# CALCUL NUMERIC – LABORATOR #1

## 1. ELEMENTE DE BAZĂ ÎN PYTHON

### 1.1. Liste

```
# Listele sunt colecţii de elemente stocate într-o anumită ordine.
  Elementele listei pot fi de orice tip de date: int, float, string.
#  Sintaxa: var = [elem1, elem2,...,elemn]

```

**Ex.** #1 Să se definească lista de întregi care conţine elementele $1, 3, 2, 5, 7, 8$ în ordinea precizată. Să se afişeze lista $a$.

```
a = [1,3,2,5,7,8]
print(a)
```

```
# Accesarea unui element dintr-o listă se face scriind numele listei urmată de
  paranteze pătrate în interiorul cărora se precizează poziţia elementului. În
  Python numerotarea indicilor începe de la 0.
Sintaxa: a[k] afişează elementul de pe poziţia k+1
```

**Ex.** #2 Să se afișeze elementul cu valoarea 2. Să se schimbe valoarea acestuia în 4 prin simpla atribuire. (Individual)

```
# Folosim funcţia f print dacă dorim să inserăm valoarea unei variabile într-un
  text.
# Sintaxa: print(f'Text1 {var} Text2')
```

**Ex.** #3 Să se afișeze mesajul: Elementul al treilea din lista $a$ este: (se va afișa valoarea acestuia). (Individual)

```
# În Python sunt definite câteva metode care pot fi aplicate listelor.
# 1. Metoda append adaugă element nou la sfârşitul listei.
# Sintaxa: a.append(elem_nou)
# 2. Metoda extend adaugă mai multe elemente noi.
# Sintaxa: a.extend(lista_nouă)
# 3. Metoda insert() adauga un element pe o poziţia indicată.
# Sintaxa: a.insert(indice,elem)
# 4. Instrucţiunea del înlatură un element de pe poziţia indicată.
# Sintaxa: del a[indice]
# 5. Metoda pop() înlatură ultimul element din listă, salvându-l pentru a fi
  folosit ulterior.
# Sintaxa: elem_inlaturat = a.pop()
# 6. Metoda pop poate înlătura orice element, menţionând indicele elementului
  drept parametru.
# Sintaxa: a.pop(indice)
# 7. Metoda remove înlatură un element după valoarea acestuia.
# Sintaxa: a.remove(valoare)
```

**Ex.** #4 (Individual)

- Să se adauge la sfârșitul listei $a$ elementul cu valoarea 10.
- Să se adauge simultan elementele 5,4,8.
- Să se insereze elementul cu valoare 8 pe poziția a 4-a.

- Să se șteargă elementul de pe poziția a 2-a.
- Să se elimine ultimul element din listă, afișându-se elementul scos din listă.
- Să se elimine elementul al 3-lea din listă, afișându-se elementul eliminat.
- Să se elimine din listă elementul cu valoarea 5.

```
# Tăierea sau extragerea unui grup de elemente dintr-o listă.
# 1. Extragerea unei liste dintr-o listă precizându-se indicele de pornire şi, respectiv indicile de final.
# Sintaxa: b = a[indice_init:indice_fin] (elementul cu indice_fin nu intră în lista extrasă)
# 2. a[:indice_fin] extrage toate elementele până la elementul cu indice indice_fin - 1, inclusiv.
# 3. a[indice_init:] extrage elementele începând cu  indice_init până la ultimul element.
# 4. a[-n:] extrage o listă formată din ultimele n elemente din lista a.
# 5. a[indice_init:indice_fin:pas]  extrage o sublistă indicând primul element, elementul final şi pasul. Ultimul element nu intră în sublistă.
```

**Ex.** #5 (Individual)

- Să se creeze lista $a = [2, 4, 3, 6, 8, 9, 4, 5]$. În baza listei $a$ să se creeze lista $b$ formată din elementele $[4, 3, 6, 8]$ folosind operația de tăiere de la 1..
- Să se extragă primele 5 elemente din $a$.
- Să se extragă ultimele 5 elemente prin două metode.
- Să se extragă elementele de pe poziția pară.

```
# Copierea unei liste. Copierea unei liste se face prin operaţia de tăiere de la primul până la ultimul element.
# Sintaxa: a_copy = a[:]
# Obs. Ce se întamplă dacă definim o nouă listă prin simpla atribuire? Prin scrierea c = a se asociază o referinţă nouă la acelaşi spaţiu de memorie fără a se construi o nouă listă cu un spaţiu propriu de memorie. Prin urmare, orice modificare adusă listei 'a' se va regăsi şi în lista 'b' şi invers.
```

