Lista és for ciklus

2025

1 List és tuple adatszerkezetek

A `list` és a `tuple` mindkettő olyan adatszerkezetek a Python programozási nyelvben, amelyek segítségével több elemet tárolhatunk egyetlen változóban. Azonban közöttük több fontos különbség is van:

1. **Módosíthatóság:**
   - `list`: A lista (list) módosítható. Ez azt jelenti, hogy hozzáadhatunk, eltávolíthatunk vagy módosíthatunk elemeket a listában.
   - `tuple`: A tuple nem módosítható. Miután létrehoztuk, nem változtathatjuk meg annak tartalmát. A tuple azonban hatékonyabb és kisebb memóriafoglalást igényel, mint a lista.
2. **Szintaxis:**
   - `list`: A lista szögletes `[ ]` zárójelek között felsorolt elemekből áll. `list = [1, 2, 3]`
   - `tuple`: A tuple sima `( )` zárójelek között felsorolt elemekből áll. `tuple = (1, 2, 3)`
3. **Méret és Teljesítmény:**
   - Mivel a tuple nem módosítható és kisebb adatszerkezet, ezért általában gyorsabb, különösen nagy adathalmazok esetén.
4. **Felhasználás:**
   - `list`: Ha olyan adatszerkezetre van szükség, amely dinamikusan változhat, akkor a lista a megfelelő választás.
   - `tuple`: Ha olyan adatszerkezetet kívánunk létrehozni, amely nem változik vagy kisebb méretre van szükségünk, akkor a tuple-t érdemes választani.

2 Lista szerkezet

A lista egy olyan adatszerkezet, amely több elemet tartalmazhat, és az elemei tetszőleges típusúak lehetnek. Megengedett az elemek ismétlődése, és az elemek a behelyezés sorrendjében kerülnek eltárolásra, nem változik meg a sorrendjük.
A lista elemei szögletes `[ ]` zárójelek között kerülnek felsorolásra, vesszőkkel `,` elválasztva egymástól. Ha nincs benne egyetlen elem sem, akkor üres listáról beszélünk.
Listát a `list()` konstruktorral is létre lehet hozni, egy másik sorozatból.

[ ]:
```
# Lista létrehozása
listam = [1, 2.0, "három", "négy", "5", 1]
ures_lista = []
masik_lista = list((1, 2, 3, 4, 5))  # tuple konvertálása listává

```

```
# Lista elemeinek kiíratása
print("A lista elemei:", listam)
print("Üres lista:", ures_lista)
print("Típusa:", type(ures_lista))
print("Másik lista:", masik_lista)
```

2.1 Referencia

A lista egy módosítható adatszerkezet, mert az elemei csak hivatkoznak más elemekre. Sőt a lista változónak is csak egy referenciája van magára a tároló objektumra.
Az objektumra hivatkozást referenciának hívjuk. A változó ezt a hivatkozást tárolja, így köti magához az objektumot. Ugyanazt az objektumot több változó is magához kötheti.

2.2 A listákon végezhető műveletek

2.2.1 Lista mérete

A lista elemeinek száma a `len()` függvény segítségével kérdezhető le.

[ ]:
```
listam = [1, 2, 3, 4, 5]

# Lista hosszának lekérdezése
hossz = len(listam)
print(f"A lista hossza: {hossz}")
```

2.2.2 Elemek kijelölése

A lista egy módosítható (mutable) adatszerkezet, ami azt jelenti, hogy az elemek átírhatók, beszúrhatók, törölhetők.
Az elemeket a lista neve után írt index `[]` operátorral tudjuk kijelölni.
Az elemek kijelölése a `0` indexel kezdődik, és `n-1`-ig tart. Nagyobb indexű elem kijelölése (amely nem létezik) hibát fog okozni, mert kicímzünk a listából.
Negatív indexek is értelmezve vannak Pythonban, ekkor a lista utolsó elemétől visszafelé haladva kell értelmezni az indexet.

Az index operátorban megadhatunk tartományt is, amivel résztartományokat lehet kijelölni. (később részletesen ismertetésre kerül)

[ ]:
```
listam = ["alma", "körte", "banán", "narancs", "kivi", "dinnye", "mangó"]

# a lista első eleme
elso = listam[0]
print(f"Az első elem értéke: {elso}")

# a lista utolsó eleme
utolso = listam[len(listam) - 1]
utolso = listam[-1]
print(f"Az utolsó elem értéke: {utolso}")
```

```
# Lista elemeinek módosítása: narancs helyett legyen manadarin
listam[3] = "mandarin"
print("Módosított lista:", listam)
```

2.2.3 Elemek hozzáadása és beszúrása

A lista végéhez az `append` függvény segítségével adhatunk hozzá egy új elemet.
A lista végéhez az `extend` függvény vagy a `+=` operátor segítségével adhatunk hozzá több elemet.
Közbenső pozícióba az `insert` függvény segítségével szúrhatunk be új elemet, (index, elem) paraméterekkel.

