Listák létrehozása

2025

1 Lista szeletelése (List sliced)

A lista szeletelése (sliced) azt jelenti, hogy a lista egy részhalmazát kiválasztjuk és azt egy új listában kapjuk meg.
A részhalmaz kijelölése a `[start:stop:step]` szintaxist használja, ahol: - start: A szelet kezdeti indexe (ez az index is benne lesz a szeletben). - stop: A szelet végső indexe (ez az index már nem lesz benne a szeletben). - step: A lépésköz, amellyel kijelöli az elemeket (alapértelmezett értéke 1).

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

# Egy elem kijelölése
print(f"Második elem: {gyumolcsok[1]}")
print(f"Utolsó elem: {gyumolcsok[-1]}")

# Szeletelés kezdeti és végső indexekkel
print(f"Tartomány [2:5]: {gyumolcsok[2:5]}")

# Ha nincs kezdeti vagy végső index megadva, akkor az alapértelmezett az␣
 ↪elejétől vagy a végéig tartó szeletelés
print(f"Tartomány [:4]: {gyumolcsok[:4]}")
print(f"Tartomány [2:]: {gyumolcsok[2:]}")

# Szeletelés kezdeti és végső negatív indexekkel
print(f"Tartomány [-4:-1]: {gyumolcsok[-4:-1]}")

# Szeletelés kezdeti és végső negatív indexekkel, negatív lépésközzel
print(f"Tartomány [-1:-4:-1]: {gyumolcsok[-1:-4:-1]}")

# Teljes lista átadása másolással
print(f"Összes: {gyumolcsok[::]}")

# A kijelölt elemek átírhatók, törölhetők
gyumolcsok[1:3] = ["mandarin", "málna"]
gyumolcsok[4:] = []
print(f"Módosított lista fordított sorrendben: {gyumolcsok[::-1]}")
```

2 Lista vagy tuple kicsomagolása (Unpack)

A lista vagy tuple kicsomagolása (unpacking) egy olyan mechanizmus a Pythonban, amely lehetővé teszi egy lista vagy tuple eleminek szétosztását különálló változókba vagy egyéb iterálható objektumokba. Ez a szintaxis segít egyszerűbben és olvashatóbban kezelni a listák vagy más iterálható objektumok elemeit.
A változókat vesszővel elválasztva kell felsorolni, pontosan annyit amennyi eleme van a listának.

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

# Elemek szétszedése: különböző változókba
alma, korte, banan = gyumolcsok

print(alma)
print(korte)
print(banan)
```

Amelyik változóra nincs szükségünk, annak a helyét az alulvonás _ jellel kihagyhatjuk.

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

# Elemek szétszedése: elemek kihagyásával
_, _, banan = gyumolcsok

print(banan)
```

Több elem esetén, lehetőség van a csillag * jellel jelölni a változó előtt, hogy az összes hátralévő elemet abba a változóba pakolja bele egy listaként. Egy kicsomagolásnál a csillag * karakter csak egyszer szerepelhet, de kombinálható az alulvonás _ jellel.

[ ]:
```
gyumolcsok = ("alma", "körte", "banán", "narancs", "kivi", "dinnye", "mangó")

# Elemek szétszedése: több elem listába gyűjtésével
alma, korte, *tobbi, mango = gyumolcsok

print(alma)
print(korte)
print(tobbi)
print(mango)

# Elemek szétszedése: több elem kihagyásával
alma, korte, *_, mango = gyumolcsok
```

A csillag * jellel a jobb oldalon egy becsomagolás során azt jelölhetjük, hogy az elemeket ne egyben, hanem külön-külön értelmezze.

[ ]:
```
gyumolcsok1 = ("alma", "körte", "banán")
gyumolcsok2 = ("narancs", "kivi", "dinnye", "mangó")
```

```
# listába csomagolás
gyumolcsok = [gyumolcsok1, gyumolcsok2 ]
print(gyumolcsok)

# listába csomagolás kibontott elemenként
gyumolcsok = [gyumolcsok1, *gyumolcsok2 ]
print(gyumolcsok)

# listába csomagolás kibontott elemenként
gyumolcsok = [*gyumolcsok1, *gyumolcsok2 ]
print(gyumolcsok)

# lista elemeinek kiíratása kibontott elemenként
print(*gyumolcsok, sep = ", ")
```

3 Új lista létrehozása egy meglévő alapján

Ha az a feladat, hogy egy meglévő lista elemeinek felhasználásával, szűrésével vagy módosításával hozzunk létre egy másik listát, akkor azt a következőképpen tehetjük meg.

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

# a számok négyzetei
for szam in szamok:
    negyzet.append(szam**2)

print(f"Négyzetek: {negyzet}")
```

Ha nem minden elemet szeretnénk felhasználni, akkor egy szűrési feltétellel kiválogathatjuk a szükséges elemeket.