**Ex.** #6 Fie lista de la Ex. #5. Să se creeze două liste $b, c$ în baza listei $a$, mai întâi folosind operația de tăiere, iar apoi folosind operația de atribuire. Să se modifice în lista $a$ un element și să se afișeze conținutul celor două liste $b$ și $c$. Ce observați?

```
# Funcţia range() crează un şir de numere întregi. Atentie! Variabila construită în baza funcţiei range nu este o lista, astfel că nu putem folosi metodele aferente listelor. Putem totuşi crea o listă în baza unui şir de numere folosind funcţia list()
# Sintaxa: a = range(n1,n2,pas) (ultimul element nu intră în şir)
# b = list(a) construieste o listă în baza şirului de numere.
```

**Ex.** #7 Să se construiască un șir de numere format din numerele pare de la 2 la 20. Să se creeze o listă în baza șirului de numere. (Individual)

```
# Elementele unor liste pot fi la rândul lor alte liste.
# Exemplu:
A = [[1,2,3],['a','b','n']]
A[0] # Extrage prima listă din lista mare.
A[0][0] # Extrage primul element din prima listă.
```

### 1.2. Structuri de date de tip array

```
# Construirea obiectelor de tip array din modulul array.
```

```
# funcţia array construieşte tablouri unidimensionale sau n-dimensionale având elemente de acelaşi tip de date. Obiectele de tip array se aseamănă foarte mult cu listele, excepţie fiind precizarea tipului de date al elementelor ca o condiţie restrictivă. Toate metodele/funcţiile utilizate în cazul listelor se pot extinde şi la tablourile de tip array.
#Sintaxa:
#import array as arr
# a = arr.array('tip_date',[elem1,elem2,...,elemn])
# Printre valorile pe care le poate lua tip_date pot fi 'i' pentru numere întregi, 'f' sau 'd' pentru numere zecimale. Pentru memorarea variabilelor de tip 'd' se folosesc minim 8 biti (precizie mai mare), în comparaţie cu tipul de date 'f' care foloseşte minim 4 biţi.
# Exemplu:
B = arr.array('d',[1.3,2.4,5.5])
C = arr.array('i',[1,2,5])
```

### 1.3. Tupli

```
# Tuplul este o listă care ramane mereu aceeaşi (nu poate fi modificată decât printr-o nouă reatribuire). Un tuplu se defineşte exact ca o listă exeptând faptul că se folosesc paranteze rotunde în loc de cele drepte.
# Sintaxa: a = (elem1, elem2,...,elemn)
#Exemplu:
dimensiune = (10,20,30)
#Afişarea fiecărul element în parte
print(dimensiune[0])
print(dimensiune[1])
print(dimensiune[2])
# Dacă scriem dimensiune[0] = 30 vom obţine eroare la rulare. Tuplul nu acceptă să i se altereze elementele.
```

### 1.4. Dicționare

```

# Definirea dictionarelor. Dicţionarele sunt obiecte care stochează un volum mare de informaţii. Acestea pun în corelaţie o listă de cuvinte cu intelesul/valoarea acestora.
#Sintaxa: dictionar = {'Cheia1':valoarea1, 'Cheia2':valoarea2,...,'Cheian':valoarean}
```

**Ex.** #8 Să se construiască un dicționar *datepers* care să conțină drept listă de cuvinte: 'Nume', 'Prenume','Varsta', iar valorile lor fiind datele dumneavoastră. Să se afișeze dicționarul creat.

```
# Accesare unui element din dicţionar se face cu sintaxa date_pers['Cheia'].
# Adăugarea unei noi informaţii: nume_dict['Cheie_noua'] = valoare
```

**Ex.** #9 Să se adauge dicționarului *datepers* o nouă intrare 'inaltime' având drept valoare, înălțimea dumneavoastră. Să se afișeze dicționarul cu toate intrările.

## 2. MODULUL NUMPY

### 2.1. Definirea vectorilor/matricilor folosind modulul numpy.

```
import numpy as np
# Prin vector înţelegem un tablou unidimenional, iar prin matrice înţelegem un tablou bidimensional.
# Funcţia np.array() construieste un vector în baza unei liste sau o matrice în baza unei liste de liste.
# Exemplu:
a = np.array([1,2,4,6])
b = np.array([[1],[2],[4],[2]])
```

```
A = np.array([[1,2,3],[4,3,2],[2,3,6]])
# 'a' reprezintă un vector (tablou unidimensional)
# 'b' reprezintă matrice cu o singură coloană
# 'A' reprezintă o matrice cu 3 linii şi 3 coloane
# Accesarea unui element dintr-o matrice se va face fie folosind sintaxa similară listelor, fie folosind sintaxa A[indice1,indice2]. În cazul unui vector este suficient să precizăm poziţia elementului, i.e. a[indice].
A[0][0] #similar listelor, sau
A[0,0] #folosind doi indici scrişi între paranteze drepte
a[1]
```