[ ]:
```
listam = [1, 2, 3, 4, 5]

# Elemek hozzáadása: adjuk hozzá a 6 és 7 számokat
listam.append(6)
listam.append(7)
print("Az új lista:", listam)

# Egy másik lista elemeinek hozzáadása: adjuk hozzá a masik_lista elemeit
masik_lista = [9, 10]
listam.extend(masik_lista)
print("Hozzáadott lista:", listam)

# Egy másik lista elemeinek hozzáadása: adjuk hozzá a [11, 12] lista elemeit
listam += [11, 12]
print("Hozzáadott lista:", listam)

# Elem beszúrása egy adott pozícióba (3. helyre szúrjunk be 10-et)
listam.insert(2, 10)
print("Beszúrt elem:", listam)
```

2.2.4 Elemek eltávolítása

A `remove()` függvény eltávolítja a lista megadott elemét. Ha a megadott elem nem része a listának, akkor hibát ad (*ValueError: list.remove(x): x not in list*).
A `pop()` függvény eltávolítj a lista megadott indexű elemét. Ha olyan indexet adunk meg, amely túlmutat a lista indexein, akkor hibát ad (*IndexError: pop index out of range*).
A `pop()` függvény index nélküli meghívásával az utolsó elem fog törlödni.
A `clear()` függvény törli a lista összes elemét.

[ ]:
```
listam = [1, 2, 3, 4, 5, "valami"]

# Lista adott elemének eltávolítása (töröljök a "valami" szöveget)
listam.remove("valami")
print("Eltávolított elem:", listam)

# Lista utolsó elemének eltávolítása
listam.pop()
```

```
print("Utolsó elem törlése:", listam)

# Lista elemének index alapján való eltávolítása (töröljük a 2. elemet)
torlendo_index = 1
listam.pop(torlendo_index)
print("Elem törlése index alapján:", listam)

# Lista ürítése
listam.clear()
print("Üres lista:", listam)
```

2.2.5 Elemek keresése

Egy gyűjtemény tagságát az `in` operátorral lehet megvizsgálni. Ha a vizsgált elem része a listának, akkor igaz (True), különben hamis (False) értékkel tér vissza.
A gyűjtemény adott elemének az indexét az `index()` függvénnyel kérdezhetjük le. Ha a vizsgált elem megtalálható a gyűjteményben, akkor megadja annak indexét, különben egy hibát ad.

[ ]:
```
listam = [1, 2, 3, 4, 5]

keresett_elem = 3

# Megtalálható-e a listában a keresett elem
megtalalhato = keresett_elem in listam
print("Megtaláltó-e a listában:", megtalalhato)

if megtalalhato:
    # Keresett elem indexe a listában
    megtalalt_index = listam.index(keresett_elem)
    print("Megtalált elem indexe:", megtalalt_index)
```

2.2.6 Elemek rendezése

Ha a lista összes eleme azonos típusú, akkor a `sort` függvénnyel növekvő vagy csökkenő sorrenbe rendezhetjük az elemeket. A `reverse` paraméter `True` értékével lehet beállítani, a csökkenő sorrendet. *Szövegek esetén az abc sorrend sorrend számít, de a nagy betű előrébb van, mint a kisbetű. Számok esetén az értékek számítanak.*
A `reverse` függvénnyel pedig megfordíthatjuk az elemek sorrendjét.
Ezek a függvények magát a listát módosítják.

[ ]:
```
listam = [2, 3, 4, 5, 1]

# Lista elemeinek rendezése növekvő sorrendbe
listam.sort()
print("Rendezett lista:", listam)

# Lista elemeinek rendezése csökkenő sorrendbe
listam.sort(reverse=True)
```

```
print("Csökkenő sorrendbe rendezett lista:", listam)

# Lista elemeinek megfordítása
listam.reverse()
print("Fordított lista:", listam)
```

Listák rendezésére használható még a `sorted()` függvény is. Ez egy általános rendező függvény, amely nem csak listákra használható. A függvény nem módosítja az eredeti lista sorrendet, hanem visszaad egy rendezett sorrendű új listát.