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

# a páros számok
for szam in szamok:
  if szam % 2 == 0:
    paros_szamok.append(szam)

print(f"Páros számok: {paros_szamok}")
```

3.0.1 Listakifejezés (list comprehension)

A listakifejezés egy egyszerű és olvasható módszer arra, hogy egy új listát hozzunk létre egy meglévő lista vagy más iterálható objektum elemeinek átalakításával vagy szűrésével.

```
uj_lista = [kifejezés(elem) for elem in lista]
```

[ ]:
```
szamok = [1, 5, 2, 7, 9, 4]
# a számok négyzetei
negyzet= [szam**2 for szam in szamok]
print(f"Négyzetek: {negyzet}")
```

szűrés alkalmazása:
`uj_lista = [kifejezés(elem) for elem in lista if feltétel(elem)]`

[ ]:
```
szamok = [1, 5, 2, 7, 9, 4]
# a páros számok
paros_szamok = [szam for szam in szamok if szam % 2 == 0]
print(f"Páros számok: {paros_szamok}")
```

3.1 N elemű lista generálása

N darab azonos elemből álló lista könnyedén létrehozható a * műveleti jel segítségével.

[ ]:
```
N = 5
# N darab 0-át tartalmazó lista
vektor = [0] * N
print(vektor)
```

Használhatjuk a for ciklusos megoldást is, ebben a kezdeti értékek kiszámíthatók, nem kell feltétlenül mindegyiknek azonos értékűnek lennie.

[ ]:
```
N = 5
# N darab 0-át tartalmazó lista
vektor = [0 for _ in range(N)]
print(vektor)
# N darab négyzetszámokat tartalmazó lista
negyzet = [i**2 for i in range(N)]
print(negyzet)
```

3.2 Többdimenziós lista létrehozása

Egy lista nem csak alaptípusokat hanem más listákat (azok referenciáját) is tartalmazhat(ja). Ebben az esetben az index [] operátort kétszer kell használni, hogy a második lista elemét elérjük. Listák esetén mindig csak hivatkozunk az adatra, így akár több helyről is használhatjuk ugyanazt.

[ ]:
```
vektor1 = [0, 0, 0]
vektor2 = [1, 1, 0]
vektor3 = [2, 3, 0]

vektorok = [vektor1, vektor1, vektor2, vektor3]

vektorok[1][1] = 5

print(f"vektorok: {vektorok}")
print(f"vektor1: {vektorok[0]}")
```

```
print(f"vektor1 második eleme: {vektorok[0][1]}")
```

A fenti megfontolás alapján, ha szeretnénk készíteni egy nxm dimenziós mátrixot, akkor a * művelettel történő generálás nem lesz jó, mivel ezzel a módszerrel mindegyik sor ugyanarra az egy listára fog hivatkozni.

[ ]:
```
N, M = 4, 3

vektor = [0] * M        # ez még jó
matrix = [vektor] * N   # rossz, mert minden sor ugyanarra a vektor listára fog␣
 ↪hivatkozni

matrix[0][1] = 5 # első sor, 2. eleme
print(matrix)

#Másik próbálkozás
matrix = [[0] * M] * N  # rossz megoldás, minden sor ugyanarra fog hivatkozni

matrix[0][1] = 5 # első sor, 2. eleme
print(matrix)
```

2 dimenziós (nxm méretű) lista generálható 2 for ciklus egymásba ágyazásával, ahol a belső ciklusban az oszlopokat jelentő értékeket kell megadni, amiket a külső ciklus bepakol a sorokba.

[ ]:
```
N, M = 4, 3

matrix=[]
# 4x3-as zérus mátrix előállítása
for _ in range(N):
    sor = []
    for _ in range(M):
        sor.append(0)
    matrix.append(sor)

print(matrix)

# Elemek kiválasztása két index operátor segítségével történik
matrix[1][2] = 1  # 2. sor 3. eleme

# kiíratás soronként
print("Matrix:")
for sor in matrix:
    print(sor)
```

Itt is alkalmazhatunk rövidítést, de ebben az esetben 2 for ciklust kell egymásba ágyazni.

[ ]:
```
N, M = 4, 3
# 4x3-as zérus mátrix
```