### 2.2. Definirea tablourilor n dimensionale folosind modulul numpy.

```
# Similar matricelor se pot construi tablouri cu mai multe dimensiuni.
# Exemplu:
B = np.array([[[1,2],[2,4],[3,5]],[[4,3],[3,4],[4,2]],[[2,3],[3,5],[4,6]]])
# Obs.: np.array în acest caz are drept parametru o listă cu 3 liste fiecare listă având la rândul ei 3 elemente de tip listă cu 2 elemente.
# Pentru a extrage un element din tabloul B, vom indica succesiv poziţia elementului în fiecare listă pornind cu cea superioară.
B[0][0][1]
# Se extrage mai întâi lista  [[1,2],[2,4],[3,5]] din care se extrage lista inferioară [1,2] şi în final se alege elementul cu indice 1, adică elementul cu valoarea 2.
```

### 2.3. Câteva metode/funcții specifice tablourilor (vectori, matrice, tablouri n dimensionale).

#### 1. Afișarea dimensiunii vectorilor, matricilor și a tablourilor n dimensionale.

```
# Sintaxa: a.shape - returnează un tuplu cu valori egale cu numărul de elemente pe fiecare axă/dimensiune.
a = np.array([1,2,4,6])
b = np.array([[1],[2],[4],[2]])
A = np.array([[1,2,3],[4,3,2],[2,3,6]])
B = np.array([[[1,2],[2,4],[3,5]],[[4,3],[3,4],[4,2]],[[2,3],[3,5],[4,6]]])
print(f'Dimensiunea vectorului \n {a} \n este tuplul cu un singur element {a.shape}','\n')
print(f'Dimensiunea matricei de o singură coloană \n {b} \n este reprezentată de valorile tuplului cu două elemente {b.shape}','\n')
print(f'Dimensiunea matricei \n {A} \n este reprezentată de valorile tuplului cu două elemente {A.shape}','\n')
print(f'Dimensiunea tabloului \n {B} \n este reprezentată de valorile tuplului cu trei elemente {B.shape}','\n')
```

#### 2. Numărul de elemente ale unui tablou.

```
# Metoda size returneaza numarul de elemente dintr-un tablou.
B.size
#Obs.: Un tablou de tipul (m,n,p) are mxnxp elemente.
```

#### 3. Numărul de axe/dimensiuni ale unui tablou.

```
# Un vector are o singură axă, matricea- 2 axe, iar un tablou n - dimensional are n axe.
a.ndim
b.ndim
A.ndim
B.ndim
```

#### 4. Setarea tipului de date pentru tablouri.

```
print('Generarea unei matrice cu elemente de tip float')
A = np.array([[1,2,3],[4,3,2],[2,3,6]], dtype =float)
print(A)

print('Generarea unei matrice cu elemente de tip întreg')
A = np.array([[1,2,3],[4,3,2],[2,3,6]], dtype = int)
print(A)

A = np.array([[1,2,3],[4,3,2],[2,3,6]], dtype = complex)
print('Generarea unei matrice cu elemente de tip complex')
print(A)
```

### 5. Alte funcții folosite pentru generarea matricilor.

```
#1. np.zeros creeaza o matrice cu toate elementele egale cu zero
# Sintaxa: np.zeros((n,m)) - parametrul este un tuplu, reprezentând dimensiunea tabloului
A1 = np.zeros((2,3)) #  Matrice nulă cu 2 linii şi 3 coloane
#2. np.ones creează o matrice cu toate elementele egale cu 1.
#Sintaxa: np.ones((n,m))
A2 = np.ones((2,3)) # Matrice cu 2 linii şi 3 coloane, elementele căreia sunt toate egale cu 1.
#3. np.diag generează o matrice cu diagonala dată ca parametru
#Sintaxa: np.diag(lista)
A3 = np.diag([1,2,4,5]) # Matrice având diagonala egală cu [1,2,4,5]
#4. Crearea unor matrice care moştenesc proprietaţile altor matrice.
# Câte odată este necear de a creea noi matrice care sa aibă aceleaşi proprietăţi (formă, tip de date) ca a altor matrice.
A = np.array([[1,2,3],[4,3,2],[2,3,6]], dtype =float)
A4 = np.ones_like(A)
A5 = np.zeros_like(A)
#5. Matricea identitate.
#Sintaxa: np.identity(n) - creează o matrice pătratică care are 1 pe diagonală.
A6 = np.identity(5)
```

