Tema 5.
Liste și tupluri în Python

Ce ne așteaptă?

1. Liste în Python

2. Tupluri în Python

1. Care sunt caracteristicile unei liste în Python?
2. Cum se creează listele?
3. Cum se accesează elementele unei liste?
4. Cum se adaugă elemente într-o listă?
5. Cum șterg elemente din listă?
6. Cum se reordonează elementele listei?
7. Cum se creează copia unei liste?
8. Ce operatori suportă listele?
9. Cum se creează listele prin procedura “comprehensions”?
10. Care sunt caracteristicile unui tuplu în Python?
11. Cum se creează tuplurile?
12. Cum se accesează elementele unui tuplu?
13. Ce operatori suportă tuplurile?
14. Care sunt funcțiile și metodele de bază ale tuplurilor?
15. În ce constă împachetarea și despachetarea unui tuplu?

1. Liste în Python

Particularitățile listelor în Python

- **Lista – o colecție ordonată de obiecte ce poate fi modificată.**
- **O listă este definită de următoarele caracteristici:**

- este heterogenă – poate conține elemente de același tip sau de tip diferit

- este dinamică – nu necesită specificarea lungimii, adică lungime a poate fi modificată

- este ordonată – ordinea in care sunt incluse elementele este și ordinea de afișare a acestora

- permite prezența repetată a obiectelor

- este mutabilă – permite modificare conținutului după crearea acesteia

- permite apelarea obiectelor folosind indexul acestora, inclusiv indexul negativ

- elementele listei se includ în cadrul parantezelor pătrate “[ ]” separate prin virgulă

Crearea listelor

- **Utilizând parantezele pătrate “[ ]”**

  - crearea unei liste goale

```
list1 = []
print(list1)
print(type(list1))
```

  - crearea unei liste cu elemente

```
list2 = [30, 20, "UTM", 53, True]
print(list2)
print(type(list2))
```

- **Utilizând funcția *list()***

```
string1="Pavel"
list3=list(string1)
print(list3)
print(type(list3))
```

Accesarea elementelor listei prin intermediul indexului

- **Căile de accesarea a elementelor listei:**
  - Prin utilizarea indexului
  - Prin utilizarea operatorului de feliere
  - Prin utilizarea buclelor
- **Exemplu de accesare utilizând indexul:**

```
nume = ["Nicolaev", "Pavel", "Simion"]
print(nume)
print(nume[0])
print(nume[1])
print(nume[2])
print(nume[-1])
print(nume[-2])
print(nume[-3])
print(type(nume))
print(type(nume[1]))
print(nume[4])
```

Accesarea elementelor listei prin intermediul operatorului faliere

- **Sintaxa :**

  nume_listă[start:stop:pas]
  - start – indexul de început al porțiunii selectate în cadrul listei (valoare implicită = 0)
  - stop – indexul primului element al listei care nu se include în porțiunea selectată (valoare implicită = ultimul element al listei)
  - pas – pasul de selecție a elementelor din listă (valoare implicită = 1)

- **Exemplu de utilizare:**

```
n = [1, 2, 3, 4, 5, 6]
print(n)
print(n[2:5:2])
print(n[4::2])
print(n[3:5])
print(n[:3])
print(n[:5:2])
print(n[4:1:-1])
print(n[::-1])
```

Accesarea elementelor listei prin intermediul buclei

- **Accesarea utilizând bucla for**

```
a = [100, 200, 300, 400]
for x in a:
    print(x)
```

- **Funcția len() – returnează numărul de elemente în secvență**

```
n = [1, 2, 3, 4, 5]
print(len(n))
```

- **Accesarea utilizând bucla while**

```
a = [100, 200, 300, 400]
x = 0
while x<len(a):
    print(a[x])
    x = x+1
```

Metode de adăugare a unui element în listă

- Metoda *append()* – adaugă un element la sfârșitul listei
- Sintaxa : *nume_listă*.append(*element*)
- Exemplu de utilizare:

```
lista=[]
lista.append("Nicolaev")
lista.append("Pavel")
print(lista)
```

- Metoda *insert()* – adaugă un element pe poziția cu indexul indicat
- Sintaxa : *nume_listă*.insert(*index, element*)
- Exemplu de utilizare:

```
l=[10, 20, 30, 40]
print(l)
l.insert(1, 11)
print(l)
l.insert(-1, 22)
print(l)
l.insert(10, 33)
print(l)
```

Metode de adăugare a unei secvențe în listă

- **Metoda extend() – permite adăugarea unei secvențe de elemente la finalul listei curente**
- **Sintaxa :**

  ***nume_listă*.extend(*nume_secventa*)**

- **Exemplu de utilizare:**

```
list1 = [1,2,3]
list2 = [4,5,6]
string="Pavel"
print('list1 pana la estindere:', list1)
list1.extend(list2)
print('list1 dupa adaugarea list2',
list1)
list1.extend(string)
print('list1 dupa adaugarea string',
list1)
```