```
matrix = [[0 for _ in range(N)] for _ in range(M)]
print(*matrix, sep="\n")

# 4x3-as egység mátrix
print("Egységmátrix:")
egyseg = [[1 if i==j else 0 for i in range(N)] for j in range(M)]
print(*egyseg, sep="\n")
```

4 Identitás vizsgálat

Az identitás vizsgálat az `is` operátor segítségével történik úgy, hogy referenciákat hasonlít össze. Ezzel el lehet dönteni, hogy két változó ugyanazt az objektumot jelöli-e.

[ ]:
```
vektor1 = [0, 0, 0]
vektor2 = vektor1
vektor3 = [0, 0, 0]

# Identitás vizsgálat
print(f"vektor1 is vektor2: {vektor1 is vektor2}")  # ugyanaz az objektum két␣
 ↪különböző névvel
print(f"vektor1 is vektor3: {vektor1 is vektor3}")  # csak az értékeik egyeznek␣
 ↪meg, de különböző objektumok

# Értékek vizsgálata
print(f"vektor1 == vektor3: {vektor1 == vektor3}")  # az értékeik megegyeznek-e
```

Az `is` operátor alkalmas típusok ellenőrzésére is. A `type(érték) is típusneve` kifejezést rövidíteni is lehet, az `isinstance()` függvénnyel.

[ ]:
```
vektor1 = [0, 0, 0]
print(type(5) is int)
print(type(5) is str)
print(type('5') is str)
print(type(vektor1) is list)

print(isinstance(5, int))
print(isinstance('5', str))
print(isinstance(vektor1, list))
```

5 Iterátor

Az iterátorok olyan objektumok, amelyek lehetővé teszik az elemeik egyenkénti feldolgozását, anélkül hogy a teljes kollekciót betöltenénk a memóriába. Az iterátorokkal a `next()` függvénnyel vagy a `for` ciklussal tudunk dolgozni.
A beépített `iter()` függvény segítségével hozhatunk létre egy meglévő gyűjteményből egy iterátort.
A `next()` függvénnyel léphetünk a következő elemre.
A végigolvasott iterátort az utolsó elem után már nem szabad újra használni.

[ ]:
```python
szamok = [1, 5, 2, 7, 9, 4]
# iterátor készítése a szamok listához
iterator = iter(szamok)
print(iterator)

# első érték lekérése
szam = next(iterator)
print(szam)

# következő érték lekérése
szam = next(iterator)
print(szam)

# további értékek lekérdezése
print("Maradék:")
for szam in iterator:
    print(szam, end=" ")

#elfogytak az elemek innentől már nem szabad , hibát ad
#szam = next(iterator)
#print(szam)
```

Az iterátorok használata kényelmes és hatékony módszer a nagy adatmennyiségek feldolgozására, különösen akkor, amikor nincs szükségünk az összes elem egyszerre történő betöltésére a memóriába.

6 Gyakorló feladatok

1. Feladat

   Készíts egy programot, amely bekéri a felhasználótól egész számokat mindaddig, amíg nem ad meg egy nullát. Szétválogatja a páros és páratlan számokat két külön listába, majd írja ki mindkét listát.

[ ]:
```python
paros = []
paratlan = []

szam =  int(input("Adjon meg egy egész számot:"))

while szam != 0:
    if szam % 2:
        paratlan.append(szam)
    else:
        paros.append(szam)
    szam =  int(input("Adjon meg egy egész számot:"))

print("Megadott páratlan számok:", paratlan)
print("Megadott páros számok:", paros)
```

2. Feladat

   Készíts egy programot, amely két elemet felcserél egy listában. A felhasználótól kérjen be két indexet, majd cseréld ki azokat az indexeket a listában. Ellenőrizze le, hogy helyes indexeket adott-e meg a felhasználó, ha nem, akkor kérje be újra.

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


a = int(input(f"Kérem adja meg az indexet 0-{len(szamok)-1} között:"))
while not (0 <= a < len(szamok)):
    a = int(input(f"Kérem adja meg az indexet 0-{len(szamok)-1} között:"))

b = int(input(f"Kérem adja meg az indexet 0-{len(szamok)-1} között:"))
while not (0 <= b < len(szamok)):
    b = int(input(f"Kérem adja meg az indexet 0-{len(szamok)-1} között:"))

szamok[a], szamok[b] = szamok[b], szamok[a]

print(szamok)
```

3. Feladat

   Készíts egy listát, majd számolja meg és írja ki, hány különböző elem szerepel a listában.