[ ]:
```
listam = [5, 3, 4, 2, 1]

# Lista elemeinek rendezése egy új listába
rendezett_listam = sorted(listam)
print("Eredeti lista:", listam)
print("Rendezett lista:", rendezett_listam)

# Lista elemeinek rendezése egy új listába
rendezett_listam = sorted(listam, reverse=True)
print("Eredeti lista:", listam)
print("Fordított lista:", rendezett_listam)
```

2.2.7 Lista másolása

Lista szerkezet másolása során figyeljünk arra, hogy az átadott lista referenciája kerül csak lemásolásra, vagyis ebben az esetben a másolat is ugyanazt a lista szerkezetet használja.

[ ]:
```
listam = [5, 6, 4, 2, 5, 1]
masolat = listam  # nem igazi másolat, csak a referenciát kapta meg

listam.sort()
print("Lista:", listam)
print("Másolat:", masolat)  # megváltozott a sorrendje ennek is
```

Ha szeretnénk két független lista szerkezetet használni úgy, hogy a bennük lévő elemek lemásolásra kerüljenek, akkor használjuk a lista `copy()` függvényét, vagy a `list()` konstruktort.

[ ]:
```
listam = [5, 6, 4, 2, 5, 1]
# Lista másolatának készítése
masolat = listam.copy()  # vagy masolat = list(listam)

listam.sort()
print("Lista:", listam)
print("Másolat:", masolat)
```

Többszörös értékadásnál is ugyanazt a listát fogja használni minkét változó.

[ ]:
```
egyik_lista = masik_lista = [1, 2, 3]
egyik_lista.append(4)
```

```
masik_lista.append(5)
print(f"Lista elemei: {egyik_lista}")
```

2.2.8 Random modul

A `random` modul segítségével lehetőség van véletlenszerű elem vagy elemek kiválasztására, és a sorrend összekeverésére.

[ ]:
```
import random

# Véletlenszerű elem kiválasztása egy listából
lista = ["alma", "körte", "banán", "szilva"]
#kedvenc = lista[random.randint(0, len(lista) - 1)]
kivalasztott_elem = random.choice(lista)
print(f"Véletlenszerűen kiválasztott elem: {kivalasztott_elem}")

# Lista elemeinek összekeverése
random.shuffle(lista)
print(f"Kevert lista: {lista}")

# Véletlenszerű elemek ismétlődés nélküli kiválasztása egy listából, az␣
 ↪eredmény is lista
sorsolas = random.sample(lista, 3)
print(f"Véletlenszerűen kiválasztott elemek: {sorsolas}")

# Véletlenszerű elemek tobbszörös kiválasztása egy listából, az eredmény is␣
 ↪lista
elemek = random.choices(lista, k=8)
print(f"Véletlenszerűen kiválasztott elemek: {elemek}")
```

3 For ciklus

A `for` ciklus egy sorozat bejárását teszi lehetővé. Megadásra kerül a bejárandó sorozat és egy változó név, amely minden ciklus kezdetén megkapja a sorozat következő elemének az értékét. A ciklusmagban több utasítás is megadható, és behúzással jelöljük.

```
for elem in gyűjtemény:
    #ciklusmag
```

[ ]:
```
gyumolcsok = ["alma", "körte", "banán", "szilva"]

# Gyümölcsök lista elemeinek bejárása és kiíratása
for gyumolcs in gyumolcsok:
    print(gyumolcs)
```

A `for` ciklusban is használható a már tanult `break` utasítás, amely kiléptet a ciklusmagból.

[ ]:
```python
gyumolcsok = ["alma", "körte", "banán", "szilva"]

# Gyümölcsök lista elemeinek bejárása és kiíratása, kilépés ha talál banán-t
for gyumolcs in gyumolcsok:
    if gyumolcs == "banán":
        break
    print(gyumolcs)
```

Ugyancsak használható a `continue` utasítás is, amely kihagyja a ciklusmagban található utasításokat és a következő elemre lép.

