# Argumentarea proiectului

Tema acestui proiect la informatică este “Jocul de şah”.

Am ales această temă deoarece jocul necesită o gândire dezvoltată datorită posibilelor strategii care pot fi aplicate in timpul unui meci de şah.

Jocul de sah pe calculator are o istorie destul de vasta. La început, jocul calculatoarelor era modest, însă, cu timpul, ca urmare a creşterii performanţelor calculatoarelor, dar şi a muncii asidue a unor echipe de programatori renumiţi, programele de şah au început să joace la fel de bine ca maeştrii de şah.

După anul 2000, a apărut şi minunea, s-a putut crea un program de şah (BLUE DEPTH - rulat pe un supercomputer care făcea miliarde de mutări pe secundă) să bată, în premieră pe Kasparov, campion mondial de şah *en titre* la acea vreme.

Pe lângă o petrecere placută a timpului, şahul este o preocupare care dezvoltă intelectul copiilor şi al adulţilor, face ca ei să gândească mai temeinic, să nu acţioneze impulsiv la condiţiile exterioare, ci să reflecteze temeinic la calea cea mai bună de urmat, să prevadă ce se întâmplă cu o afacere care evoluează după o succesiune de multe evenimente.

# Descrierea limbajului de programare folosit

**Python** este un limbaj de programare dinamic, de nivel înalt, ce pune accent pe expresivitatea și înțelegerea ușoară a codului. Sintaxa sa permite implementări echivalente cu alte limbaje în mai puține linii de cod. Datorită acestui fapt, Python este foarte răspândit atât în programarea de aplicații, cât și în zona de scripting.

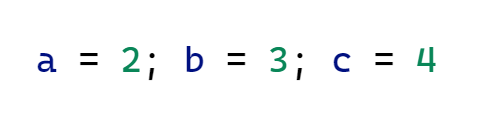
Limbajul facilitează mai multe paradigme de programare, în special paradigma imperativa (C) și pe cea orientată pe obiecte (Java). Spre deosebire de C, Python nu este un limbaj compilat, ci interpretat. Acest fapt are atât avantaje, cât și dezavantaje. Pe de-o parte, Python este mai lent decât C. Pe de altă parte, aplicațiile Python sunt foarte ușor de depanat, codul putând fi ușor inspectat în timpul rulării. De asemenea, este foarte ușor de experimentat cu mici fragmente de cod folosind interpretorul Python.

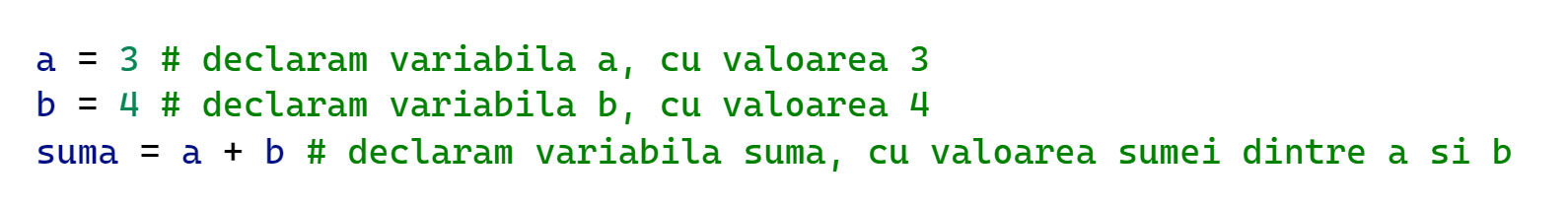
Sintaxa este gândită în așa fel încât programele Python să fie ușor de citit. Acest lucru este obținut prin folosirea de cuvinte în locul semnelor (de exemplu, and în loc de &&) și prin includerea indentării în limbaj. Astfel, în Python nu se folosesc acolade (ca în C/C++, Java), ci blocurile de cod se delimitează prin indentare. Programele Python sunt, de multe ori, foarte aproape de o “implementare” echivalentă în pseudocod.

Limbajul Python este interpretat, nu compilat. Asta înseamnă că programele Python sunt transformate într-un limbaj intermediar. Acest lucru permite codului să fie ușor de portat pe diverse sisteme de operare și arhitecturi hardware.

Codul este executat linie cu linie. Astfel, dacă - de exemplu - apelăm o funcție care nu există, vom primi un mesaj de eroare abia când se încearcă executarea liniei respective. Erorile de sintaxă sunt raportate însă înainte de rularea programului.

## **Notiunile de baza in Python**

În Python nu este necesară folosirea de ; la sfârșitul unei instrucțiuni. Folosim o linie pentru fiecare instrucțiune. În cazul în care dorim să scriem mai multe instrucțiuni pe aceeași linie, putem folosi ; ca în exemplu:

Python este un limbaj dinamic și nu lucrăm direct cu tipuri. Ele există, 2 este în continuare int, dar programatorul nu trebuie să specifice acest lucru.

**Operatori aritmetici:**

* + - adunare
* - - scădere
* \* - înmulțire
* / - împărțire
* % - operatorul modulo; a % b întoarce restul împărțirii lui a la b
* \*\* - exponent; a \*\* b este echivalent cu a la puterea b
* // - împărțire în care rezultatul se trunchiază; de exemplu: 9.0 // 2.0 = 4.0, în timp ce 9.0 / 2.0 = 4.5

**Operatori de comparare:**

Similar cu alte limbaje, aceștia sunt: *==, !=, <>, <, >, <=, >=.*

**Operatori de atribuire**

Cel mai des folosit este operatorul simplu de atribuire =, care atribuie variabilei din stânga lui valoarea expresiei din dreapta lui. Pe lângă acesta, s-au definit o serie de operatori compuși astfel: *operator aritmetic + operatorul simplu de atribuire*. Exemplu:

* += - realizează o adunare și o atribuire, astfel: a += b este echivalent cu a = a + b
* %= - asignează operandului din stânga restul împărțirii acestuia la operandul din dreapta, astfel: a %= b este echivalent cu a = a % b

Similar funcționează și operatorii: -=, \*=, \=, \*\*=, //=.