[ ]:
```
elemek = [12, 15, 12, 8, 15, 12]
egyedi_elemek = []

for elem in elemek:
    if elem not in egyedi_elemek:
        egyedi_elemek.append(elem)

print(len(egyedi_elemek), "db egyedi elem:", egyedi_elemek)
```

4. Feladat

   Készíts egy programot, amely összefűz két listát egy harmadikba. Egymás után az egyikből, majd a másikból veszünk elemet, ameddig valamelyik lista el nem fogy. Írja ki az eredménylistát.

[ ]:
```
egyik_lista = [14, 45, 78, 79, 475, 78, 78]
masik_lista = ["alma", "korte", "szilva", "banán"]

# elemek_szama = len(egyik_lista) if len(egyik_lista)<len(masik_lista) else␣
 ↪len(masik_lista)
elemek_szama = min(len(egyik_lista), len(masik_lista))

eredmeny_lista = []

for i in range(elemek_szama):
```

```
    eredmeny_lista.append(egyik_lista[i])
    eredmeny_lista.append(masik_lista[i])

print(eredmeny_lista)
```

5. Feladat

   Készítsen egy kurzuslistát. Ossza szét a hallgatókat 3-as mérőcsoportokra.

[ ]:
```
import math
hallgatok =␣
 ↪["SERXFG","XDTKVJ","8VUS8Z","GZ6JM5","ZZ8R6E","12QESL","TR44VX","CVB4FS"]

for i in range(math.ceil(len(hallgatok)/3)):
    print(hallgatok[i*3:(i+1)*3])
```

6. Feladat

   Készítsen el 4 darab (x,y) adatot tartalmazó listát. Határozza meg a síkidom kerületés és súlypontját.

[ ]:
```
import math
pontok = [(0, 0), (3, 0), (3, 4), (0, 2)]

db = len(pontok)

kerulet = 0
for i in range(db):
    dx = pontok[i][0] - pontok[i - 1][0]
    dy = pontok[i][1] - pontok[i - 1][1]
    kerulet += math.sqrt(dx * dx + dy * dy)

print("Kerület:", kerulet)

Sx = Sy = 0
T = 0
pontok += pontok[0] # bezárjuk
for i in range(db-1):
    xi = pontok[i][0]
    yi = pontok[i][1]
    xi1, yi1=  pontok[i+1]
    ter = xi * yi1 - xi1 * yi
    T += ter
    Sx += (xi + xi1) * ter
    Sy += (yi + yi1) * ter

T /= 2
Sx /= (6*T)
Sy /= (6*T)
```

```
print(f"Terület: {T}")
print(f"Súlypont: {Sx} , {Sy}")
```

7. Feladat

   A hallgatókhoz tartozó zh eredményeket tárolja el egy tuple-ben. A hallgatók, pedig kerüljenek egy listába. Keresse meg egy adott neptunkódhoz tartozó zh eredményt, és írj ki. Ha nem található a megadott neptunkód, akkor írja ki, hogy nincs ilyen.

[ ]:
```
eredmenyek = [("BATMAN", 14), ("SDFERG", 23), ("4DF5RH", 64), ("LOU7RG6", 80),␣
 ↪("SDRPH9", 67), ("SWR9IP", 34)]

keresett_neptun = "BATMAN"

for hallgato in eredmenyek:
    if hallgato[0] == keresett_neptun:
        print(f"{keresett_neptun} eredménye: {hallgato[1]}")
        break
else:
    print(f"{keresett_neptun} nem található")
```

8. Feladat

   A hallgatókhoz tartozó zh eredményeket tárolja el egy tuple-ben. A hallgatók, pedig kerüljenek egy listába. Rendezze csökkenő sorrendbe a pontszám alapján és írja ki a 3 legjobb hallgató neptunkódját.

[ ]:
```
eredmenyek = [("BATMAN", 14), ("SDFERG", 23), ("4DF5RH", 64), ("LOU7RG6", 80),␣
 ↪("SDRPH9", 67), ("SWR9IP", 34)]

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

# for i in range(3):
#     print(eredmenyek[i][0])

for eredmeny in eredmenyek[0:3]:
    print(eredmeny[0])
```