[ ]:
```python
gyumolcsok = ["alma", "körte", "banán", "szilva"]

# Gyümölcsök lista elemeinek bejárása és kiíratása, átugorja a banán-t
for gyumolcs in gyumolcsok:
    if gyumolcs == "banán":
        continue
    print(gyumolcs)
```

A `for` ciklus végén szintén elhelyezhető egy `else` ág, amely csak akkor fut le, ha a ciklus nem lett megszakítva `break` utasítással.

```
for elem in gyűjtemény:
    #ciklusmag
else:
    #normál leálláskor
```

[ ]:
```python
gyumolcsok = ["alma", "körte", "banán", "szilva"]

# Gyümölcsök lista elemeinek bejárása és kiíratása
for gyumolcs in gyumolcsok:
    if gyumolcs == "banán":
        print("Van közte banán!")
        break
else:
    print("Nincs közte banán!")
```

Az `if` , `for` és `while` szerkezetek belseje nem maradhat üresen. Ha fejlesztés közben még tartalom nélküli `for` ciklusunk van, akkor hibát fog adni. A hiba elkerülése érdekében írjuk be a `pass` utasítást, hogy ne maradjon üresen, ezzel jelezve, hogy később kitöltére fog kerülni.

[ ]:
```python
for gyumolcs in gyumolcsok:
    pass  # még nincs befejezve

print("Ez már nem a for ciklus része")
```

3.1 Range

Ha egy `for` ciklust meghatározott számú alkalommal akarunk futtatni, akkor használjuk a `range()` függvényt.

A `range()` függvény egy számsorozatot ad vissza, amely alapértelmezés szerint 0-tól kezdődik, 1-gyel nő és egy megadott számig tart.

[ ]:
```
for x in range(6):
    print(x, end=" ")
print()

print("Range:", range(6))
print("Lista", list(range(6)))
print("Típus:", type(range(6)))
```

A `range()` függvény két paraméteres hívása esetén az első paraméter a kezdő érték, a második a vég érték, amit már nem ér el.
A `range()` függvény három paraméteres hívása esetén a harmadik paraméter a növekmény mértéke.

[ ]:
```
# 2-től 30-ig 3-mal növekedve
for x in range(2, 30, 3):
    print(x, end=" ")
```

4 Meglévő lista elemeinek módosítása

Egy lista kényelmesen bejárható a `for` ciklussal, azonban ha szeretnénk módosítani az elemeinek az értékét, akkor szükségünk van az adott elem indexére is.
Egyik lehetőség, hogy az indexet úgy állítjuk elő, hogy lekérdezzük a lista elemeinek számát és egy `range` segítségével elkészítjük a 0-tól elemméretig tartó tartományt.

[ ]:
```
szamok = [1, 5, 2, 7, 9, 4]

for elem in szamok:
    elem += 10  # ez nem változtatja meg a lista elemeit

print(szamok)

# Elemek módosítása index alapján: mindegyik elemnek vegyük a négyzetét
for index in range(len(szamok)): # Megfelelő index létrehozása: 0 ... n-1
  szamok[index] **=2

print(szamok)
```

Egy másik lehetőség, hogy használjuk a beépített `enumerate()` függvényt, amely segít az iteráció során az aktuális elem mellett az indexeket is követni.
Az `enumerate()` egy új iterálható objektumot (`tuple`-t) hoz létre, amely tartalmazza az (index, elem) párokat, amelyeket a `for` ciklusban fel tudunk használni.

[ ]:
```
szamok = [1, 5, 2, 7, 9, 4]

# Az enumerate 2 értékkel tér vissza (index, elem)
for index, szam in enumerate(szamok):
```

```
  print(f'Index: {index}, Szám: {szam}')
  szamok[index] = szam * index

print(szamok)
```

5 Gyakorló feladatok

1. Feladat

   Lista elemei közül keressük meg a legnagyobb és legkisebb elemet. Készítse el a feladatott while ciklussal, for ciklussal és beépített függvények segítségével is.

[ ]:
```
del min, max
# %reset -f  #ipython reset

szamok = [2, 45, 47, 48, 53, 54, 14]

# beépített megoldás
print(f"Min: {min(szamok)}")
print(f"Max: {max(szamok)}")

# while ciklus
index = 0
min = max = szamok[0]
while index < len(szamok):
    if szamok[index] < min:
        min = szamok[index]
    if szamok[index] > max:
        max = szamok[index]
    index += 1

print(f"Min: {min}")
print(f"Max: {max}")

# for ciklus
min = max = szamok[0]
for szam in szamok:
    if szam < min:
        min = szam
    if szam > max:
        max = szam

print(f"Min: {min}")
print(f"Max: {max}")
```

2. Feladat

   Adja össze egy lista elemeit és határozza meg az átlagot. Készítse el a feladatott for

ciklussal és beépített függvények segítségével is.