Metode de ștergere a unui element în listă

- Metoda *remove()* – sterge elementul specificat din listă. Dacă sunt mai multe îl șterge pe primul, iar dacă lipsește returnează ValueError
- Sintaxa : *nume_listă*.remove(*element*)
- Exemplu de utilizare:

```
n=[1, 2, 3, 1]
n.remove(1)
print(n)
n.remove(10)
```

- Metoda *pop()* – sterge elementul de pe indexul specificat și reîntoarce acel element. Dacă indexul lipsește se șterge ultimul element, dacă este eronat returnează IndexError
- Sintaxa : *nume_listă*.pop(*index*)
- Exemplu de utilizare:

```
n=[1, 2, 3, 4, 5]
print(n)
print(n.pop(1))
print(n)
print(n.pop())
print(n)
print(n.pop(10))
```

Metode de ordonare a elementelor în listă

- Metoda revers() – inversează ordinea elementelor
- Sintaxa : *nume_listă*.reverse()
- Exemplu de utilizare:

```
n=[1, 2, 3, 4, 5]
print(n)
n.reverse()
print(n)
```

- Metoda sort() – aranjează elementele în ordine crescătoare sau alfabetică
- Sintaxa : *nume_listă*.sort()
- Exemplu de utilizare:

```
n=[1, 4, 5, 2, 3]
n.sort()
print(n)
s=['rosu', 'verde', 'alb']
s.sort()
print(s)
```

- Funcția count()– returnează numărul de apariție a elementului în secvență

```
n = [1, 2, 3, 4, 5, 5, 5, 3]
print(n.count(5))
```

Aliasing-ul și cloning-ul listei

- Aliasing – atribuirea unui alt nume. Ambele nume se vor referi la aceiași locație.
  Modificările făcute într-un listă se vor reflecta și în cealaltă
- Exemplu de utilizare:

```
x=[10, 20, 30]
y=x
print("x este", x, "iar y este", y)
print("locatia x este", id(x), "iar locatia y este",id(y))
x[1] = 99
print("x este", x, "iar y este", y)
```

- Cloning – dublarea cu un alt nume. Ambele nume se vor referi la locații diferite.
  Modificările făcute într-un listă nu se vor reflecta și în cealaltă
- Exemplu de utilizare:

```
x=[10, 20, 30]
y=x[:]
print("x este", x, "iar y este", y)
print("locatia x este", id(x), "iar locatia y este",id(y))
x[1] = 99
print("x este", x, "iar y este", y)
```

Operatori aritmetici în liste

- Concatenarea a 2 liste
- Exemplu de utilizarea:

```
a = [1, 2, 3]
b = [4, 5, 6]
c = "Pavel"
d = a + b
print(d)
e = a + c
```

- Multiplicarea unei liste
- Exemplu de utilizarea:

```
a = [1, 2, 3]
b = 3
c = "Pavel"
d = a * b
print(d)
e = a * c
```

Compararea a două liste

- **Compararea listelor numerice – se compară câte 2 elemente**

```
print([1, 2, 3] < [2, 2, 3])
print([1, 2, 3] < [1, 2, 3])
print([1, 2, 3] <= [1, 2, 3])
print([1, 2, 3] < [1, 2, 4])
print([1, 2, 3] < [0, 2, 3])
print([1, 2, 3] == [1, 2, 3])
```

- **Compararea listelor cu string-uri – se tine cont de:**
  - numărul de elemente
  - ordinea elementelor
  - continutul elementelor

```
x =["abc", "def", "ghi"]
y =["abc", "def", "ghi"]
z =["ABC", "DEF", "GHI"]
a =["abc", "def", "ghi", "jkl"]
print(x==y)
print(x==z)
print(x==a)
```

Operatori de apartenență și liste imbricate

- **Operatorii *in* și *not in* – verifică prezența sau lipsa unui element în listă**

```
x=[10, 20, 30, 40, 50]
print(20 in x)
print(20 not in x)
print(90 in x)
print(90 not in x)
```

- **Listă imbricată – listă care element al altei liste**

```
a = [80, 90]
b = [10, 20, 30, a]
print(b[0])
print(b[1])
print(b[2])
print(b[3])
```

List comprehensions

- List comprehensions – o cale simplificată de creare a unei liste prin iterația unei secvențe
- Sintaxa : *nume_listă* = [*expresie* for *x* in *secventa* if *conditie*]
- Exemplu de creare clasică a unei liste prin iterația unei secvențe

```
x = "Pavel"
y = []
for i in x:
    y.append(i*2)
print(y)
```

- Exemplu de creare a aceleași liste prin ”list comprehentions”

```
x = "Pavel"
y = [i*2 for i in x]
print(y)
```

