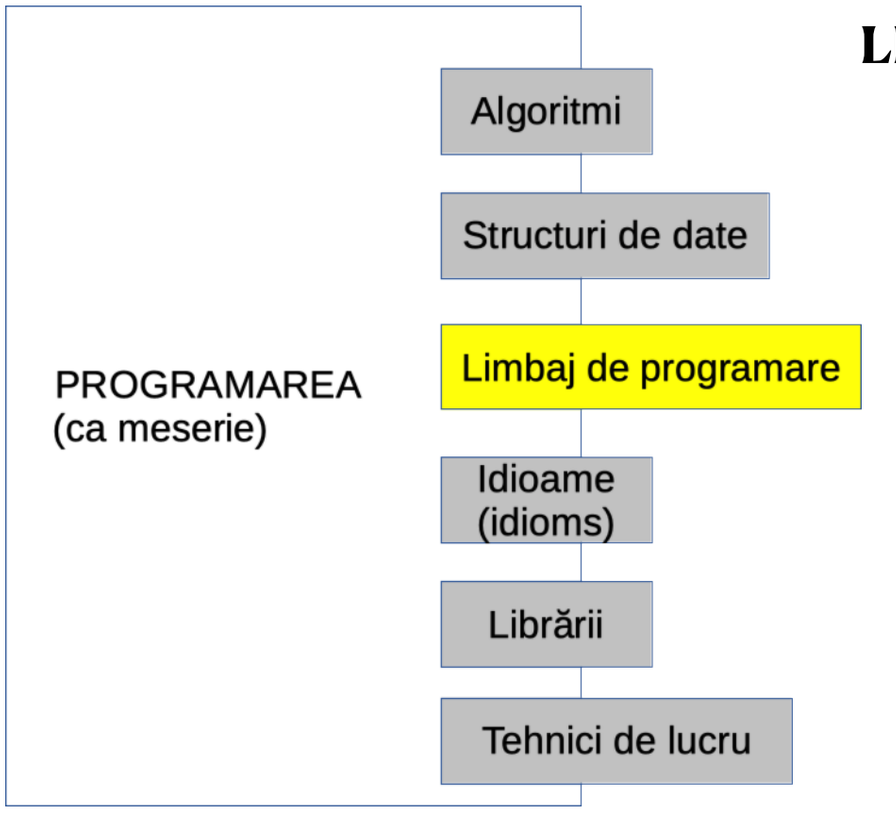
ALGORITMI

Pregătirea maionezei nu mai este o secvență liniară de pași ca la pregătirea salatei.

1. Se fierb ouăle
2. Se amestecă gălbenușurile fierte cu cele nefierte
3. Atâta timp cât este ulei de pus
    a. Se pune o linguriță de ulei în vas
    b. Se amestecă în același sens
    c. Se reia de la pasul 3

Obs: instrucțiunea 3 e compusă din 3.a, 3.b și 3.c.
Obs2: instrucțiunile 3.a, 3.b și 3.c se execută atâta timp cât (**while**) condiția 3 este adevărată

Problema se complică dacă maioneza se taie.
Dacă (**if**) avem cazul acesta se ia din maioneza tăiată și se pune iar nu din cana cu ulei.



Obs. Datorită faptului ca e weak type anumite erori se pot determina doar la execuție, ceea ce este f mare consumator de timp.
Pentru a contra-balansa acest lucru dar mai cu seamă datorită valorii adăugate de această tehnică vom folosi **TDD**: (T)est (D)riven (D)evelopment.