[ ]:
```python
szamok = [2, 45, 47, 48, 53, 54, 14]

osszeg = 0
for szam in szamok:
    osszeg += szam

atlag = osszeg / len(szamok)

print(f"Összeg: {osszeg}")
print(f"Átlag: {atlag}")

# beépített függvény
osszeg = sum(szamok)
atlag = osszeg / len(szamok)

print(f"Összeg: {osszeg}")
print(f"Átlag: {atlag}")
```

3. Feladat

   Készítsen egy ötös lottó húzó programot. Figyeljen arra, hogy kétszer nem szerepelhet ugyanaz a szám.

[ ]:
```python
import random

szamok = []

while len(szamok) < 5:
    szam = random.randint(1, 90)
    if szam not in szamok:
        szamok += [szam]

szamok.sort()
print(f"A kisorsolt számok: {szamok}")

# másik megoldás
szamok = random.sample(range(1, 90), 5)
szamok.sort()
print(f"A kisorsolt számok: {szamok}")
```

4. Feladat

   Egyik hónapban ezek voltak a napi középhőmérsékletek.
   `homersekletek = [37.9, 32.2, 35.4, 36.8, 30.5, 30.2, 28.0, 28.6, 28.3, 26.3, 24.5, 30.6, 31.7, 36.2, 29.6, 29.8, 29.1, 31.9, 35.1, 37.9, 39.9, 39.7, 39.7, 33.8, 35.3, 29.7, 33.6, 34.5, 36.6, 25.3, 20.7]`
   Határozza meg, hogy hány nap volt 30 °C felett a hőmérséklet.
   Mennyi volt az átlagos középhőmérséklet?

Volt-e olyan egymást követő 7 nap, amikor minden nap 30 °C felett volt a hőmérséklet?

[ ]:
```
homersekletek = [37.9, 32.2, 35.4, 36.8, 30.5, 30.2, 28.0, 28.6, 28.3, 26.3, 24.
↪5, 30.6, 31.7, 36.2, 29.6, 29.8,
                 29.1, 31.9, 35.1, 37.9, 39.9, 39.7, 39.7, 33.8, 35.3, 29.7, 33.
↪6, 34.5, 36.6, 25.3, 20.7]

atlag = 0
napok_szama = 0
elozo_napok_szama = 0

volt_olyan_het = False

for hom in homersekletek:
    atlag += hom
    if hom > 30:
        napok_szama += 1
        elozo_napok_szama += 1
    else:
        elozo_napok_szama = 0

    if elozo_napok_szama >= 7:
        volt_olyan_het = True

print(f"Átlagos középhőmérséklet: {atlag/len(homersekletek)}")
print(f"30 foknál melegebb napok száma: {napok_szama}")
if volt_olyan_het:
    print(f"Volt olyan hét")
else:
    print("Nem volt olyan hét")
```

5. Feladat

   Kérjen be 5 lebegőpontos számot és adja őket össze.

[ ]:
```
osszeg = 0
for i in range(5):
    szam = float(input(f"Adja meg az {i+1}. számot:"))
    osszeg += szam
print(f"Az összeg: {osszeg:.2f}")
```

6. Feladat

   Kérjen be 5 lebegőpontos számot és adja meg az átlagukat és szórásukat.

[ ]:
```
szamok = []
for i in range(5):
    szam = float(input(f"Adja meg az {i+1}. számot:"))
    szamok.append(szam)
```

```
atlag = sum(szamok) / len(szamok)
szoras = 0
for szam in szamok:
    szoras += (atlag - szam) ** 2

szoras /= len(szamok)
szoras **= 0.5

print(f"Az átlag: {atlag:.2f}")
print(f"A szórás: {szoras:.4f}")
```

7. Feladat

   Buborék rendezés módszerével rendezze növekvő sorba egy listában található számsort.

[ ]:
```
elemek = [12, 2, 4, 10, 6]

for i in range(len(elemek)):
    for j in range(len(elemek) - 1):
        if elemek[j] > elemek[j + 1]:
            elemek[j], elemek[j + 1] = elemek[j + 1], elemek[j]

print(f"Elemek rendezve: {elemek}")
```

8. Feladat

   Készítse el a következő számháromszöget for ciklusokkal: (az üres helyekre tegyen space karaktert)

```
0123456789
 012345678
  01234567
   0123456
    012345
     01234
      0123
       012
        01
         0
```

[ ]:
```
for i in range(10):

    for space in range(i):
        print(" ", end="")

    for num in range(10-i):
        print(num, end="")

    print()
```