- Exemplu de creare a liste prin ”list comprehentions” cu conditie

```
x = "Pavel"
y = [i*2 for i in x if i=="P" or i=="a"]
print(y)
```

2. Tupluri în Python

Particularitățile tuplurilor în Python

- **Tuplu (tuple) – o colecție ordonată de obiecte ce nu poate fi modificată.**
- **Un tuplu este definit de următoarele caracteristici:**

  - este heterogen – poate conține elemente de același tip sau de tip diferit
  - este ordonat – ordinea in care sunt incluse elementele este și ordinea de afișare a acestora
  - este imutabil – nu permite modificare conținutului după crearea acestuia
  - permite prezența repetată a obiectelor
  - permite apelarea obiectelor folosind indexul acestora inclusiv indexul negativ
  - elementele tuplului se includ în cadrul parantezelor rotunde “( )” separate prin virgulă

Crearea unui tuplu gol sau cu un singur element

- **Crearea unui tuplu gol**

```
tuple1 = ()
print(tuple1)
print(type(tuple1))
```

- **Crearea unui tuplu cu un element cu paranteze – vigula obligatorie**

```
tuple2 = (11,)
print(tuple2)
print(type(tuple2))
```

- **Crearea unui tuplu cu un element fără paranteze – vigula obligatorie**

```
tuple3 = 120,
print(tuple3)
print(type(tuple3))
```

Crearea unui tuplu cu mai multe elemente

- **Crearea unui tuplu cu mai multe elemente folosind cu paranteze**

```
tuple4 = (15, "Pavel", False, 120, [23,24])
print(tuple4)
print(type(tuple4))
```

- **Crearea unui tuplu cu mai multe elemente fără paranteze**

```
tuple5 = "Pavel", False, 120, [23,24]
print(tuple5)
print(type(tuple5))
```

- **Crearea unui tuplu utilizând funcția *tuple()***

```
lista = [11, 22, 33]
tuple6=tuple(lista)
print(tuple6)
print(type(tuple6))
```

Accesarea elementelor unui tuplu

- **Prin utilizarea indexului**

```
t=(10,20,30,40,50,60)
print(t[0])
print(t[-1])
print(t[100])
```

- **Prin utilizarea operatorului feliere**

```
t=(10,20,30,40,50,60)
print(t[2:5])
print(t[2:100])
print(t[::2])
```

- **Prin utilizarea buclei**

```
t=(10,20,30,40,50,60)
for i in t:
    print(i)
```

Operatori aritmetici în tuplu

- Concatenarea a 2 tupluri și crearea unui nou tuplu
- Exemplu de utilizarea:

```
a = (1, 2, 3)
b = (4, 5, 6)
c = "Pavel"
d = a + b
print(d)
e = a + c
```

- Multiplicarea unui tuplu și crearea unui nou tuplu
- Exemplu de utilizarea:

```
a = (1, 2, 3)
b = 3
c = "Pavel"
d = a * b
print(d)
e = a * c
```

Funcții și metode pentru tupluri (1)

- Funcția *len()* – returnează numărul elementelor în tuplu
- Exemplu de utilizare:

```
t=(10,20,30,40)
print(len(t))
```

- Metoda *count()* – returnează numărul de apariții a elementului specificat în tuplu
- Exemplu de utilizare:

```
t=(10,20,10,10,20)
print(t.count(10))
```

- Funcția *sorted()* – returnează o listă cu elementele sortate ale tuplu
- Exemplu de utilizare:

```
t=(40,10,30,20)
l=sorted(t)
print(t)
print(l)
```

Funcții și metode pentru tupluri (2)

- Metoda *index()* – returnează indexul elementului specificat sau ValueError dacă lipsește
- Exemplu de utilizare:

```
t=(10,20,10,10,20)
print(t.index(10))
print(t.index(30))
```

- Funcția *max()* – returnează elementul de valoare maximă
- Exemplu de utilizare:

```
t=(40,10,30,20)
print(t)
print(max(t))
```

- Funcția *min()* – returnează elementul de valoare minimă
- Exemplu de utilizare:

```
t=(40,10,30,20)
print(t)
print(min(t))
```

Împachetarea și despachetarea tuple

- **Împachetarea – crearea unui tuplu dintr-un grup de variabile**
- **Exemplu de utilizare**

```
a=10
b=20
c=30
d=40
t = a, b, c, d
print(t)
```

- **Despachetarea – crearea unui grup de variabile conform elementelor unui tuplu**
- **Exemplu de utilizare**

```
t=(10, 20, 30, 40)
a, b, c, d = t
print("a=",a , "b=", b," c=", c ,"d=",d)
```

Tuple comprehensions

- **Tuple comprehensions – nu este suportat în Pyton returnând un generator nu un tuplu**
- **Exemplu de creare a generator prin ”tuple comprehentions”**

```
x = [1,2,3,4,5,6]
y = (i*2 for i in x)
print(type(y))
for j in y:
    print(j)
```