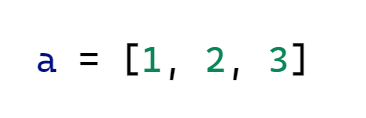
**Operatori logici**

De remarcat că acești operatori sunt foarte aproape de limbajul natural, ceea ce face codul ușor de citit și înțeles:

* and - operatorul ȘI
* or - operatorul SAU
* not - operatorul negație

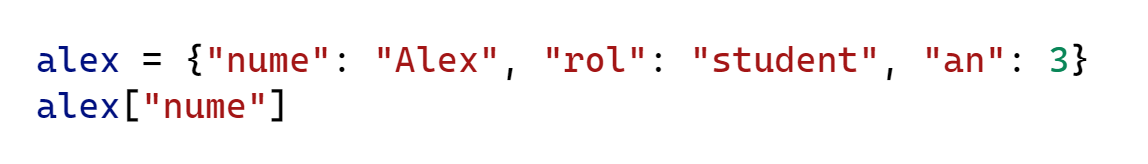
**Liste (Vectori)**

Listele sunt echivalentul vectorilor dinamici din STL (din C++). Ele sunt similare cu string-urile.

Se pot declara folosind sintaxa:

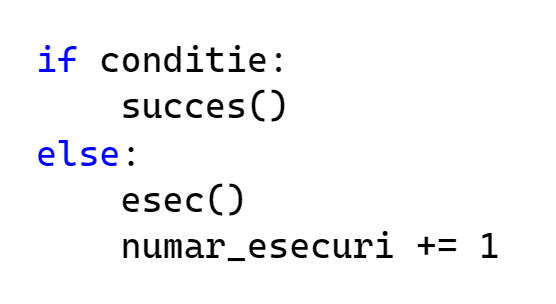
**Dicționare**

Dicționarele sunt asocieri *cheie - valoare*, asemănător hashmap-ului din STL. Cheile pot fi string-uri, numere, sau alte obiecte nemodificabile (de exemplu nu putem folosi o listă drept cheie). Valorile pot fi orice.



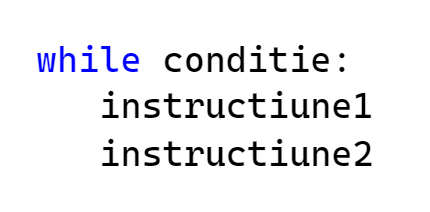
## Tipuri de instrucţiuni

### **Condiționale**

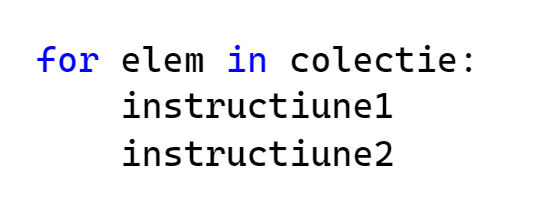
Condiționalele sunt expresii care întorc o valoare A sau o valoare B în funcție de valoarea de adevăr a unei expresii. Precum în majoritatea limbajelor de programare, și în Python există structura if/else.

După cum se observă, este folosit : în loc de {, iar blocurile de cod ce vor fi executate pe ramurile respective sunt indentate. De asemenea, nu este nevoie de paranteze în jurul condiției verificate.

### **While**

Structura while din Python este la fel ca cea din C, ținând cont de diferențele de sintaxă:

### **For**

For este diferit în Python. Sintaxa este:

*elem* va lua pe rând ca valoare fiecare element din colecție.

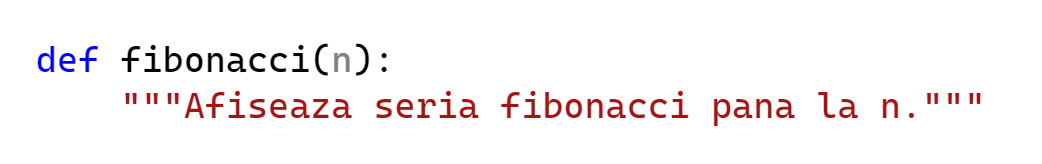
## **Funcții**

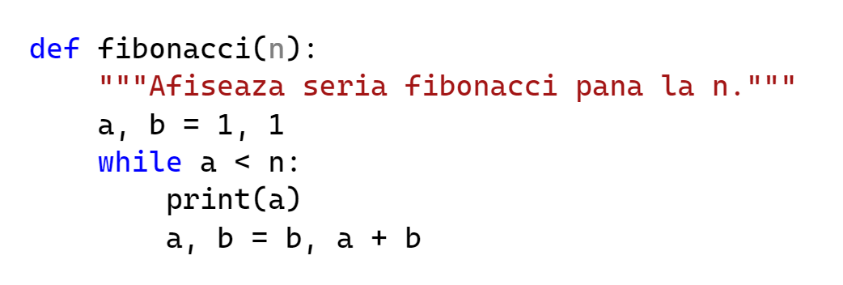
**Python** are funcții, la fel ca majoritatea celorlalte limbaje de programare. În continuare vom discuta despre particularitățile modului în care sunt create și utilizate funcțiile în Python.

### Definirea unei funcții

O funcție se declară folosind cuvântul cheie *def* astfel:

Documentarea unei funcții se face printr-un *doc string*:



 Corpul unei funcții se scrie indentat: