

02_PythonBev_kollekciok

August 22, 2024



1 Gyűjtemények; Konverziós lehetőségek; Adat beolvasás és kiírás

Gyűjtemények (Kollekciók): lista, tuple, szótár, halmaz. Ezek tulajdonságai. Indexelése.

Konverziós lehetőségek a különböző adattípusok közt. Például `int -> float`; `str -> int`.

Adat beolvasás és kiírás a képernyőre: `input()` és `print()`, alapvető formázási konvenciók.

1.1 Gyűjtemények (Kollekciók)

Négy különböző, a pythonban alapvető gyűjteménnyel ismerkedünk meg. Ezek a tuple, lista (list), szótár (dict) és halmaz (set) típusok.

1.1.1 Néhány általános megfontolás

- A gyűjtemény elemeket vesszővel választjuk el.
- A gyűjtemény típusát a határoló zárójel jelzi: `()` `{}` `[]`
- Egy gyűjteményt típusát is le lehet kérdezni: `type()`
- A gyűjteményben lévő elemek számát a `len()` mutatja meg
- Ha értelmes indexelni, akkor az 0-tól kezdődik, akár csak a sztringek esetében
- Tartalmazás vizsgálat az `in` kulcsszóval

1.1.2 Tuple

- A tuple természetes számokkal indexelhető, **nem módosítható tömb**.
- **Az elemeknek nem kell azonos típusúnak lenniük.**
- Az indexelés $O(1)$, a tartalmazásvizsgálat $O(n)$ időben fut le, ahol n a tuple elemszáma.
- [Tuple a python dokumentációban](#)

```
[8]: # Hozzunk létre egy t nevű, 3 elemű tuple változót!  
# Az elemeknek nem kell azonos típusúnak lenniük.  
t = (1, 2, 1.23, True, 'alma')  
t
```

```
[8]: (1, 2, 1.23, True, 'alma')
```

```
[9]: # Ellenőrizzük t típusát!  
type(t)
```

[9]: tuple

```
[10]: # Az elemek számát a len függvénnyel kérdezhetjük le.  
len(t)
```

[10]: 5

```
[11]: # Tuple elemeinek elérése (az indexelés 0-tól indul).  
t[1]
```

[11]: 2

```
[12]: # Az elemeken nem lehet módosítani!  
t[1] = 16
```

```
-----  
TypeError                                Traceback (most recent call last)  
/tmp/ipykernel_65232/1175813823.py in <module>  
      1 # Az elemeken nem lehet módosítani!  
----> 2 t[1] = 16  
  
TypeError: 'tuple' object does not support item assignment
```

```
[13]: # A t változó persze kaphat új értéket.  
t = (1, 2, 3, (12, 24))
```

```
[14]: # Tartalmazásvizsgálat.  
print(12 in t)  
print(1 in t)
```

False

True

```
[15]: # Amennyiben nem okoz kétértelműséget, a ( és ) határoló elhagyható!  
t2 = 1, 2, "alma"  
print(t2)
```

(1, 2, 'alma')

```
[16]: # Üres tuple létrehozása.  
t0 = ()  
len(t0), type(t0)
```

[16]: (0, tuple)

```
[17]: # Egy elemű tuple létrehozása. A zárójel el is hagyható.  
t1 = ("s",)  
type(t1) , len(t1)
```

```
[17]: (tuple, 1)
```

1.1.3 Lista (list)

- A tuple módosítható változata:
 - új elemet is hozzá lehet adni, ill.
 - meglévő elemeken is lehet módosítani.
- Az indexelés $O(1)$, a tartalmazásvizsgálat $O(n)$ időben fut le itt is.
- [List a python dokumentációban](#)

```
[22]: # Hozzunk létre egy l nevű, 4 elemű listaváltozót!  
# Az elemeknek nem kell azonos típusúnak lenniük.  
l = [1, 1.23, True, 'alma']  
l
```

```
[22]: [1, 1.23, True, 'alma']
```

```
[23]: # Ellenőrizzük l típusát, és kérdezzük le az elemek számát!  
type(l), len(l)
```

```
[23]: (list, 4)
```

```
[24]: # Lista elemeinek elérése (az indexelés 0-tól indul).  
l[1]
```

```
[24]: 1.23
```

```
[25]: # Listaelem módosítása.  
l[1] = 16  
l
```

```
[25]: [1, 16, True, 'alma']
```

```
[26]: # Listába elemként beágyazhatunk másik listát  
[1, 2, 'alma', [3, 4]]
```

```
[26]: [1, 2, 'alma', [3, 4]]
```

```
[27]: # Tartalmazásvizsgálat.  
'alma' in l
```

```
[27]: True
```

```
[28]: # Üres lista létrehozása.  
10 = []  
type(10), len(10)
```

```
[28]: (list, 0)
```

Lista módosítása: Eljárások: `.append()`, `.index()`, `.insert()`, `.pop()`

```
[18]: l = [1, 1.23, True, 'alma']
```

```
[19]: # Elem hozzáfűzése a lista végére: append()  
l.append(134)  
l
```

```
[19]: [1, 1.23, True, 'alma', 134]
```

```
[20]: # Elem beszúrása a lista közepére: insert()  
l.insert(1, "körte")  
l
```

```
[20]: [1, 'körte', 1.23, True, 'alma', 134]
```

```
[21]: # Elem indexének meghatározása (az első előfordulásé) .index()  
l.index("alma")
```

```
[21]: 4
```

```
[22]: # Adott indexű elem törlése .pop()  
l.pop(5)  
l
```

```
[22]: [1, 'körte', 1.23, True, 'alma']
```

```
[23]: # Utolsó elem törlése.  
l.pop()  
l
```

```
[23]: [1, 'körte', 1.23, True]
```

```
[24]: # Két lista összefűzése egy új listába.  
[1,2,3] + ['alma','körte']
```

```
[24]: [1, 2, 3, 'alma', 'körte']
```

```
[25]: # Lista többszörözése.  
['alma','körte']*3
```

```
[25]: ['alma', 'körte', 'alma', 'körte', 'alma', 'körte']
```

1.1.4 Halmaz (set)

- A halmaz adattípus a matematikai halmazfogalom számítógépes megfelelője, azaz egy elem csak 1x szerepelhet!
- Halmazt indexelni nem lehet, a tartalmazásvizsgálat $O(1)$ időben fut le.
- [Halmaz a python dokumentációban](#)

```
[40]: # Hozzunk létre egy s nevű halmazváltozót!  
# Az elemek típusa nem feltétlenül azonos.  
  
s = {1, 1, 3, "alma", 1.25, 1,5}  
s
```

```
[40]: {1, 1.25, 3, 5, 'alma'}
```

```
[37]: # Ellenőrizzük s típusát és elemszámát!  
type(s) , len(s)
```

```
[37]: (set, 5)
```

```
[38]: # Tartalmazásvizsgálat.  
1 in s
```

```
[38]: True
```

```
[39]: 2 in s
```

```
[39]: False
```

```
[102]: # Üres halmaz létrehozása. {} nem jó, mert az szótárt készít.  
s0 = set()  
type(s0), len(s0)
```

```
[102]: (set, 0)
```

Halmaz módosítása, halmazműveletek:

- Eljárások: `.add()` és `.remove()`
- Műveletek: únió (`|`), metszet (`&`), különbség (`-`)

```
[26]: s = {1, 1, 3, "alma", 1.25, 1, 5}
```

```
[27]: # Elem hozzáadása a halmazhoz.  
s.add(12)  
print(s)
```

```
{1, 1.25, 3, 5, 'alma', 12}
```

```
[28]: # Elem eltávolítása.  
s.remove(1)  
s
```

```
[28]: {1.25, 12, 3, 5, 'alma'}
```

```
[32]: # unió  
{1, 2, 3} | {1, 5, 2}
```

```
[32]: {1, 2, 3, 5}
```

```
[33]: # metszet  
{1, 2, 3} & {1, 5, 2}
```

```
[33]: {1, 2}
```

```
[34]: # halmazkivonás  
{1, 2, 3} - {1, 5, 2}
```

```
[34]: {3}
```

1.1.5 Szótár (dict)

- A szótár kulcs-érték párok halmaza, ahol a kulcsok egyediek.
- Indexelni a kulccsal lehet, $O(1)$ időben.
- A kulcs lehet egyszerű típus, tuple vagy bármely módosíthatatlan adatszerkezet.
- [Szótár a python dokumentációban](#)

```
[ ]: # Hozzunk létre egy d nevű szótárváltozót!  
d = { "alma": 120, "körte": 12.5, "barack": 4}
```

```
[35]: # ha az olvashatóság úgy kívánja mehet a kollekció több sorba!  
d = {  
    "alma": 120,  
    "körte": 12.5,  
    "barack": 4  
}
```

```
[36]: print(d)
```

```
{'alma': 120, 'körte': 12.5, 'barack': 4}
```

```
[50]: # Ellenőrizzük le d típusát és elemszámát!  
type(d) , len(d)
```

```
[50]: (dict, 3)
```

```
[51]: # Létező kulcshoz tartozó érték lekérdezése.  
d["alma"]
```

```
[51]: 120
```

```
[107]: # Nem létező kulcshoz tartozó érték lekérdezése.  
d["meggy"]
```

```
-----  
KeyError                                Traceback (most recent call last)  
<ipython-input-107-02e06c1aef53> in <cell line: 2>()  
      1 # Nem létező kulcshoz tartozó érték lekérdezése.  
----> 2 d["meggy"]  
  
KeyError: 'meggy'
```

```
[66]: # Kulcshoz tartozó érték módosítása.  
d["alma"] = 100  
d
```

```
[66]: {'alma': 100, 'körte': 12.5, 'barack': 4}
```

```
[67]: # Új kulcs-érték pár beszúrása.  
d["barack"] = 48  
d
```

```
[67]: {'alma': 100, 'körte': 12.5, 'barack': 48}
```

```
[68]: # Kulcs-érték pár törlése.  
del d["alma"]  
d
```

```
[68]: {'körte': 12.5, 'barack': 48}
```

```
[111]: # Benne van-e egy kulcs a szótárban?  
2 in d
```

```
[111]: True
```

```
[52]: # Üres szótár létrehozása.  
d0 = {}  
type(d0)
```

```
[52]: dict
```

Néhány hasznos eljárás a szótárokkal:

- kulcsok lekérdezése: `.keys()`
- értékek lekérdezése `.values()`
- kulcs-érték párok kigyűjtése `.items()`

```
[37]: d = { "alma": 120, "körte": 12.5, "barack": 4}
```

```
[38]: # szótár kulcsok lekérdezése
d.keys()
```

```
[38]: dict_keys(['alma', 'körte', 'barack'])
```

```
[39]: # szótárban tárolt értékek lekérdezése
d.values()
```

```
[39]: dict_values([120, 12.5, 4])
```

```
[41]: # kulcs érték párok
d.items()
```

```
[41]: dict_items([('alma', 120), ('körte', 12.5), ('barack', 4)])
```

1.2 Konverzió

Minden eddig tanult adattípushoz tartozik egy függvény, amely az adott adattípusra konvertál bármely más adattípusról, amennyiben a konverciónak van értelme.

```
[53]: int(1.234)      # float => int
```

```
[53]: 1
```

```
[57]: float(1)        # int => float
```

```
[57]: 1.0
```

```
[54]: int('123')      # str => int
```

```
[54]: 123
```

```
[58]: str(123)         # int => str
```

```
[58]: '123'
```

```
[56]: float("1.234")  # str => float
```

```
[56]: 1.234
```

```
[59]: # list => tuple
l = [1, 2, 3]
```



```
tuple(1)
```

```
[59]: (1, 2, 3)
```

```
[60]: # tuple => list
t = 1, 2, 3
list(t)
```

```
[60]: [1, 2, 3]
```

```
[70]: # tuple => set
t = 1, 2, 3, 3
set(t)
```

```
[70]: {1, 2, 3}
```

```
[62]: # párok listája => dict
dl = [('alma', 1), ('körte', 2)]
dict(dl)
```

```
[62]: {'alma': 1, 'körte': 2}
```

```
[83]: # dict => párok listája
d = {'alma': 1, 'körte': 2}
list(d.items())
```

```
[83]: [('alma', 1), ('körte', 2)]
```

