CURS 4
Colecții de date

1. LISTE

Listă = o secvență de valori indexată de la 0

- **O listă are o dimensiune variabilă (elementele se pot șterge/insera în mod dinamic).**
- **Valorile pot fi de tipuri de date diferite (neomogene).**
- **Listele sunt MUTABILE => se pot modifica valorile pe care le conțin!!!**

Crearea unei liste:

```
#listă vidă
L = []
print(L)

#valori
L = [1, 2, 5, 7, 10]
print(L)

# elemente neomogene
L = [1, "Popescu Ion", 151, [9, 9, 10]]
print(L)

#secvențe de inițializare (list comprehensions)
L = [x + 1 for x in range(10)]
print(L)

#secvențe de inițializare (list comprehensions) cu placeholders (_)
L = [_ + 1 for _ in range(10)]
print(L)

#secvențe de inițializare (list comprehensions)
L = [x**2 for x in range(10) if x % 2 == 0]
print(L)

L = [x**2 if x % 2 == 0 else -x**2 for x in range(10)]
print(L)

# secvențe de inițializare (list comprehensions)
L1 = [1, 3, 5, 6, 8, 3, 13, 21]
L2 = [18, 3, 7, 5, 16]
L3 = [x for x in L1 if x in L2]
print(L3)

# secvențe de inițializare (list comprehensions)
L = [int(x) for x in input("Valori: ").split()]
print(L)

# suma cifrelor unui numar natural
print("Suma cifrelor:", sum([int(c) for c in input("x = ")]))
```

https://www.programiz.com/python-programming/list-comprehension

Accesarea elementelor unei liste:

a) prin indici pozitivi sau negativi

L = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

	0	1	2	3	4	5	6	7	8	9
L	10	20	30	40	50	60	70	80	90	100
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

L[3] == 40 == L[-7]

b) prin secvență de indici pozitivi sau negativi (slice)

L[1 : 4] == [20, 30, 40] == L[-9 : -6]

inclusiv **exclusiv** **Listele sunt mutabile!!!**

L[2] = -10 => L = [10, 20, -10, 40, 50, 60, 70, 80, 90, 100]
L[: 4] == L[0 : 4] == [10, 20, 30, 40]
L[4 :] == [50, 60, 70, 80, 90, 100]
L[:] == L
L[5 : 2] == [] **(pentru că 5 > 2)**
L[5 : 2 : -1] == [60, 50, 40]
L[: : -1] == [100, 90, 80, 70, 60, 50, 40, 30, 20, 10] **(lista oglindită)**
L[-9 : 4] == [20, 30, 40]
L[-k :] == lista formată din ultimele k elemente

L[1 : 3] = [-2, -3, -4] => L = [10, -2, -3, -4, 40, 50, 60, 70, 80, 90, 100]
L[1 : 3] = [] => L = [10, 40, 50, 60, 70, 80, 90, 100] **(ștergere)**
L[1 : 1] = [-2, -3] => L = [10, -2, -3, 20, 30, 40, 50, 60, 70, 80, 90, 100]

c) ștergerea unui element / unei secvențe

del L[1:3] **<=>** **L[1 : 3] = []**

Operatori pentru liste:

a) concatenare: +
b) concatenare și atribuire: +=
c) multiplicare (concatenare repetată): *
Exemplu: [1, 2, 3] * 3 == 3 * [1, 2, 3] = [1, 2, 3, 1, 2, 3, 1, 2, 3]
d) testarea apartenenței: in, not in
e) operatori relaționali: <, <=, >, >=, ==, !=

Elementele testate trebuie să fie comparabile!

Funcții predefinite pentru liste (apelare prin: funcție(listă)):

a) `len(lista)` — len([10, 20, 30, "abc", [1, 2, 3]]) = 5
b) `list(secvență)` — list("test") = ['t', 'e', 's', 't']
c) `min(lista) și max(lista)` — Toate elementele listei trebuie să fie comparabile între ele!

Metode pentru liste (apelare prin: listă.metodă()):

a) `count(valoare)` = numărul de apariții ale valorii respective în listă

```
L = [x % 4 for x in range(22)]
print(L)
n = L.count(2)
print(n)
```

b) `append(valoare)` = adaugă în listă valoarea respectivă

```
L = [x + 1 for x in range(5)]
print(L)

L.append("abc")
print(L)

L.append([10, 20, 30])
print(L)
```