LIMBAJ DE PROGRAMARE

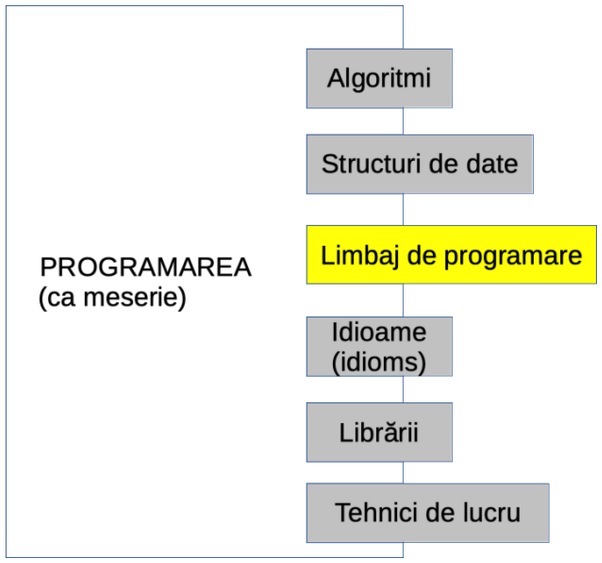
Ca limbaj de programare am ales **python**.

Python este cel mai popular limbaj de programare al momentului cu o multitudine de aplicații:
- machine learning
- scripturi de administrare
- site-uri web django
- aplicații în general

Avantaje:
- se scrie foarte puțin
- flexibil - datorită faptului ca e weak type
- type inference (deducerea tipului variabilei din context)

Dezavantaje:
- e weak type - compilatorul nu poate face validări pe bază de tip multe erori se detectează la rulare

LIMBAJ DE PROGRAMARE



Orice limbaj de programare oferă suport executării pașilor unui algoritm punând la dispoziție instrucțiuni de programare.

Unele instrucțiuni sunt simple, altele compuse.

Ca instrucțiuni simple avem:

- atribuire (se asociază unui nume o valoare)

```
Ex: a = 7
mesaj="Rezultat eronat"
```

- importul unei biblioteci

Ex:

```
import unittest
```

- întoarcerea unei valori simple dintr-o funcție

Ex:

```
return True
return 1
```

Ca instrucțiuni compuse avem:

- Expresii numerice, booleene, șiruri de caractere
- Instrucțiuni repetitive (**`while`**, **`for`**)
- Instrucțiuni de decizie (**`if`**)
- Definirea de funcții (**`def`**)
- Definirea unei clase (**`class`**)
- Apelarea unei funcții sau metode

PYTHON. EXPRESII

În mod uzual se folosesc expresii care se reduc la o valoare simplă fix în același mod în care rezolvăm o problemă de algebră în care înlocuim variabilele cu valori și executăm operațiile matematice.

Ex: x^2 + 3x + 5 calculăm delta
delta = b^2 - 4ac = **3**^2 - 4***1****5** = 9 - 20 = -11

După tip avem expresii de tip:

- număr întreg
- număr float
- șir de caractere
- boolean (True sau False)

Expresiile se pot folosi oriunde o constantă de tip întreg, float, șir sau boolean poate fi folosită.

PYTHON. INSTRUCȚIUNI COMPUSE

```
if <cond>:
  <bloc> #pt cond adevarata
elif <cond2>:
  <bloc2> #pt cond2 adevarata
else:
  <bloc3> #pt cond si cond2 false
```

cond, cond2 - conditii booleene
bloc, bloc2, bloc3 - blocuri de instructiuni

```
for <name> in <iterator>:
  <block>
```

<name> - nume variabila
<iterator> - un iterator.

Def: o sursa de date, finita (sau infinita! cum e multimea N a nr naturale), care intoarce o noua valoare la fiecare apel. De obicei este o lista. Poate fi un interator pe cheile unui dictionar sau pe perechile unui dictionar. In toate aceste cazuri particulare iteratorul va ajunge in situatia in care nu va mai avea ce sa intoarca si atunci va arunca o exceptie care e prinsa de instructiunea for si se trece la instructiunea urmatoare lui for.

Script python

Pasul 1. Cream un script care este un fisier de forma

<nume_problema_de_rezolvat>.py

Obs: Nu este

numeProblemaDeRezolvat.py , numele de fisier sunt scrise cu lowercase.

Pasul 2. este sa evidentiem partea destinata testarii. Acea parte a scriptului va fi executata doar daca scriptul este invocat (chemat) direct cu python3 <nume_problema_de_rezolvat>.py

```
if __name__ == “__main__”:
  <bloc_de_test>
```

Pasul 3. Ne pregatim sa folosim TDD