1.3 Adat bekérés a billentyűzetről és kiíratás a képernyőre (Standard adatfolyamok)

Az operációs rendszer indításkor minden folyamathoz hozzárendel 3 szabványos adatfolyamot: a [standard bemenetet](#), a [standard kimenetet](#), és a [standard hibakimenetet](#). Alapértelmezés szerint a standard bemenet a billentyűzettel, a standard kimenet és hibakimenet pedig a képernyővel van összekötve. Ez a beállítás módosítható, pl. a standard bemenet érkezhethet egy fájlból vagy egy másik programból, a standard kimenet és hibakimenet pedig íródhat fájlba vagy továbbítható más programnak.

1.3.1 Adat bekérés a billentyűzetről (Standard bemenet)

- A standard bemenetről adatokat bekérni az `input()` függvény segítségével lehet.
- Az eredmény sztring típusú. Ha más adattípusra van szükség, akkor a konvertálni kell.
- [input függvény a dokumentációban](#)

```
[72]: # Sztring típusú adat beolvasása.
text = input("Írjon be egy szöveget: ")
```

```
[73]: print(text)
```

Ez egy szöveg

```
[74]: # Egész típusú adat beolvasása.  
n = int(input("Adjon meg egy egész számot: "))  
print ("A megedott egész", n)
```

A megedott egész 12

1.3.2 Kiírás a képernyőre (Standard kimenet)

- A standard kimenetre kiírni a `print()` függvény segítségével lehet.
- [print függvény a dokumentációban](#)

```
[1]: # Kiírás a standard kimenetre.  
n = 200  
print(n)  
print()    # extra soremelés  
print(n)
```

200

200

```
[2]: # Kiírás soremelés nélkül.  
print(n, end = "\t")  
print(n)
```

200 200

```
[3]: # Egyetlen soremelés kiírása.  
print()
```

1.3.3 Formázott kiírás

```
[7]: x1 = 2  
x2 = 2**0.5  
  
# eddigi megoldásunk:  
print ("A megedott x1 egész", x1)  
print ("A megedott x2 valós:", x2)  
  
# Formázott kiírás f-sztringgel.  
print (f"A megedott x2 valós: {x2}")  
  
# Formázott kiírás helyettesítő karakterrel.
```

```
print ("A megedott x2 valós: %f"%(x2) )
```

```
A megedott x1 egész 2
A megedott x2 valós: 1.4142135623730951
A megedott x2 valós: 1.4142135623730951
A megedott x2 valós: 1.414214
```

```
[ ]: # Kiírás 1 tizedesjegy pontossággal.
print (f"A megedott x2 szám: {x2:0.1f}")

# Vagy 5 tizedesjegy pontossággal.
print (f"A megedott x2 szám: {x2 : 0.5f}")
```

```
A megedott x2 szám: 1.4
A megedott x2 szám: 1.41421
```

```
[ ]: # szép kiíratás egészre: 4 karakternyi helyre
n = 10
print(f"A megdott egész:{n:4}")
print(f"A megdott egész:{n*10:4}")
print(f"A megdott egész:{n*100:4}")
```

```
A megdott egész: 10
A megdott egész: 100
A megdott egész:1000
```

```
[ ]: # Egész szám ill. sztring kiírása.
n=3
s="körte"
print(f"ez az egész: {n}, ez meg a sztring: {s}")
```

```
ez az egész: 3, ez meg a sztring: körte
```