Crearea unei liste din elemente citite:

```
import time

nr_elemente = 100_000

start = time.time()
lista = [x for x in range(nr_elemente)]
stop = time.time()
print("   Initializare: ", stop - start, "secunde")

start = time.time()
lista = []
for x in range(nr_elemente):
    lista.append(x)
stop = time.time()
print("Metoda append(): ", stop - start, "secunde")

start = time.time()
lista = []
for x in range(nr_elemente):
    lista += [x]
stop = time.time()
print("  Operatorul +=: ", stop - start, "secunde")

start = time.time()
lista = []
for x in range(nr_elemente):
    lista = lista + [x]
stop = time.time()
print("   Operatorul +: ", stop - start, "secunde")
```

Timpi de executare:

```
   Initializare:  0.0039899349212646484 secunde
Metoda append():  0.010996818542480469 secunde
  Operatorul +=:  0.012456417083740234 secunde
   Operatorul +:  7.092353820800781 secunde
```

c) `extend(secvență)` = adaugă în lista curentă, pe rând, toate elementele din secvența dată ca parametru

```
L = [x + 1 for x in range(5)]
print(L)

L.append("test")
print(L)
L.extend("test")
print(L)

L.append([10, 20, 30])
print(L)
L.extend([10, 20, 30, [40, 50]])
print(L)
```

d) `insert(poziție, valoare)` = inserează valoarea înaintea poziției indicate

```
L = [x + 1 for x in range(5)]
print(L)

L.insert(3, "test")
print(L)

L.insert(30, "abc")
print(L)
```

e) `remove(valoare)` = șterge din lista curentă prima apariție, de la stânga la dreapta, a valorii date sau lansează o eroare (ValueError) dacă valoarea nu apare în listă!

f) `index(valoare)` = furnizează poziția primei apariții, de la stânga la dreapta, a valorii date sau lansează o eroare (ValueError) dacă valoarea nu apare în listă!

```
L = [x + 1 for x in range(5)]
print(L)

x = 3
if x in L:
    p = L.index(x)
    print(x, "apare in lista pe pozitia", p)
else:
    print(x, "nu apare in lista!")
```

```
L = [x + 1 for x in range(5)]
print(L)

x = 30
try:
    p = L.index(x)
    print(x, "apare in lista pe pozitia", p)
except:
    print(x, "nu apare in lista!")
```

g) **pop(poziție)** = furnizează elementul aflat pe poziția respectivă și apoi îl șterge

```
L = [x + 1 for x in range(5)]
print(L)

p = L.pop(3)
print(p, L)

p = L.pop()
print(p, L)
```

h) **clear()** = șterge toate elementele din listă (echivalentă cu L=[])

i) **reverse()** = oglindește lista

```
L = [x + 1 for x in range(5)]
print(L)

L.reverse()
print(L)
```

```
L = [1, 2, 3, 4, 5, 6]
print(L)

aux = L[1:3]
aux.reverse()
L[1:3] = aux
print(L)
```

j) **sort()** = sortează crescător elementele listei

```
L = [1, 2, 3, 2, 1]
print(L)

L.sort()
print(L)

L.sort(reverse=True)
print(L)
```

Copierea unei liste:

Varianta greșită (se copiază doar referința):

```
a = [1, 2, 3, 4, 5, 6]

# se copiază doar referința!
b = a

print("Listele initiale:", a, b, sep="\n")

a[3] = 100
print("\nListele noi:", a, b, sep="\n")
```

Metoda copy() realizează o copie superficială (shallow copy) care rezolvă problema anterioară:

```
a = [1, 2, 3, 4, 5, 6]
b = a.copy()
print("\nListele initiale:", a, b, sep="\n")

a[3] = 100
print("\nListele noi:", a, b, sep="\n")
```

Metoda copy() realizează o copie superficială (shallow copy) care nu funcționează corect dacă lista inițială conține referințe:

```
a = [1, 2, 3, [4, 5, 6]]
b = a.copy()
print("\nListele intiale:", a, b, sep="\n")

a[2] = 100
a[3][1] = -100
print("\nListele noi:", a, b, sep="\n")
```

Metoda deepcopy() realizează o copie în profunzime (deep copy) care rezolvă toate problemele anterioare (dar este lentă!):

```
import copy
a = [1, 2, 3, [4, 5, 6]]

b = copy.deepcopy(a)
print("\nListele intiale:", a, b, sep="\n")

a[2] = 100
a[3][1] = -100
print("\nListele noi:", a, b, sep="\n")
```

Crearea unui tablou bidimensional (matrice):

```
m = 3   #numarul de linii
n = 5   #numarul de coloane

# variantă incorectă: toate liniile vor conține aceeași referință!
# a = [[0] * n] * m
a = [[0 for x in range(n)] for y in range(m)]

print("Matricea initiala:")
for linie in a:
    print(linie)

print("Matricea modificata:")
a[1][3] = 7
for linie in a:
    print(linie)
```