### 6. Generarea unui vector in baza functiilor np.arange și np.linspace

```
# np.arange(m,n,pas) construieşte un vector (tablou unidimensional) având elementele m, m+pas, m+2*pas,....,m + k*pas < n
b = np.arange(3,10,2)
b
# Se afişează array([3, 5, 7, 9])

# np.linspace(amin,amax,n) - împarte intervalul [amin, amax] în n puncte, numite şi noduri. Obs.: Elementul amax intră în discretizare.
c = np.linspace(0,1,10)
print(c)
```

### 7. Atribuirea unei noi etichete la același spațiu de memorie folosind operația de 'slicing', sau construirea unui nou tablou cu un spatiu de memorie propriu, folosind funcția copy.

```
#Obs.: Dacă în cazul listelor operaţia de tăiere (slicing) genera o nouă listă cu un spaţiu propriu de memorie, în cazul tablourilor definite cu np.array acest lucru numai rămâne valabil.
a = np.array([1,2,4,5,6,7,8,3,4,5])
b = a[0:2] #Adaugă o referinţa (etichetă) la zona de memorie unde sunt stocate elementele a[0], a[1].
```

```
b[1] = 100 # Modifică continuţul stocat in spaţiul de memorie asociat a[0:2]
print(a) # Modificarea se regăseşte şi în vectorul original 'a'.
# Dacă dorim să construim un nou vector cu spaţiul propriu de memorie în baza vectorului 'a' vom folosi  metoda copy.
b = a[0:2].copy() # Construieşte o nouă structură 'b' şi orice modificare adusă acesteia, nu va afecta structura originală 'a'.
b[0] = 51
print(a) # vectorul 'a' nu se modifică.
```

### 8. Extragerea elementelor dintr-un vector dacă se precizează indicii.

```
c = a[[0,1,4,5]] # Se construieşte o nouă structură de acelasi tip cu 'a' formată din cele patru elemente a[0], a[1], a[4] şi a[5].
# Obs.: Această atribuire se comportă ca o copie, NU ca o referinţă (se creează un nou spaţiu de memorie pentru 'c')
```

### 9. Operații aritmetice cu vectori/matrici de tip ndarray.

```
# Operaţiile de adunare, scădere, înmulţire, împărţire şi ridicare la putere a matricilor de aceeaşi dimensiune sunt nişte operaţii vectorizate. Aceasta înseamnă că cele 5 operaţii acţionează la nivel de fiecare element în parte. Mai numim şi operaţii de tip 'element cu element'.
#1. Adunarea a două matrice: Cij = Aij + Bij pentru orice i, j
A = np.array([[1,2,3],[3,5,6],[2,34,4]],dtype = float)
B = np.array([[8,7,3],[3,2,6],[2,34,4]],dtype = float)
C = A + B #Adunarea uzuală a două matrice
print(C)
#2. Înmulţirea element cu element a două matrice (NU este la fel cu înmulţirea clasică dintre doua matrice): Dij = Aij * Bij
D = A * B
print(D)
#3. Împărţirea element cu element a două matrice Eij = Aij/Bij pentru orice i,j cu Bij ~= 0 (diferit de zero)
E = A/B
print(E)
#4. Ridicarea la putere: Fij = Aij^3
E = A**3
print(E)
#4. Alte funcţii pentru operaţii cu matrice element cu element
# np.add(A,B) echivalent cu A + B
# np.subtract(A,B) echivalent cu A - B
# np.multiply(A,B) echivalent cu A * B
# np.power(A,n) echivalent cu A**n
# np.remainder(A,B) păstrează restul la împărţire
# np.sign(A) semnul fiecarui element în parte, daca Aij<=0, sign(Aij) = 0, in rest sign(Aij) = 1
# np.abs(A) valoarea absoluta |Aij|
# np.floor(A) calculează cel mai aproape întreg spre -infinit,
# np.ceil(A) calculează cel mai aproape întreg spre +infinit
# np.round(A) rotunjeşte la cel mai apropiat întreg
```

### 10. Funcția np.vectorize

```
# De multe ori este nevoie să definim funcţii care operează element cu element. Funcţia np.vectorize transformă orice funcţie care acceptă ca parametri numere, într-o funcţie care  acţionează element cu element asupra vectorilor/matricilor.
#Sintaxa: np.vectorize(functie)
#Functia np.heaviside(x1,x2) este definită astfel: 1, daca x1>0; x2, dacă x1 = 0; 0, dacă x1<0
```

```
#Această funcţie acceptă ca argument un scalar, nu se poate aplica în cazul unei matrice
#Dacă dorim ca functia np.heviside() sa devină o operaţie care actionează element cu element, aplicam acestei funcţii metoda np.vectorize.

x = np.linspace(-10,10,21)
Heaviside = np.vectorize(np.heaviside)
H = Heaviside(x,0.5)
print(H)
```

### 11. Alte funcții specifice matricilor

```
#np.sum(A) = calculează suma tuturor elementelor matricei 'A'.
#np.sum(A,axis = 0) = calculeaza suma pe coloane şi returneaza un vector cu 3 elemente reprezentând maximul ficărei coloane în parte.
#np.sum(A,axis = 1) = calculeaza suma pe linii şi returnează rezultatele sub formă de vector.
#np.min(A) = calculează minimul tuturor elementelor matricei A.
#np.min(A,axis = 0) = minimul pe coloane
#np.min(A, axis = 1) = minimul pe linii, returnând atâtea valori câte linii are matricea A.
#np.all(A) returneaza True dacă toate elementele matricei A sunt nenule
#np.any(A) returneaza True dacă există cel puţin un element nul.
```

### 11. Operații cu vectori și matrici

```
#np.transpose(A) calculează transpusa matricei A
#np.dot(A,B) calculează produsul clasic între două matrice.
#Atentie! În Python operatorul * este folosit pentru înmulţirea element cu element. Pentru a calcula înmulţirea a două matrice folosim funcţia np.dot.
#np.dot se foloseste si la produsul între o matrice şi un vector, obţinându-se la rezultat un vector.
A = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
B = np.array([[8,7,3],[3,2,6],[2,34,4]])
C = np.dot(A,B)
print(C)
a = np.array([1,2,3])
b = np.dot(A,a) #b este un vector (un tablou unidimensional) care are aceeaşi formă cu vectorul a.
#np.inner(a,b) calculează produsul scalar între doi vectori
p = np.inner(a,b)
print(p)
#np.cross(a,b) calculează produsul vectorial dintre doi vectori
c = np.cross(a,b)
#np.linalg.inv(A) calculează inversa unei matrice nesingulare
Binv = np.linalg.inv(B)
print(Binv)
#np.linalg.det(A) calculează determinantul matricei A
det = np.linalg.det(A)
print(det)
```

**Ex.** #8 Să se calculeze produsul $C = BAB^{-1}$, unde $B$ este o matrice inversabilă, iar $A$ și $C$ sunt două matrice oarecare.

```
A = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
B = np.array([[7, 1, 2], [3, 3, 1], [2, 3, 1]])
C = np.dot(B, np.dot(A, np.linalg.inv(B)))
#Obs.: Pentru a evita  scrierea de funcţie în funcţie, putem folosi o alternativă a funcţiei np.dot, operatorul @.
```

```
C = B @ A @ np.linalg.inv(B)
```

### 2.4. Expresii condiționale.

```
Python este prevazut cu o serie de operatori condiţionali, cum ar fi >, <, >=, <=, == si !=. Acesti operatori actionează element cu element.
```

**Ex.** #9 Fiind dați doi vectori $a$ si $b$, în urma comparării celor doi vectori să se afiseze urmatoarele mesaje: Daca $a < b$, toate elementele vectorului $a$ sunt mai mici decat elementele vectorului $b$. Daca $a > b$, toate elementele vectorului a sunt mai mari decat elementele corespunzatoare vectorului $b$, în caz contrar, unele elemente din $a$ sunt mai mici decat corespondentii lor din $b$.

```
a = np.array([1,3,5,2,5,6])
b = np.array([3,41,4,6,1,3])
c = a < b
if np.all(c)==True:
    print('Toate elementele vectorului a sunt mai mici decat elementele corespunzatoare din vectorul b.')
elif np.any(c)==True:
    print('Unele elemente din a sunt mai mari decat corespondentii lor din b.')
else:
    print('Toate elementele vectorului a sunt mai mari decat elementele corespunzatoare din b')
```

**Ex.** #9 Folosind operatorii condiționali să se construiască vectorul $y$ conform formulei:

$$y = \begin{cases} x^3, \text{ dacă } x <= 0 \\ x^2 + 2x, \text{ dacă } x > 0 \end{cases}$$

unde $x$ reprezintă discretizarea intervalului $[-5, 5]$.

```
x = np.linspace(-5,5,20)
c1 = x <= 0
c2 = x > 0
y = c1 * x**3 + c2 * (x ** 2+2 * x)
```